VALUE- AND DEBT-AWARE SELECTION AND COMPOSITION IN CLOUD-BASED SERVICE-ORIENTED ARCHITECTURES USING REAL OPTIONS

by

ESRA FAWAZ AHMAD ALZAGHOUL

A thesis submitted to the
University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

School of Computer Science
College of Engineering and Physical Sciences
University of Birmingham
July 2015

Abstract

This thesis presents a novel model for service selection and composition in Cloud-based Service-Oriented Architectures (CB-SOA), which is called CloudMTD, using real options, Dependency Structure Matrix (DSM) and propagation-cost metrics. CB-SOA architectures are composed of web services, which are leased or bought off the cloud marketplace. CB-SOA can improve its utility and add value to its composition by substituting its constituent services. The substitution decisions may introduce technical debt, which needs to be managed. The thesis defines the concept of technical debt for CB-SOA and reports on the available technical debt definitions and approaches in the literature. The formulation of service substitution problem and its technical debt valuation is based on options, which exploits Binomial Options Analysis. This thesis looks at different option types under uncertainty. This thesis is concerned with some scenarios that may lead to technical debt, which are related to web service selection and composition that has been driven by either a technical or a business objective. In each scenario, we are interested in three decisions (1) keep, (2) substitute or (3) abandon the current service. Each scenario takes into consideration either one or more QoS attribute dimension (e.g. Availability). We address these scenarios from an option-based perspective. Each scenario is linked to a suitable option type. A specific option type depends on the nature of the application, problem to be investigated, and the decision to be taken. In addition, we use Dependency Structure Matrix (DSM) in order to represent dependencies among web services in CB-SOA. We introduce time and complexity sensitive propagation-cost metrics to DSM to solve the problem. In addition, CloudMTD model informs the time-value of the decisions under uncertainty based on behavioral and structural aspects of CB-SOA.

To My Beloved Parents,

To Ahmad, Alaa, Aseel and Mohammad

with love...

This PhD is a fulfillment of my childhood dream, which was inspired by the PhD of my lovely dad, Prof. Fawaz Alzaghoul. My childhood dream was accomplished by using the writing skills that I gained from my lovely mom. Mom and dad, you have always been my inspiration. You have always been my "number one" supporter.

I dedicate my PhD to you both with love. (Elhamdulelah)

Acknowledgment

When you have a dream, the sky is the limit. PhD... My childhood dream... Here we go... PhD is a worthwhile journey full of experiences in many aspects of life that is not possible without support and encouragement of many people. And as the end of my journey is approaching, I would like to take the opportunity to express my sincere thanks and appreciation to all wonderful people who helped me along this journey.

I would like to thank my family for the continuous love and support. During my stay in the UK, I realized that I really CANNOT live without you! Words will not be enough to thank you and will not be enough to express how much I love you. Thanks for your continuous encouragement. Thanks for giving me the pride to do a PhD. I'm so proud to have a family like you. I dedicate this thesis to you all with love.

I'm glad to be part of the Software Engineering Research Group and having the opportunity to work under Rami Bahsoon supervision. I'm thankful to his support and guidance over my PhD years. He has always been willing to assist his students in research at anytime. I would like to thank Dr. John Bullinaria and Dr. Peter Lewis for the continuous help and support during my thesis group meetings.

Special thanks go to Reem Al-Fayez. Thanks for your continuous support. Thanks for staying by my side. I'm also thankful to Waseela and Sara Al-zu'bi. I would like to thank Salsabeel, Heba and Kaziwa. Thanks for your continuous support. Thanks for always being there during happy and sad times. I would like also to thank Niloufar, Sally, Khulood, Maria, Guru, Shereen, Siti, Sarah, Xiaofen, Shen and many others. And I would like to thank many other colleagues and friends in computer science department.

Finally, I would like to thank The University of Jordan[1] for sponsoring me, and Mrs. Enas Dababneh for her continuous help and support.

[1]ju.edu.jo/home.aspx

Publications arising from this Thesis

- Esra Alzaghoul and Rami Bahsoon. Evaluating Technical Debt in Cloud-based Architectures using Real Options. The 23rd Australasian Software Engineering Conference (ASWEC) co-located with the 11th Working IEEE/IFIP Conference on Software Architecture (WICSA), Sydney, Australia, 2014 (acceptance rate: 32%).

- Esra Alzaghoul and Rami Bahsoon. Economics-driven Approach for Managing Technical Debt in Cloud-Based Architectures. In proceeding of the sixth IEEE/ACM International Conference on Utility and Cloud Computing (UCC), Dresden, Germany, 2013(acceptance rate: 25%).

- Esra Alzaghoul and Rami Bahsoon. CloudMTD: Using Real Options to Manage Technical Debt in Cloud-Based Service Selection. In proceeding of the Fourth International Workshop on Managing Technical Debt, in conjunction with 35th International Conference on Software Engineering (ICSE), San Francisco, CA, 2013.

Contents

List of Figures

1 Introduction

1.1 Overview

Cloud-Based Service-Oriented Architectures (CB-SOA) are typically composed of services which are offered by cloud service providers in cloud marketplace. In such architectures, non-functional requirements (e.g. availability) are unpredictable as they tend to fluctuate frequently. Therefore, such requirements are likely to evolve and change over time. As a result, such change can affect not only user needs, but also other requirements in the same architecture. For CB-SOA decision makers (e.g. architects and stakeholders), coping with change in these requirements is an ultimate goal for the purpose of maximizing profits and reducing cost in general. In this sense, we argue that CB-SOA decision makers should respond to changes and act upon them as if they are new opportunities. In CB-SOA, decision makers may need to replace (substitute) one of the constituent web services, which is not satisfying requirements in some circumstances at specific time due to changes (e.g. demand change). The need for service substitution could be driven by business objectives or technical ones. For example, the need can be attributed to changes in quality of service (QoS) requirements, the need to reduce operational cost, upgrade to a new web service, etc.

On the other hand, in CB-SOA, services can be offered from various cloud service providers with different cost, QoS, features and functionalities, and varying flexibility in supporting business processes. This can make the problem of selecting and composing CB-SOA benefiting from the cloud marketplace complex and challenging. This is because the selection and composition decision need to consider multiple providers, their diverse provisions and the likely long-term value-added of a candidate selection. In this thesis, the value-added has many dimensions. For example, it can be attributed to the likely increase in revenues generated by improving a specific QoS expressed as utility.

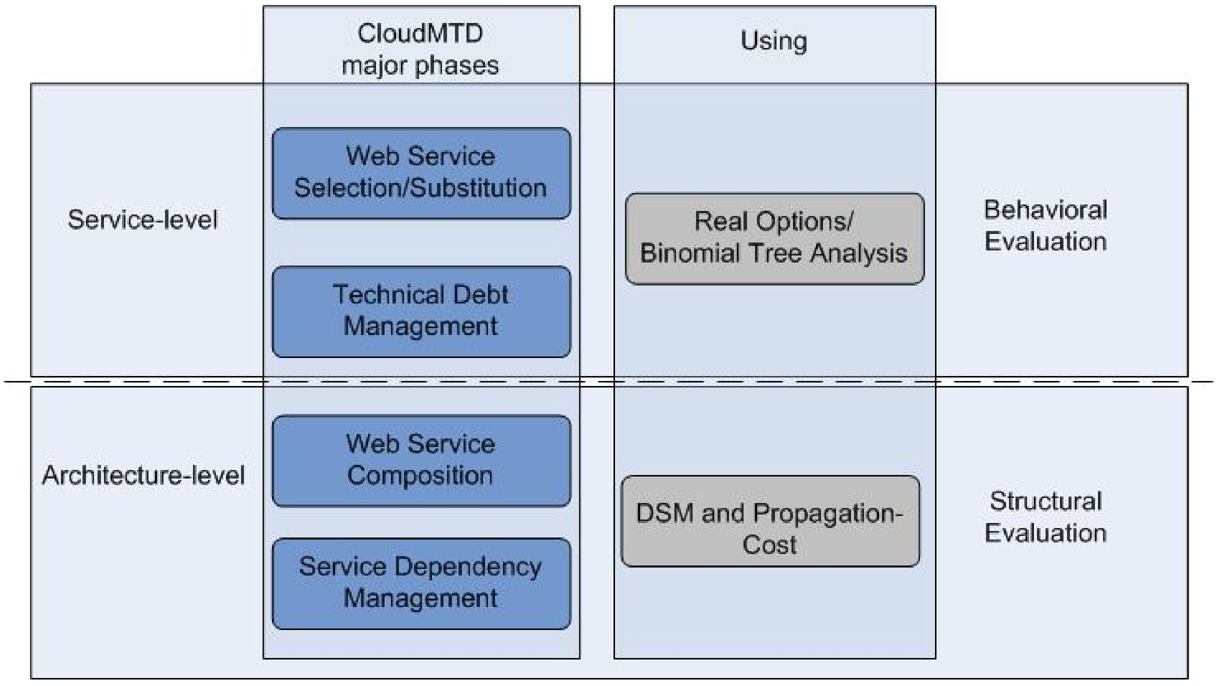
We argue that service selection decision may come with a technical debt - *an operational liability which may incur an interest if not managed, cleared and transformed from liability to value.* Technical debt can span several dimensions, which are related to selection decision or service provision. For example, technical debt can be related to situations, where the service capacity is underutilized and the cost of maintenance outweighs the revenue streamed from using the service. Technical debt can be also attributed to situations, where the service is at risk because of SLA non-compliance. In addition, technical debt can present in situations where the selection and/or substituting decisions are not fully justified for value-added. Also, it can be attributed to poor and unjustified service selection decisions, which are not long-term geared. Following a similar argument to Guo and Seaman: little level of debt "is not bad", as it can help developers speed up the development process [93]. This perspective can be valid for the case of service selection in CB-SOA, where less attractive services may come with a technical debt, which can be transformed to future value if properly managed.

This thesis presents a model (CloudMTD), which has the following major phases (Figure 1.1): (I) inform the selection and composition decisions in CB-SOA and associate it with long-term value under uncertainty, (II) technical debt management, if any, (III) inform the time-value of the decisions under uncertainty based on behavioral and structural aspects of CB-SOA, and (IV) services dependencies management. CB-SOA decision makers (e.g. application architect) can make use of our model to inform the decisions of service selection and composition in CB-SOA, which has the promise to add long-term value and reduced technical debt, if any. In addition, the model can be used to assess the time-value of the substitution decisions relative to possible changes of the structure over time. The aim is to reduce unnecessarily investment actions, which may incur unmanageable technical debt and unlikely to be utilized for value creation.

1.2 Problem

The Cloud-Based Service-Oriented Architecture can improve its utility and add value to its composition by substituting its constituent services dynamically. Moreover, one of the constituent services may become inappropriate, in terms of satisfying requirements, and should be substituted (replaced). The substitution decision should take into consideration

Figure 1.1 CloudMTD in a General View



the goal of improving utility and adding value to CB-SOA structures. The substitution decision is either driven by business or technical objectives. For example, change in requirements, end of service contract, desire to improve one or more QoS attributes, need to upgrade to a new web service that has been released in cloud marketplace recently, etc. We argue that these drivers are affecting the utility of a given CB-SOA structure. Furthermore, selecting a suitable alternative service (e.g. replacement of the inappropriate service with another one with better QoS) in order to accommodate a change (e.g. new demands or new requirements) at specific time is a challenging problem. We view the cloud as a marketplace [55], which offers different web services with different QoS, cost and SLA promises from different cloud providers. In this thesis, we are interested in dealing with one type of cloud-based services, which is SaaS: software as a service. The complexity of the decision making intensifies with the number of candidate services and their attributes. On the other hand, we argue that selection decision may come with a technical debt. In this thesis, we define the technical debt as the gap between "expected-level" of QoS, expressed as utility, and the "current-level". Technical debt can be manageable (creates value) or unmanageable. CloudMTD model is managing technical debt in the view of realizing future value using Real Options. Obviously, managing technical debt could entail a cost. However, some future value maybe lost when ignoring technical debt and the future opportunities created by options.

1.3 Solution

We view selection and composition decisions in CB-SOA form two levels; service-level for service selection and architecture-level for service composition. This is done in the view of achieving the following goals: (1) creating long-term value-added, (2) maximizing utility, and (3) managing technical debt (if any). In order to achieve the previous goals, some consequences/aspects may appear and need to be addressed, such as: (1) Technical Debt, (2) Future Changes (Uncertainty), and (3) Dependency.

The first part of the solution is concerned with the selection decision and the likely incurred technical debt. As we are taking decision under uncertainty, the solution should be flexible enough to manage such uncertainties. In fact, option analysis is valuable where there are uncertainties associated with the decision to be taken. Option theory is typically applied to such decision because managing uncertainty requires flexibility in dealing with such decisions when there are multiple possibilities for future events [21].

An option, in general, is the right to have the freedom of choice. *"A Real Option, particularly, is the right - but not the obligation - to undertake some business capital investment decision; typically the option to make, abandon, expand, or contract a capital investment" [30].* Real options analysis emphasizes the value of flexibility under uncertainty, which can be viewed in the form of future options it creates [77]. In this sense, flexibility creates future opportunities, which take many forms, such as business expansion (growth), delay, abandon and so on. In web service selection context, future opportunities are linked to the candidate web service being able to accommodate future changes under uncertainty. Real options are contingent decisions [21], which gives the decision maker the opportunity to make a decision after events are unfold. In addition, real options is a proactive thinking [21], which is useful in strategic decision making in environments full of uncertainties, such as service selection in CB-SOA. In this sense, when we take an option-based approach in such environments: "uncertainty creates opportunities". This is because; CB-SOA decision makers are interested in knowing the range of possible outcomes before the decision date (proactively), as some late decisions can be costly. We argue that the key to decision making in CB-SOA is to link uncertainty to time-value and possible future opportunities that options create. In this sense, real options analysis gives the power to decision makers by giving them the flexibility to act upon unfolding events by using options under uncertainty [21]. However, this flexibility comes with a cost. Here, we say that the CB-SOA decision maker has the flexibility to act upon changes when s/he doesn't know what is hidden in future. And accordingly, unlike traditional approaches, an option-based perspective leads to higher value of the selection decision in CB-SOA.

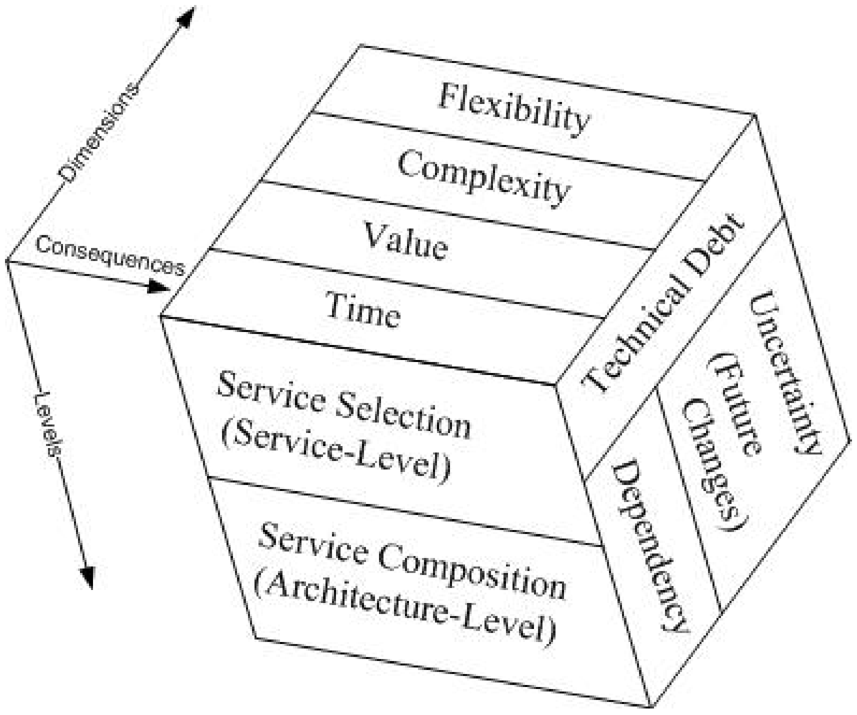
In addition, in CB-SOA, services can be offered from various cloud service providers with different cost, QoS, and features. The complexity of the decision making intensifies with the number of candidate services and their attributes. This can make the problem of selecting and composing CB-SOA benefiting from the cloud marketplace complex and challenging. Therefore, we have applied the K-means clustering technique, which presents data (web services) in a structured way. By the use of k-means clustering, services can be classified into different groups based on their QoS attributes' similarities (e.g. high availability and low response time). In this sense, we argue that the K-means clustering technique improves the service selection process in CB-SOA in terms of (1) scalability, (2) relevancy, and (3) QoS correlation.

The second part of the solution is concerned with service composition. After selecting a suitable service and managing any likely technical debt, the service should be integrated with the rest of the services in a given CB-SOA composition. In this sense, the integration process may introduce a change to the CB-SOA structure in case of any dependencies among constituent services. We use Dependency Structure Matrix (DSM) in order to represent dependencies among web services in CB-SOA. In CB-SOA context, we use the propagation cost metric for computing the percentage of web services that may be affected on average, either directly or indirectly, when a change occurs to a given web service. In this sense, propagation on the architecture can be related to changes to the structure. For this thesis, the propagation cost quantifies the additional rearchitecting cost to integrate the candidate web service into the architecture. Despite the fact that two services may be dependent, the complexity of the dependency may vary. For example, higher complexity may signal higher rework cost, lower complexity may indicate otherwise. For this reason, we build on the work of [145, 144] and we introduce time and complexity sensitive propagation cost metrics to DSM to solve the problem.

1.4 Scope

The CloudMTD model is a middleware between multiple cloud-based services (SaaS) and a given CB-SOA. In this thesis, we distinguish a cloud-based service from any other type of software, by considering some characteristics those are related to quality aspects of a given service, which are defined using QoS. Since these services are offered by the cloud, they come with varying QoS and price, and as a result, they have different implications on CB-SOA structure. This thesis is concerned with some scenarios that may lead to technical debt, which are related to web service selection and composition in CB-SOA that has been driven by either technical or business objectives. In each scenario, we are interested in three decisions (1) keep, (2) substitute or (3) abandon the current service. Each scenario takes into consideration either one or more QoS attribute dimension. We address these scenarios from an option-based perspective. Each scenario is linked to a suitable option type. A specific option type depends on the nature of the application, problem to be investigated, and the decision to be taken. For example, investment's expansion decision can be formulated as a growth option, on the other hand, when-to-invest decision can be formulated as a defer option, and so on. We are interested in four option types in this thesis: (1) growth-options and we link it to expand and substitute decision, (2) switch-option and we link it to substitute decisions, (3) defer-option and we link it to wait and substitute decision (taking into consideration the flexibility of delaying decisions in CB-SOA until some uncertainty is resolved), and (4) abandon-option and we link it to service abandon decision. We will illustrate the use of these options in later chapters using case study and illustrative examples, which are driven by scenarios of possible web service selection challenges that may lead to technical debt in CB-SOA. These scenarios have some sources of uncertainty. We are interested in reducing such uncertainties in CB-SOA decision making. We are also interested in valuing flexibility under uncertainty in CB-SOA decision making. This thesis is concerned with the following uncertainties: (1) Behavioral Aspects, such as QoS fluctuation, (2) Structural Aspects, such as complexity of services dependencies, and (3) Environmental Aspects, such as cloud uncertainties.

Figure 1.2 CloudMTD Cube


Dimensions
Consequences
Levels
Flexibility
Complexity
Value
Time
Service Selection (Service-Level)
Service Composition (Architecture-Level)
Technical Debt
Uncertainty (Future Changes)
Dependency

1.5 Research Questions

This thesis is concerned with answering the following research questions:

1. How is the CloudMTD model capable of enhancing the selection and composition decisions in CB-SOA under uncertainty? More specifically, how can CloudMTD model improve the execution time of the selection and composition process?

2. How does the Binomial Option analysis (option thinking) add value to selection decisions in CB-SOA taking into consideration technical debt? Is real option theory capable of valuing web services investments?

3. Does technical debt have an impact on the selection decision? Does CloudMTD model make technical debt explicit in CB-SOA?

4. How is the CloudMTD model capable of giving a good recommendation about time-value of selection and composition decisions in CB-SOA taking into consideration the behavioral and structural aspects of CB-SOA?

5. How can K-means clustering approach enhance the scalability of the model? Does the clustering approach make QoS correlations, recall and relevancy explicit in CB-SOA selection and composition?

1.6 Thesis Contributions

This thesis's contributions are summarized as follows:

1. A novel model (CloudMTD) for services selection and composition in CB-SOA based on options theory, DSM and propagation-cost metrics. The novelty of CloudMTD model is based on different aspects, such as (1) a value-driven model for Managing Technical Debt in Cloud-Based Service-Oriented Architectures, (2) the use of Binomial model for modeling selection decisions in CB-SOA, and (3) CloudMTD model quantifies the time-value of selection and composition decisions in CB-SOA,

technical debt and dependencies they can imply on the structure. The analysis is done taking into consideration the structural and behavioral aspects of CB-SOA. A distinctive feature of the CloudMTD model is that it links time-value of the substitution decision to both the behavior and the structure of CB-SOA.

2. The use of k-means clustering in service selection in CB-SOA. K-means clustering technique is capable of presenting data in a structured way, which enables the decision maker to investigate the structure of each group (cluster) in the given web services dataset. K-means clustering technique can improve service selection in CB-SOA by improving the following:

   (a) Scalability: K-means improve scalability by reducing the search-space in the cloud marketplace and accordingly will reduce searching-time. And consequently it will be easier to consider the context of services composition.

   (b) Relevance and Recall: K-means improve the relevancy of the retrieved web services from a given cluster. In addition, recall metric measures the percentage of the relevant retrieved web services in a given cluster.

   (c) QoS correlation: K-means is capable of making QoS correlation explicit by finding structures in data in a given web services dataset.

3. A novel Service-level Technical Debt in Cloud-Based Service-Oriented Architectures: We describe the concept of technical debt for cloud-based service selection and composition and discuss its causes. We introduce a new novel dimension of technical debt explicating service-level in CB-SOA.

4. Literature Review:

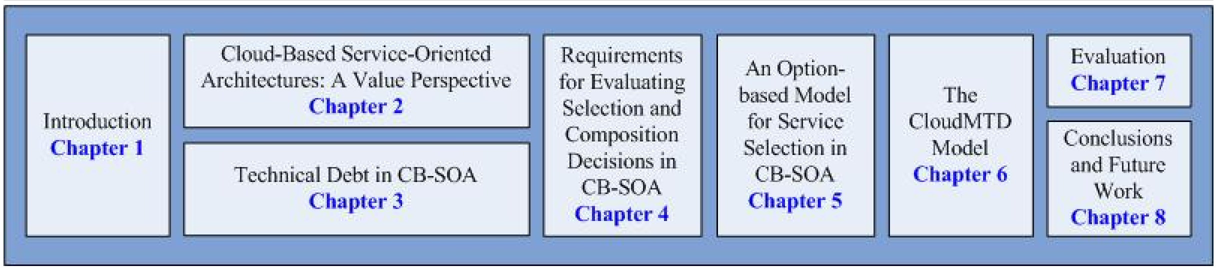
   (a) Web service selection and composition: A representative sample of research that has been carried out in the field of web service selection and composition.

(b) Technical Debt and Managing Technical Debt: This review presents the available definitions of technical debt in different fields and on different levels in the literature. The review also presents the available approaches, which have been investigating technical debt. It provides a comparison of different approaches dealing with technical debt on different levels, dimensions, causes, solutions, and evaluation methods.

(c) Real Options: A review of the previous research that investigates option theory in software engineering, economics, SOA and IT investments.

1.7 Thesis Structure

Figure 1.3 Thesis Chapters Organization


Introduction
Chapter 1
Cloud-Based Service-Oriented Architectures: A Value Perspective
Chapter 2
Technical Debt in CB-SOA
Chapter 3
Requirements for Evaluating Selection and Composition Decisions in CB-SOA
Chapter 4
An Option-based Model for Service Selection in CB-SOA
Chapter 5
The CloudMTD Model
Chapter 6
Evaluation
Chapter 7
Conclusions and Future Work
Chapter 8

The thesis structure is as follows (Figure 1.3):

- Chapter 2: presents a representative sample of research works that have been done in the field of web service selection and composition. The chapter gives a brief background on Service-Oriented Architecture (SOA) and cloud computing paradigms. Then, we present the problems with the traditional selection and composition approaches. We also motivate the need of why a value-based perspective is needed for solving the web service selection and composition problem in Cloud-Based Service-Oriented Architectures (CB-SOA).

- Chapter 3: introduces and defines a new dimension of architectural debt explicating service-level in CB-SOA. We present the term technical debt and its available definitions in the literature in different fields and on different levels. In this chapter, we motivate the need for technical debt-aware web service selection in CB-SOA. Then, we present a review of the available research works in which technical debt was investigated and we provide a comparison of the available approaches in the literature that have been used to solve technical debt-related problems. In this chapter, we also discuss different origins of technical debt, which are related to service selection decisions in CB-SOA. We also present the main drivers and situations that lead to technical debt on service level in CB-SOA. And finally, we present technical debt quantification on service-level in CB-SOA.

- Chapter 4: presents the requirements for evaluating a selection and composition decisions and managing the likely technical debt in CB-SOA from an economics value-driven perspective. The requirements cover two levels in CB-SOA: (1) service-level and (2) architecture-level. We also present some concepts and consequences, which arise once we take a decision in CB-SOA, such as flexibility, uncertainty, and services dependencies.

- Chapter 5: presents the theory of real options, its related definitions and background. We present the different option types and we relate them to different service selection scenarios in CB-SOA. In this chapter, we present and discuss different options valuation model available in the literature. Then we discuss the reasons why we have chosen binomial option valuation model in order to address the requirements presented in chapter 4. We also present the service selection problem from an option perspective and we present the analogy between real option and service selection. After that, we present a review of the available research works that investigate the option theory in software engineering. We finally present the related work that combines real options, technical debt and service selection in cloud environment.

- Chapter 6: presents our main model that is called CloudMTD. CloudMTD is an option-based model for selecting and composing service in CB-SOA, taking into consideration the likely technical debt. In this chapter, we explain the steps of CloudMTD model. We also present illustrative examples with different scenarios, which are related to web service selection and composition. Each scenario is driven by either a technical or a business objective and takes into consideration either one or more QoS attribute dimension (e.g. improving scalability). In each scenario, we are investigating three decisions (1) keep, (2) substitute or (3) abandon the current service. We address these scenarios from an option-based perspective and link each one to an appropriate option type.

- Chapter 7: presents the evaluation process of the CloudMTD model and the related decisions in CB-SOA. It also reports on the implementations and different technologies that have been used. This chapter also evaluates the applicability of the CloudMTD model based on a case study. In addition, it evaluates the scalability of the CloudMTD model based on k-means clustering using a real web service dataset. It evaluates the purity, precision and recall of the results. This chapter also presents some results, which are related to performance, behavioral evaluation and structural evaluation.

- Chapter 8: concludes, revisits contributions and present future work and directions.

2

Cloud-Based Service-Oriented Architectures (CB-SOA): A Value Perspective

2.1 Overview

In this chapter, we present a general background on SOA and Cloud Computing. We also review the state of the art of the work done so far in SOA and present a representative sample of research work that has been done in the field of web service selection and composition. We also present some drivers these are affecting and influencing selection and composition decisions in CB-SOA. We mainly focus on drivers and factors affecting and influencing the selection decision in cloud-based service-oriented architectures depending on three main dimensions: time, cost and value (and value-added). We classify traditional approaches and discuss their limitations. This chapter provides some concepts and approaches those are related to cloud and SOA, so we know where we stand. These concepts include SaaS, Cloud, SOA, etc. We are mainly interested in dealing with one type of the cloud, which is SaaS: software as a service. This chapter is linked to next chapter as follows: this chapter presents the current approaches in dealing with service selection and composition. We present the problems of current approaches. Then, we motivate the need for a novel model in dealing with the problem of service selection and composition in CB-SOA. It also provides factors and dimensions that affect and influence the selection and composition decisions in CB-SOA. Next, we motivate the need for non-technical factors that affect and influence the selection decision in chapter 3.

2.2 SOA and Cloud Computing

In this section, the link between Service-Oriented Architectures (SOA) and Cloud Computing is investigated. We begin by the definition of SOA by Sommerville, which is: "*Service-Oriented Architectures (SOAs) are a way of developing distributed systems where the system components are stand-alone services, executing on geographically distributed computers* [203]." Service-oriented software engineering is one of the most dominant approaches in the business world [203]. Accordingly, we use SOA as an architectural style for building applications. Moreover, web services are used as a technology for implementing SOA. The notion of service-oriented computing came out as a result of the previous technological drawbacks. Later on, SOA was supported by the notion of “component-as-a-service” [203]. The main difference between the traditional concept of a “component” and the new “component-as-a-service” is that services are treated as stand-alone units. Each service in the SOA architecture provides an interface for its functionality.

Furthermore, SOA and Cloud Computing complement each other [138]. In general, cloud potentials are defined as services. There are many factors supported the adoption of cloud by consumers, such as virtualization technology, subscription-oriented computing, and pay as you go model [53]. Cloud Computing is defined as: “*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* [23].” Cloud Computing is a relatively recent paradigm that was developed out of many technologies, such as grid computing, virtualization and web tools [55]. Cloud Computing paradigm includes infrastructures, platforms, and applications. Cloud applications have many characteristics, such as: on-demand, flexible, and scalable applications with pay per use policies [55]. The beauty of the cloud comes from the previous factors in addition to the fact that users can access services, anywhere in the world, based on their requirements. Cloud computing allows the usage of web resources on demand over the internet, such as: databases, applications, email, etc. Traditionally, both data and software used to reside on the client-side. However, in cloud computing, the client-side has relatively no software or data, and processes occur away from client-side, somewhere on the network [23]. To explain cloud computing in simple words, the analogy of *“Public Utilities”* is used (e.g. electricity, gas, and water). In such utilities, generating electricity and pumping water is done away from the client. Similarly, in cloud computing, certain software and hardware installation and maintenance are performed away from the end-user. Gmail is an example of a daily used cloud service. There are three Cloud Computing service models, which are [91]:

1. Software as a Service (SaaS): SaaS are services that are accessed by end-users via web browsers, such as: Facebook, Google Calendar, Google Docs, Google Mail, etc.

It provides customers some capabilities to use web applications which are running on cloud. Customers do not deal with the cloud infrastructure, such as: storage, network, operating systems or servers. In this thesis, we are interested in SaaS.

2. Platform as a Service (PaaS): PaaS are services that allow developers to develop custom applications and deploy them onto the cloud infrastructure, such as Google App Engine and Microsoft Windows Azure. Developers do not deal with the cloud infrastructure, such as: storage, network, operating systems or servers.

3. Infrastructure as a Service (IaaS): IaaS provides some resources, such as processing and storage, in which customers are able to deploy and run their own software, which includes applications, such as Amazon's Elastic Compute Cloud (EC2). Customers do not deal with the cloud infrastructure, such as: storage, network, operating systems or servers. This category includes meeting requirements such as memory, CPU, storage, etc. [53].

There are three Cloud Computing deployment models, which are [23]:

1. Private Cloud, where the infrastructure is running exclusively for a private business.

2. Public Cloud, where the infrastructure can be used publicly.

3. Hybrid Cloud, where the infrastructure is combined of the two previous models (Private Cloud and in Hybrid Cloud).

2.2.1 Web Services

In general, a web service is a special form of a service. According to World Wide Web Consortium (W3C), a web service is: "a software system designed to support interoperable machine-to-machine interaction over a network"[1]. Each web service has a functional description and an interface description. Normally, web services are either based on SOAP or REST standards. For the nature of the selection and composition problem in

[1]www.w3.org

CB-SOA, we implemented web services using SOAP instead of REST. SOAP was chosen as it is language- and platform-independent protocol, while REST requires the use of HTTP (note: transport protocol for information exchange includes [202]: HTTP, HTTPS, SMTP, etc.). We have also used WSDL, which is an interface description based on XML, for the purpose of describing the functionality of our workflow and web services, which will be described in later chapters.

Quality of Services (QoS). In general, a web service is defined and described by its functional and non-functional requirements. Functional requirements refer to what the service should provide, and how it will behave and react. Moreover, QoS defines the quality aspects of a given web service (i.e. non-functional requirements).

2.3 Web Service Selection and Composition: a Review

Since our main research question is related to enhancing the selection and composition process of web services, we present a review of the approaches available in the literature those are dealing with the problem of selecting and composing web services. The objective is to explore the landscape of proposed approaches dealing with the problem. We also explore the state of the art in the field of service selection and composition, and then identify the gaps.

First, we start by defining some related terms that are used in this thesis from our perspective. We refer to web service selection as the act of selecting a qualified web service among other suitable candidate ones, based on user preferences and some other criteria. We consider the web service selection procedure as a decision making process, which is influenced by different factors and attributes. We consider Quality of Services (QoS) as one of the main determinant of web service selection process, in addition to cost and technical debt factors (technical debt, QoS gap, will be explained in more details in chapter 3). We also refer to web services' composition as the act of composing individual web services in order to form a workflow. We view the composition as a collection of related constituent web services of specific system architecture. In this sense, service selection and service composition are two interdependent phases.

The major two categories for distinguishing services in the literature are functional and non-functional requirements. In addition, as there has been a massive work in the literature dealing with service selection and composition, we narrow down our review and exclude some categories and criteria. We focus more on the available work dealing with the service selection and composition problem in SOAs based on QoS. The following review will be answering the following research questions:
RQ1: What are the current proposed techniques for solving web service selection and composition problems? (e.g. Reducing the search space)
RQ2: What are the available approaches dealing with web service selection and composition problems? (e.g. Genetic Algorithm)
RQ3: What are the possible limitations of current approaches? (e.g. Scalability)
RQ4: What are the available gaps in the literature dealing with web service selection and composition problems? (e.g. decisions are not long-term value-driven nor technical debt-driven)

The research space of selecting and composing web services can be classified into different categories depending on different aspects dealing with the problem such as context-aware web service compositions [6], planning-based approach [244], goal oriented composition [25], process-driven composition [237], ontology-based (e.g. [96, 124]), semantic-based (e.g. [143, 142, 160]), and syntactic including qualitative aspects (e.g. QoS-based approaches). Moreover, various research works has been done in the field of web services description and discovery (e.g. [177, 10, 12]), trust-based service discovery (e.g. [5]), selection based on user preferences over non-functional requirements (e.g. [187, 102, 107]), web service substitution in term of context-aware substitution [174, 67], location-based service selection (e.g. [223]), truthful mechanism (e.g. [240]), QoS negotiation (e.g. [154]) and many other fields.

In this thesis and according to our main research questions, we are interested in QoS-based selection and composition approaches. Moghaddam and Davis [159] have classified the research space, related to QoS, into three categories: (1) Technical domain-independent attributes (e.g. availability), (2) Non-technical domain-independent attributes (e.g. reputation), and (3) Domain-dependant attributes (e.g. refresh time [63]). Our goals of mapping the available research works in the literature into different categories are to recognize the general classes and identify the main theme of each approach, and the second goal is to identify the available models in the literature dealing with web services' selection and composition problems.

Inclusion and exclusion criteria. As we previously mentioned, the search space of the following review was reduced based on specific criteria. We selected papers from conferences, journals, and workshops. First, we have selected papers from a well respect electronic databases, such as Google-Scholar, ACM, IEEE, Science-Direct, Springer. Second, papers were selected according to well-known publishers, such as ACM, IEEE, etc. Third, we have presented representative samples of some specific selection and composition categories. This is because; some of the available work may not be fully directly related to our work. Fourth, we have excluded unavailable papers, such as papers with abstract view only. However, some unavailable papers were given by main authors via email.

2.3.1 Technical domain-independent attributes

In this category, approaches are classified according to non-functional service properties and are considered to be technical domain-independent attributes. In general, web service selection approaches related to this class of work can be further classified as follow; QoS Modeling, Greedy web service selection (e.g.[181]), Discarding non-optimal subsets of web services, and pattern-wise web service selection.

The most well-known works in the literature that presented different QoS categories for the purpose of distinguishing web service is the ones presented by Menasce [155] and Zeng et al. [238]. Menasce proposed that QoS can be a combination of many other attributes of a service [155]. Zeng et al. [238] have moved the QoS-based selection problem into an optimization problem. They presented a QoS-aware middleware for supporting web services composition based on either local optimization or global planning. Their work is based on Integer Linear Programming (ILP), which can lead to scalability and performance issues due to their linear exhaustive-search algorithm (high time complexity). In addition, their approach is not fulfilling the global constraints in all cases, as some cases are treated by selecting services randomly. Similarly, Alrifai et al. [13, 15] presented a hybrid approach which combines global optimization with local selection techniques. Their approach is based on mixed integer programming (MIP) for the global optimization. The second part of their approach is concerned with service selection based on QoS values with the focus on the performance of the selection technique. The idea is to decompose QoS global constraints into a set of local constraints and accordingly finding services that satisfy these local constraints. However, their method is based on linear programming and in [13] they handle sequence execution. Accordingly it can entail poor scalability.

In addition, Al-Masri and Mahmoud [9] proposed an approach for discovering and ranking web services. They introduced a Web Service Relevancy Function (WSRF), which is a relevancy ranking function that rank web services depending on user's preferences and some QoS metrics[9]. In WSRF matrix, each QoS cell value is compared to the maximum value of each QoS/column. Reddy et al. [181] presented a QoS model for rating web services based on a Greedy approach. The model consists of three levels; Business level, Service level view and System level view. They have formulated the local optimal choice at each level, in order to find the global optimum one (optimization problems). However, the main problem of such Greedy Algorithms is that they fetch for local optimum solutions, without considering any future change. This will affect the optimization problem accordingly. Later on, Dumas et al. proposed an approach which characterizes the behavior of services based on contracts [73]. These contracts are set of allowed sequence of operations. Their interest was in enhancing the survivability of the system by substituting the deficient service with another one with least functionalities. They also proposed a method for degraded substitution. Recently, Fu el al. [84] proposed an approach for discovering an admissible set of services using an empirical distribution function based on historical transactions. They have formulated the problem of QoS-aware service selection as a decision problem under uncertainty. They have adopted the skyline dominance notion and they applied a stochastic dominance rules for service comparisons.

2.3.1.1 QoS Modeling

We are more interested in QoS Modeling category of approaches and the QoS aggregation function. We extend the classification of [159] by going deeper into the categorization process and classify the QoS modeling approaches into three groups: (1) dynamic or static approaches (2) approaches based on service semantic description or syntactic description (3) fuzzy or precise approaches based on user preferences.

Jaeger et al. [106] presented a mechanism for aggregating QoS of individual services in order to compute the overall QoS of a specific composition. They model the composition based on seven different structural elements, which they named as the composition patterns; Sequence of service executions, Loop, AND split followed by AND join, AND split followed by an m-out-of-n join, XOR split followed by a XOR join, OR split followed by OR join, and OR split followed by an m-out-of-n join. Their composition patterns were based on the workflow patterns work done by Van Der Aalst et al. [220].

In addition, Jaeger and Ladner [105] presented a pattern-wise QoS aggregation approach. They have used the seven structural element patters presented by Jaeger et al. [106] for QoS aggregation. For example, the maximum of the execution time attribute can be aggregated by considering the larger value of parallel tasks first, and after that the sum of sequential tasks is calculated, and so forth. They have also proposed some aggregation rules for each composition pattern as well as for every QoS category.

Likewise, Hwang el al. [104] proposed a unified probabilistic model for QoS aggregation based on the context of workflows. According to their approach, when selecting a constituent web service for a specific workflow, the QoS of the workflow is calculated based on the QoS values of the constituent web services. However, they assumed that each QoS value of a given constituent service in a given workflow is independent of the other QoS values.

On the other hand, some other selection approaches were based on QoS prediction models. For example, Shao et al.[194] proposed a personalized QoS prediction model for web service selection using the collaborative filtering algorithm. The prediction of QoS missing values is based on previous consumers' experiences and similarity mining. Likewise, Zheng et al. [242] proposed a hybrid recommendation approach depending on users contributions using collaborative filtering algorithm. They used the user-contribution mechanism in order to collect QoS information and predict the personalized QoS values by running a set of large-scale experiments using real dataset.

2.3.2 Non-technical domain-independent attributes

Kokash et al. [120] proposed a culture approach system, which matches web services with user requests. The idea behind this approach is grouping users with similar needs and interests according to their requests. So those groups are culture-like community. Kokash et al. system was implemented depending on the history of decisions, which were made by other users with the same requirements and requests previously. They mentioned that instead of evaluating the web service by a trusted third party, which is expensive and inefficient, we can use the recommendation and reputation systems. Their system was implemented based on the idea of grouping users of common interest. This approach was similar to the work of Blanzieri et al. [43], where the system is used for filtering web services based on their reputations. In addition, web services can be rated by users optionally. However, their approach is not suitable for cases where the user is requesting a new web service which has neither knowledge nor data history.

In this case, we need to have a web service benchmarks or metrics that calculate the new web service's characteristic, such as performance. Consequently, in this scenario, web service providers can prove the reliability of their new web services. We believe that the rating technique is still not reliable in some cases, as many users may rate web services inaccurately and imprecisely. This approach can be enhanced to be able to store information about measurements of QoS parameters in the future and Service Level Agreement (SLA). They evaluated their system by the use of Precision, Recall and F-Measure measurements. They also used the Term Frequency Inverse Document Frequency (TF-IDF) metric based on vector space model. Their results seem to be good quality. The source of data they used was "XMethods.com". Their system was tested with four users and 100 requests. The experiment criteria factors were based on an estimated standard rank. On the other hand, Kiran et al. mentioned web services evaluation filling the gap between web services themselves and customer requirements. They claimed that web service evaluation doesn't provide a fair-enough evaluation. Moreover, they believed that a trusted web service broker can evaluate the web services in a trustworthy way. However, this way is inefficient and expensive as was mentioned previously by Kokash et al. [120].

Xu et al. [230] proposed a web service composition and discovery approach based on an index-based algorithm. Their approach was founded on two categories; finding a web service that satisfies user needs, or combing two or more accessible web services. We believe that combing available web services technique will add flexibility to any system. We noticed that this approach was a pure Natural Langue Processing (NLP), as they used the indexing and occurrences methods. The Composition Algorithm was based on the bottom-up manner. The main disadvantage of the bottom-up approach is that it is driven by the existing infrastructure rather than the current processes.

On the other hand, some other approaches were based on the user location. When publishing a new service in a new country/location, the availability of the service may vary depending on user network connectivity. For example, Wang et al. [223] proposed a fast match selection approach based on QoS prediction for selecting a global optimal service. The selection process is performed in terms of locations and communication links of multiple users. Specifically, they predict the missing multi-QoS values according to the historical QoS experience taken from different users. Similarly, Chen et al. [60] proposed a region model for web service's recommendation based on a scalable and hybrid collaborative filtering algorithm. They argued that user location is a great influence of the selection. They grouped users hierarchically into similar groups based on different regions and QoS history. Likewise, Tang et al. [211] presented a location-based hybrid approach for service recommendation based on QoS prediction and services relationships.

Furthermore, some other approaches are classified as context-aware selection and composition. Service context may span different dimensions. For example, a service could be selected based on the location as a context, or a user budget as a context, etc. Some other approaches are based on the context of specific information. For example, a specific web service may be ranked as the optimal service among other candidates; however, this service may be not available at a specific location. Here, we say that this service is not an optimal selection in this context. Some other work select services based on the context of the user's device [173].

2.3.3 Classification and Clustering Techniques

In this section, we present a representative sample of research works that have been done in the fields of service classifications and clustering. In early work of web services classification and clustering, Hess and Kushmerick [100] presented web services classification algorithm based on classifying HTML forms into semantic categories using Bayesian network. Their goal is to discover semantic categories in web services. After that, Dong et al. [72] presented a web services searching algorithm based on services operations similarity. Their matching algorithm is based on two similarity criteria: (1) textual description of operations and (2) parameters' names of operations. They have proposed a clustering algorithm based on agglomerative clustering for the purpose of grouping parameters' names of web service operations based on semantically meaningful concepts. The agglomerative clustering normally merges any two clusters that have two closely associated terms. On the other hand, Ma el al. [142] presented a clustering semantic algorithm for removing irrelevant web services based on a specific query. The purpose of their approach is to make a match between semantics hidden in a specific query and descriptions of services on concept-level after eliminating irrelevant services. Furthermore, Elgazzar et al. [76] proposed a technique for service clustering based on mining the documents of Web Service Description Language (WSDL). They extract features from WSDL documents, which describe services' semantics and behaviors, such as content, types, ports, messages and the name of the service. More recent, Makhlughian et al. [147] presented a service selection and ranking framework based on semantic matching. The first part of their framework classifies candidate web services according to users' requirements and preferences of QoS using an association classification algorithm; Classification Based on Associations (CBA). The second part of their framework is concerned with ranking candidate web services based on semantic-link similarities. Here, they mentioned that they are building on the notion of a semantic-link, which is the semantic connection between an output of a service and an input parameter of another service [147]. In this work, services classification is based on some QoS constraints that have been set by users. In this sense, services are classified into classes based on the distance between a service and user demand of QoS. Moreover, Gai and Du proposed a service discovery algorithm using ontology and Petri nets [85]. On the other hand, Kumara et al. [126, 125] presented a web services clustering approach using semantic similarity based on Associated Keyword Space (ASKS). They calculate services similarity based on hybrid term similarity method using ontology learning and information retrieval. The idea behind using semantics in previously mentioned research works is that the keywords search is insufficient anymore, as the number of retrieved services can be huge.

Later on, Xia et al. [228] presented a service selection approach based on a density-based clustering algorithm and BPEL tree structure. The idea is to cluster services based on specific tasks, where tasks are functionality achieved by services. Their algorithm is based on OPTICS algorithm presented by Ankerst et al. [22], which has a density-based clustering structure. Here, if a point is not positioned in a dense-enough environment, it is considered to be a noise. Their selection is based on QoS values of composite services depending on users QoS constraints. Their QoS aggregate function is based on the composition structure (e.g. fork, loop, etc.). The algorithm reduces candidates at every step of service composition in the BPEL tree model presented by [62]. Accordingly, they improve the efficiency of their algorithm. Alternatively, Wang et al. [225] presented a service recommendation approach based on clustering. They have applied k-means algorithm for the purpose of clustering tenants of Software as a Service (SaaS) based on their QoS requirements. Furthermore, clustering was also applied by Liu et al. [140], who developed a web service aggregation platform in the relational database (RDB). They have used self-join operation for the purpose of clustering services. Their clustering approach is based on concept semantic reasoning relationships. Their approach performs services matching based on services interfaces (input and output).

Recently, the notion of skyline has been investigated in the field of selection and composition of web services. The concept was introduced by Borzsony et al. [47] in database systems, as the process of filtering some interesting points out of a large set of data points. For example, Skoutas et al. [200] presented an approach for ranking and clustering services. The first part of their approach is concerned with ranking services according to their dominance relationships. A score is assigned to each service in the repository for the purpose of quantifying the suitability of each service for a given user request. The second part is concerned with clustering services based on matched parameters (input and output parameters). They claim that they do not use clustering for the purpose of identifying relevant services; instead, clustering is used for the purpose of grouping search results based on request parameters. They have adopted the notion of skyline. Their matchmaking algorithm is based on two factors; (1) parameter degree of match (PDM) and (2) service degree of match (SDM). SDM is computed based on aggregating individual PDMs. They claimed that assessing PDMs can be either by treating parameter description as documents and accordingly applying string similarity measurements for computing PDMs, or by following semantic web vision. On the other hand, Alrifai et al. [16] focused on services composition based on cutting down some of candidate services by dividing the global constraints to local ones. Their goal was to find an efficient solution that gets a close to optimal selection. Their approach was based on the concept of skyline. In this sense, they formulate the selection problem as a skyline filtering for the purpose of focusing on "interesting" services only. They have also addressed dominance relationships among services based on QoS values. Moreover, they have assumed that QoS parameters are anti-correlated. On the contrary, we test QoS correlation among services. However, their method is based on mixed linear programming (MIP) and handles sequence execution. Accordingly it can entail poor scalability. Nevertheless, they removed all non-skyline services in order to cope with this limitation. In this sense, when composing services they consider a subset of the skyline. They applied the hierarchical clustering technique for the purpose of selecting a representative set of skyline services, in case if the skyline is too large. Their main idea of using clustering is to cluster skyline services into a specific number of clusters, and then one representative service is going to be selected from each cluster. They have also used the hierarchical k-means clustering algorithm for building a representative tree. However, their approach deals with a two-dimensional space. Similarly, Zhang et al. [239] presented an analytic algorithm for exploring services in service registries based on pattern recognition. They have applied a pattern recognition algorithm for the purpose of extracting embedded service attributes. Then, they apply a hierarchical k-means clustering algorithm based on tree structure for acquiring a multi-level services hierarchy. Next, in this thesis, we show how a different category of k-means clustering techniques can be used in multi-dimensional space and for different purposes, such as finding QoS correlations.

Table 2.1: representative approaches comparison.

Ref.	Selection Dimension	Approach	Selection Criteria
[64]	Privacy-aware	Ranking Algorithm w.r.t. privacy level.	Purpose, sensitivity, visibility and retention period of information.
[164]	Value-based	value model based on VSDL	functional and non-functional properties, service reputation, and provisioning channels.
[96]	Ontology	1) Similarity reasoning, 2) Equivalent reasoning, and 3) Numerical reasoning.	Similarities among services
[6]	Context-aware	Guiding dynamic evolution of service compositions	Adaptation based on a predefined actions guide according to known contexts
[244]	planning-based	Temporal planning and numerical optimization	Optimal overall QoS value
[223]	location-based	predicting the missing multi-QoS values based on historical QoS experience from users	Global optimal
[139]	QoS-Aware	Ant Colony Optimization and Genetic Algorithm	Requirement satisfaction
[57]	QoS-Aware	Genetic Algorithm	Weighted fitness function
[110]	QoS-Aware	Weighted Euclidean distance	Global optimal
[194]	Based on personalized QoS Prediction	Collaborative filtering	Predicting missing QoS values based on consumers' experiences
[108]	QoS-Aware	user similarity measurement	similarity between users
[131]	location-based	hybrid Collaborative Filter algorithm	Highest degree of similarity
[211]	location-based	Collaborative Filter	Top QoS values w.r.t. users location and services' relationship
[141]	QoS-Aware	Predicting missing QoS values	user-based regularization term and a service-based regularization term
[231]	QoS-Aware	Matchmaking	fuzzy classification
[137]	Consumer-Centric QoS-aware	Similarity analysis	fuzzy opinion similarity and QoS preference
[222]	QoS-Aware	Capturing vagueness using intuitionistic fuzzy sets (IFS)	Linguistic weights of QoS criteria based on participants' preferences
[193]	Context-Based	Semantic matchmaking	Ranking and prioritization
[75]	Transactional and QoS-Aware	Matching services properties with the user's desires	Transactional behavior of the application and user's preferences satisfaction expressed as weighted QoS criteria
[16]	QoS-Aware	Services skyline	Services skyline
[234]	QoS-Aware	Multi-attribute optimization skyline	Most interesting service providers based on user desires
[63]	QoS and Contract specification	Matchmaking algorithm	Services' ability of fulfilling requestor requirements and budget constraint
[64]	Privacy-Aware	Privacy-preserving	Privacy-level and compliance
[107, 106]	QoS-Aware	Pattern-based aggregation	Abstract composition patterns
[14]	QoS-Aware	Heuristic algorithm	Satisfying business requirements and acceptable costs
[129]	QoS-Aware	Voting algorithm	Minimal response time and latency

2.4 Current Approaches: Gap Analysis

Service-oriented systems provide flexibility by supporting business processes[203]. Service-oriented systems allow web services to be integrated in order to create a web application that offer more extensive functionalities. One critical challenge in service-oriented systems is the dynamic web service selection. Another challenge is maintaining the QoS values of a specific service. Service selection is a procedure, which involves screening available options and investigating tradeoffs among them. It is much easier to evaluate available choices when there are only few services. However, evaluation becomes overwhelming when the selection decision requires critical thinking of considering uncertainties (e.g. future changes). In addition, the complexity of the decision making intensifies with the number of candidate services and their QoS attributes. In this context, this thesis will be introducing a model which investigates service selection and composition in CB-SOA under uncertainty. In this thesis, we view decision-making in CB-SOA from three main perspectives: long-term value, embedded future options, and technical debt. These three values are used for informing the decision of selecting the most qualified service at a specific time in CB-SOA. In this context, we motivate the need for a new selection and composition model in CB-SOA. We argue that decision-making in CB-SOA must take into consideration the following: (1) long-term value, (2) flexibility of the decision under uncertainty (will be discussed in details in subsequent chapters), and (3) the likely technical debt (will be discussed in details in chapter 3).

This thesis investigates the following gap analysis in the literature:

- The majority of the available research works that deal with service selection and composition problems in the literature tend to be "value-neutral". They focus on technical aspects in selecting web services by looking at QoS and cost. Observations from the review showed that the majority of the available research works are value-neutral and do not take long-term value-added into consideration. On the contrary, we quantify decisions in CB-SOA by looking at long-term value.

- None of the available approaches take into consideration the "technical debt" that may be associated with decisions in CB-SOA. Other than that, the majority focus on technical aspects such as, QoS and cost. We argue that we may lose future opportunities by ignoring unattractive choices, or by solving problems for short-term. When taking a decision of selecting a suitable service, some services may appear less attractive and may come with a debt. However, we argue that this debt can be transformed to future value if properly managed.

- Classical approaches deal with web service registries as repositories, such as Universal Description, Discovery and Integration (UDDI). On the contrary, we follow a similar argument of Buyya et al. [55] and we view the cloud as a marketplace. In this sense, the cloud offers different web services with different QoS, cost and SLA promises from different cloud providers.

- The majority of the available research works ignores the value of flexibility of selection and composition decisions with respect to likely future changes under uncertainty.

2.5 Selection and Composition Decisions: Cloud Marketplace Perspective

In previous sub-sections, we reviewed the work in the literature that investigated web service selection and composition problem from a service perspective, i.e. criteria and dimensions based on the web service itself. However, in this section, we review that available research works in the literature, which look at the problem from the provider perspective, i.e. cloud service provider.

The idea of a cloud marketplace was introduced by Buyya et al. [55, 53], which came from the notion of a cloud as a virtual market, in the view of understanding the cloud computing vision. In this sense, the cloud broker plays an important role among different providers and consumers, where the cloud broker is responsible for selecting and composing services from the cloud marketplace. Examples of cloud marketplaces include: (1) Oracle Marketplace, (2) AWS Marketplace, and (3) Microsoft Windows Azure Marketplace.

Buyya et al. defined the cloud as follows: “A Cloud is a type of parallel and distributed system consisting of a collection of interconnected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resources based on service-level agreements established through negotiation between the service provider and consumers [54].” In [52], Rajkumar Buyya presented an approach for managing market-oriented resource allocation in cloud. Their work focus was mainly on resource allocation; however we are concerned with service selection and composition in CB-SOA. Recently, some research works have been done on market-oriented cloud in different fields. For example, Toosi et al. [217] proposed a financial option-based resource pricing model for pricing cloud resources in federated cloud. Their model can help providers in managing reserved cloud resources. On the other hand, Nallur and Bahsoon [165] presented a decentralized mechanism for dynamic re-composition and self-adaptation using market-based heuristics based on continuous double-auction. Mainly they have presented a market-mechanism for service-based application to self-adapt to changes in QoS requirements. Also, Abolfazli et al. [3] proposed a market-oriented architecture based on SOA to stimulate publishing, discovering, and hosting mobile services.

Based on Buyya’s definition of cloud, we view the cloud as a marketplace. We are adopting Buyya et al. [54] notion of cloud, by presenting the web service selection and composition problem in the view of cloud marketplace for services trading. I.e. when dealing with web services, we view the cloud as the marketplace for service trading. Trading, in general, refers to exchanging commodities for money. In finance, it refers to exchanging securities for money. Nevertheless, we will investigate and understand the activities of buying and selling web services from an option perspective in chapter 5.

2.6 Selection and Composition Decisions: an Economic Value Perspective

In Software Engineering, a decision maker should analyze what are the outcomes and ramifications of any decision, as every decision must be associated with a benefit or positive impact on the system (architecture). In addition, we say that every decision should be linked to not only current value bust also future one. In this thesis, we are interested in the economic value of selection and composition decisions in CB-SOA. We approach service selection and composition problem from an Economics-Driven Software Engineering perspective. We adopt the value-based software engineering presented by Barry Boehm [44] in order to introduce the economic aspects to service selection and composition in CB-SOA. In CB-SOA selection and composition, we view each service with a distinguishable economic-based business value. In addition, we argue that the time-value of selection and composition decisions is a main contributor to service value. We say that when dealing with strategic and long-term decisions, we should think of either creating or maximize the economic value. Furthermore, once we are analyzing value, it means that we select and compose services, for the purpose of improving the utility. In this sense and as we are considering value, we cannot look at selection and composition in isolation as separate phases.

On one hand, some research work look at value and value-added concepts from a composition point of view. More precisely, they view composing more than one service is adding more value to customers than interacting with one service (e.g. [150, 164, 147]). For example, Nakamura and Aoyama [164] investigated the problem of web service composition from a Value-Based Composition (VBC) point of view. Their model consists of three building blocks; (1) a value model, (2) a value-meta model, and (3) a service broker. They used the proposed value-based model for a dynamic web service composition that is based on some criteria extracted from functional and non-functional requirements' values. They have also presented a value meta-model and its representation language;

Value-based Service Description Language (VSDL), in order to define relationships among values. They used values to define services QoS. And they defined a service as an activity of exchanging a provided-value with a requested-value. They have also presented an architecture, which consists of value added service broker, in order to dynamically compose web services. Their idea is based on composing web services depending on customers' interaction with the service broker. This is because the service broker has information and knowledge about service providers, so it is easier to find a match to each customer request. This service broker has two different repositories intended for storing templates; one is designed for the value model and the other one is designed for the process model. Their work seems to be still in progress, as it just provides an idea without a detailed example. However, their approach is based on pure QoS values. And their usage of the value-added concept is based on the idea of composing services instead of providing a single service to customers. On the other hand, we deal with value and value-added from different perspectives with respect to current and future benefits of candidate web services.

In this thesis, we view web service selection and composition as an investment opportunity with specific value determinants factor: time, cost, flexibility (created by options), and technical debt. The value of each candidate web service will be demonstrated using binomial tree in chapters 6. When considering service selection and composition in CB-SOA, decision makers have different preferences regarding the value of a service, as value is subjective and expands different dimensions. Each dimension has a weight which is specified by decision makers. In our context, these dimensions inform the selection and composition decisions in CB-SOA. Indeed, a web service is intangible "software", and accordingly its value cannot be observed in a straightforward manner. Value can be captured from different perspective and can be created in different dimensions, such as benefits of accumulated savings in switching costs or business value generated by improving the Quality of Service (QoS), enhancing opportunities for future strategic expansion, service capacity utilization management. On the other hand, we attribute value-added in

CB-SOA to ($\Delta V = V_{t1} - V_0$). Value-added may not be "immediate" and can be related to the improved Quality of Service (QoS), new revenue streams by enabling new business models, reduced operational cost, future savings in maintenance costs, etc.

All in all, this thesis views the selection decision as a value-creation and utility-maximization procedure. We view value as one of the main factors that influence service selection and composition decisions for the following reasons: (1) when selecting a service, there should be a balancing between time and cost against value/value-added. (2) For some scenarios, if a service is being down for a minute, this may have a specific lose in its business value. (3) The value can be used as a metric for informing and influencing the selection and composition decisions in CB-SOA. In later chapters, value will be associated with the enhanced flexibility to cope with future uncertainty.

2.7 Conclusions

This chapter presented a background on SOA and Cloud Computing. It also presented some factors that should be taken into consideration when selecting and composing services in CB-SOA. We say that these factors are affecting and influencing selection and composition decisions in CB-SOA. The contributions of this chapter are as follows: (1) providing a representative sample of the available approaches in the literature dealing with service selection and composition (2) we view the cloud as a marketplace, instead of the traditional repository of services. (3) Long-term selection decision based on three values; long-term value, embedded future options, and technical debt.

3 Technical Debt in CB-SOA

3.1 Overview

In the previous chapter, we presented the Cloud-Based Service-Oriented Architectures (CB-SOA) and the web service selection and composition problem. In this chapter, we explain the technical debt concept and give details on previous fields which have investigated technical debt term. First, we begin by giving a general background on technical debt. Then, we motivate the need for considering the technical debt concept on service-level in CB-SOA while taking the decision of web service selection and substitution in section 3.3. Then, we identify the technical debt concept generally and we present available definitions of technical debt in the literature. Then, in section 3.5 we present the available fields in which technical debt was investigated and provide a comparison of the available approaches in the literature, which was used to solve technical debt-related problems. After that we discuss technical debt origins from different researchers' perspectives in section 3.6. Then, we discuss different technical debt levels those were investigated in the literature in section 3.7. After that, we define a novel technical debt level explicating service-level in Cloud-Based Service-Oriented Architectures in section 3.8.

Moreover, we discuss the main drivers and situations that lead to technical debt on the service-level; those are affecting the service selection decisions, in section 3.8.1. We also show how to quantify technical debt on service-level in section 3.9. In this chapter, we will be answering the following research questions: (1) How do we know that there exists a technical debt on service-level in cloud-based service-oriented architecture? (2) How to quantify this debt? (3) What are the drivers and contributors to technical debt on service-level?

In next chapters, we will demonstrate how to manage technical debt on service-level in CB-SOA. More specifically, we propose a novel model for managing technical debt in the view of realizing future value using Real Options. We will answer the following question: How technical debt can contribute to service selection decisions and how it can be compared against other tradeoffs?

3.2 Technical Debt: Background

Technical debt is a concept that generally stands for significant consequences that may take place because of a quick and/or a bad choice. Cunningham's article [68] is considered as the starting point of technical debt metaphor. Cunningham introduced the technical debt concept as: "Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite. Objects make the cost of this transaction tolerable. The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt. Entire engineering organizations can be brought to a stand-still under the debt load of an unconsolidated implementation, object- oriented or otherwise [68]."

At the beginning, technical debt concept was simply describing "unclean" code. After that, the term technical debt has been developed broadly and covered wider aspects associated with the overall systems development lifecycle (SDLC). Based on recent studies [216, 130], research works on technical debt has been dramatically increased in 2010 that can be due to the initiation of Managing Technical Debt (MTD) workshops. Then, Brown et al. proposed that "like financial debt, technical debt incurs interest payments in the form of increased future costs owing to earlier quick and dirty design and implementation choices [48]."

On the other hand, Nord et al. has defined that technical debt management as: *"navigating a path that considers both value and cost, to focus on overall return on investment over the lifespan of the product [167]."* Later on, some research works presented ontology of technical debt terms for the purpose of presenting common vocabularies such as [17]. Technical debt has been discussed in different fields, such as software architecture [48, 132, 134], code-quality [42], software design [97], documentation [198], rework [167], software testing [183], refactoring [83, 221, 166, 115], compliance debt [171], social debt [210], agile development process [92], etc.

3.3 Motivation: Why taking on Technical Debt?

As mentioned before, Cloud-Based Service-Oriented Architectures (CB-SOA) are normally composed of services; those are offered by the cloud marketplace for trading. CB-SOA can improve its utility and add value to its composition by switching among its constituent services. We say that service selection decision may come with a technical debt - *an operational liability which may incur an interest if not managed, cleared and transformed from liability to value.*

In this section, we motivate the need for considering technical debt in CB-SOA selection decisions. We look at the technical debt in different scenarios calling for web service substitution in response to future uncertainties (e.g. QoS fluctuation). Following a similar argument to Guo and Seaman: a little level of debt "is not bad", as it can help developers speed up the development process [93]. We view this perspective as a valid argument for the case of web service selection and substitution in CB-SOA. Unlike previous work, we posit that web service selection and substitution decisions should not only be QoS-aware, but also long-term value- and technical debt-aware. If a little level of debt is acceptable, then the selection decision shall aim at seeking ways for clearing technical debt. In this sense, this thesis introduces a new model for identifying and quantifying technical debt in CB-SOA service selection.

In this thesis, we may lose future opportunities by ignoring unattractive choices, or by solving problems for short-term (these arguments are linked to options in chapter 5). We are interested in two perspectives when taking on technical debt in CB-SOA:

1. Long-term strategic decision. The selection decision should solve the problem with long-term perception instead of short-term one. In this sense, the decision should take into consideration future value-added that may be associated with the flexibility of the selection decision under uncertainty in CB-SOA. The selection decision shall leverage on the flexibility that a strategic decision can buy from the cloud marketplace and the value-added on CB-SOA architecture as a result.

2. Future opportunities that selection decisions create. When taking a decision of selecting a suitable web service in CB-SOA, some services may appear less attractive and may be ignored. However, we say that such services may come with a technical debt in the view of bringing future opportunities, seeing that this debt can be transformed to future value if properly managed.

Furthermore, we view investment in web services as a loan, which may incur interest by time and signal a probable technical debt. This technical debt needs to be tracked and managed for value creation and utility maximization. Given it is a loan; we view it as a technical debt which needs to be tracked and managed for value creation. In fact, it is a loan that is spent on two main bases when selecting a service in CB-SOA; (1) improving a specific QoS dimension for the purpose of achieving a satisfied-level of utility, i.e. improving utility expressed as QoS (QoS dimensions impart the utility of architecture), or (2) cost of achieving the satisfactory-level of utility. The loan is the compensation, which is paid when utility-level is less than the satisfactory level of utility. Interest may be incurred on the loan if technical debt is not properly managed by unlocking the architecture potential for value creation.

McConnell [153] reported that if the debt is paid back quickly enough, it will not accumulate interest. He added that this might be true for the software case. Furthermore, he added that deciding to take on a debt is not a simple binary *"debt/no debt"* problem. However, we should take interest payments into consideration; how often do we have to pay off or might we never pay off? According to [153], when a decision is made to take on the debt, we should be able to understand, track, and define a strategy for resolving this debt [153]. Following similar argument to McConnell [153], when taking the decision of taking on technical debt for strategic reason in CB-SOA, we should discuss some properties such as (1) what are the benefits and consequences of incurring technical debt? (2) What is/are the impact(s) on business value of the decision? Assessing technical debt and its impact on future development is difficult, especially when we are dealing with uncertainty (e.g. QoS fluctuation). In this sense, we advocate a predictive approach for anticipating and managing technical debt. A predictive approach can be applied during the early stages of the selection process to predict the debt, its impact on utility, when it will be incurred, when it will pay off, and the interest, if any. Classical approaches to managing technical debt in software development lifecycle tend to be retrospective. Unlike retrospective approaches, predictive approaches are preventive. The effort to valuation is justified and the evaluation is generally cost effective, as when compared to the retrospective ones. This is specifically true for the case of web service selection and substitution, where unjustified selection decisions can be costly to revert; retrospective analysis can be costly as a result.

3.4 Technical Debt Definitions

Given that the technical debt concept was widely spread in different fields, the concept obtained different definitions from different perspectives depending on the research context and domain. Table 3.1 presents some of the available technical debt definitions in the literature.

Table 3.1: Technical Debt Definitions from different perspectives depending on the research context and domain.

Definition	Ref
"The idea is that developers sometimes accept compromises in a system in one dimension (e.g., modularity) to meet an urgent demand in some other dimension (e.g., a deadline), and that such compromises incur a 'debt': on which 'interest' has to be paid and which the 'principal' should be repaid at some point for the long-term health of the project."	[48]
"Technical debt is a term that has been used to describe the increased cost of changing or maintaining a system due to expedient shortcuts taken during its development."	[119]
"This includes things like bugs, design issues, and other code-quality problems that are potentially introduced with every addition or change to the code. These issues have a detrimental impact on developer productivity."	[42]
"Technical debt describes the effect of immature software artifacts on software maintenance - the potential of extra effort required in future as if paying interest for the incurred debt."	[93]
"The technical debt (TD) concept describes a tradeoff between short-term and long-term goals in software development."	[236]
"The technical debt metaphor conceptualizes this tradeoff between short-term and long-term value: taking shortcuts to optimize the delivery of features in the short term incurs debt, analogous to financial debt, that must be paid off later to optimize long-term success."	[167]
"Almost invariably in software projects, developers can be so focused on accomplishing the needed functionality that the software itself grows less understandable, more complex, and harder to modify. Since this system deterioration usually reflects a lack of activity spent in refactoring, documentation, and other aspects of the project infrastructure, we can view it as a kind of debt that developers owe the system."	[198]
"The short iteration approach and the pressure to meet deadlines can create problems for the development this pressure can encourage shortcuts concerning code maintenance that lead to accumulation of technical debt, that is, a backlog of deferred technical problems."	[218]
"Technical debt within the enterprise should be viewed as a tool similar to financial leverage, allowing the organization to incur debt to pursue options that it couldn't otherwise afford."	[119]
"Making a decision about whether to fix or defer fixing a defect is important to software projects. Deferring defects accumulates a technical debt that burdens the software team and customer with a less than optimal solution."	[201]
"This may lead to increasing maintenance costs and the quality of the end product is undermined. Furthermore, the cost increases the longer the technological shortcomings go uncorrected, until the code becomes unmanageable and essentially has to be completely rewritten. This escalating problem is often called design debt"	[97]
"Architectural technical debt is incurred by design decisions that consciously or unconsciously compromise system-wide quality attributes, particularly maintainability and evolvability."	[133]
"Technical debt represents quality problems in software, and we use the quantification of such problems to determine the value of software."	[71]
"Technical debt is the sum of remediation costs for all non-compliances"	[128]

3.5 Technical Debt: A Review

Different approaches have been applied for managing technical debt such as technical debt prioritization [191, 128, 127], dependency mapping based on Design Structure Matrix (DSM) and Domain Mapping Matrix (DMM) [50], portfolio management [93], code metrics [46, 81], threshold monitoring [74], refactoring [8], etc. In particular, Brown et al. [48] has established the need for models and techniques which can identify and manage technical debt, its causes, and its trade-offs based on the economic impacts. They argued that compromises should be taken in a way where there should be a balance between the short-term deadlines and the long-term sustainability. Managing technical debt is crucial as technical debt may cause long-term problems if unmanaged, such as increasing the maintenance cost. They have associated the relation between the technical debt and the financial debt as they considered that it is the increase of future maintenance cost, due to earlier quick and dirty design and implementation choices. They also added that this can be viewed as a form of interests. They argued that refactoring may reduce future interest payments. With the popularity of the term “technical debt”, such an approach is needed, as different system developers have different objectives which may cause significant consequences of design decisions made previously.

Another approach that was proposed for managing technical debt on the architectural level is the one suggested by Brown et al. [50]. The approach was built on the idea of technical debt assessment in an iterative release planning process based on dependency analysis. They introduced a dependency mapping method for analyzing three types of dependencies in the iterative release planning: 1. Dependencies between requirements using DSM, 2. Dependencies between requirements and architectural elements using DMM, and 3. Dependencies between architectural elements using DSM. They investigated two paths in the view of value-maximization and cost minimization. Their method consisted of different phases: 1. identifying the requirements using DSM and DMM, 2. identifying another development and improvement paths, and 3. Cost and Value analysis of each path. According to their objectives, the path with the highest value and least architectural elements dependencies is the first one to be developed. Their method can help in decreasing the accumulating debt and can contribute to architectural decision-making. Based on [50], Nord et al. [167] looked at a new metric for managing the structural technical debt based on the propagation cost, which is called Change Propagation Metric. After that and based on Nord et al. approach [167], Schmid [188] proposed a method for formalizing technical debt using a cost perspective with formal analysis.

Another approach was proposed by Nugroho et al. [170] for managing technical debt based on an empirical assessment method of software quality. From an economic perspective, they quantified the IT investment cost in two extremes; debts and interest. They viewed debt as the cost of repairing technical quality problems by improving the technical quality level to an ideal one, which is the highest reachable level of quality in the software model that is approved by the institute. In particular, they view the debt as the difference between the current quality level and the ideal/highest one. In this sense, technical debt is viewed as the repair cost, which is the cost paid to achieve the ideal level, if the repair is carried out (Repair Effort). The other extreme is the interest, which they viewed as the additional maintenance cost which is paid because of technical quality problems such as low quality level. They added that ignoring technical debt contributes to some other problems such as inefficiency in running software systems and may obstruct the process of extending such systems in future. Furthermore, their software quality quantification procedure was carried out using an empirical assessment method based on software quality developed. This method was used for computing and rating the system's technical quality, between 0.5 and 5.5, based on a layered model. They used different metrics for analysis such as Lines of Code (LoC) and functional size. Then they mapped these metrics onto ratings values. The ratings are then converted to a number of stars between one and five. Their approach was built on a static analysis. Similarly, Letouzey and Ilkiewicz [128, 127] proposed a model for estimating technical debt on the source code level. The method is called SQALE (software quality assessment based on lifecycle expectations) and it is based on refactoring prioritization. Furthermore, Table 3.2 presents a comparison of some available approaches in the literature dealing with technical debt in different fields on different levels. Moreover, Skourletopoulos et al. [199] presented an approach for technical debt prediction and quantification in software development process taking into consideration budget and service capacity constraints.

On the other hand, the link between technical debt and options has been explored in [48, 49, 192] (Real Options will be discussed in chapter 5). In addition, Seaman et al. [192] discussed four decision approaches to deal with technical debt: Cost-Benefit Analysis, Analytic Hierarchical Process (AHP), Portfolio Management Model and Options. They have looked at investment decisions at code and design levels. For example, they viewed the value of investing in refactoring as a form of options, where purchasing the option facilitates the change to the software in the future, but without immediate profits. Likewise, Brown et al. [48] proposed that some technical debt appears to create options to invest without obligations. In addition, [49] proposed that the architectural investment decisions can be optimized by analyzing uncertainty and tradeoffs between incurred cost and anticipated value based on Real options and technical debt. However, the linkage between technical debt and Real Options for the case of web service selection in CB-SOA has not been previously explored. Recently, Kazman et al.[115] presented an industrial case study for identifying and quantifying technical debt, which is originally caused by some design flaws that they refer to as architecture roots. Their focus was on the structure among these design flaws. In addition they have examined the evolution history of the project in question, instead of examining a single version. They suggested that refactoring is the solution for removing these flaws.

However, none of the available approaches in the literature has mentioned technical debt in cloud-based service selection domain nor their link to long-term value-added. Furthermore, there is a general lack for models and techniques for identifying and quantifying technical debt in CB-SOA. Our work [18, 19, 20] was an early attempt in characterizing technical debt and understanding how the debt can be quantified in CB-SOA.

Table 3.2: Comparison of different Technical Debt available approaches in the literature.

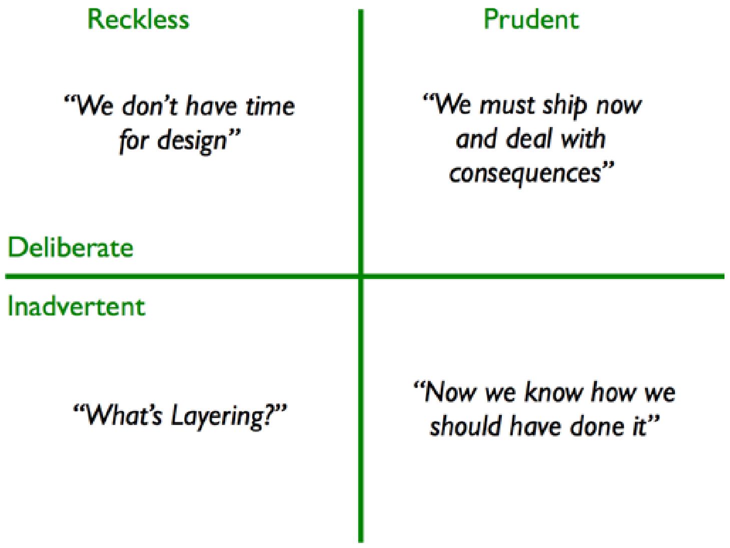
Ref	TD Level	Approach	Dimensions	Solution	Evaluation
[48]	Architectural	Offered a foundation for managing the trade-offs	Visibility, Value, debt accretion, environment, origin and impact of debt	Proposed open research questions	-
[49]	Architectural	1. DSM and DMM Dependency analysis, 2.Real Options analysis (to distinguish between decisions concerns; immediate or wait) and 3. Technical debt management	Anticipated value and Dependencies Between: 1. Stories 2. Stories and architectural elements 3. Capabilities (requirements) and architectural elements	Reaching architectural agility through the "just enough" informed anticipation in architectural decisions	User stories in Agile software Development
[42]	Code-level	formal methods	Agile software development	Reducing technical debt rapidly	-
[94]	Software project lifecycle	Retrospective Technical Debt management framework through tracking.	Maintenance	Cost/Benefit association in release planning based on simulated decisions that are retrospectively made depends on historical data.	Case Study
[198]	Documentation and code	-	Code smells, code duplication, and out-of-date documentation	Set of recommendations regarding some corrective actions to be taken against skeptical teammates or managers.	-
[167]	Architectural Rework	Architectural rework metric	Changing dependencies	Quantifying architecture quality	Case Study
[83]	Code-level	Refactoring techniques		Addressing the paying back technique of technical debt through applying some local changes to code	
[50]	Refactoring	DSM and DMM Dependency mapping: 1. Requirements, 2. Requirements and Architectural elements, 3. Architectural elements	Cost and value of modifiability		Acceptance Case of the Management Station Lite (MSLite) Exploratory analysis based on dependency and propagation cost
[170]	Code quality/software development level	Empirical assessment method of software quality	Maintainability	Reaching the ideal level	Case Study on a real software system
[192]	Decision making on incurring, paying off, or deferring the technical debt	Discussion of four approaches: 1. Cost-Benefit Analysis, 2. Analytic Hierarchical Process (AHP), 3. Portfolio Management Model and 4. Options	Maintainability	Technical debt management should be taken into consideration as an additional factor in release planning decision making.	Exploring the applicability of the four approaches on decision making considering technical debt.
[135]	"Ground level"	Empirical	Causes, symptoms, and effects.	Characterize and find out how technical debt "looks like"	Interviews with software practitioners
[119]	Enterprise level	Preliminary ethnographic study	Software technical architects experience levels.	Understanding the factors that affect the decision of taken on a technical debt.	Four preliminary interviews with experienced software technical architects.
[128]	application's source code	SQALE (software quality assessment based on lifecycle expectations)	Complexity and eight other qualities	Technical debt is the sum of remediation costs which is acquired through linking requirements to a remediation method.	Example
[122]	TD lack of definition.	A proposed technical debt landscape	-	A call for a theoretical foundation.	-
[236]	Code	Development team questionnaire.	Technical Debt Identification.	Human and automated TD identification comparison.	Case study
[93]	Investment	Portfolio	Release planning	Managing technical debt through prioritization by finding out its most important items (incomplete tasks) which must be held or incurred.	Classical scenario

3.6 Technical Debt Origin

Technical Debt concept was classified into different categories and sub-categories according to its origin. The most well know and main categories were classified by Steve McConnell [152] and Martin Fowler [82]. McConnell and Fowler had similar views and agreed on technical debt is not a simple binary decision "debt/no debt". Rather than that, technical debt should be investigated out of its source; whether if it originated from an impulsive or a strategic act. They also argued that it should be planned in the view of interests and its pay off time. On one hand, McConnell [153] classified Technical Debt into two main types: (1) Unintentional: incurred unintentionally due to low quality work, and (2) Intentional: incurred intentionally. McConnell categorized the intentional type into two sub-categories: (2.1) Short-term debt: incurred reactively for tactical reasons, and (2.2) Long-term debt: incurred proactively for strategic reasons. Furthermore, McConnell navigated deeper and deeper by classifying the short-term debt into: (2.1.1) Focused: individually identifiable shortcuts (e.g. car loan), and (2.1.2) Unfocused Short-Term debt: numerous tiny shortcuts (e.g. credit card debt). On the other hand, Fowler introduced the Technical Debt Quadrant and classified technical debt into four quadrants, Figure 3.1. Fowler mentioned that technical debt should be planned from four main angles, in the view of: (1) prudent (2) reckless (3) deliberate and (4) inadvertent. In fact, both classifications of Fowler and McConnell can be mapped to each other, except for one type, which is prudent and inadvertent debt by Martin Fowler. For example, unintentional debt can be viewed as reckless and inadvertent, and intentional debt type is similar to prudent and deliberate debt. In addition, some other bloggers categorized technical debt into almost similar categories of McConnell's and Fowler's classifications. For example, Cartwright mentioned the concept of transparent or opaque, and Marick referred to frivolous debt and debt-as-investment, etc.

In addition, Philippe Kruchten introduced the four colors backlog in terms of accumulation of incomplete work [121]. It organizes technical debt elements into four different categories. Kruchten linked technical debt to YAGNI activities: You Ain't Gonna Need

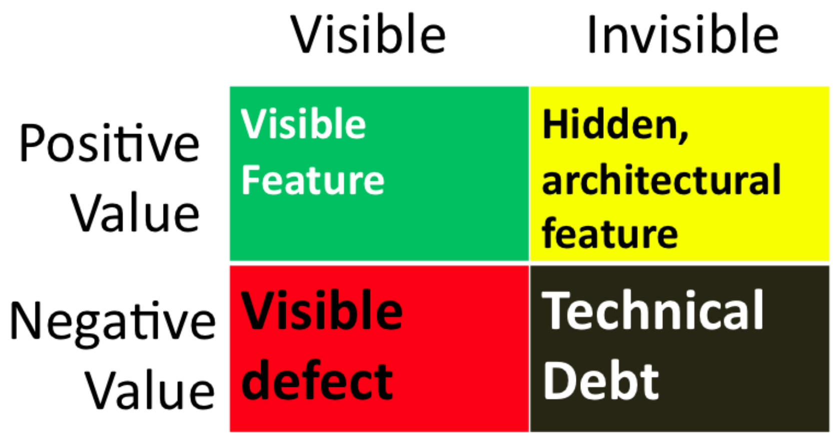
Figure 3.1 Technical Debt Quadrant by Martin Fowler [82].


Reckless
Prudent
"We don't have time for design"
"We must ship now and deal with consequences"
Deliberate
Inadvertent
"What's Layering?"
"Now we know how we should have done it"

It. He added that technical debt is the accumulation of too many YAGNI decisions. Figure 3.2 illustrates the backlog color areas which represent the intersections between elements (visible or invisible) and value (positive or negative). The areas' colors of elements identify four different possible future improvements with the purpose of increasing value. (1) Green represents adding new features, (2) yellow represents investing in the architecture, and (3) red represents the act of reducing negative effects on value of defects, or (4) black area is the technical debt, which is the intersection between invisible elements and negative value [121]. Kruchten et al. [122] differentiated between two kinds of elements in technical debt landscape: visible elements and invisible ones (this will be discussed subsequently).

In web service selection and substitution context, we distinguish between two main categories of technical debt; those are related to unintentional and intentional acts on service-level in Cloud-Based Service-Oriented Architectures. (1) The first category is the Unintentional Technical Debt: this type occurs when a non-strategic and impulsive web service selection decision is taken. Here the decision maker may not know that the technical debt has occurred for long time. We say that decision maker can be either the system architect or a stakeholder. (2) The second category is the Intentional Technical Debt: in this type, the decision maker knows that there is a technical debt accompanies the

Figure 3.2 Backlog by Philippe Kruchten [121].



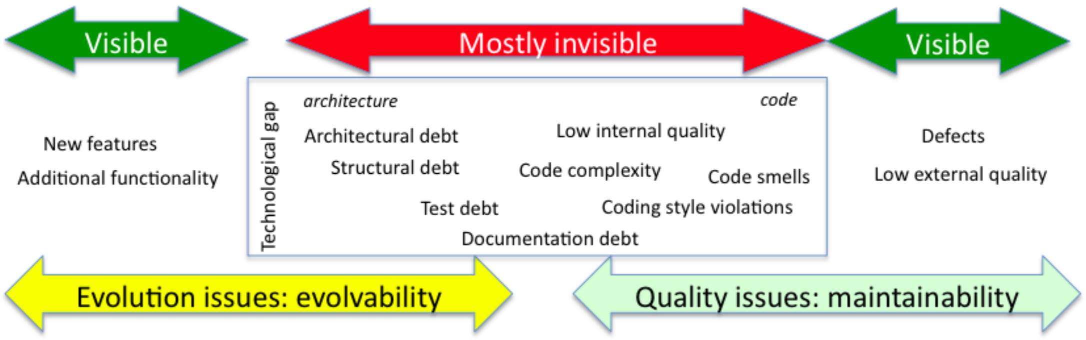
selection decision of a web service, either on the service itself or on some other dimensions that will be discussed shortly. However, the decision of taken on such technical debt can be linked to a strategic long-term benefit (value-added). In later chapters, we link this type of technical debt to time using the binomial tree, in order to obtain the time-value of the selection decision. Besides, we will track both intentional and unintentional technical debt against the value of services investments in CB-SOA.

3.7 Levels of Technical Debt

In this section, we present a representative sample of the available technical debt levels that has been investigated by researchers in the literature. After discussing these levels, we introduce a new level of technical debt explicating service-level in Cloud-Based Service-Oriented Architectures, which is a special form of architectural-debt.

Rothman [183] mentioned that there are different levels of technical debt those are related to system development life cycle (SDLC) phases, such as design debt, development debt, testing debt, etc. In addition, Krucheten et al. presented the Technical Debt Landscape, which includes visible and invisible elements [122]. For example, adding new functionality is considered as a visible element while low internal quality and code-style violation are forms of invisible elements. Figure 3.3 illustrate Technical Debt Landscape.

Figure 3.3 Technical Debt Landscape [122].



In fact, the majority of work has been investigating technical debt at the code-level. In this context, technical debt can be incurred either because of postponing some important code activities such as documentation or because of a bad code writing. It can be also related to updating, improving, rewriting an existing code, or poor code structure (e.g. [74, 205]). Code-level technical debt span across different problems those are related to code artifacts such as design, bugs, code-quality, etc. Technical debt on code-level can be incurred, for example, because of poor code quality or lack of documentation, where there is no supporting documentation provided with a specific code. Another example is the absence of comments or even poor comments writing or poor programming practices (e.g. [191]). In addition, lack of experience or knowledge of the code developer, when writing codes, can be a reason of incurring technical debt on code-level. Lack of experience can lead to further problems including increased code complexity, poor performance, low maintainability, fragile code, etc. [196].

The second level of technical debt is the design-level (e.g. [123, 79]), where technical debt is either related to code design or system design. Here technical debt is linked to quick and dirty design choices and design shortcuts that compromise the design quality [48]. The consequences of such choices and shortcuts will slow down the intended project. In software engineering, system design refers to the process of identifying the system and its components, interfaces, modules etc. in order to fulfill requirements [202]. Any rush or impulsive decision at system design phase may lead to technical debt (design debt). Some other design well-known problems that may lead to technical debt are anti-pattern and design flaws [148]. The concept anti-pattern defines a commonly used act as a solution for a specific problem and as a result it generates negative consequences [51]. This could be a result of an inexperienced manager or developer with little knowledge about the problem. Here, the manager or the developer keeps on mapping a general form of solution to a specific class of problems. For example, the act of applying the same design pattern to different problems in an inappropriate context. The second design problem is the design flaw. In object-oriented software engineering, design flaw refers to the act of developing class entities in an improper way that lead to maintenance difficulties [149]. Some design flaws that were published by Marinescu [148] such as intensive coupling (when a method calls other methods too many times from the same or different classes), large class (it is called the God class as well, which is the centralized class that calls other classes and uses its data), data class (those are data containers with little or no functionality at all), brain class (a very complex class), code duplication (the duplication of lines of code in different methods), etc. Such flaws can lead to technical debt on design-level as well.

The third level that will be discussed is the Testing-level. In systems development life cycle (SDLC), testing has many types such as unit testing, system testing, regression testing, black-box testing and white-box testing [203]. In software engineering, system testing refers to the process of evaluating the system in terms of fulfilling requirements. Here, technical debt can be incurred as a result of incomplete test strategies (e.g. [61]). Testing debt refers to the act of reducing or not developing testing phase of a specific software [183]. Lack of test plans and test suits may lead to testing debt. Sometimes when automated test is used with the purpose of reducing testing cost some problem can occur which lead to testing debt [227]. Here the automated test may imply some advantages such as cost reduction, as well as some disadvantages such as incurring technical debt that should be managed. Testing debt was defined as: "feature testing to attain one aspect while simultaneously ignoring-either knowingly or not-other aspects that may prompt up later with increased rework and cost [7]."

The fourth level of technical debt that will be discussed is the Architectural-level. Software architecture, specifically, is the system's high-level structure [87]. Kruchten et al. mentioned that: "technical debt isn't related to code and its intrinsic qualities but to structural or architectural choices or to technological gaps [122]." Technical debt on the architectural-level could be related to different dimensions such as design artifacts. Some architecture activities may lead to technical debt if not completed properly, such as architecture evolution, architecture maintenance, etc. What we mean by architecture evolution is the process of developing and enhancing the exiting architecture, either for the purpose of improvement, fixing defects, or as a result of adaptation to new environmental changes in order to maintain requirements satisfaction. In addition, the architecture evolution process includes adding new features, elements, functionalities, etc. or editing existing ones. Any non-strategic or quick decisions of these activities can lead to technical debt on the architectural-level. Such decisions need a continuous analysis, as any reckless act may cause many problems and increase maintenance costs accordingly. In subsequent sections, we discuss a special level of architectural debt explicating cloud-based service-oriented architectures.

Furthermore, many other researchers have been discussing technical debt on many other levels such as defect debt [201, 111]. For example, Snipes et al. proposed that when decisions about fixing some defects are deferred, a technical debt will be accumulated. However, such approaches can be mapped to the available levels and can be considered as a sub-level of the available ones. For example, Snipes et al. can be mapped under code-level debt. Nevertheless, we have presented the main four levels of technical debt. As we have seen, descriptions of some of technical debt levels have overlaps among each other and some others are interrelated somehow. Generally in software engineering, this is normal as these levels can be seen as sub-levels of the main root, which is system debt (software, hardware, etc.) in systems development life cycle (SDLC). All in all, lack of standards on different levels can lead to technical debt.

3.8 Service-level Technical Debt

Most of research works, which was discussed above, linked technical debt on various levels to code writing and code development. Unlike previous work, we introduced a new form of architectural debt explicating service-level in Cloud-Based Service-Oriented Architectures (CB-SOA). Specifically, we view services as "black-box" of code, when dealing with technical debt on service-level in CB-SOA (more details about web services and CB-SOA can be found in chapter 2). We relate technical debt to service selection or substitution decisions in CB-SOA and investigate the consequences of such decisions. We define technical debt on service-level in CB-SOA as:

the gap between "expected-level" of utility, expressed as QoS, and the "actual-level".

As mentioned before, we view investment in web services as a loan, which may incur interest by time and signal a probable technical debt. In this sense, we define the loan as:

The compensation, which is paid for achieving the highest satisfactory level of utility, either for the purpose of achieving better business or technical objectives, when utility-level is less than the satisfactory level.

And accordingly, interest may be incurred on the loan if technical debt is not properly managed by unlocking the architecture potential for value creation. We achieve a satisfied-level of utility by improving a specific QoS dimension, i.e. improving utility expressed as QoS, as QoS dimensions impart the utility of the architecture in CB-SOA. When improving QoS, we may pay some fees such as rework cost (in terms of time and money). Utility gap is related to one or more QoS dimensions (e.g. availability, scalability, capacity, etc.) depending on service selection scenario. For example, service capacity can incur technical debt if it is less than what is needed, or more than what is needed. Here, we should either switch to a service with more capacity or less capacity respectively, for the purpose of achieving better utility-levels.

In Cloud-based Service-Oriented Architectures, technical debt on service-level can be (1) *unintentional* that is because of bad engineering practices, and (2) *intentional* that is when we decide to take on the technical debt for generating future value. We view the intentional technical debt as a strategic tactics for gaining future value. The debt can be visible in different situations, which are related to service-selection in CB-SOA. For example, in case of poor selection the debt can be accumulated as a liability and interest on the web service. To manage and clear the debt, we may call for switching. More details of situation leading to technical debt on service-level will be discussed shortly.

Figure 3.4 illustrates three different possible situations of Technical Debt on the Service-Level in CB-SOA. It describes patterns, which are related to some selection challenges (challenges will be discussed shortly). For example, the unintentional technical debt may occur due to unsuitable web service selection and may incurred on the current web service before switching (TD_{WScurr}), illustrated in Figure 3.4-part(A). The second category of technical debt, which is the intentional one, that may occur on the rework cost (TD_{RW}) due to switching as we decide to take on the technical debt in order to gain future value-added, illustrated in Figure 3.4-part(B). Furthermore, Figure 3.4-part(C) illustrates our third case scenario when we decide to minimize the unintentional technical debt by switching to another service, which can carry its own intentional technical debt (TD_{RW}). Though the decision is deemed to curry a technical debt, it is likely to unlock future value-added (V^t_{WSnew}) on the structure and clear the debt (this will be linked to options in chapter 5). We formulated technical debt and manage it with respect to Equations 3.1 and 3.2. In any of the previous cases, when we decide to take on an intentional technical debt either through switching or on the new service (Figure 3.5); we expect the intentional debt to be less than the current technical debt (equ. 3.1). On the other hand, we expect the expected future value-added to cover for the intentional debt; otherwise we do not take the decision of incurring a debt (equ. 3.2).

$$TD_{RW} + TD_{WSnew} << TD_{WScurr} \tag{3.1}$$

$$V^t_{WSnew} >> TD_{WScurr} + TD_{RW} \tag{3.2}$$

Figure 3.4 Sample of Technical Debt positions on service-level in CB-SOA.

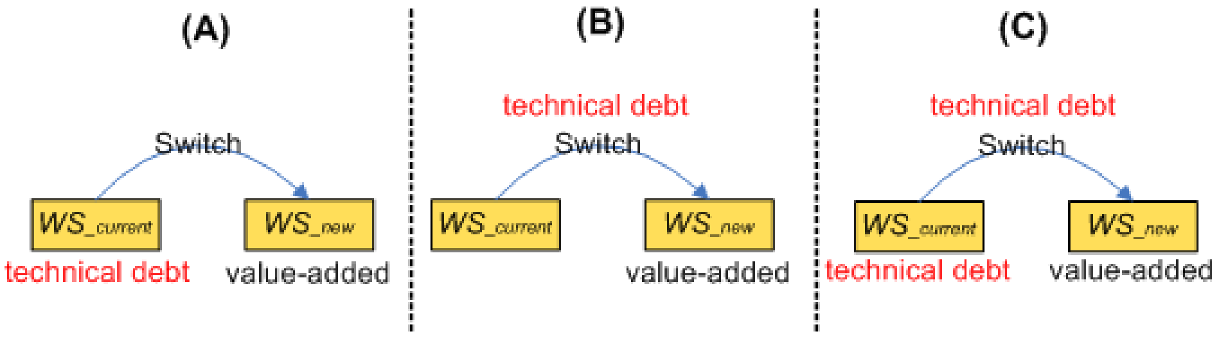
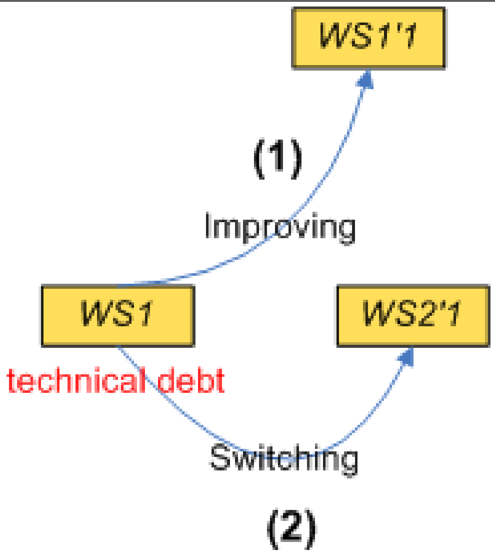


Figure 3.5 Managing Technical Debt on service-level in CB-SOA.



For a candidate web service to be selected, we may encounter two possible debt types when making a selection decision: (I) Solvable technical debt (manageable): occurs when the strategic decision of taking on a technical debt will pay off in future. For example, if we consider a scalability scenario, if the candidate web service is providing more than the currently required capacity. We may accept a technical debt in the view that the capacity will be utilized to generate a value-added in support of probable future demands (scaling up). (II) Unsolvable technical debt (unmanageable): when technical debt will not pay off in future. Back to previous example, here the capacity will continue to be underutilized and accordingly the value is unable to cover for the cost.

3.8.1 Technical Debt Drivers in CB-SOA

Before quantifying technical debt, we need to discuss how it occurs on service-level in CB-SOA. In this section, we discuss service selection challenges, which lead to technical debt on service-level in CB-SOA. This can be related to different situations that will be discussed shortly. We look at technical debt on service-level from two main angles: (1) Act and (2) Actor. The first angle is the Act, where the act is the action being taking on service-level. Here we differentiate between three different classifications of acts when taking the decision of service selection or substitution in CB-SOA. (1.1) Strategic acts that lead to intentional technical debt. (1.2) Impulsive acts that lead to unintentional technical debt. (1.3) Accidental acts that incur technical debt accidentally, either because of a cloud service provider or an environmental condition. The second angle is the Actor, where the technical debt is linked to the performer of the act. (2.1) The first actor is the CB-SOA decision maker (e.g an architect). (2.2) The second actor is the provider or the publisher of the service, which is the cloud service provider in our case. (2.3) The third actor that we consider in this thesis is "Environmental conditions", where the act is involuntary. We are mainly concerned about selection scenarios related to CB-SOA decision maker (e.g. an architect), however we will discuss some other acts, which are performed by other actors and affect the architect decision.

3.8.1.1 CB-SOA Decision Maker

The first type of actors is the CB-SOA decision maker (e.g. an architect). Here, technical debt can be attributed to different web service selection challenges on service-level. This can be related to different situations such as:

(1) Budget restriction: sometimes, when architects need to select a specific service, budget limitation can restrict the selection process. For example some services may be attractive but "unaffordable" due to some budget constraints. Budget can be restricted due to different development plans such as preserving cost, reducing startup principal, development expense postponement, etc. Budget restriction has different extremes that

lead to technical debt in web service selection. First, budget restriction may influence the architect's selection decision by selecting unsuitable web service, i.e. select the affordable service within the budget. Second, technical debt can be incurred when the selected service is expensive and its expenses outweigh the revenue streamed from using such web service. Here, over-expenditures can lead to technical debt on service-level.

(2) Selecting and improving the overall Quality of Service (QoS): QoS is a significant factor that distinguishes web services. The key to web service selection decision is to understand the future change in QoS. If we consider the service availability attribute, for example, the service should support availability requirements by accommodating specific amount of users at specific time. Any failure in meeting availability requirements can contribute to lose in business value. As a result, the utility is affected. And in this sense, the difference between the "expected utility level" and the actual one can be viewed as technical debt. Some other key factors that should be taken into consideration: the cost of achieving the "expected-level" of QoS, expressing the utility. Here, technical debt can be cleared, for example, by fully utilizing the selected service's QoS (e.g. capacity). As mentioned previously, we view the investment in web services as a loan, which may incur interest by time. In this scenario, as the number of users benefiting from the web service increases, the loan is said to be paid through installments reducing the interest on the loan. An installment is said to be paid by active users, which are charged for using the web service. The web service selection decision will pay off, when revenues of the selected service cover the technical debt.

(3) Underutilization: technical debt can be related to situations, where the service capacity is underutilized and the operational cost outweighs the revenue streamed from using the service. For example, web service underutilization occurs when the available capacity is not being used to its fullest potential, i.e. the web service is underutilized by having much less load than what its capacity can normally handle. In this case, the selected web service is said to be over-flexible. Technical debt can also be related to situations where the web service capacity does not match the demand requirements

because of inappropriate management. In this case, the flexibility of the selection decision in response to changes is unlikely to add.

(4) Swift selection: Poor and quick decisions may add value in the short-term, but can introduce long-term debt in situation where service substitution is unavoidable. Sometimes the decision maker (e.g. an architect) may avoid exhaustive search, which could be a time-consuming process, due to time restriction such as deadlines and timely critical decisions. Candidate web services may be selected from different web service providers and therefore a process of discovering the suitable provider for each candidate can be a time-consuming task for architects. In this case, technical debt can be incurred because of two different acts from architects. The first one is when the architect intends to save searching-time. Here, the architect may select the first available web service that satisfies requirements. Sometimes when there are many candidate web services satisfying requirements, a random selection is carried out. On the other hand, the second one is when the architect intends to save money. In this case, web service selection decision may insure the minimum price of the selected web service; however, it ignores the long-term value. There must be a balance of both acts, as any irrational decision can lead to unsuitable service selection and an unmanageable technical debt as a result. We argue that the service selection decision requires a tradeoff analysis among cost, value and lifetime of the candidate web service (SLA contract). Any ignorance of these three tradeoffs can lead to unsuitable web service selection.

(5) Sometimes technical debt can be incurred intentionally as a result of solving an earlier inappropriate decision. In this case, the architect may have already made a former wrong decision (unsuitable web service selection), which has incurred an unintentional technical debt. So s/he might think of minimizing such unintentional technical debt by taking on an intentional debt. Here, we see the intentional technical debt as creating future opportunities; however, it costs an “opportunity fees” (this will be linked to flexibility and options in chapter 5).

(6) Technical debt on service-level can be also incurred due to lack of long-term vision

and planning. This could be the case when dealing with inexperienced CB-SOA decision makers (e.g. architects).

3.8.1.2 Cloud Service Provider

The second type of actors that we discuss in this thesis is the Cloud Service Provider. In this part, we discuss acts performed by the service provider, however affectingCB-SOA decision makers (e.g. architects).

(1) Deadlines and schedule constraints: commitments to customers with wrong estimation could create the fear of the risk of not delivering the service on time. Technical debt can be incurred accidentally (not by architect) when service providers rush the release of their web services to the market under the pressures of meeting deadlines. This can be, for example, through falsely accelerating the velocity by reducing testing and verification of the features delivered with web services. In this context, when web services are selected based on the sole trust of the service provider, the choice may accidentally introduce a technical debt, which is often left to the application developer to visualize and manage.

(2) Trust issue and Service Level Agreement (SLA) violations: when we subscribe to a web service, we actually borrow a “black-box” from the web service provider. Technical debt can be attributed to situations, where the selected web service is at risk of SLA non-compliance. Technical debt can be also attributed to situations, where future technical support and maintenance is not clear and not fulfilled by the web service providers. Furthermore, in the absence of publicly available benchmarks and historical performance data to verify web service provision and its compliance, uninformed service selection decisions may carry likely risks, which can lead to SLA violations.

(3) Cloud marketplace competitiveness: as mentioned before, we view the cloud as a marketplace for trading web services. The rule is to release services as quick as possible, in order to “reserve a place” in the cloud marketplace. Cloud marketplace is very competitive for web service providers, as web service consumers have the choice to select providers who satisfies their demands. Technical debt can be incurred when the competition is high

and services must be released as soon as possible, e.g. cut down time to market.

3.8.1.3 Environmental Conditions

The third type of actors that may be the performer of incurring technical debt on service-level is "Environmental conditions". This can be attributed to:

1. Requirements mismatch: when the selected web service features do not fully match the requirements of the application. In this case, fixes may be required to reengineer the solution to better fit the changes in requirements.

2. QoS fluctuation and utility levels variations: here, technical debt can be incurred due to the difference between the utility "expected-level" and "actual-level".

3. Changing or new environment: (e.g. moving toward mobile cloud).

4. Technology debt on service-level, such as the rework cost needed to make this service available to new operating systems.

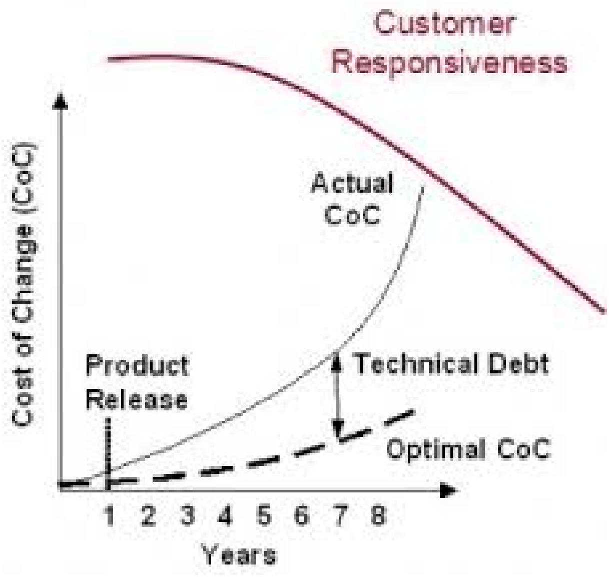
5. Adjusting priorities list: this can happen because of receiving new information from the marketplace or during unexpected situation such as changing requirements.

3.9 Technical Debt Quantification on Service-Level

We quantify technical debt in order to inform the service selection decision in CB-SOA. Once the selection decision is made, the technical debt is said to be active and needs to be managed. We look at technical debt on service-level in order to anticipate the future value of a candidate web service by quantifying the cost of achieving the intended utility-level. The aim is to quantify an estimate of technical debt on service-level during the selection and substitution procedure.

Some available research work has examined technical debt on various levels and quantified it in difference ways with respect to time and cost. The most well-known work is the technical debt curve presented by Highsmith [101], which is quantifying technical debt as the difference between the actual and the optimal Cost of Change (COC) with respect to time, illustrated in Figure 3.6. Likewise, Gat presented a rating approach that links technical debt to development cost [88].

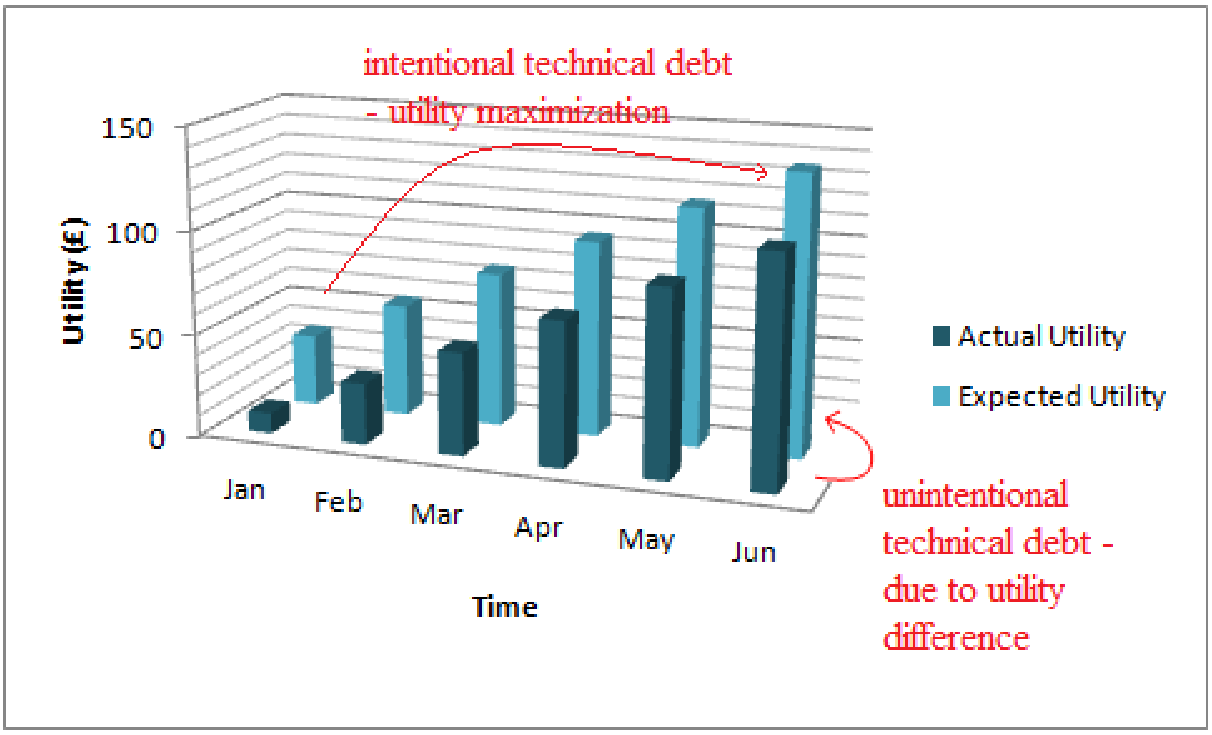
Figure 3.6 Technical Debt Curve: technical debt as the difference between the actual and the optimal Cost of Change (COC) [101].



Other example is the work presented by CAST [69], in which they calculated technical debt as the cost (in pounds) of addressing the percentage of violations of architectural and coding good practice rules. They had three different levels of violations; high, medium, and low severity. Nugroho et al. [170] estimated technical debt as the amount of rework (in man-month), which is required in order to improve the software to a higher level of quality.

However, we look at technical debt from satisfactory utility-level perspective. We define and deal with three different values on service-level: (1) Technical Debt value, (2) Utility value (where QoS values impart the utility of the architecture), and (3) Option value (Real Options will be discussed in chapter 5). Moreover, quantifying technical debt on service-level is much harder, as we are dealing with “black-box” of code. However, other approaches estimate technical debt with respect to line of codes, for example, which is easier in term of quantifying numerical values. In order to accept a specific amount of technical debt, we define technical debt threshold and satisfaction level of web service’s value, which is expressed as QoS and impart the utility of the architecture. Here, the difference between the utility "expected-level" and "actual-level" is triggering a technical debt (in terms of rework cost and time) that is likely to increase if the value of the selected web service keeps on degrading. We quantify technical debt as the loan, which is paid for achieving the highest satisfactory level of utility, i.e. moving from the current unsatisfactory level of utility to the "expected-satisfactory-level". We view this cost as a loan, which is anticipated to pay off in future. Taking on technical debt in order to reach the highest satisfaction level will pay off in future in terms of flexibility and reducing cost. After quantifying technical debt in terms of pounds (time and cost), we use this quantified value of technical debt as a *metric*. Technical debt metric is a monetary unit (in pounds), which includes time, effort and calendar months. Subsequently, this metric can be used by CB-SOA decision maker (e.g. system architects) against other tradeoffs in order to quantify the decision of selecting a specific service at a specific time in CB-SOA. Figure 3.7 illustrates our two possible cases of incurring technical debt (intentional and unintentional) on service-level in CB-SOA.

Figure 3.7 Technical Debt due to utility differences.



We say that technical debt has a lifetime, which we link to the web service expiry date. When a candidate web service is selected, the service level agreement (SLA) mandates an expiry date of the service. The time until expiration is used for tracking the value and clearing the technical debt. When selecting a service, we are specifically interested in tracking how fast the loan will be paid back. We posit that technical debt on the service-level has a three phases lifecycle; (1) when it starts, (2) when interests are incurred (if any), and (3) payback process: when the selected web service is expected to clear out the technical debt and generate future options. We argue that technical debt is sensitive to some tradeoffs such as operational cost and value generated as a result of acquiring the web service, which are associated with web service lifetime. We formulated technical debt using the following equation:

$$TD = \max[-1 * (OP - RW), 0] \tag{3.3}$$

Where *TD* is the technical debt, *OP* is the option value (discussed in chapter 5), and *RW* is the rework cost, which includes but not limited to the upfront cost of acquiring the new web service plus the accumulated interest over time, if any (for some cases, e.g. option-to-defer). We assume that when option value starts to exceed the rework cost, technical debt will be cleared out. In this case, the technical debt is said to be zero. In chapter 5, we use options valuation based on binomial tree analysis as a way to track and manage the technical debt and to visualize its lifecycle. We also demonstrate how CloudMTD model will be ranking web services according to technical debt value and its clearing date.

3.10 Discussions and Conclusions

To sum up, cloud marketplace offers web services with varying QoS and cost. CB-SOA is composed of these web services. In such architectures, technical debt may be incurred while substituting the constituent services. Considering technical debt in service selection is motivated by the need for taking strategic tactics for gaining long-term value-added. In this sense, we have introduced a new level of technical debt, which is the service-level in CB-SOA. Technical debt in CB-SOA service selection can be attributed to different situations such as poor and swift selection decisions, mismatches in applications' requirements with that of the service provision, falsely accelerating the velocity of the integration, testing and deployment process due to budget restrictions, and/or through accidentally acquiring the debt via an untrusted provider. Technical debt concept is a metaphor that has been examined in different fields taking into account different dimensions. However, none of the available work has discussed technical debt in service selection in CB-SOA. We argue that technical debt on service-level can create future value-added to CB-SOA architectures if managed properly. In this sense, our work is different from the available approaches in some point as follows:

- First, from concept point of view, when taking on technical debt on the current service, we assume that the cost of today (e.g. development cost) may be cheaper than tomorrow (using options thinking). On the contrary, McConnell mentioned that cost of development today is more expensive than the cost in future [152].
- Second, in 2008, McConnell mentioned that the intentional technical debt is a decision for optimizing for the present rather than for the future. On the contrary, we are optimizing for the future when taking the decision of selecting a web service, taking into consideration future uncertainty and changes (e.g. QoS fluctuation).
- Third, from a technical debt level perspective, we are managing technical debt by taking decisions on service-levels in CB-SOA. Moreover, we are dealing with the selection decision as a long-term investment, which takes technical debt into consideration strategically and proactively.
- Fourth, some research work such as the work of Software Engineering Institute (SEI)[1], takes on the technical debt decision mainly for shorting time to market

[1]www.sei.cmu.edu

and preserving time and principal. However, we are not dealing with releasing software; instead we are dealing with a service as a stand-alone component; black-box. In addition, when dealing with starting up scenarios, we say that a new startup company may scarify some short-term business value for the benefit of starting their business and gaining long-term value (e.g. low availability and losing customers).

- Fifth, some other research works discussed technical debt for the reason of delaying development expenses. However, we will show in next chapters that we may be dealing with “defer-option” for the purpose of quantifying the time-value of the selection decision. We are more dealing with “solvable” technical debt in order to quantify the benefits, rather than “unsolvable” technical debt which may incur bad consequences and impacts on the structure. When technical debt is unsolvable, we either abandon the current service or select another one.

The contributions of this chapter are as follows: (1) We have provided a comparison of different technical debt approaches those are available in the literature, in terms of TD level, dimensions, solutions, and evaluation. (2) We have also reported on some of the available technical debt definitions in the literature. (3) After that, we have introduced and defined a new form of architecture technical debt explicating service-level for Cloud-Based Service-Oriented Architectures. (4) We have also introduced the likely drivers of technical debt on service-level in CB-SOA. (5) Finally, we reported on the formulation of technical debt on service-level in CB-SOA.

4 Requirements for Evaluating Selection and Composition Decisions in CB-SOA

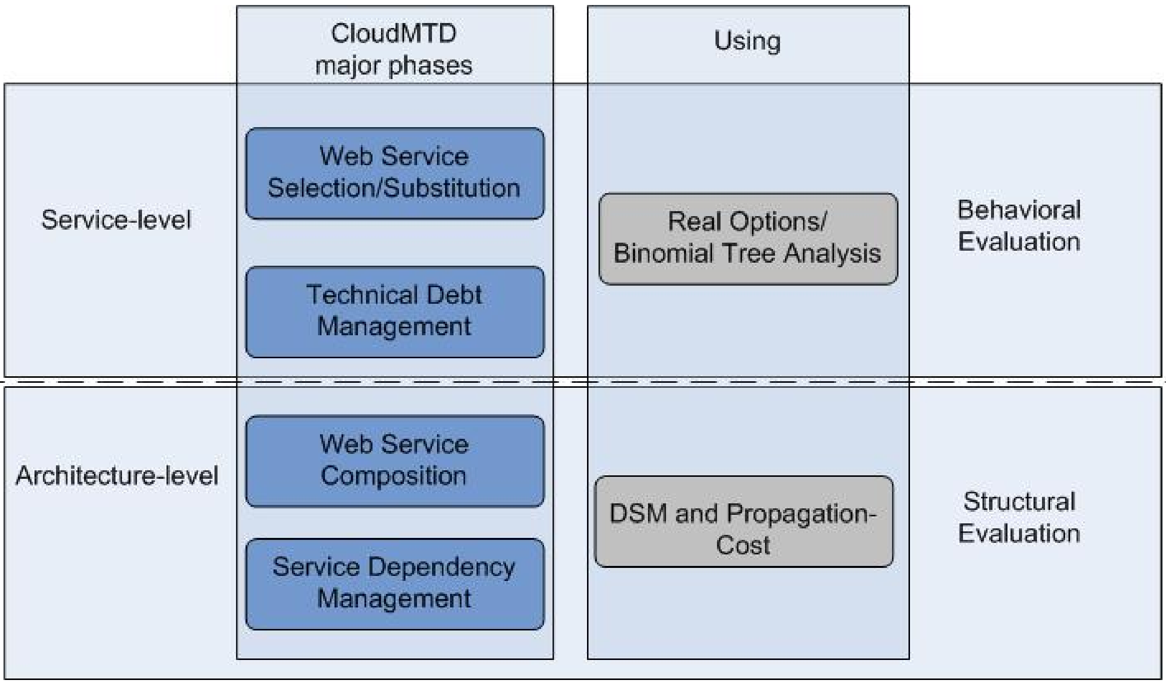
4.1 Overview

In previous chapters, we have presented a review of research work on (1) web service selection and composition, and (2) technical debt and its available management approaches. In this chapter, we state the requirements for evaluating a selection and composition decisions in CB-SOA. This process consists of (1) managing the likely technical debt on service-level and (2) managing services dependencies on architecture-level. Satisfying user requirements, business requirements, and services' QoS requirements are not the only concerns in web service selection and composition approaches. Instead, in this chapter, we would like to present other aspects and dimensions that should be considered as major characteristics when taking the decision of selection and composition in CB-SOA.

We adopt Garlan's notion of a Software Architecture by viewing it as a bridge between requirements and implementation [86]. Besides, Software Engineering Institute (SEI) [40] mentioned that an architecture is a suitable bridge to link business goals and software systems. In this sense, we believe that the architecture is an appropriate level of abstraction for evaluating the strategic selection and composition decisions in CB-SOA. Since this level is believed to be general enough to present the applicability of the evaluation process in order to be used in solving other related problems in other domains (e.g. the prediction of likely behaviors in future). In this sense, this thesis presents the CloudMTD model (ch. 6), which is developed based on a set of requirements that we believe it meet the needs of a major number of potential CB-SOA decision makers. In this sense, we investigate the requirements of evaluating a selection and composition decisions in CB-SOA with respect to two levels; service-level and architecture-level (illustrated in Figure 4.1). This is done in the view of achieving the following goals: (1) Creating value-added (long-term), (2) Maximizing utility and (3) Reducing Technical Debt (if any). In order to achieve the previous goals, some consequences may appear and need to be addressed, such as: (1) Dependency, (2) Future Changes (Uncertainty) and (3) Technical Debt. This thesis is concerned with the following uncertainties:

1. Behavioral Aspects, such as QoS fluctuation.
2. Structural Aspects, such as complexity of services dependencies.
3. Environmental Aspects, such as cloud uncertainties.

Figure 4.1 CloudMTD in a General View



4.2 Service-Level Requirements: Selecting a Web Service

In this section, we present requirements for the "behavioral" evaluation that is performed on service-level of the architecture based on Real Options analysis (Real Options is presented in chapter 5). Following the argument of Kevin Sullivan [209], we view a web service as an asset and its value includes not only the likely generated revenues, but also the value of options it creates. In this sense, on service-level of CB-SOA, we justify the selection decision based on the notion that part of the value of a web service is in the form of embedded options.

4.2.1 Flexibility

As we mentioned before that part of the value of a web service is in the options it embodies. In this sense, we say an option has a value as it gives the decision maker the flexibility to decide about a web service (an asset) whose future value is unknown today. However, flexibility comes with a price. In this sense, in order to know whether or not to invest in flexibility, we need to weight the cost against the value of the selection decision with the purpose of maximizing utility. However, this is challenging as usually the cost is tangible (quantifiable), but value is not (as value is subjective). In options terms, "Flexibility is nothing more than the collection of options associated with an investment opportunity, financial or real [219]." In next chapters, we will show how we treat flexibility in web services' "investments" as an option and value it as such. In this context, we will be reasoning about the "value of flexibility" by quantifying the "value of an option". On service-level, we view flexibility from different perspectives: (1) accommodating future changes, (2) uncertainty, and (3) time-value. Our model is capable to show how and under what conditions investments in flexibility add value.

4.2.1.1 Flexibility of Accommodating Changes

On service-level, future changes are attributed to change in QoS attributes. Here, the act of change is done by CB-SOA decision maker with the purpose of satisfying technical or business objective, such as improving scalability. When taking a decision of selecting a web service, we should consider flexibility, which is the ability of a web service to accommodate these changes and take into consideration their possible impacts. In this sense, what constrains the success of the selection decision is the ability of the candidate service to accommodate likely future changes.

4.2.1.2 Value of Flexibility under Uncertainty

Here, we posit that options provide decision maker with a valuable flexibility to invest in web services by substituting services under uncertain conditions. Here, flexibility can be related to the ability of CB-SOA decision maker to substitute web services under demand uncertainty. This thesis presents a model that supports our claims and provides deeper understanding of web service selection and composition principles by connecting the value of a given web service to value of flexibility under uncertainty. In decision theory [169], uncertainty is defined as the lack of certainty because of having limited knowledge about the possible future outcomes. In **web services context**, uncertainty on service-level in CB-SOA is related to the behavioral aspects of a service. This can be related to uncertain willingness of a web service to be able to provide its provision with the intended QoS attributes' values, for example. In this sense, uncertainty on service-level in CB-SOA is related to QoS fluctuation and future changes. Here, uncertainty of availability, for example, can be related to how critical downtime with respect to business value is. In **CB-SOA context**, we define three major types of uncertainty on service-level, which need to be addressed when evaluating the selection decision: (1) Future Changes and (2) Value/Value-added.

Future change is one of the major sources of uncertainty. In this context, uncertainty arises from the presence of future changes in many dimensions those are related to selecting a web service. For example, this can be related to unpredictable fluctuation in load and likely fluctuation in QoS requirements. Furthermore, future changes, on service-level, can be related to uncertain behaviors of multiple QoS attributes and their interactions in future. Here, we view flexibility as the candidate/selected web service being able to accommodate future changes. We assume that flexibility creates value in the form of future embedded options. For example, selecting a web service with respect to future business expansion carries flexibility in the form of growth options. In this sense, we view the "flexibility" as one of the main contributors to value-added.

The second type of uncertainty is related to value and value-added of a candidate web service, which is uncertain and may take long time to be obtained. In web service selection context, the value-added can be attributed to the likely increase in revenues generated by improving the QoS expressed as utility ($\Delta V = V_{t1} - V_0$), or reduced operational cost, etc. Here, we say that the selection decision is taken by a decision maker to achieve a desired level of QoS, which in turn imparts the utility of the architecture. For the behavioral evaluation, service value is expressed based on one or more QoS attribute according to the selection scenario (e.g. improved throughput, availability, scalability, etc). We assume that CB-SOA acquire flexibility by having the ability of substituting its constituent services for the purpose of improving its utility. In this type of uncertainty, we value flexibility by looking at options it creates through substituting with the view of improving utility, the value of the structure and reducing technical debt under uncertainty. In this context, when investing in web services, the value-added is strategic in essence, which may not be immediate and may take long time to be acquired.

Furthermore, As we are taking decision under uncertainty in CB-SOA, the solution should be flexible enough to manage such uncertainties. Uncertainty makes Real Options theory an ultimate choice to be used as a valuation technique, since other valuation techniques has some shortages in dealing with the value of flexibility under uncertainty. Option theory is typically applied to such decision because managing uncertainty requires flexibility in dealing with such decisions when there are multiple possibilities for future events [21].

4.2.1.3 Time-Value of a Selection Decision

The third perspective is concerned with the flexibility of the selection decision with respect to its time-value, i.e. when to select or substitute a candidate web service. In general, options theory is concerned with deciding on the optimal timing of a given investment decision [208]. In chapter 6 we quantify the time-value of the selection decision, which has the promise to add long-term value and reduced technical debt, if any. Deciding when to invest in web service's flexibility is transformed into a decision about when to exercise the option in the view of maximizing utility. In option terms, this is providing the decision maker by flexibility to exercise the option at any time. In web service selection context, the flexibility is related to the decision maker being free to exercise the option and acquire the web service at any point of time without obligation. This is useful especially when the outcomes of the selection decision are uncertain. When decision makers have the flexibility whether to invest now or wait, the question is how to quantify the time-value of such decision with the purpose of maximizing value. In next chapters, we will be answering such question.

One of the problems in web service selection is that we are often encountered by scenarios where we need more information about the candidate service (to be selected). Such information may help us in making a better decision; however they are sensitive to time and entail many dimensions, especially when uncertainty takes place. This information can be related to QoS attributes, such as a service reputation. Time-value of the selection decision can be also related to the nature of demand; peak or off-peak daily usage; peak or off-peak seasons, etc. In the next chapter, we will link such time-sensitive decision to option thinking. For example, "time-value" in the context of waiting for more information can be linked to "option-to-wait" scenarios in option terminologies.

4.2.2 Managing Technical Debt

When evaluating the selection decision of a web service in CB-SOA, we should consider the likely technical debt that may incurred. In previous chapter, we have defined technical debt on service-level as the gap between "expected-level" of QoS, expressed as utility, and the "current-level". And we mentioned that the service selection decision may come with a technical debt. CB-SOA decision makers are normally interested in selecting the most "qualified" service. However, practically in many cases, once we select a web service we are often encountered by situations which lead to major problems. We view these problems from two angles. The first one takes place when the selected service deviates from and lags behind the "ideal-level of utility". Where the second one is related to service's provision, where we are interested in situation when the service is either providing more than what we require (over-provisioning) or less (under-provisioning). In either case, we refer to these scenarios as technical debt. More details on technical debt scenarios were discussed in chapter 3, section 3.8.1. We argue that our selection and composition model (CloudMTD) shall predict technical debt in such situation before it takes place (before we hold the debt). Here, we insist that using a retrospective approach may be too expensive.

Managing technical debt is one of the major requirements that we consider when we evaluate a selection decisions. Managing technical debt is a strategy for clearing technical debt and likely interests on a selection decision using CloudMTD model. This act may add a strategic long-term value that may not be immediate. Here, we say that while managing technical debt, we should take into consideration the cost of achieving the highest satisfactory level of utility, either for the purpose of achieving better business or technical objectives, or for the purpose of fixing some unintentional QoS problems expressed in utility-levels. In this context, technical debt is an operational liability which may incur an interest if not managed, cleared and transformed from liability to value. Technical debt is one of the main contributors to value-added once it is cleared. Technical debt was explained in details in chapter 3. This phase is performed on the **service-level**.

4.3 Architecture-Level Requirements: Composing Web Services

In this section, we present requirements for the "structural" evaluation that is performed on the architectural-level of CB-SOA. The structural aspect evaluation for a candidate web service is carried out using Dependency Structure Matrix (DSM, also known as Design Structure Matrix) and the Propagation-cost metric for the purpose of visualizing the "ripple" impact of the change on the structure. We will be using DSM in order to model the structure of CB-SOA and its constituent services' relationships. Propagation Cost metric will be used for computing the percentage of system elements that may be affected, on average, when a change is made to a randomly selected element [144]. Detailed formulation of both DSM and propagation-cost metric are shown in chapter 6. Composing services is performed on the architecture-level of CB-SOA. And the evaluation on this level differs mainly on how we treat dependency and its accompanied complexity.

4.3.1 Addressing Services Dependency

Dependencies among web services are quantified using DSM, where one means dependent and zero otherwise. We take into consideration that the structure can change by time and so dependencies among constituent services in a given architecture. On the architecture-level, when quantifying dependency, we take into consideration rework cost (e.g. cost of change), which increases in case if other services are affected by the change. In chapter 6, we report on the formulation of dependencies using DSM and visualize time using Binomial Tree.

4.3.1.1 Services Dependency Complexity

Services dependency's complexity levels is treated on the **architecture-level**. We assume that CB-SOA decision makers (e.g. architects) can decide on the level of complexity per dependency through voting mechanisms backed-up by their experience and knowledge of the system. The complexity level of dependency between any arbitrary services may vary based on the complexity of the change [20]. We assume that the complexity of the change can be quantified based on the effort required. For example, code development, configuration, etc. Furthermore, we assume that the service instantiation of the composing service abstraction tends to imply different complexity on a given architecture [20]. This is to reflect dynamic changes of the structure and its composition at varying time intervals. As a result, the time value of the decision of composing candidate web services shall take into consideration the different complexity of the structure and its propagation cost at different times. Despite the fact that two services may be dependent, the complexity of the dependency may vary. For example, higher complexity may signal higher rework cost, lower complexity may indicate otherwise. This can be due to the effort required for code development, rewriting, configuration, data migration, new/throwaway maintenance, fixing mismatches, changes to legacy code, etc.

4.3.2 Flexibility

In this section, we consider the structural aspect of flexibility. Again, we posit that an option has a value by giving the decision maker the flexibility to substitute web services under uncertain conditions and unknown future value. Furthermore, the selection and composition decisions shall leverage on the flexibility that a web service substitution can buy from the cloud marketplace and the value-added on the cloud-based architecture as a result. On the architecture-level, we examine flexibility from three different perspectives those are related to the structural aspect of a given architecture: (1) accommodating future changes, (2) uncertainty, and (3) time-value.

On architecture-level, future changes can be attributed to change in requirements those are stated by stakeholders. Here, we mean the structural change to a given architecture that may happen in future. We should take into consideration the likely impact of such changes on other services. Furthermore, analyzing services' dependencies and the accompanied complexity at different time intervals have provided means of flexibility to evaluate the time-value of the substitution decision relative to changes in the structure.

On the other hand, we examine the value of flexibility under uncertainty, which is related to structural aspect of a given architecture. On the architecture-level, we attribute uncertainty to different factors those are related to the environment, in which the composition works, new market demands and conditions, new users' preferences, some economic constraints, etc. In addition to the dynamic nature of the cloud environment that exhibits uncertainty.

4.4 Strategic Decision

What we mean by a strategic decision is that we are improving and sustaining a specific value and QoS (e.g. performance) overtime (service lifetime) aligned with meeting technical and business goals while selecting and composing services. During the evaluation phase of selection and composition decisions, we should take into consideration not only operational benefits but also strategic ones. Strategic benefits may be reflected on the composition overtime with respect to future changes. On the operational side, selection and composition decisions may come with long-term operational benefits, such as maintenance cost reduction over time. However, such strategic and operational benefits have uncertain payoff and they are not immediate and may take time. Furthermore, evaluating selection and composition decisions in CB-SOA should address certain strategic dimensions: (1) timeline: in our case timeline is modeled as the service lifetime (contract). Likely future changes must be analyzed during the whole service lifetime. (2) Long-term value/value-added. (3) Impact of selection and composition decisions on other services in a specific composition.

4.5 Retrospective vs. Predictive Approach

In this thesis, we advocate a predictive approach for service selection and composition in CB-SOA. The predictive approach aids in anticipating service value and managing the likely technical debt in CB-SOA with respect to possible behaviors of the composition. Classical approaches tend to be retrospective. Unlike retrospective approaches, predictive approaches are preventive. The effort to valuation is justified and the evaluation is generally cost effective, as when compared to the retrospective ones. This is specifically true for the case of web service selection as we are dealing with impropriety solutions, where inappropriate and unjustified selection decisions can be costly to revert; retrospective analysis can be costly as a result. In this sense, the predictive approach saves time and money. With the increase number of service and the wide spread of "e-world", retrospective approach is no longer sufficient for solving the problem. This is because; the reaction may be too late and serious risk has occurred, or addressing the problem may be too costly if a reconfiguration of the composition is unavoidable, for example. On the other hand, the problem of dealing with uncertainty, which was explained in previous sections, motivates the need for a predictive approach. As if we are certain about the "output" of the selection and composition decisions, the process will be straightforward. And as a result, there is no need for such predictive approach. However, as we are uncertain of the "output" of the selected service, there is a need for an "intelligent" technique and an economic-driven approach that takes into consideration the previous problems when selecting and composing web services in CB-SOA.

4.6 Summary

We have highlighted the requirements for evaluating a selection and composition decisions, and managing any likely technical debt, in CB-SOA from an economics-driven software engineering perspective. We have also investigated some consequences and difficulties which take place as a result of the decision making in CB-SOA. Requirements and the associated consequences motivate the need for a predictive approach of web service selection and composition in CB-SOA.

5 An Option-based Model for Service Selection in CB-SOA

5.1 Overview

In the previous chapter, we presented the requirements for evaluating the selection and composition decisions in CB-SOA. We also presented some consequences and difficulties which are associated with selection and composition decisions in CB-SOA. In this chapter, we are going to present an option-based approach for addressing these requirements. In next chapter, we present a novel model that exploits options theory for evaluating selection and composition decisions in CB-SOA and managing the likely technical debt that may be incurred as a result of the selection decision. The model is referred to as CloudMTD [18, 19, 20]. Its phases will be described in details in next chapter (Chapter 6). The model builds on Real Options theory, Design Structure Matrix (DSM) and propagation cost metric for valuation purposes. In this chapter, we mainly focus on Real Options.

We first present background on Real Options theory as it is necessary to understand some concepts in our approach. We also provide its use in Software Engineering and highlights ongoing research in the field of Real Options. We also present the related work.

By using options thinking, we aim to answer the following questions: (1) why is a new service selection and composition technique needed? (2) Has anyone proposed an option-based approach before to solve it? (3) How does it differ from classical approaches available in the literature? (4) How worthwhile is it to invest in flexibility with the purpose of accommodating future changes? When does investing in flexibility create value?

5.2 Real Options: A Brief Background

5.2.1 Definitions

In this subsection, we present some definitions which are related to option theory. The option[1] concept is the core of real options approach, which must be mentioned and recognized. An option, in general, is the right to have the freedom of choice. In finance,

[1]Financial option is an option on stock and bonds, while real options is an option on real assets[219].

an option indicates that the owner has the right- with no obligation- to buy (or sell) an underlying asset at a specific price (strike/exercise price) on (or before) a particular date (Expiration date) depending on the type of the option; American or European [21]. Mainly, an option on a non-financial asset is called a Real Option, such as tangible investments, buildings, and software project. *"A Real Option, particularly, is the right - but not the obligation - to undertake some business capital investment decision; typically the option to make, abandon, expand, or contract a capital investment" [30].* A call option is a choice that gives the right - with no obligation- to buy an asset at a predefined price before/at a given date. On the other hand, a put option is a choice that gives the right - with no obligation- to sell an asset at a predefined price before/at a given date [219]. Moreover, when the option is exercised on the expiration date, it is called a European Option, and alternatively, the American Option can be exercised at any time before the expiration date [190].

5.2.2 Why Real Options?

As discussed in previous chapters, we are investigating the problem of web service selection and composition in CB-SOA from an economics-driven value perspective. We have also investigated uncertainties associated with the selection and composition decisions in CB-SOA. From value-based software engineering point of view, real options theory is suitable for addressing various software engineering problems with inherent uncertainty linked to the technical decisions [208, 45]. We present some characteristics, which makes real options a suitable approach for addressing service selection and composition in CB-SOA.

First, real options emphasize the value of flexibility under uncertainty. In addition, real options techniques help in quantifying and characterizing the value in environments full of uncertainties [208]. In previous chapter we explained in details the uncertainties associated with the selection and composition decisions in CB-SOA. In addition, we argue that cloud environments, such as CB-SOA, exhibit plenty of uncertainties, which are attributed to the emerging behavior of concurrent and unexpected modes of interactions of service and the environment. These are attributed to the dynamic elasticity and continuous evolution of the cloud topology (e.g. new services, mashups, unpredictable modes of service use, fluctuation in QoS and users' requirements). In addition to that, the accelerating pace of change is very high in cloud environment. From an economic perspective, flexibility of a selection and composition decision in CB-SOA has value under uncertainty. From this point, we are investigating one of the economics techniques, i.e. real options, in cloud environment, particularly CB-SOA. What links these aspects together is uncertainty. From a real options perspective, uncertainty is costly, however it creates valuable future opportunities [21].

Second, real option theory is a significant way of thinking that provides the power of valuing investments that involve strategic decisions from an economics perspective [162]. We argue that we need such a strategic vision when investing in web services, in particular, selection and composition strategic decisions in CB-SOA. In this sense, real options theory is compatible with web services' strategic investments which entail uncertainty and evolving conditions. We argue that such investments in CB-SOA must be proactively managed by responding to the changes.

Third, real options valuation identifies the value of the investment, which lies not only in the expected direct revenues, but also in the future opportunities that flexibility creates [208]. Economics-Driven Software Engineering researches posit that engineering activities need to be examined by their contribution to value-added and value-creation [45]. In addition, real options helps in capturing value in new investments and adds flexibility to decisions-making process which will take into consideration unexpected future developments [209]. Here we say that decision makers in CB-SOA can create value by exercising available options which are linked to web services investments.

Fourth, inputs to real options techniques are not subjective [21]. Rather than that, real options valuation is based on data from market and it takes into consideration the variance of the value of a given asset over time (web service in our case).

All in all, we appeal to real options thinking to provide us with better insights and understandings on the key principles of decision-making in the field of web service selection and composition in CB-SOA. This is because real option theory captures the future value of flexibility and interprets it in terms of "options", i.e. treating flexibility as an option. The values of these options are estimated using option pricing techniques. The idea is to treat the flexibility of a selection decision as an option and then to value such flexibility using option pricing techniques. Notably, we are interested in valuing flexibility under uncertainty in CB-SOA selection and composition decision-making. However it is not the only goal, where other goals are related to value-added and technical debt management, if any. Accordingly, we say that flexibility and value-added are correlated; however both entail cost (flexibility was discussed in details in previous chapters).

5.2.3 Option Pricing Plot in Brief

Option Pricing Theory has a long history that commenced in 1900 [156]. Afterward, many contributions have been published in the area of option pricing. On the other hand, the real option theory is originated from the financial option theory, which mainly deals with options on financial assets. Black-Scholes is the most famous formula for pricing European options [41]. Black-Scholes pricing model was not obvious at the beginning, until Stewart C. Myers published his article [161]. The "Real Option" concept was coined by Myers. The notion of valuing investments using real options was originated by Myers [161] when he noted that corporate assets can be viewed as call options. The value of options relies on the discretionary future investments of a specific corporation. He viewed these investment opportunities as growth options. He mentioned that part of the value of a given firm is justified by the present value of future growth opportunities [161].

In 1970, Robert C. Merton[2] was the Professor of Finance at the MIT Sloan School of Management [3]. Merton[156] extended the Black-Scholes options pricing model [41] and expanded the mathematical understanding of the model. Merton contributed to Black-Scholes model by investigating the value of derivatives and examining the effect of

[2]Merton was a student of Myers

[3]http://mitsloan.mit.edu

dividends on the warrant price [156]. He is well-known for his no-arbitrage argument and continuous-time option pricing model. What Merton emphasized is that the output of the model must be free of arbitrage opportunities, i.e. enforces the Law of One Price [21]. After that, the Black-Scholes-Merton model won the 1997 Alfred Nobel Memorial-Prize in Economics Sciences. However, the prize was received by Merton and Scholes as Fischer Black died in 1995. Robert C. Merton's lecture, "Applications of Option-Pricing Theory Twenty-Five Years Later", was published in 1998 [157].

After that, many other researchers have contributed to the area of option pricing significantly. One of the most important contributions was published by Lenos Trigeorgis in 1996 [219]. Trigeorgis presented a way to practice real options by discussing different evaluation methods in his book. Similarly, Amram and Kulatilaka contributed to this field and presented useful examples in their book [21]. And many others such as Cox et al. [66] and Robert S Pindyck [175]. In the context of web services, we are having a model which is inspired by financial options but it serves the real options field.

5.3 Options Types and Service Selection Decision

Real Options has been applied to various domains, such as manufacturing, Information Technology, etc. A specific option type depends on the nature of the application, problem to be investigated, and the decision to be taken. For example, investment's expansion decision could be formulated as a growth option, when-to-invest decision could be formulated as a defer option, and so forth. This thesis is concerned with some scenarios, which are related to web service selection and composition that has been driven by either a technical or a business objective. In each scenario, we are interested in three decisions (1) keep, (2) substitute or (3) abandon the current service. Each scenario takes into consideration either one or more QoS attribute dimension. We address these scenarios from an option-based perspective. Each scenario is linked to a suitable option type.

We are interested in four option types in this thesis: (1) growth-options and we link it to expand and substitute decision, (2) switch-option and we link it to substitute decisions, (3) defer-option and we link it to wait and substitute decision, and (4) abandon-option and we link it to service abandon decision. We will illustrate the use of these options later on using case study and illustrative examples, which are driven by scenarios of possible web service selection challenges that may lead to technical debt in CB-SOA. These scenarios have some sources of uncertainty, such as payoff, QoS values of the candidate web service and the time-value of the selection decision (uncertainty was explained in details in the previous chapter). We are interested in reducing such uncertainties. We are also interested in valuing flexibility under uncertainty. Next, we explain the four types of options that are used in this thesis.

5.3.1 Growth option

Growth option is a type of real options, which gives the right to expand with strategic importance [161]. Growth options are significant in every infrastructure-based or strategic industry, which has a multiple-product generation or application [219]. In web service selection and composition, we assume that the selection decision may carry growth options in support of a specific QoS (e.g. scalability). In this context, growth options is a real option on an asset (web service) with the view to unlock the architecture value potentials through supporting more users and constrained by the service inherent capacity. Here, the value of the selection decision is derived not only from the expected direct revenues, but also from the future growth opportunities it may unlock. In this context, a selection decision is modeled as a call option to acquire a web service with additional capacity by paying an exercise price (scalability scenario), opening up future growth opportunities. Several research works have discussed the significance of growth options as a source of value, such as [162, 116, 175, 213]. For example, Myers [162] suggested that options pricing can value investments with strategic options, seeing that growth options open future opportunities. Based on Myers' idea, Kester [117] investigated strategic aspects of growth opportunities. Likewise to Kester, Taudes [213] evaluated software growth options with the purpose of valuing software benefits. In particular, he proposed an option-based model for evaluating software growth options, which are generated by Information Systems functions in a given software system.

5.3.2 Option-to-Switch

Option-to-switch gives the right to react upon the change market conditions and change input or output accordingly [219]. In web services context, you have the right to switch between services if the current service does not work as expected (as agreed in the SLA). Here, the service provider provides flexibility to the decision maker to switch among services in case if they are not happy with the current web service and if they are willing to pay an extra charge for flexibility. Accordingly, the decision maker has the right without obligation to switch a service. For example, if the decision maker is looking for improving the utility, this can be done through switching web services with the view of improving QoS and reducing technical debt, if any.

5.3.3 Option-to-defer

Option-to-defer is defined as holding a lease on a valuable asset and wait x time units (e.g. months, years, etc.) [219]. It is also called option-to-wait [21] and learning-option [189]. Option-to-defer is valuable in sectors with high uncertainty and long investment horizon, such as real estate developments. Option to defer can assess the time-value of waiting and it has many applications such as waiting to expand, waiting to enter a new market, etc. In web service selection context, option-to-defer gives the right to decision makers (e.g. architects) to defer the selection decision and wait for sometime before exercising the option (e.g. wait-and-watch the cloud marketplace). For example, if we consider option-to-defer, we have the right to delay the investment decision till we get more data from cloud marketplace. This data can be related to the likely future potentials of the candidate web service, and accordingly we will be able to estimate the potential of the selection decision. We can also defer the selection decision until specific uncertainty is resolved (e.g. demand uncertainty) seeing that deferral can reduce uncertainty. In this context, we view differing a selection decision as buying time for investment in web services, hoping that market information will change and new information will be added, which aid in informing whether if the service in question is suitable or not. The question is whether or not to invest at time t? Option-to-defer is appropriate when the value of waiting exceeds the value of immediate exercise (e.g. immediate substitution of a web service). Option-to-defer is important when there is a lack of information or when there is no enough information either about the usage conditions or the candidate web service itself (e.g. service reputation information). Here, it is better to wait. In this context, the time-value is important in the context of "waiting" in order to have more available information and accordingly make a decision. We may kill the option if we exercise it immediately. In later chapters, we will link this type of options to time-value of the selection decision (e.g. time-value of waiting). Furthermore, we predict the time-value of waiting to defer the decision of investing in web services substitution. We will also answer the question of when it is beneficial, in terms of value and technical debt, to exercise the option. We will also track the time-value of waiting and make a comparison in different extremes, such as peak time and off-peak time. We consider "timing" as a significant driver in web service selection decision making process. We are optimizing the time-point of the selection decision and making the selection decision flexible in terms of time.

5.3.4 Option-to-abandon

Option-to-abandon gives the right to the management to abandon the current operations when market conditions are declined severely [219]. Myers and Majd investigated the significance of abandonment value in a given project's life [163]. In a web service context, option-to-abandon gives the right to a decision maker to abandon the current web service when the value is unable to cover for the cost and as a result signaling unmanageable technical debt. This type of options is important in many strategic industries such as airlines and railroads. For example, some destinations of Easyjet airlines were stopped due to high airport's tax (e.g. Amman-Jordan destination was stopped in May 2014). In option terms, we say that the value, which Easyjet used to acquire from Amman destination, is no more covering the cost and as a result signaling unmanageable technical debt, and should be abandon.

5.4 Option Valuation Models

There are many valuation models that are used for option pricing. In these models, options are valued using different techniques. Each technique has different assumptions and tools for the purpose of: (1) capturing and modeling uncertainty, (2) estimating flexibility, and (3) tracking the value of the asset over time. These techniques differ in the way they choose values and represent uncertainties. Each option valuation model is useful at a particular area and has its advantages and disadvantages. The most well known ones are the Black-Scholes-Merton [41] and [156] for valuing options in continuous time, and the Binomial option pricing for valuing options in discrete time by Cox et al. [66]. These two models will be discussed in details in subsequent subsections.

The third valuation approach is related to simulation analysis, such as Monte Carlo simulation models, which are also used for option valuation [103]. However, this model is based on pseudo-random numbers that are generated by computers to iteratively create many possible future paths of a given stock price. In this approach, the simulation analysis may generate thousands of possible paths. These paths are difficult to handle, as each option valuation is performed on one path at a time. In addition, these random generated numbers do not incorporate real market data. Recently, many other valuation models emerged in different fields, such as Net Option Valuation (NOV) by Baldwin and Clark [30], which was based on the notion that modularity adds value in terms of design options it creates. These models are used for the purpose of valuing a specific option. More recently, Sullivan and other researchers in the field of "Design Modularity" extended the NOV model by introducing environment parameters (environmental variables) as main contributors that affect the value of a software design [207] and [56]. These valuation approaches are out of the scope of this thesis.

5.4.1 Black-Scholes-Merton Model

From a financial market, Black-Scholes-Merton is a mathematical model that is used for valuing an option in terms of stock prices. The model was based on the notion that almost all corporate liabilities can be viewed as combinations of options [41]. The model deals with options that have (1) a particular expiration date (European-style Option) and (2) a single source of uncertainty. In this model, the option value is computed from a value of a portfolio of traded securities that already exists in the market, which has the same payoff as the option and has similar fluctuations in its value over time. This process is called tracking portfolio [21] (it is also called replicating portfolio and twin security). By forcing the Law of One Price and "no arbitrage" condition, they ensure that the value of an option equals to the value of the portfolio as the stock price evolves, i.e. two assets with exactly the same future payoff have the same current value [21]. The following inputs to this model are needed when valuing an option: (1) current value of the underlying asset, (2) time to expiration (time to the decision date), (3) exercise price (strike price), (4) risk-free interest rate, and (5) volatility of the underlying asset, which is estimated out of the fluctuation over a period of time.

Black-Scholes equation is well known by its speed as it takes short time for option calculation. It is also well known by its simplicity when it characterizes the relationship between risk and value [21]. However, it has some disadvantages. First, it used when there is only a single source of uncertainty and a single expiration date. That means that Black-Scholes is a closed formula that does not scale, especially when we encounter staged decisions with multiple decision points, such as service selection in CB-SOA. Another significant limitation of this model is that it is suitable only for European-type options and is not suitable for American-type options. Some approaches made some adjustment to Black-Scholes equation for the purpose of approximating the value of an American-type option, such as Pseudo-American method [219]. However, these approaches are used for valuing call options and they work under certain limits and conditions. We argue that Black-Scholes is an inflexible model since its output is a single number that summarize all expected behavioral information about an option. Additionally, another drawback is that Black-Scholes model cannot be used when options have no underlying assets in the market to replicate. Finally, market volatility is assumed to be constant in Black-Scholes model.

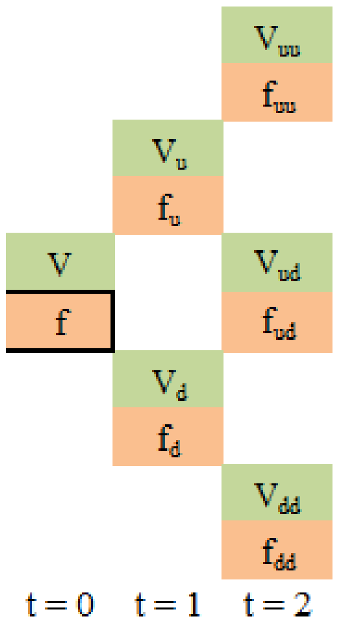
5.4.2 Binomial Tree Model

In 1976, Cox and Ross [65] developed a binomial approach that facilitates options valuation in discrete time. After that in 1979, Mark Rubinstein contributed to Cox and Ross approach as he realized that everyone has different risk preferences that are independent of the option values [66]. Accordingly, the same valuation will be applied whether it is risk-neutral or not. This approach is different than other valuation techniques in how to choose relevant values and represents uncertainty. Binomial Tree options valuation was introduced in different fields such as refactoring [151], prototyping [208], extreme programming [77] and Software Reuse [80]. Section 5.6 provides a review on the available research works in the field of real options valuation.

The binomial tree represents all possible paths that the value of an asset can take during the option life (Figure 5.1). The tree is called a binomial tree as the value of the asset will either move up or down at each decision point. The ups (favorable outcome of the uncertain variable) and downs (unfavorable outcome of the uncertain variable) movements of the value are generally determined by the probability. Each node of the tree shows the probability that asset's value may move up or down and accordingly influencing the value of options on the asset. The different levels of the binomial tree correspond to discrete time points in the future and the probabilities indicates how likely it is that certain jumps in the tree will occur. Each branch in the tree can be viewed as a decision possibility or flexibility.

Binomial Tree construction begins with dividing the expiration date (T) by the number of time steps (n) of duration Δt, where $\Delta t = \frac{T}{n}$. The model is then constructed to identify the likely future values at the end of each time period, and the movements (ups and downs) from one time period to another subsequently. One of the Binomial Tree properties is that any up move followed by a down move has exactly the same effect on value as a down move followed by an up move.

Figure 5.1 General Binomial Tree.



When valuing an option using the binomial model, two main phases are constructed; asset value calculations and options value derivation. First, the values tree is constructed starting from present to expiration date. Next, options are calculated at each point in the binomial tree by going back from expiration date to the present, i.e. options are valued inductively by stepping back (folding back) through the binomial tree from the expiration date to the present. Option values in the binomial tree are derived depending on the value tree development. Each option value at each step of the binomial tree is used as an input for deriving the option value of the next step. Asset values and option values are derived using the binomial tree coefficients (binomial tree input parameters) that will be discussed shortly.

Based on [66, 21, 103] general Binomial notation, which was adapted by [172], we can generalize the web service selection problem by viewing the web service as an asset which has a value V and an option on this asset which has a price f (Figure 5.1). We assume that the option lasts for time T and during this time (option life) the asset value can either go up to V_u, where $u > 1$, or down to V_d, where $d < 1$. When asset price moves upward to V_u, we assume that the expected payoff is f_u and when asset price moves down to V_d, we assume that the expected payoff is f_d , as seen in Figure 5.1. Expected payoffs are calculated based on ups and downs movements. In our interpretations, u is said to be the increase multiplicative percentage of the web service value and d is the decrease multiplicative percentage of the web service value. Our calculations are based on the Binomial Tree Valuation presented by [219] and [172]. Complete steps of the Binomial Tree in the context of web service selection is explained in details in chapter 6, section 6.1.4.1.

5.4.2.1 Binomial Tree Coefficients

Binomial tree coefficients can be calculated based on different methods. They can be (1) elicited from stakeholders, or (2) estimated from historical data or based on a given underlying asset valuation. This thesis is concerned with the first method of acquiring binomial tree coefficients, as we assume that the binomial tree coefficients are elicited from decision makers (e.g. stakeholders) based on their experiences. In CB-SOA context, the interpretations of these coefficients depend on the context and scenario of the selection decision. Table 5.1 summarizes the binomial tree coefficients and their definitions. It also presents the coefficients mapping in terms of real options and cloud-based web services.

On the other hand, if we are estimating these coefficients, some formulae are used. Based on [21], the coefficients values of u and d are calculated using equations 5.1 and 5.2 respectively, where σ is the volatility, which is the standard deviation of expected return [21]. Volatility is out of scope of this thesis. After that, the probability coefficient is calculated using equation 5.3. As mentioned before, these equations are not used in

Table 5.1: Binomial Tree Coefficients mapping

	Real Options	Web services
Δt	Length of time periods in months, years, etc.	Length of time periods in months, years, etc ("staging-view"). This is used to calculate the time period between each decision point in the tree, where each decision point correspond to an identified hypothetical time that is used to monitor the service behavior during the life time of the option (T).
p	Favorable probability coefficient (up movement probability)	Favorable probability coefficient (up movement probability). It implies the likelihood of the next up movement of the candidate service value in the tree.
1- p	Unfavorable probability coefficient (down movement probability)	Unfavorable probability coefficient (down movement probability). It implies the likelihood of the next down movement of the candidate service value in the tree.
V_{t0}	Initial asset value	Initial web service's value at t_0 (explained in details in chapter 6).
u	Percentages increase in the asset value.	Percentages increase in the likely web service value in supporting improvement in one or more QoS, which in turn imparts the utility of the architecture.
d	Percentages decrease in the asset value.	Percentages decrease in the likely web service value, which in turn imparts the utility of the architecture.

CloudMTD model, as we assume that the binomial tree coefficients are elicited from decision makers based on their experiences. However, the inputs to CloudMTD can be change and these equations can be considered if needed.

$$u = \mathrm{e}^{\sigma} \tag{5.1}$$

$$d = \mathrm{e}^{-\sigma} = \frac{1}{u} \tag{5.2}$$

$$p = \frac{(1+r) - d}{u - d} \tag{5.3}$$

In CB-SOA context, we say that u and d are the percentages increase and decrease, respectively, in service's value, which imparts the utility of the architecture. Here we say that the interpretation of u and d depend on the context of the scenario in question. For example, if we consider growth options in a business expansion decision that is associated with scalability scenarios, the u will imply the percentage increase of service's value in supporting scalability. In this context, the selection decision is driven by requirements to scale up and support more users. In this sense, the selection decision carries growth options in support of more users. Here, we assume that the service value will be improved by £1 in supporting an additional user. Accordingly, more fulfilled users' requests will improve the utility value of the architecture through supporting more users. On the other hand, if we consider availability scenarios. Here we say that u coefficient is the percentage increase of service's value in supporting availability. For example, when (u = 1.4), 100% availability is likely to increase the service value by an average of 40% per month. Here, we assume that the service value will be improved by £1 in supporting more availability. In this context, availability goals take into consideration the downtime of the service (unavailability), which affects the business value in supporting availability scenarios. For more details on each scenario refer to illustrative examples in section 6.2.

5.4.3 Discussion

We have adopted the binomial model for the reason that it has many advantages over the other valuation models. Even though the other models are used for the same purpose, option valuation, the binomial model has a wider range of applications and is easier to use [99]. The binomial model has a primary advantage over Black-Scholes model, for instance, which is the ability of pricing American options precisely [21]. This is attributed to the ability of the Binomial model to exercise an option at any point (step) in the binomial tree during option life. In addition, unlike Black-Scholes model, binomial model provides discrete approximations[219]. Accordingly, the binomial model provides the decision maker with discrete space, which helps in quantifying the decision at each point in the tree. In this sense, the Binomial model is open in terms of flexibility. In addition, the binomial model handles large range of complex real options, generates good visual images and involves risk-neutral valuation [21]. The main disadvantage of the binomial model is that it is considered to be slow when calculating thousands of prices in a few seconds. We argue that the binomial model is more accurate than other valuation models, as it incorporates real data (e.g. market-based information). For example, as mentioned in previous section, Monte Carlo simulation models are based on pseudo-random number generator.

We view the binomial tree as a suitable model to be used in CB-SOA selection decisions as CB-SOA environments exhibit a lot of uncertainties (e.g. demand changes). In addition, CB-SOA selection decisions are not made at a single predetermined deadline and need continuous evaluation. Accordingly, the binomial approach is a good fit as it models an option by splitting its time to expiration into number of intervals with different flexible decision points. Accordingly, by using the binomial tree we provide the decision maker by a "staging-view" along the life of the option. What is meant by the "staging-view" is that we can analyze and interpret the results during different time periods (intervals of time), for example month by month or year by year that depends on needs and contracts length, which is explained in SLA. Moreover, because of the dynamic nature of the selection problem there is a need for such a flexible approach with flexible decision points. The dynamic nature of the selection problem is driven by different factors which are related to uncertainty and its dimensions. For example, the continuous fluctuation in QoS, change in needs and requirements, etc. (detailed explanation of uncertainty and its dimensions were discussed in chapter 4). In this sense, the binomial tree fits the dynamic decision making in CB-SOA by allowing a continuous step-by-step assessment and judgment.

5.5 The Approach

We assume that the goal of a decision maker (e.g. architect) is to guide the selection and composition decisions in CB-SOA, and manage any likely technical debt. We view the web service selection and composition as value-seeking and value-maximizing activities. We "stage" the valuation of the candidate web service in the view of managing the technical debt using binomial options model.

5.5.1 Real Options Analogy

In this section, we sketch our interpretation of the related web service selection and composition principles in real options terms. We will be spot-lighting on some significant aspects and dimensions, for example, the case of optimal-time selection decision.

Table 5.2: Analogy of Financial Options, Real Options and Web services CloudMTD

Financial Options	Real Options	Web services CloudMTD
Stock price	Value of the expected cash flows of a given project	Web service's value in supporting a specific QoS
Exercise price/Strike price	Investment cost	Cost of the option, which depends on the scenario. For example, cost of switching, cost of waiting to switch, etc.
Expiration Time	Time until opportunity disappears	Expiry date of the service contract, which is viewed as the expiry date of the option

5.5.2 Web Service Selection: Options Perspective

In order to understand web service selection and composition using an economic approach, we need a valuation technique that is capable of: (1) strategic and long-term valuation, (2) accounting for flexibility, and (3) quantifying the value of options, which is created by flexibility, for the purpose of making the value of the selection decision tangible. Accordingly, we posit that real options theory satisfies these requirements. We present an option-based approach with the aim of understanding the economic potential of CB-SOA selection decisions. In this thesis, real options are used with the purpose of modeling situations related to selection decisions in CB-SOA, where options exist when investing in an "architectural asset", i.e. a web service. In the context of web service selection, a real option is the right but not the obligation to make a selection decision in future ($t = T$) by taking into account further opportunities related to web services, such as option to expand, option to defer, option to abandon, and so on.

The economist, Myers, noted that "Part of the value of a firm is accounted for by the present value of options to make further investments on possibly favorable terms [161]." Similarly, investment opportunities of web service selection can be viewed as a set of options. We would like to highlight the importance of determining the potential value of new investments in web services. The expected value V of an inflexible web service selection decision is: $V = B - C$, where B is the expected benefits and C is the expected cost. While the expected value V of a flexible web service selection decision is its benefits B in addition to the value of the future option O to accommodate uncertainties (e.g. future change), minus its cost: $V = B + O - C$. The best case scenario of a flexible web service selection decision is when the option is worth more than its cost. Note that the difficulty is in evaluating the value of the option as the cost is generally tangible and known, while the value is not and it needs time and effort to be grasped. For example, reconfiguring the architecture by substituting one of its constituent web services incurs upfront cost; however the value is elusive and may take long-time to be grasped. Web service selection decision should be able to weigh the cost against the uncertain value. Traditional software engineering economics approaches were limited in terms of flexibility, i.e. do not account for the value of flexibility. Conversely, real options theory builds an analogy to financial options theory for valuing flexibility and hence making it tangible. Indeed, if benefits of web service selection decision were certain, the decision would be much easier by a simple comparison process. Taking a decision is more complex when the value and payoff depend on uncertain future conditions with different possible outcomes. Furthermore, the ability to reconfigure a given CB-SOA may create future options by decreasing costs and improving efficiency, for example. However, the reconfiguration process shall take place only if the expected benefits, through cost saving, for example, exceed the required expenditures (e.g. upfront cost). In this sense, the decision maker (e.g. an architect) must be able to reason about when and under what conditions investments in web services add value.

All in all, evaluating web service selection decision under uncertainty makes real options an ultimate choice as a valuation technique, since other valuation techniques has limitations in managing value of flexibility under uncertainty. Accordingly, integrating service selection with options thinking can provide a powerful tool for: (1) better reflection on the value of the candidate web service and hence the utility of the architecture, (2) situations leading to web service substitution, (3) time-value of the selection decision.

5.5.3 Technical Debt Management: Binomial Tree Visualization

Kruchten et al. [122] presented the technical debt landscape and its elements. They said that the most challenging part of the process is deciding about future changes. They argued that the decision is about making a balance between cost and value. In this sense, they suggested that technical debt landscape can be unified using an economic or financial model. Some of these models were explored previously in the literature section in chapter 3. In CB-SOA, we manage technical debt on service-level by substituting constituent services. We relate the selection/substitution decision to technical debt drivers, those were explained previously in chapter 3. For managing technical debt, the substitution decision can be one of the following: (1) selecting another web service, (2) keeping the current web service, or (3) abandon the current web service. Technical debt clearing out stages is tracked using the binomial model. We also use the binomial tree to visualize the technical debt lifecycle that was discussed in chapter 3: (1) when it starts. (2) when interest are incurred, if any. (3) Payback; when the technical debt is expected to be cleared out and/or when it is likely to generate future options if cleared. All in all, integrating the technical debt analysis with options thinking can provide a powerful tool for: (1) better reflection on the value of the debt under uncertainty, (2) situations leading to clearing out the debt, (3) lifetime cycle of the likely debt.

5.6 Real Options in Software Engineering Practices: A review

Valuing software engineering related investments using economics theory is not new. Barry Boehm pioneered the use of economics in software engineering in early 80s (e.g. Boehm1981 [45]). Subsequently, Real Options theory has been investigated in many fields, such as software design [208], software architecture [172], systems design and engineering [30], software refactoring [29, 151], architectural stability [26], COTS-centric development [78], Extreme Programming [77], security of IT systems [2], Agile requirements engineering [179], and many others. In this section, we provide a representative sample of the available research works in the field of software and system engineering, which are related to software engineering practices.

Baldwin and Clark [30] established the use of real options in systems design and engineering. They pointed out that modularity creates flexibility in system design as it influences the evolution of the system designs. In particular, Baldwin and Clark's theory was derived from the notion that system modularity creates value in terms of real options. They calculated the option value of design modularity using the Design Structure Matrix (DSM) [30]. Likewise, Sullivan et al. [207] established the use of real options theory in software engineering. They said that real options techniques offers insights about modularity, projects structures, and delaying of decisions in software design [209, 208]. They presented an option-based analysis of the spiral-model for software development. They reported that the spiral-model provides flexibility in system structure. The aim of each phase is to reduce uncertainty and to take a decision about whether to invest in the next phases derived from information from previous phases. Moreover, within each phase it points out alternatives and creates an option to choose the most appropriate one. In real options perspective, this is called options to defer a decision of investment until a specific optimal time. In addition, Erdogmus et al. [78] presented the valuation of strategic flexibility, using real options, in software development (including COTS). They apply two quantitative valuation techniques: (1) Net Present Value (NPV) and (2) real options, for assessing software development projects, which are based on COTS. The objective is to investigate the economic incentive of choosing COTS-based strategies in a project. The result shows that real options theory is better than NPV, as NPV neglects the value of the flexibility in COTS-centric projects making it appear less attractive. After that, Cost Benefit Analysis Method (CBAM) was improved by including the use of real options theory in reasoning about the value of delaying a decision of an investment related to architectural strategy [114].

Real options theory was also used in the field of software refactoring. For example, Bahsoon and Emmerich [27] used the Black-Scholes model for software refactoring valuation, based on the assumption that a decision has a fixed expiration date to be made for a given application. In [28], they have used mining technique based on a financial analogy of real options for analyzing architecture scalability under uncertainty. Moreover, Mavridis et al. [151] examined the selection of an optimum refactoring strategy based on real options analysis. Their objective was to select a strategy that maximizes the endurance of a given architecture with respect to future imposed changes. They mentioned that refactoring carries economic value in terms of real options, which can be viewed as the right, but not the obligation, to select a refactoring strategy in a given period of time, where the refactoring strategy (to be selected) is viewed as a real asset [151]. They answered the question of when and under which conditions a given refactoring should take place. They addressed the following uncertainties: (1) value associated with candidate refactoring, (2) the expected added value gained from a given refactoring, and (3) some conditions that affect the value of the extended system (e.g. developers' ability and number of features to be added).

Furthermore, Real Option analysis was applied extensively as a decision making tool for IT investments and SOA projects, and then the application of Real Option analysis started to revolve around the notion of valuing uncertainties in software engineering practices in general. For instance, the used of real option theory in Information Technology (IT) investments decisions was driven by different factors that are related to IT investments characteristics, such as the long payback periods of IT investments, the associated uncertainties, and the continuous change of business conditions. Therefore, many research works have applied real option models to support IT investments decisions for assessing the value of IT assets (e.g. IT applications).

An early work of applying real options theory to IT investments decisions was done by Alfred Taudes [213]. This approach was a foundation of software platforms valuation using option theory. In particular, he examined some techniques to evaluate sequential exchange options for approximating software's growth option's value. However, in this work Alfred Taudes did not mention the difference between Information Systems functions and value generating applications. In addition, his early work was limited to general conclusions and guidelines without any real world cases. Later on, Taudes et al. [214] investigated the value of implementation opportunities based on options theory. They presented the practical advantages of using options analysis in software platform decisions valuation. They demonstrated these advantages by presenting a real world case based on Black-Scholes-Model. In particular, they viewed a software platform as a bundle of functions, which forms the building blocks of a given application whose value changes over time. Accordingly, the value of a given software platform is determined by the options it creates by implementing these applications.

Michel Benaroch, Robert J. Kauffman and their colleagues have used different types of real option theory in different problems related to IT investments. They have been employing real option theory for evaluating IT investments and managing the associated risk in different IT projects contexts (IT portfolio). In 1999, Benaroch and Kauffman [37] started by providing a formal theoretical background on the validity of using the Black-

Scholes model, as a range of capital budgeting techniques, for assessing IT investments. They presented that both Black-Scholes and the binomial models can be used for evaluating different IT investment situations that can be analogous to capital budgeting ones. Specifically, they used Black-Scholes Model and Blacks approximation for calculating the optimal exercise time in IT projects. They have posited that deferring an investment can reduce risk. This is due to the flexibility that is created by deferring an investment commitment, which is useful when it enables learning about the nature of uncertain payoffs. In [33], they have also presented the flexibility to stage an investment. This is valuable when there are different types of risk, such as complexity risk, user involvement, architectural compliance, etc. Then, Kauffman and Li [112] developed a continuous-time stochastic model in a real options theory framework. Their model aids in determining the optimal time strategy for a specific managerial adoption. Similarly, Bardhan et al. [32] presented a methodology for valuing a portfolio of IT investments based on real options. Particularly, they looked at applications of nested real options for the purpose of valuing and prioritizing a portfolio of IT investments, taking into consideration interdependencies among them.

After that, Benaroch et al. [39, 36] presented IT investment decisions in terms of risk management characteristics that are associated with such decisions. Particularly, Benaroch et al. [39] proposed an option-based risk management framework (OBRiM). In [38], they have empirically tested the OBRiM framework. The framework is based on real options theory for the purpose of: (1) quantifying the monetary consequences of risk, (2) measuring flexibility in terms of real option value, and (3) linking risk and options to investment value. In [36], they have investigated another dimension of IT investments' problems that is related to complex sequential investments, which involves different types of risks for a given firm. The OBRiM framework can be used by decision maker in order to find a suitable real option that can be used in an IT investment. Their objective is to manage and control risk, and to maximize the investment value. The framework was evaluated based on interviewing senior managers from multiple firms. Similarly, Kauffman and Sougstad [113] investigated contracts of a portfolio of IT services for the purpose of evaluating the tradeoffs between contract profitability and service-level risk.

More recently, Benaroch and his colleagues looked at other dimensions that are related to IT investments: (1) risk management practices optimization [35], and (2) service-level agreement contract flexibility [34]. Their objective was again to control risk and maximize project (investment) value. Benaroch and Goldstein [35] modeled risk management decisions using real options theory. They presented an economic-based optimization approach for the purpose of optimizing risk management practices. In particular, they addressed the problem of managing the risk of system development following an economical optimization point of view. Benaroch et al. [34] examined the flexibility that is associated with IT services contracts using real option theory. They looked at situations where IT service providers offer customers by the option to abandon a specific contract. In addition to determining the optimal exercise time, they used real options for evaluating the service-level agreement contract flexibility. In their work, they concluded that the use of binomial model for valuing multi-stage IT investments is more accurate, seeing that a simple Black-Scholes model provides incorrect results.

Furthermore, Tansey and Stroulia [212] proposed a methodological approach for estimating the value of an SOA project. In particular, they proposed a procedure in order to estimate business value's potential in evolving service-oriented applications. They used COCOMO II for estimating the development cost of services in terms of programmer's effort (e.g. creating or modifying a service in SOA). They proposed used of option theory for valuing flexibility and reusability of a service composition. The object was to maximize return-on-investment (ROI) in a given SOA project. Moreover, Su et al. [206] introduced an option-based methodology for evaluating the decision of choosing among different service's transformation alternatives for the purpose of implementing shared services. The methodology aids the decision maker in a given firm to estimate the value of each transformation alternative. They have viewed the decision of service's transformation as a European call option. The value of managerial flexibility was calculated using

Black-Scholes model. Similarly, Lin and Kang [136] proposed an option-based mathematical decision-making model to evaluate the decision of whether to invest or not to invest in a free internet service. They proposed that a given firm has the option to invest (or not to invest) in the free software market. Similarly, Sanchez and Milanesi [186] presented an option-based framework for estimating the volatility of IT investments taking into consideration the impact on the project value. Their suggested method can be used by IT managers for making decisions related to software development investments. It can be also used for understanding the interactions among software processes, market environment, financial issues and options value. On the other hand, Lukas Auer [24] analyzed the applicability of real options theory in IT investment planning. Particularly, he examined a management approach for supporting decision making in SOA investments. Auer posited that SOA investments should be associated with suitable evaluation methods for capturing flexibility and uncertainty. He also identified the limitations associated with commonly used SOA investment decision-making techniques. However, the presented work was a simple option-based decision framework that presents guidance for decision making in SOA investment.

Furthermore, Gaynor et al. [90] proposed an option-based model for examining the problem of designing standards for implementing network-based services. The model was based on the assumption that market demand of network-based services has a degree of uncertainty. They quantified the economic value of standards that is attributed to the flexibility of choice of a management structure. They conducted experiments for determining the best match of services based on market conditions in the context of popularity of features. The objective was to maximize the overall gain when designing standards for implementing network-based services in markets with high uncertainty. Likewise, Gayno and Bradner [89] proposed an option-based framework for evaluating architectural choices of network infrastructures and linking value of flexibility to market uncertainty. In particular, they compared the value of distributed architectures against the benefits of centralized control based on cost and value differentials. They illustrated the potential of real options in quantifying the economic value of networks, protocols, and service architectures (network services/applications). They defined experiments as attempts of finding customers' needs and accordingly creating options by permitting users to make a choice. In this context, they have investigated uncertainty that they attributed to users preferences and call it market uncertainty. They have linked the economic value of a given experiment to market uncertainty as follows: "uncertainty is the inability of the experimenter to predict the value of the experiment, i.e. predict what the customer wants [89]." They examined five factors: (1) capability of experimenting, (2) experiment's level of difficulty, (3) market uncertainty, (4) advantages related to business and technical aspects, and (5) architecture's evolution over time. In addition, they linked network infrastructures to market uncertainties. The evaluation was conducted based on voice and email services case studies. Distributed architectures were preferred by users in case of high market uncertainty. On contrary, centralized architectures were preferred by users when market uncertainty is low. They concluded that distributed end-to-end applications and services are required for maximizing value, which is attributed to satisfying users' needs in markets full of uncertainties. More recent, Yoo et al. [233] proposed an approach for evaluating a given firm's decision of choosing between two different types of Business to Business (B2B) marketplaces; public or private. Their approach was based on a game-theoretic analysis and a real options theory, taking into consideration future managerial flexibility, uncertainty and network externalities. They analyzed the impact of options thinking on decision making in IT investments.

In addition, some research works used real options theory for the purpose of justifying and formalizing agile software development practices. For example, Racheva and Daneva [178] investigated the use of real options theory in agile software developments, seeing that such investment decisions can be discussed in terms of options. Particularly, they investigated the problem of making decision in inter-iteration time for a given client. Likewise, Racheva et al. [180] proposed a formal decision making approach based on real options analysis and some other measurement techniques. Specifically, their approach is used under uncertainty in order to assist clients in choosing among alternative sets of requirements. When decisions are formulated as options, decision makers look at many alternatives. Accordingly, they can compare them and then make a decision by taking into consideration business goals and current software functionalities. They applied real options in decision-making from two perspectives related to agile aspects. First, from client's perspective, real option is used for requirements prioritization at the beginning of each iteration and accordingly optimizing business value. Second, from developers' perspective, real option is used in order to support the implementation of the prioritization process. In this context, they view agile developments as a sequence of decisions and treat them as a set of options before or after each iteration. In [179], Racheva and Daneva evaluated the work by conducting a case study based on eight agile development organizations. They have also performed interviews about the decision-making process with different practitioners. They have described different types of options in terms of agile development, such as option to abandon and option to defer a client's decision. In the same way, Klein et al. [118] suggested an option-based method for analyzing decisions related to requirements of common platforms for systems of systems. They suggested that existing economic and probabilistic models can be used for modeling uncertainty in evolution requirements. Moreover, Erdogmus and Favaro [77] employed real options for valuing the flexibility in Extreme Programming (XP). In their context, XP is a lightweight process and is suitable to respond to future changes and future opportunities. They use real options to reason about the You Aren't Going to Need It principle (YAGNI). The YAGNI principle emphasizes the value of postponing an investment decision under uncertainty (option to delay). In this sense, uncertain features implementation is delayed till uncertainty is resolved.

Subsequently, Real Options theory was also applied to other fields such as, product line, mobile networks and many others. For example, Chaiworawitgul and Sutivong [59] illustrated how to apply real options analysis to hardware design decision for valuing the expected payoff of the technology shift from Object-Oriented to Aspect-Oriented concepts.

After that, Gustavsson and Axelsson [95] examined the use of real options in quantifying the value of a flexible design decision of a given automotive systems - vehicle product lines. They presented an option-based evaluation method for the purpose of analyzing the cost of designing for flexibility for handling a product line growth in future, depending on the future value of functionality. They applied real options analysis using a case study from an automotive industry for evaluation purposes. Their objective was to improve the usability and to help practitioners (e.g. developers).

5.7 Related Work

Our work is related to different correlated fields, which are (1) web service selection and composition, (2) managing technical debt, and (3) real options. In chapter 2, we presented a representative sample of works that have been done in the field of service selection and composition. In chapter 3, we presented a representative sample of works that have been done in the field of technical debt and technical debt management. In the previous section, we presented a representative sample of works that have been done in the field of real options. In this section, we present the related work, which includes the approaches that have linked service selection and composition, technical debt, and options.

On the cloud side of option theory, real options have been used in businesses migrating to cloud [232] and in pricing cloud compute commodities [195]. Mainly, Sharma et al. [195] proposed an approach for pricing cloud resources (cloud compute commodities C^3) using financial option theory by assuming the cloud resources are underlying assets. Their model was used for capturing the value of cloud compute commodities and providing users with QoS guarantees. In addition, Yam et al. [232] have addressed the decisions of businesses migrating to the cloud using options. In particular, they have looked at the decisions, which driven by security requirements. Specifically, they have answered: "The question that the company is faced with is should they switch from their internal IT to Cloud?", whereas our approach looks at the options in improving the utility of cloud-based architectures through switching web services with the view of improving QoS, the value of the structure and reducing technical debt, with respect to flexibility and time-value of the switching decision under uncertainty. Up to our knowledge, we are the first to explore the link between Real Options and technical debt for the case of web service selection in the cloud marketplace.

On the other hand, the link between technical debt and options has been explored in [48, 49, 192]. Seaman et al. [192] discussed four decision approaches to deal with technical debt: Cost-Benefit Analysis, Analytic Hierarchical Process (AHP), Portfolio Management Model and Options. They have looked at investment decisions at code and design levels. For example, they viewed the value of investing in refactoring as a form of options, where purchasing the option facilitates the change to the software in the future, but without immediate profits. Likewise, Brown et al. [48] proposed that some technical debt appears to create options to invest without obligations. In addition, [49] proposed that the architectural investment decisions can be optimized by analyzing uncertainty and tradeoffs between incurred cost and anticipated value based on Real options and technical debt. However, the linkage between technical debt and Real Options for cloud-based web service selection has not been previously explored.

The link between Real options and web service selection has been explored in a published bachelor thesis from the University of Zurich [182]. The published bachelor thesis investigated the applicability of Real Option Analysis on some projects in the area of Service Oriented Computing (SOC). Their approach examined the SOC projects on the infrastructure level. They have investigated growth options using an illustrative example for evaluation.

Heinrich et al. [98] investigated the question of whether to develop a web service internally or buy it from different provider externally (outsourcing and make-or buy decision). They have improved the traditional make-or-buy web services approaches by introducing a make-and-sell or buy approach using call options. They added that a specific company has further opportunities by considering sell options as follows: “If a company decides to internally develop a web service (make decision), it has additionally the right (but not the obligation) to sell it on the electronic web service market [98].” They claimed that they have chosen Black-Scholes-Merton over Binomial Model due to its ease of calculation. They have illustrated their approach by providing an evaluation for a real case example.

Skourletopoulos et al. [199] presented an approach for technical debt prediction and quantification in software development process taking into consideration budget and service capacity constraints.

6 CloudMTD: A Model for Evaluating Selection and composition Decisions in CB-SOA

6.1 CloudMTD Model

In the previous chapter, we presented an approach based on options for addressing web service selection and composition requirements. In this chapter, we present and describe the CloudMTD model and its phases. In the next chapter, we will present the implementation and evaluation of CloudMTD model. CloudMTD model provides insights into the evaluation phase of the selection and composition decisions in CB-SOA based on different concepts such as, uncertainty, flexibility and technical debt (these concepts were discussed in previous chapters). In each phase of the CloudMTD model, the input is either from previous phase or taken from decision makers based on their experiences, such as Binomial Tree coefficients (as described in previous chapters). Figure 6.1 represents the CloudMTD Cube, which describes CloudMTD scope in general. In this thesis, we take into consideration the following dimensions: (1) time, (2) Value, (3) Complexity, and (4) Flexibility. The figure shows that CloudMTD phases are conducted on two levels: Service-Level for service selection, and (2) Architecture-Level for service composition. As mentioned in chapter 4, web services selection and composition should take into consideration the following factors and consequences: (1) Technical Debt (intentional and unintentional), (2) Uncertainty (Future Changes), and (3) Dependency.

Figure 6.1 CloudMTD Cube

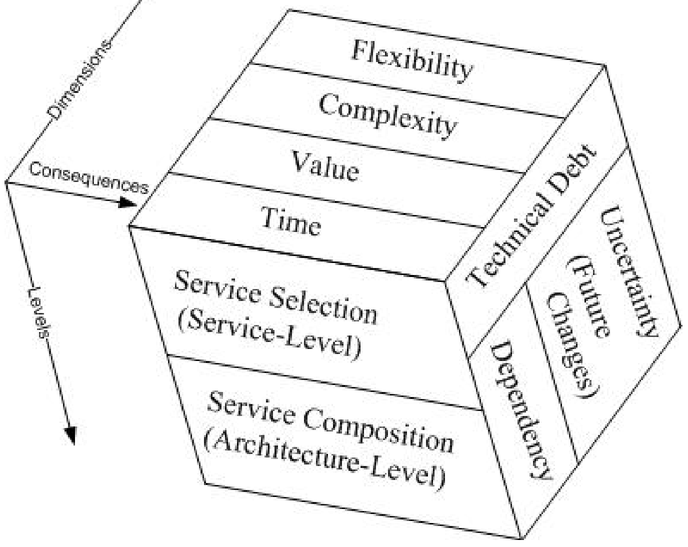
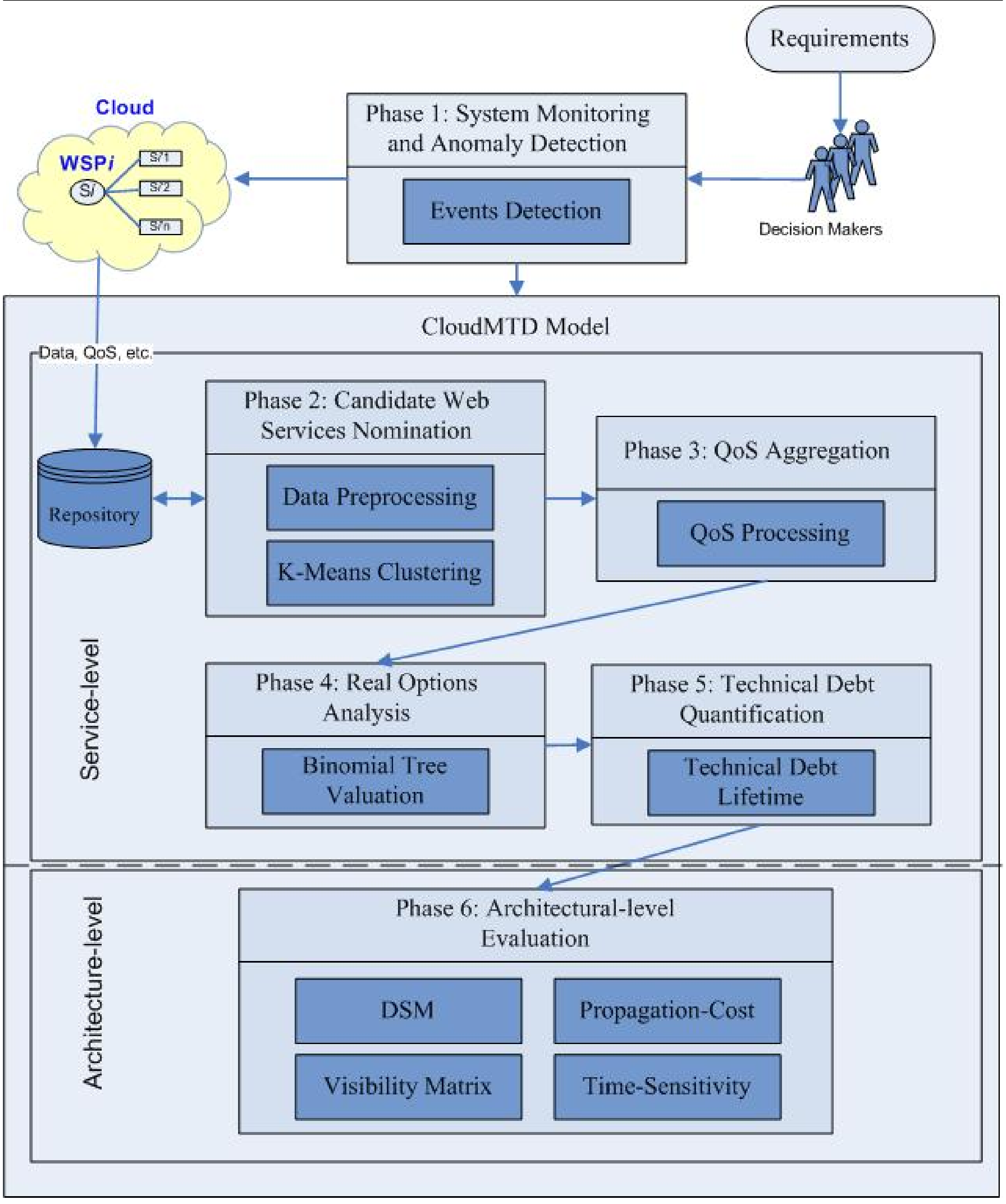

Dimensions
Consequences
Levels
Flexibility
Complexity
Value
Time
Technical Debt
Uncertainty (Future Changes)
Dependency
Service Selection (Service-Level)
Service Composition (Architecture-Level)

Figure 6.2 illustrates the CloudMTD model and its phases, which are:

1. **Phase (1): System Monitoring and Anomaly Detection.** This is a preliminary phase, where CloudMTD model will be called in case of any event detection, such as end of service contract.
2. **Phase (2): Candidate Web Services Nomination.** In this phase, the cloud marketplace will be visited for fetching candidate services.
   - Data Preprocessing. In this sub-phase, services' data, which was extracted from cloud marketplace, is preprocessed.
   - Data Clustering using K-Means Algorithm. Services are clustered into groups. Each group contains services with similar features.
3. **Phase (3): QoS Aggregation.** Services' QoS attributes are aggregated.
4. **Phase (4): Real Options Analysis.** An option-based analysis takes place using Binomial Tree.
   - Binomial Tree Valuation. Here, a binomial tree will be constructed for each candidate service.
5. **Phase (5): Technical Debt Quantification.** Technical debt is quantified based on option values at each month.
6. **Phase (6): Architectural-level Evaluation.** The structural evaluation is performed based on:
   - Dependency Structure Matrix (DSM)
   - Visibility Matrix
   - Propagation-Cost Metrics
   - Time-Sensitivity

Figure 6.2 CloudMTD Model



6.1.1 Phase (1): System Monitoring and Anomaly Detection

This is a preliminary phase where the system is monitored for the purpose of detecting and analyzing the occurrence of anomalies in a given composition, i.e. stimuli that triggers the need of using CloudMTD model. We assume that the monitoring process takes place on two levels; architecture-level and web service-level. In web service substitution context, we encounter “anomalies” when an event occurs that lead to the deviation of the utility of a given service or composition from what is standard, normal, or expected. These events may affect a specific web service, QoS or the architecture as a whole. Such events are either directly related to web services (e.g. end of contract) or to an observation that is related to web services (e.g. change in QoS, decreasing performance, continuous unavailability, etc.) in a given architecture. In this phase, the identification of events and any other observations, which do not conform to the expected utility, takes place. These events may call for new requirements implementation or a specific web service substitution and accordingly an architecture reconfiguration. These events must be defined and described previously by system experts, in order to be easily recognized and managed. An event in CloudMTD concepts is a description of some anticipated changes in the architecture that call for web service substitution. Such events can increase the understanding of CloudMTD model and its use. Before calling CloudMTD model, this information must be available (e.g. list of events with priorities). Such events trigger the need for calling the CloudMTD model based on critical priorities.

Here, we assume that the decision maker (e.g. an architect) has been encountered by an event, which calls for web service substitution. Events could be driven by either business objectives or technical ones, such as end of web service contract, desire to improve utility of the architecture, change in users' requirements, need to upgrade to a new released web service in cloud marketplace, listen to demand changes, new resources are released, etc. We assume that the event is critical and establishes the need for evaluation. The system monitoring process for events detection can be either off-line or online. In either case, the decision maker needs to evaluate the events and their consequences against the requirements. In addition, the decision maker needs to evaluate the implications of such events on the architecture's utility. The monitoring process is beyond the scope of this thesis. We assume that this phase is done by third party. Some research works have been investigating the monitoring problem. For example, execution monitoring in web services compositions [176, 31], QoS monitoring for checking the compliance of the offered service [215], QoS monitoring mechanism based on clients ratings [109], monitoring SOA applications at runtime [185], and many others.

Input: list of critical events, objectives and requirements.

Process: monitoring the system and detecting critical events which will lead to web service substitution (e.g. service is not satisfying the requirements anymore).

Action: calling CloudMTD model for improving a specific service by substitution. Here, we say that the selection decision is taken by a decision maker to achieve a desired level of QoS, which in turn imparts the utility of the architecture.

Output: ID of the service in question that should be substituted.

6.1.2 Phase (2): Candidate Web Services Nomination

In this phase, the cloud marketplace will be visited for searching for candidate web services. We report on two techniques for nominating candidate web services. The first technique depends on the decision maker (e.g. architect), where we assume that the architect will shortlist candidate web services of comparable functionalities, which may come with different non-functional requirements (QoS), cost and SLA categories. The second technique is based on a clustering algorithm (clustering will be discussed in details shortly). The clustering phase is introduced to improve the scalability of CloudMTD model. In addition, the use of clustering will reduce the search-space in the cloud marketplace and accordingly will reduce searching-time. We use the data clustering technique with the aim of analyzing the structure of a given dataset and finding the common attributes in data and then assign them to suitable clusters. For instance, if a web service of high availability is needed, we can search the cluster which meets the search criteria instead of visiting the whole cloud marketplace. The clustering technique consists of different challenging steps: (1) identifying the number of clusters in a dataset, (2) determining to which cluster each member belongs, and (3) finding the suitable distance function. The reason why we have chosen clustering technique is because it reduces the complexity of the selection and composition model. Furthermore, each service, if selected, may have different implications on the utility of the composition. As we are dealing with a marketplace, it is imperative for the selection decisions to be time-sensitive: a service offered through a market may look attractive now but otherwise in the near/far future. Furthermore, the CB-SOA tends to be more dynamic as they tend to be composed of market-leased services, which can change by time and consequently affect the composition. Therefore, a selection decision shall take into consideration possible changes to the structure overtime (if any) due to changes to constituent services.

6.1.2.1 Data Preprocessing

Before we process and analyze QoS datasets, we have to ensure data integrity in many dimensions, as QoS attributes vary in units, ranges, and magnitude. For that reason, equation 6.1 can be used based on [238], in order to make the data uniform and standardized. Equation 6.1 has two parts, the first one deals with descending values and the second one deals with ascending values. Some of the QoS attributes have decreasing values, which means the lower the value, the higher the quality (e.g. response time). And some other QoS attributes have increasing values, which means the higher the value, the higher the quality (e.g. availability). And accordingly, treating these two values as one will produce incorrect results.

$$QoS_{i,j} = \begin{cases} \frac{max(Q_j)-x}{max(Q_j)-min(Q_j)} & \text{if } QoS_j \text{ is decreasing} \\ \\ \frac{x-min(Q_j)}{max(Q_j)-min(Q_j)} & \text{if } QoS_j \text{ is increasing} \end{cases} \tag{6.1}$$

6.1.2.2 Data Clustering using K-Means Algorithm

K-means clustering algorithm is a famous partitioning algorithm. K-means is a popular algorithm and known for its speed and simplicity [146]. Clustering is the assignment of objects with similar features into groups (clusters). K-means classifies objects, based on some features, into K clusters (K should be positive integer). In web services context, cloud-based web services can be classified into different groups depending on their QoS attributes' similarities, e.g. high availability and low response time. K-means clustering algorithm organizes data points (web services) into similar groups (clusters). And accordingly, data points in one cluster have similar characteristics and data points in different clusters have different characteristics [146].

Figure 6.3 illustrates the k-means clustering algorithm. The algorithm takes two inputs; (1) a set of data points and (2) the number of clusters (K). Then, it clusters data points into k clusters using a specific distance function. In the first round, the algorithm chooses some data points as centroids for each cluster randomly. Then, the distance between centroids and each data point, in a given dataset, is calculated. Accordingly, data points are allocated in clusters taking into consideration the closest centroid distance. After assigning all data points in the dataset, centroids are recomputed again for each cluster based on its data points. The assignment procedure is repeated till one of the stopping conditions takes place [146]. These are: (a) data points are not re-assigned to other clusters anymore (or minimum re-assignments), or (b) centroids are not recomputed anymore (or minimum change). The output of this algorithm is a matrix of clusters' indices for each data point in a given dataset. MatLab[1] k-means function is shown in

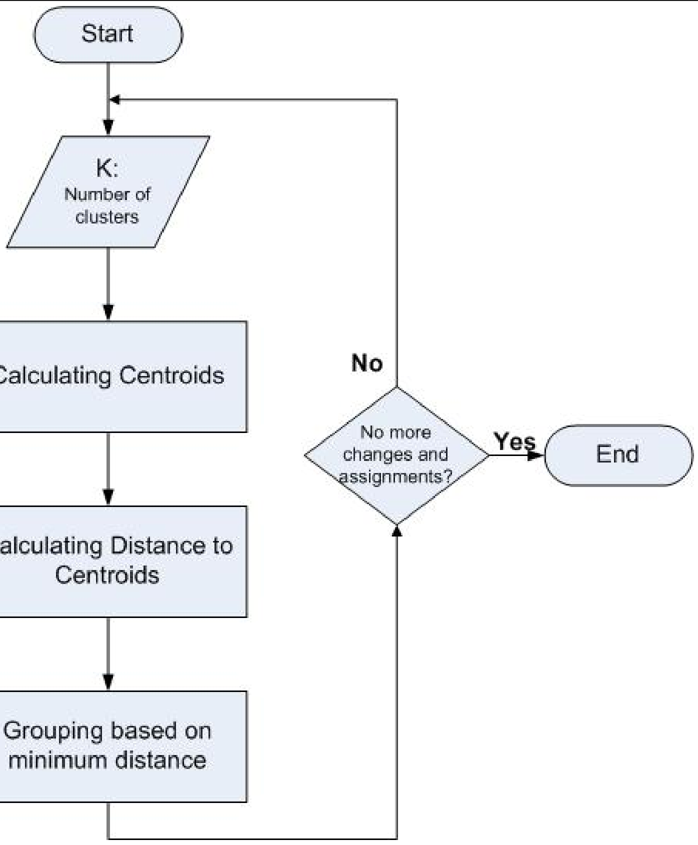
[1]http://www.mathworks.co.uk/products/matlab

formula (6.2).

$$IDX = kmeans(X, k), \tag{6.2}$$

where X is a matrix of web services and their QoS attributes and K is the number of clusters. *IDX* is the output matrix of the *kmeans* function, which includes the clusters' indices for each data point in a given dataset. MatLab *kmeans* function uses the standard Euclidean distance as a default function.

Figure 6.3 K-means Clustering Algorithm Flowchart


Start
K:
Number of clusters
Calculating Centroids
Calculating Distance to Centroids
Grouping based on minimum distance
No
No more changes and assignments?
Yes
End

6.1.2.3 Number of Clusters (K) Selection

To recognize the most suitable number of clusters for a given data points, Silhouette Coefficients is employed. Silhouette is used to plot clustered data points. Silhouette is a graphical representation of data points in each cluster [184]. It shows whether data points are well-positioned in each cluster or not. It is also used for comparing the quality of each cluster. Equation (6.3) [184] represent Silhouette Coefficients (*s(i)*) calculation for data point i in cluster A.

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}}, \tag{6.3}$$

based on [184], *a(i)* represents the average dissimilarity between a given data point i and the other data points in one cluster. *b(i)* represents the average dissimilarity between a given data point i and the other data points in other clusters. *s(i)* lies somewhere between *-1* and *1*, i.e. *-1* $\leq$ *s(i)* $\leq$ *1*. The value of *s(i)* has three cases [184]:

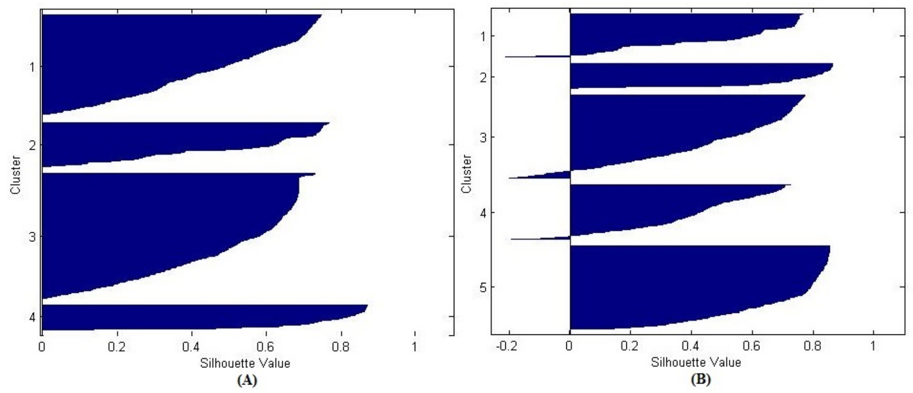
1. If the result of *s(i)* is close to **1**: in this case the data points are "well-clustered" and they were allocated to suitable clusters.

2. If the result of *s(i)* is around **zero**: in this case, data points can be allocated to other clusters those are close to their current cluster. In this case, the distance between the data point i and both clusters are equally the same.

3. If the result of *s(i)* is close to **-1**: in this case, data points are "misclassified". Accordingly, we consider them as outliers.

Figure 6.4 represents two examples of silhouette coefficients plots: (A) well-clustered data points and (B) includes some outliers and some misclassified data points.

Figure 6.4 Examples of Silhouette Coefficients Plots



After finding the appropriate number of clusters, these clusters have to be assessed and analyzed. That means metrics must be used to measure how good the clustering techniques are and to evaluate the quality of clusters. For clusters evaluation and comparison, some of the commonly used evaluation methods are: Purity, Precision and Recall. These metrics will be discussed in details in chapter 7.

6.1.3 Phase (3): QoS Aggregation

As mentioned before, when we subscribe to a web service, we actually borrow a "black-box" from the web service provider, with some related information about this service. When we compare web services, we need to value them using such information. Some information is taking from service providers (e.g. QoS and price), and some other information is taking from stakeholders (e.g. QoS weights, which convey the relative importance of a specific QoS). QoS aggregation is used for two purposes: (1) the first one is for valuing a specific service on service-level using equation 6.4, and (2) the second one is for computing architecture utility on the architecture-level using equation 6.5.

$$V_{WSi}(t) = (V_{qos_1}(t) \times w_1) + (V_{qos_2}(t) \times w_2) + ... + (V_{qos_n}(t) \times w_n) \tag{6.4}$$

$$V_{arch}(t) = \sum_{i=1}^{n} V_{WSi}(t), \quad \forall\, WSi \quad \exists\, d(WSi, WSj),\ i\, and\, j \in n \tag{6.5}$$

where $V_{WSi}(t)$ is the total value of web service at time t. This value will be used in next phase for the purpose of calculating web service value before substitution. $V_{qos_{n}(t)}$ is the QoS attribute, such as scalability, availability, etc. at time t. w_n represents the weight (relative importance), which is acquired from stakeholders. $V_{arch}(t)$ represents the architecture utility at time t. *d(WSi, WSj)* represents the dependency among web services in the architecture (dependency will be described subsequently). There are many web services QoS aggregation functions available in the literature some of them are based on QoS metrics that are provided by the service providers themselves and the other are calculated from a broker's viewpoint (e.g. [106, 107, 104]). We say that QoS metrics should be calculated from an unbiased point of view, such as a trusted service agent (third party broker). For that reason, we have taken our data from [9].

6.1.4 Phase (4): Real Options Analysis

In the previous chapter, we have discussed the Real Options theory, its background, its use, etc. We have also discussed different option valuation techniques and investigated the reason why we have chosen Binomial Model for option valuation. In this thesis, we look at different types of options: growth option, option-to-switch, option-to-abandon, and option-to-defer. We model the selection decision using the binomial model with respect to different scenarios those are related to the previous mentioned option types. Here, scenarios are used in order to aid decision makers in understanding what the architecture will be like.

As mentioned in chapter 4, the "behavioral" evaluation is performed on the service-level using real options. We use options (1) to quantify the value of the selection decision, and (2) to quantify the time-value of the decision and its implication on service value and architecture utility accordingly. Option thinking is fit for such evaluation as the non-functionalities tend to fluctuate. In fact, when valuing options, time is a significant factor that must be taken into consideration, as it helps in mapping out the structure and sequence of options, which in turns facilitate the valuation process. Here, we say that framing the selection decision by real options involves three steps: (1) describing the problem to be solved (scenario), (2) identifying option type that suits the problem, and (3) describing the "contingent" decision to be evaluated.

6.1.4.1 Binomial Tree Valuation

Our option analysis is based on the binomial tree. One binomial tree will be constructed for each candidate web service. One of the important characteristics of the binomial model that is produces graphical images, which helps in modeling and understanding options. In this context, we use the binomial tree for modeling different aspects which are related to web services selection, such as (1) the value of the candidate web service (when improving one or more QoS), (2) option value associated with each candidate web service across the option lifetime, (3) time-value of the selection decision, and (4) technical debt lifetime. As mentioned before, the binomial tree provides the “staging-view”, which we use in modeling these factors across a discrete-time of the option lifetime. Furthermore, the Binomial model identifies the likely future value at the end of the life of an option.

In this subsection, we provide the basics of option valuation using the binomial tree. Before plotting binomial trees, some information must be identified, such as tree coefficients, to be used as inputs to the process of building the tree (binomial tree coefficients were explained in pervious chapter). We built on the work of Ozkaya et al. [172] who has adapted the binomial model to value design decisions of architectural patterns, which was documented in [66, 21, 103]. We have adapted their model and introduced the weight dimension to their utility model. We have also introduced the dynamic dimension for computing the service value using u and d coefficients.

The subsequent steps will be used to produce a binomial tree for each candidate web service using CloudMTD model:

Inputs: m, which represents number of months and u, d and p coefficients.

1. Calculating the service value at t=0.
2. Calculating service value at each node of the tree (*up and down movements*).
3. Calculating options at terminal nodes.
4. Calculating options at each node of the tree (*backward induction/folding back process*).
5. Calculating final option value.

Now, we provide the details of the previous steps of building a binomial tree for each candidate service. **First**, we start by calculating the service value at t=0 by using the following equation 6.6.

$$V_{t0} = V_{WSi} + \Delta V_{qos_n} \tag{6.6}$$

where V_{t0} is the candidate service value at t = 0 (following substitution). V_{WSi} is the current service value prior to any improvement (before substitution, where it is acquired from equation 6.4). ΔV_{qos_n} is the improvement in service value when improving a specific QoS (e.g. Scalability). This value represents the difference in the V_{qos_n} between previous service (to be substituted) and the new one following substitution (Δ).

Second, the asset value, which is the candidate service's value in our context, is calculated at each node of the binomial tree, starting with t_1 at the left end and moving toward the right end t_m of the tree. Asset values are calculated using u, d and p coefficients. The asset value is calculated using formula 6.7. The first part of the formula is used for calculating an asset value for up movement in the tree and the second part is used for down movement in the tree.

$$V_{t_m} = \begin{cases} V_{t_{m-1}} \times u & \text{for up movement in the tree, where } m \geqslant 1 \\ \\ V_{t_{m-1}} \times d & \text{for down movement in the tree, where } m \geqslant 1 \end{cases} \tag{6.7}$$

where V_{t_m} is the total value of the candidate web service at time t_m. $V_{t_{m-1}}$ is the service value at t = m-1, which represents the service value in a previous step in the tree. u and d are respectively the percentages increase and decrease in service's value, which imparts the architecture utility, where each service is a contributor to the architecture utility. As mentioned before, u, d and p could be elicited using stakeholders input, historical data, or based on a given underlying asset valuation.

Third, options at the end nodes (terminal nodes) is calculated using equation 6.8, as we are modelling a call option.

$$f_m = \max(0, V_{t_m} - C) \tag{6.8}$$

where f_m is the option at terminal node m, V_{t_m} is the service value at terminal node m, and C is the cost of the option (exercise price), which depends on the option type.

Fourth, the *backward induction/folding back* process is completed through calculating options at each node by folding back from right-side toward the left-side of the binomial tree. The option price is calculated using equation 6.9:

$$f = \frac{p * f_u + (1 - p) * f_d}{(1 + r)} \tag{6.9}$$

where f is the option, and p is the probability coefficient. f_u is the expected payoff when service value goes up and f_d is the expected payoff when service value goes down. r is the interest rate.

Finally, after calculating options at each node of the tree, the final value is computed. We extract the rework cost out of the option to get the final value, i.e. *final value = option - rework cost*. Where this value is used to compare services and choose accordingly, in addition to technical debt dimension that will be described in next step.

6.1.5 Phase (5): Technical Debt Quantification

In chapter 3, we have discussed technical debt, its definition, dimensions, etc. in detail. We have also introduced a new dimension of technical debt explicating service-level in CB-SOA. In this subsection, we show how CloudMTD model uses option thinking for the purpose of modeling the technical debt lifetime using binomial tree. Technical debt on service-level is quantified based on three values: (1) option values at each month, (2) rework cost, and (3) time.

CB-SOA decision maker (e.g. an architect) will decide on accepted level of technical debt, when evaluating candidate web services for substitution. This technical debt needs to be monitored, tracked and managed for value creation. We posit that coining options thinking with technical debt analysis on service-level provides better assessment on (1) how web service substitution can clear technical debt and create value and future options (2) how web service substitution is likely to introduce an intentional debt in some cases, (3) when it is optimal to substitute a candidate web service, while tracking the debt against service value (more details in Section 3.8).

Algorithm 1 Technical Debt quantification and management for a given candidate service.

```
Require: $WS_i$ : Candidate service ID
Require: $OP_{month}$ : List of option values for each month of the candidate service $WS_i$
Require: $ED_{WSi}$ : Expiration Date of the candidate service $WS_i$
Require: $RW_{WSi}$ : Rework Cost of the candidate service $WS_i$
  for $month = 1 \rightarrow ED_{WSi}$ do
    Calculate TD per month: $TD_{month}$
    $TD_{month} = \max[-1 * (OP_{month} - RW_{WSi}), 0]$
    if $TD_{month} = 0$ then
      TD is managed at month: month
      Value-added is created at month: $month + 1$
      Exit Loop
    end if
  end for
```

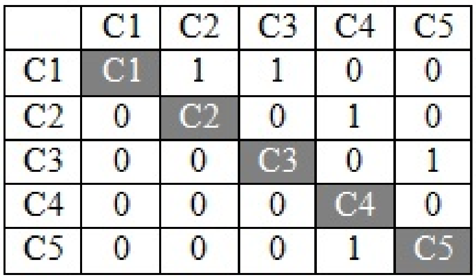
6.1.6 Phase (6): Architectural-level Evaluation

The previous phases were conducted for evaluating the *behavioral aspects* of CB-SOA on service-level and selecting a web service. We have also examined candidate web services nomination and the likely technical debt that may be incurred as a result of selecting a specific web service at a specific time. In this phase, we investigate the *structural aspect* evaluation of CB-SOA on the architectural-level and composing the selected web service. The structural aspect evaluation is carried out using three types of related metrics; those are (1) Dependency Structure Matrix (DSM), (2) Visibility Matrix, and (3) the Propagation-cost, for the purpose of visualizing the “ripple” impact of the change on the structure (CB-SOA). We use these metrics in order to determine the consequences of a change in any web service on the rest of web services in a given CB-SOA.

6.1.6.1 Dependency Structure Matrix (DSM)

Dependency structure matrix (DSM) is a way of representing dependencies between elements in a matrix format. It was introduced by Don Steward in 1981 [204] for the purpose of determining and managing interdependent variables in complex systems. After that, many research works have been using DSM in evaluating software design choices (e.g. [207]). Some available research works use DSM in order to understand how elements should be organized in a given system. In contrast, we use DSM in order to analyze existing CB-SOA structures. DSM is a simple square matrix that represents dependencies among elements of the *same type*, where 1 means dependent and zero otherwise (Figure 6.5). Some approaches blackout the diagonal elements in a given DSM for the purpose of mentioning that these elements are of no significance, such as [197]. Figure 6.6 present an example of building DSM.

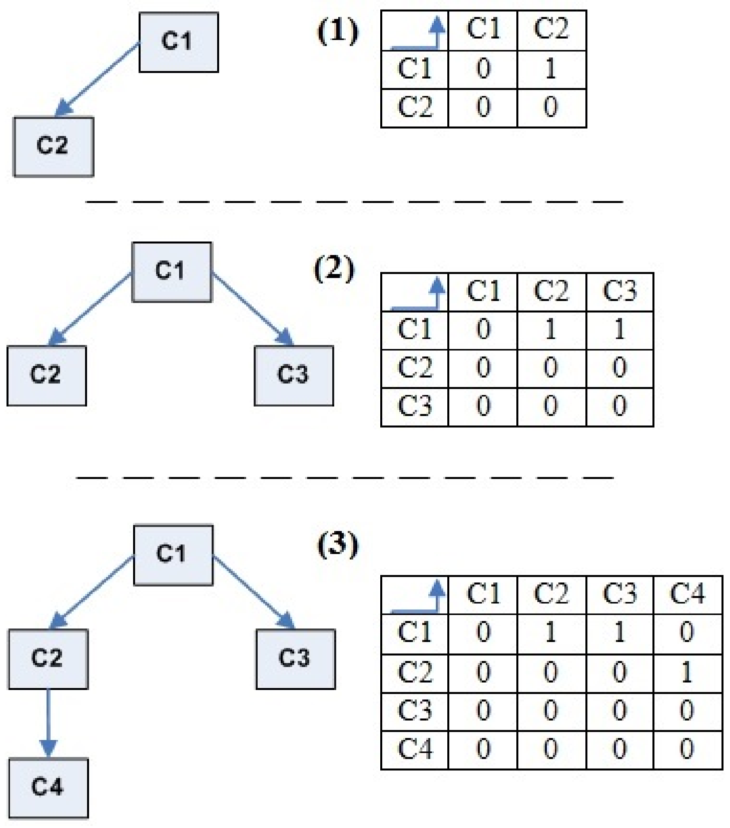
Figure 6.5 DSM Example



	C1	C2	C3	C4	C5
C1	C1	1	1	0	0
C2	0	C2	0	1	0
C3	0	0	C3	0	1
C4	0	0	0	C4	0
C5	0	0	0	1	C5

[1]DSM is also referred to as Design Structure Matrix

Figure 6.6 Example of building DSM



	C1	C2
C1	0	1
C2	0	0

	C1	C2	C3
C1	0	1	1
C2	0	0	0
C3	0	0	0

	C1	C2	C3	C4
C1	0	1	1	0
C2	0	0	0	1
C3	0	0	0	0
C4	0	0	0	0

In our context, we use DSM in order to represent dependencies among web services in CB-SOA architectures using the architectural view of the composition/system. In this context, when there is dependency between two web services in CB-SOA, any change to one of them may affect the other. Equation 6.10 represents the dependency mapping between an arbitrary web services, C_i and C_j, in a given CB-SOA structure. The first part of the equation represents a dependency between web service C_i and web service C_j, where C_i depends on C_j, or C_i calls C_j. The second part indicates that there is no relation between C_i and C_j. In this context, we use DSM in order to visualize web services dependencies and track the consequences of any change that take place in CB-SOA on the architectural-level. In this sense, DSM demonstrates how web services influence each other in a given CB-SOA structure. In addition, we compute some other metrics from DSM in order to characterize CB-SOA structure and figure out the consequences of a change on the architectural-level. These metrics are visibility and propagation-cost, which will be explained in details shortly.

$$Dependency(C_i, C_j) = \begin{cases} C_i \mathbb{R} C_j, & C_i \longrightarrow C_j \\ \\ C_i \overline{\mathbb{R}} C_j, & \text{No relation} \end{cases} \tag{6.10}$$

Furthermore, we treat DSM in CB-SOA as source of information for the purpose of interpreting architectural-level dependency analysis. Such information, which can be extracted from DSM, can be related to (1) web services dependencies and (2) the likelihood of change propagation (the consequences of change and dependencies related to the change), and (3) cost of change. After managing such information, benefits/value-added can be inferred when considering web service substitution in CB-SOA. We argue that benefits/value-added cannot be measured directly, however they are related to many dimensions that were discussed in previous chapters, such as cost saving. Here, we say that benefits/value-added can be measured from DSM by quantifying the cost of dependencies among web services in CB-SOA. In this sense, the cost of substituting or modifying a web service in a given CB-SOA increases if there are any other web services that are affected by the substitution or modification decisions, i.e. if there are any direct or indirect dependencies among web services. In this sense, the ability of analyzing change propagation can be interpreted in two aspects; cost and benefits/value-added.

6.1.6.2 Propagation-Cost Metrics

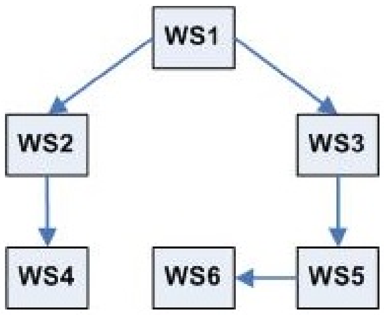
The architectural view that is presented by the DSM exposes a map of web services dependencies, which we use for analyzing CB-SOA structure. However, we must use metrics in order to characterize these maps among web services for comparison reasons (time-driven comparison will be discussed later on). For achieving these objectives, the propagation-cost metric is employed. Propagation Cost metric computes the percentage of the affected elements, on average, in a given system, when a change takes place to a given element in the same system [144]. Propagation cost metric has been used in many fields such as software error propagation cost [4], architecture modifiability [58, 168], characterizing the differences in design structure between software products [145], Iterative Release Planning [50], etc. In CB-SOA context, we use the propagation cost metric for computing the percentage of web services that may be affected on average, either directly or indirectly, when a change occurs to a given web service. In this sense, propagation on the architecture can be related to changes to the structure. For this thesis, the propagation cost quantifies the additional rearchitecting cost to integrate the candidate web service into the architecture.

We build on the work of [145, 144] and we introduce time and complexity sensitive propagation cost metrics to solve the problem. Their work was built on the concept of visibility introduced by Sharman and Yassine [197], which is based on the concept of reachability matrix that was introduced by John Warfield [226]. The reachability matrix demonstrates the needed number of steps to reach a given element from another one in a given network [226]. Based on the concept of reachability matrix, Sharman and Yassine presented the twin notions of visibility and dependency in a given architecture, where they posited that all architectures possess a visibility-dependency signature [197]. The concept of a visibility matrix is based on matrix multiplication techniques[2], in order to identify the "visibility" of a given element (web service in our context) for any given path length [145]. In this sense, a given DSM is raised to successive powers of n, i.e. a matrix is multiplied by itself for n times. The results of this procedure expose direct and indirect dependencies among web services for successive path lengths in a given CB-SOA structure. After that, by summing the resulting matrices, from M^0 to M^n, we get the visibility matrix V. Note that $n = 0$ indicates that any change to a given web service will always affect itself. Thereafter, we compute the architecture's propagation cost from the density of the visibility matrix V. Finally, we build on the this work by introducing the time and complexity sensitive propagation cost metrics for the purpose of computing the final weighted density matrix by reflecting the dependency complexity levels.

[2]For more information about matrix multiplication check: http://www.purplemath.com/modules/mtrxmult.htm

Figure 6.7 An example of an architecture and its DSM.

	ws1	ws2	ws3	ws4	ws5	ws6
ws1	0	**1**	**1**	0	0	0
ws2	0	0	0	**1**	0	0
ws3	0	0	0	0	**1**	0
ws4	0	0	0	0	0	0
ws5	0	0	0	0	0	**1**
ws6	0	0	0	0	0	0



6.1.6.3 Deriving Propagation-Cost from DSMs in CB-SOA

In web service substitution context, from the DSM of a given CB-SOA structure, we can conclude which web services are affected by substituting a specific web service and then we can compute the propagation cost in the entire architecture. In order to understand the use of DSM, visibility matrix and propagation cost metrics in CB-SOA, consider the following example. Figure 6.7 represents web services relationships in a given architecture and its corresponding DSM. It is clear that the web service WS1 depends on both WS2 and WS3. And accordingly, any change to web service WS2 or WS3 will affect web service WS1. Similarly, web service WS2 and WS3 depends on WS4 and WS5, respectively. Here, any change to web service WS4 may have a direct impact on WS2 and indirect impact on WS1 with a path length of 2. Similarly, any change to web service WS6 may have a direct impact on WS5, an indirect impact to WS3 with a path length of 2, and an indirect impact on WS1 with a path length of 3.

First, we plot the initial matrix, which is M^0 that contains ones along the diagonal of the matrix. Based on [144], including the matrix of the power n = *0* indicating that each web service depends and has an impact on itself with path length of zero. **Second**, we need to find the DSM of the intended architecture, which will be used as M or M^1, for n = *1*. **Then**, we derive the rest of the matrices from M^0 to M^n. For example, M^2 = M X M, and so forth. The multiplication process is stopped when matrix M^n is filled with zeros, or when M^n equals M^{n+1}. **After that**, we get the sum of all matrices from M^0 to M^n, using equation 6.11. Then, the architecture's propagation cost is computed from the density matrix, which is extracted by converting all non-zero cells in V matrix to ones. Figure 6.8 shows the derivation of the visibility matrix V based on the approach described above. The propagation cost of a given CB-SOA is given by equation 6.12 in (%), where nzc is the number of non-zero cells and nc is the total number of cells in the density matrix. This means that $x\%$ of the web services is likely to be affected, on average, by the change. However, this percentage does not provide a meaningful indication of the complexity of a change that could affect the cost in a given CB-SOA. For this reason, in next step, we introduce the time and complexity sensitive propagation cost metrics to compute the final weighted density matrix by reflecting the dependency complexity levels.

$$V = \sum_{i=0}^{n} M^i \tag{6.11}$$

$$PropCost_{arch} = \frac{nzc}{nc} \tag{6.12}$$

Figure 6.8 The Derivation of the Visibility Matrix V.

M^0

	ws1	ws2	ws3	ws4	ws5	ws6
ws1	**1**	0	0	0	0	0
ws2	0	**1**	0	0	0	0
ws3	0	0	**1**	0	0	0
ws4	0	0	0	**1**	0	0
ws5	0	0	0	0	**1**	0
ws6	0	0	0	0	0	**1**

M^1

	ws1	ws2	ws3	ws4	ws5	ws6
ws1	0	**1**	**1**	0	0	0
ws2	0	0	0	**1**	0	0
ws3	0	0	0	0	**1**	0
ws4	0	0	0	0	0	0
ws5	0	0	0	0	0	**1**
ws6	0	0	0	0	0	0

M^2

	ws1	ws2	ws3	ws4	ws5	ws6
ws1	0	0	0	**1**	**1**	0
ws2	0	0	0	0	0	0
ws3	0	0	0	0	0	**1**
ws4	0	0	0	0	0	0
ws5	0	0	0	0	0	0
ws6	0	0	0	0	0	0

M^3

	ws1	ws2	ws3	ws4	ws5	ws6
ws1	0	0	0	0	0	**1**
ws2	0	0	0	0	0	0
ws3	0	0	0	0	0	0
ws4	0	0	0	0	0	0
ws5	0	0	0	0	0	0
ws6	0	0	0	0	0	0

M^4

	ws1	ws2	ws3	ws4	ws5	ws6
ws1	0	0	0	0	0	0
ws2	0	0	0	0	0	0
ws3	0	0	0	0	0	0
ws4	0	0	0	0	0	0
ws5	0	0	0	0	0	0
ws6	0	0	0	0	0	0

$$V = \sum_{i=0}^{4} M^i$$

	ws1	ws2	ws3	ws4	ws5	ws6
ws1	**1**	**1**	**1**	**1**	**1**	**1**
ws2	0	**1**	0	**1**	0	0
ws3	0	0	**1**	0	**1**	**1**
ws4	0	0	0	**1**	0	0
ws5	0	0	0	0	**1**	**1**
ws6	0	0	0	0	0	**1**

The structural evaluation of a given CB-SOA architecture is given by the previous steps for the purpose of evaluating the entire architecture and its propagation cost. However, these steps do not give an indication of time and complexity sensitivities of a given dependency. Despite the fact that two services may be dependent, the complexity of the dependency may vary. For example, higher complexity may signal higher rework cost, lower complexity may indicate otherwise. This can be due to the effort required for code development, rewriting, configuration, data migration, new/throwaway maintenance, fixing mismatches, changes to legacy code, etc. For this reason, we introduce the time and complexity sensitive propagation cost metric to compute the final weighted density matrix by reflecting the dependency complexity levels. In this sense, decision makers (e.g. architects or stakeholder) can decide on the level of complexity per dependency through voting mechanisms backed-up by their experience and knowledge of the system (Table 6.1). The complexity level of dependency between any arbitrary web services WS_m and WS_n may vary depending on the complexity of change. This can be quantified based on the effort required (e.g. for code development, configuration, data migration, etc). The results of the process will be multiplying weights for adjusting dependencies, which we refer to as the "Complexity Density Matrix" (Figure 6.9).

Table 6.1: Density Matrix Level of Complexity

Scheme	Complexity level	Multiplication factor
A	Very high	1
B	High	0.7-0.9
C	Moderate	0.5-0.69
D	Low	0.3-0.49
F	Very low	0.1-0.29
0	No dependency	0

6.1.6.4 Time-Sensitivity

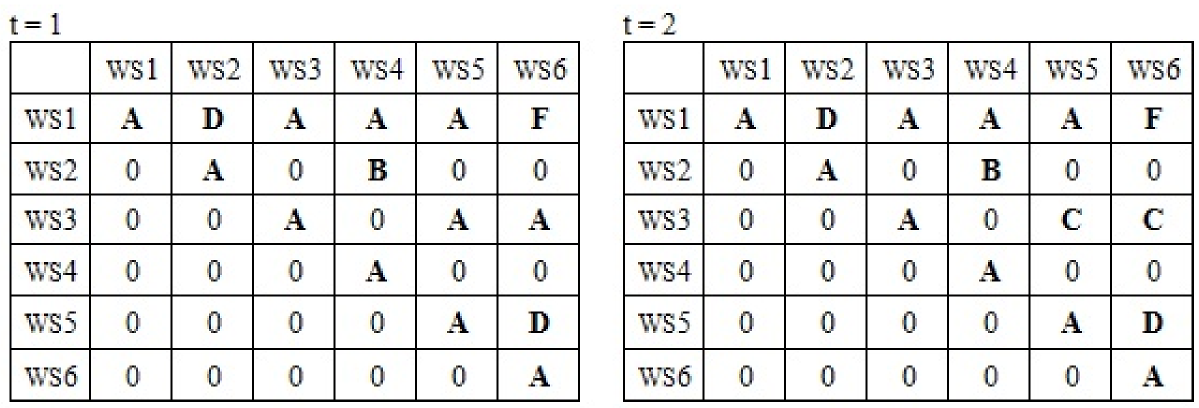
We posit that the "Complexity Density Matrix" can vary between time instances. This is to reflect the dynamic changes of the structure and its composition at varying time intervals. We assume that web service instantiation of the composing service abstraction

Table 6.2: Complexity Density Matrix values at t=1 and t=2

Level	t = 1	t = 2
A	11/15 = 73%	9/15 = 60%
B	1/15 = 7%	1/15 = 7%
C	0/15 = 0%	2/15 = 13%
D	2/15 = 13%	2/15 = 13%
F	1/15 = 7%	1/15 = 7%
0	No dependency	No dependency

tends to imply different complexity on the structure. Table 6.2 shows the complexity levels values in the Complexity Density Matrix at t=1 and t=2. The dependency analysis for t_1 and t_2 corresponds to taking the decision at either t=1 and t=2. This is helpful when considering option-to-defer a decision (an example of a detailed scenario will be discussed). After mapping levels of complexity of each web service from Table 6.1 we get the level of complexity of each dependency among web services. Table 6.2 shows that by substituting a given web service at t_1 that $\frac{15}{36} = 0.416\%$ of the affected web services by the change are of level complexity A(73%), B(7%), C(0%), D(13%) and F(7%). If we take the substitution decision at t_2, the percentage of elements classified under complexity level A has decreased. This can be attributed, for example, to upgrades to one of the legacy services in the composition, which has eliminated the need for designing wrappers for integration. As a result, the time value of the decision shall take into consideration the new complexity of the structure and its propagation cost at t_2, if we decide to take the decision at t_2.

Figure 6.9 Complexity Density Matrix at t=1 and t=2



t=1

	WS1	WS2	WS3	WS4	WS5	WS6
WS1	A	D	A	A	A	F
WS2	0	A	0	B	0	0
WS3	0	0	A	0	A	A
WS4	0	0	0	A	0	0
WS5	0	0	0	0	A	D
WS6	0	0	0	0	0	A

t=2

	WS1	WS2	WS3	WS4	WS5	WS6
WS1	A	D	A	A	A	F
WS2	0	A	0	B	0	0
WS3	0	0	A	0	C	C
WS4	0	0	0	A	0	0
WS5	0	0	0	0	A	D
WS6	0	0	0	0	0	A

6.2 Illustrative Examples

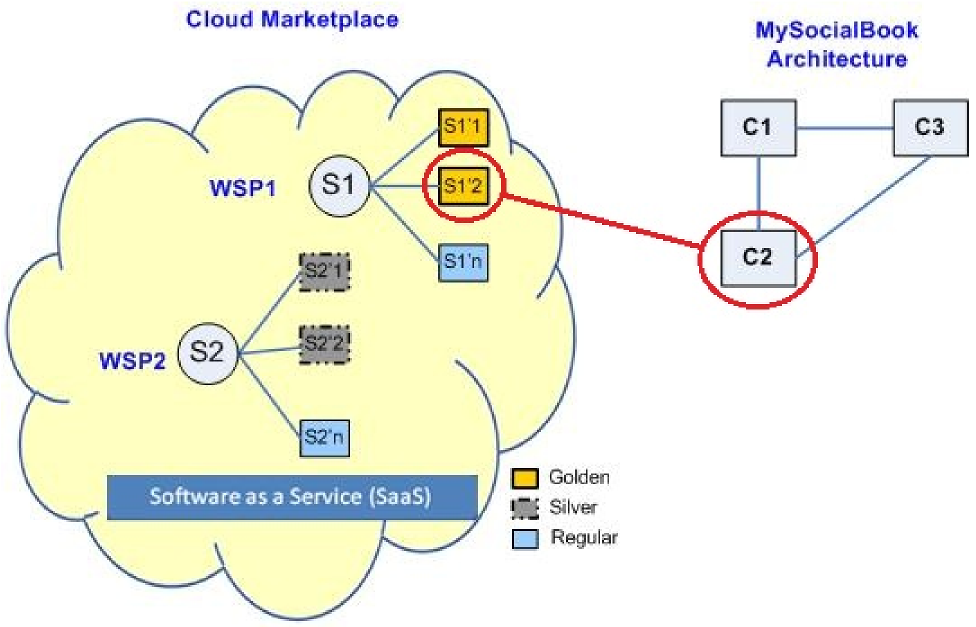
MySocialBook is a CB-SOA social network which is running in a highly uncertain environment, as the load tends to fluctuate and the demand for its services is uncertain. Its architecture is composed of different web services that are leased or bought off the cloud marketplace. MySocialBook offers different online web services, such as post, comment and ads. MySocialBook stakeholders are always looking for ways of maximizing the revenues. In CloudMTD context, this means maximizing utility and in options terms, it means maximizing the economic-value. Such structures, which are based on CB-SOA, can improve their utility and add value to their composition by substituting their constituent services. We argue that these systems can seek value by substituting their constituent services with web services offered for trading in the cloud marketplace. In the following scenarios, we assume that MySocialBook needs to substitute one of its constituent web services for a given critical stimulus that established the need for evaluation. This stimulus is driven by either business or technical objectives, such as end of web service contract, desire to improve utility of the architecture, change in users' requirements, need to upgrade to a new released web service in cloud marketplace, demand changes, etc.

We view the investment of MySocialBook in substituting one or more of its constituent web services as buying options and consequently long-term flexibility on an asset (Web service). The substitution can introduce a technical debt, which needs to be tracked, cleared and transformed from liability to value-added. Technical debt can cover several dimensions, which are related to selection, composition, and operation of the service that was explained in detail in chapter 3. As discussed previously, we consider two extremes while valuing flexibility and managing technical debt; *option in-the-money* and *option out-of-the-money*. When a candidate web service is selected, the service level agreement (SLA) mandates an expiry date of the service. We view this date as a lifetime that is associated with the acquisition of a web service, which we view as the life of the option. We view the expiry date of the service (i.e. end of contract) as the expiry date of the option. The system is monitored and the technical debt is tracked during the life of the option. The time until expiration is used for tracking the value and clearing the technical debt.

The general hypothesis is that substituting a web service at a specific time may embed flexibility under uncertainty to accommodate future changes (e.g. load fluctuation scenario) and generate value. The substitution decision may incur a technical debt, which can be translated into future options, if properly managed. CloudMTD analysis can provide the decision maker (e.g. application architect) with insights into economic viability of the technical solution, expressed in value-added and technical debt. Based on different scenarios, decision maker needs to assess the following while selecting a web service: (i) how web service substitution can clear an existing technical debt and create future options (ii) how web service substitution is likely to introduce an intentional technical debt, and (iii) when it is optimal to substitute a given web service, while tracking the technical debt against value. The decision shall leverage on the flexibility that a web service substitution can buy from the cloud marketplace and the value-added on CB-SOA architecture as a result. In each of the following scenarios, we take an option-based approach to inform the selection of candidate web services with varying costs and value. For every selection, we quantify the technical debt and the extents to which it can be cleared out and provide future options.

For the web service in question, MySocialBook has been offered different web services from different cloud service providers. Candidate web services are of comparable functionalities and may come with different non-functional requirements (QoS), cost and SLA provisions, Golden and Silver, as shown in Figure 6.10. In each scenario, Golden and Silver SLAs may vary in price and QoS. CloudMTD model informs the decision of selecting a candidate web service which will add value and can clear the technical debt more rapidly. We base the analysis on binomial tree as described previously. The CloudMTD will aid MySocialBook to quantify the possible future options and the associated technical debt with respect to time. We are interested in situations, where the options will start to pay off the loan (and the interest on the loan, if any) as a way for managing the technical

Figure 6.10 Web service substitution for CB-SOA



debt. In each scenario, CloudMTD option-based analysis aims at answering the following:

- Which of the offered web services will be adding value to MySocialBook system?
- When and how fast the technical debt will be cleared out during the lifetime of the acquired option?
- When is the optimal time for taking the decision?

In each scenario, we consider the case where MySocialBook needs to substitute only one of its constituent web services to meet changes. Our model, CloudMTD, tests the implication of substituting one web service. MySocialBook system will be evaluated with respect to the following assumptions:

- MySocialBook business goal is to improve a specific QoS in each scenario and accordingly maximizing the utility of the structure. Form CloudMTD point of view, MySocialBook business goal must take into consideration long-term value and should be debt-aware.
- In each scenario, the investment is viewed as a loan, that may incur interest and

signal a probable technical debt, which needs to be tracked and managed for value creation.

- Once a web service is acquired, there is an expiry date associated with the acquisition (end of contract), which we view as the expiration date of the option. The system will be monitored and the debt will be tracked during the life of option (e.g. 6 months).
- Flexibility is attributed to a web service being able to accommodate the likely future changes based on a specific scenario (e.g. load fluctuation).
- Value-added at time t_n is attributed to the likely increase in revenues generated by improving the QoS, which in turn imparts the utility of the architecture ($\Delta V = V_{t1} - V_0$).
- CB-SOA can have a base utility describing its current value.
- Substitution decision shall demonstrate a value-added on the utility; otherwise, the substitution may carry a technical debt on the attribute sought (e.g. scalability) and hence needs to be managed.
- The web service selection/substitution decision can be viewed as buying option (flexibility) on an asset (web service) and can be modelled as a call option. As mentioned in previous chapter, Call Option gives the right *-without the symmetric obligation-* to buy an asset (web service) of uncertain future value for an exercise price, where there is a potential benefit associated with exercising this option.

Based on the previous assumptions, a binomial tree will be built for each candidate web service, using the steps in the previous section. The scenarios below illustrate show the CloudMTD analysis for one selection. In each binomial tree, upper cells represent the total value of the candidate web service, generated from formula 6.7. The lower cells represent the option value, based on formula 6.9. In each scenario, we track the option and check when it will start to clear out the technical debt, if any. The scenarios show

how the analysis can link technical decisions in selecting web services to their value-added and likely technical debt. Poor and quick selection decisions may add a value in the short-term. In this context, we argue that the substitution operation should generate not only present value but also future one. As mentioned before, value could be created in various dimensions including savings in operational cost, improved Quality of Service (QoS), etc. We have reported on the following scenarios and results in [18, 19, 20].

6.2.1 Scenario (1): Growth-Option

In this scenario, we assume that MySocialBook becomes popular, and it may scale up in order to accommodate more active users benefiting from its services. Here, changes in scalability requirements trigger the need of substituting one of MySocialBook constituent services. In response to a scaling-up scenario, we select a substitutable web service with comparable functionality but with higher capacity. In our case, we view investment in scalability as a loan, which may incur interest by time. As the number of users benefiting from the scalability increases, the loan is said to gradually pay off through subsequent installments reducing the interest on the loan. An installment is said to be paid by active users, which are charged for using the service. Given it is a loan; we view it as a technical debt which needs to be monitored and managed for value creation. The selection decision of improving scalability will start to pay off when the technical debt is cleared out. In this scenario, uncertainty is attributed to probable fluctuation in load and the value generated as a result of accommodating scalability scenarios. In this scenario, we assume that scalability is the only QoS of concern when taking the decision of selecting a web service. Here, we specifically look at the technical debt that is incurred due to web service substitution decisions based on scaling up scenarios. We look at the technical debt in scenarios calling for substitution in response to changes in scalability requirements. Scaling up decision may need to be utilized for the expected load on the web service; otherwise, underutilization and the inappropriate management of the capacity of the service can lead to another technical debt.

In CloudMTD context, we view MySocialBook investment in substitution as buying strategic long-term growth options. We model scaling-up scenario of MySocialBook as a *call option* to acquire a web service with additional capacity by paying an exercise price. In this case, we are interested in the expected outcome which utilizes the capacity and generates value. Here, we say that MySocialBook needs to scale up to accommodate a load, and accordingly it will improve its utility through substituting one of its constituent web services to support higher load. In this context, we assume that the selection decision carries growth options in support of the load. Growth options is a real option on an asset with the view of unlocking the architecture value potentials through supporting more load and constrained by the service inherent capacity (growth options was discussed in details in chapter 5). But since the load fluctuates, the benefits tend to be uncertain. In addition, technical debt can be created if the payoff of acquiring the option continues to lag behind its cost for situations where the acquired web service capacity continues to be underutilized. In this scenario, we look at technical debt, which is related to situations where the service capacity is underutilized and the operational cost outweighs the revenue streamed from using the service.

We “stage” the valuation of the candidate web service in the view of managing the technical debt using the binomial tree. We assume that the selection is driven by requirements of scaling up and supporting more users. Each stage can reveal a value-added, which can be expressed as an option and can be used to manage and clear the technical debt. In this sense, the candidate web service may unlock future opportunities and accordingly enabling new business opportunities, which in turn can assist in managing the technical debt. In this scenario, MySocialBook current system is currently accommodating 100 users. The current value of the system is (£100). Table 6.3 represents the candidate web services that have been offered from the cloud marketplace.

Based on previous assumptions, two Binomial Trees were built; one for each offered web service, as seen in Figure 6.11 and Figure 6.12. The Binomial Trees were built using the coefficients attributes in Table 6.4.

Table 6.3: Shortlist of the offered Web services to be analyzed

Attributes	Web Service1	Web Service2
Expiry Date	6 months	6 months
Rework Cost	£130.00	£120.00
Exercise Price	£100.00	£80.00
Capacity	1140	1140
Availability	99.9 %	90 % - 99.9 %

Table 6.4: Binomial Tree Coefficients attributes for the candidate Web services

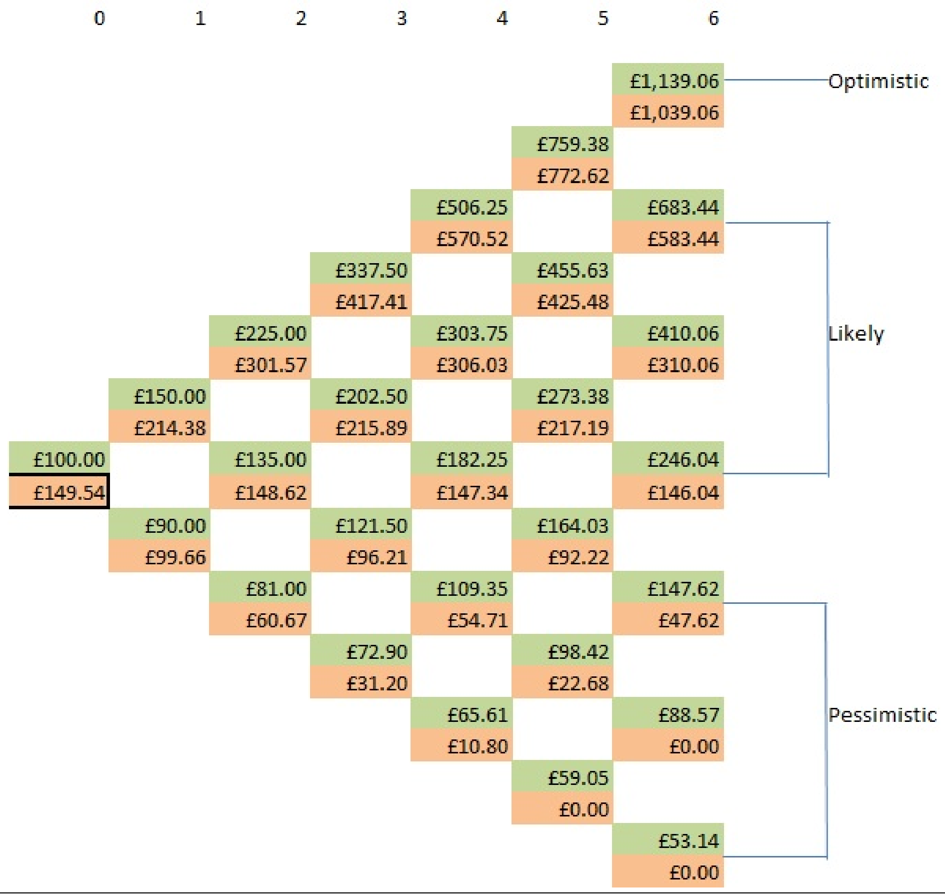
Attributes	Web Service1	Web Service2
V_{t0}	£100.00	£100.00
r	5 %	5 %
u	50 %	50 %
d	-10 %	-40 %
p	0.5	0.5

6.2.1.1 Growth Option Results and Discussion

If selection decision is made, the technical debt is said to be active and needs to be managed. The candidate web service may appear to be less attractive in the first five stages of the investment as it carries a technical debt (see Figure 6.11), the technical debt which can be manageable and be cleared out when the embedded flexibility of the structure starts to pay off through supporting extra active users on the configuration. For example, we can see that in the sixth stage the growth options value starts to exceed the rework cost clearing out the technical debt. The technical debt is said to be zero (see Figure 6.13). However, if the number of users taking advantage of the structure won't materialize, then the technical debt may require more time to clear out. The technical debt can even increase for scenarios, where debt will accumulate interest if the expected load continues to be underutilized.

Binomial valuation showed that the first web service will start to clear out the technical debt and add value during the sixth month (Table 6.5). Here, the value of the call option

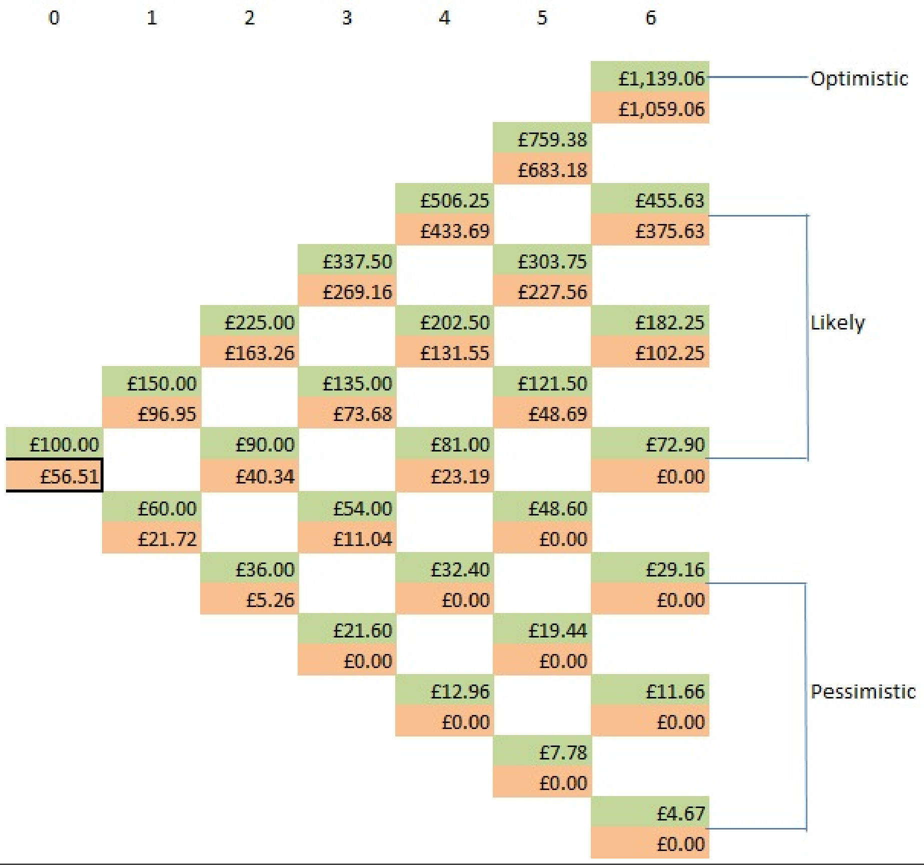
Figure 6.11 Six months evaluation of the first candidate web service



is an indicative measure of the selection decision in unlocking future growth opportunities and generating extra value. Here we say that the option is *in-the-money*. This is because, the value of the call option exceeds the rework cost, and the flexibility of the selection decision relative to the change in load is likely to pay off if the option is exercised. Here, real options analysis can further indicate that the technical debt can be manageable as the investment is likely to unlock future opportunities and creates growth options, which can cover for the debt.

In the second Binomial Tree (Figure 6.12), we see how acquiring second candidate web service will provide extra flexibility to scale up. However, that flexibility was underutilized by having much less load than what the web service capacity can normally handle. In this case, the value of the option is *out-of-the-money*. This is because the cost of acquiring the option continues to be higher than its expected payoff (as shown in Table 6.6). Here, the value of the call option is an indicative measure of the selection decision in incurring

Figure 6.12 Six months evaluation of the second candidate web service



debt. This is because the value of the call option drops to zero, and the flexibility of the selection decision in response to the change in load is unlikely to add value. Scaling-down by switching to a lower capacity web service can be an alternative solution for managing the technical debt.

Figure 6.13 Technical Debt vs. Option Value over Six months evaluation of the candidate web services

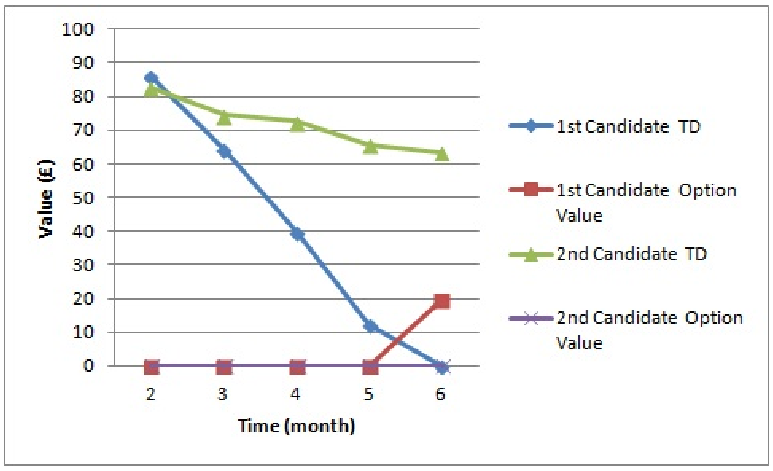


Table 6.5: Option Values of the first candidate web service

Month	Option Price	Technical Debt	Option Value
2	£44.22	(£85.78)	£0.00
3	£65.81	(£64.19)	£0.00
4	£90.09	(£39.91)	£0.00
5	£117.81	(£12.19)	£0.00
6	£149.54	£0.00	£19.54

Table 6.6: Option Values for the second candidate seb service.

Month	Option Price	Technical Debt	Option Value
2	£37.4150	(£82.59)	£0.0000
3	£45.6214	(£74.38)	£0.0000
4	£47.4211	(£72.58)	£0.0000
5	£54.1888	(£65.81)	£0.0000
6	£56.5091	(£63.49)	£0.0000

6.2.2 Scenario (2): Switch-Option

In this scenario, we consider the case of a switch-option. Here, MySocialBook has subscribed to a web service and has accidently taken a quick decision focusing on the short-term cost and value. This is due to deadline restrictions. In CloudMTD context, this is called an unintentional technical debt. Since the selected service features do not fully match the requirements of MySocialBook application; service switching may be necessary in order to better fit MySocialBook requirements. MySocialBook has considered a new web service provider and needs to model switching decision in relation to technical debt (whether to switch or not).

6.2.2.1 Switch Option Results and Discussion

CloudMTD analysis will be carried out to know whether if the value of the new web service will cover the rework cost (which is incurred due to switching) and will deliver a value-added relative to the current web service. We use CloudMTD to model the current web service's value against the candidate web service's value. Let us assume a scenario, where we need to model the technical debt and the value-added on the structure, expressed in utility. Let us look at the case of availability, where 100% availability is likely to increase the service value by an average of 40% per month. Similarly, we assume that the architecture is likely to experience 10% unavailability on average during the valuation period leading to an average of 40% decrease in its utility. If the service capacity will be fully utilized, the situation is assumed to be optimistic benefiting from 40% increase in utility. The option value is likely to exceed that of the operational cost clearing any debt. Alternatively, when the operational cost exceeds that of the option value, the debt will be visible. In this scenario, CloudMTD model is used to inform the decision of substituting to another web service with higher availability, which has the potential to improve the option value and reduce the technical debt. The analysis shows that the current system value (utility) is decreasing on average due to the unavailability of the service and lost opportunity in supporting expected load (Figure 6.15). Figure 6.14

shows that technical debt started to be visible as of the third month while showing a decreasing trend in the option value reaching zero on the fourth months. We argue that the utility may be improved by switching from the current web service to another one with higher availability (Table 6.7). Though short-term cost-benefit analysis shows that switching looks to be unattractive decision for the first four months, option calculation reveals likely value-added following the fourth month. The case justifies that taking on an *intentional* technical debt due to rework is likely to create an option, which will outweigh the cost of rework.

Figure 6.14 Clearing TD vs. Option Value over 6-month evaluation

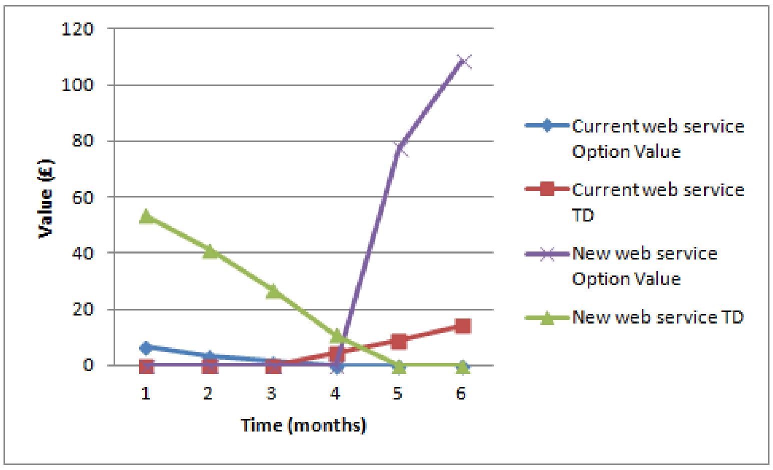


Figure 6.15 Current web service (a) first and (b) second months option value.

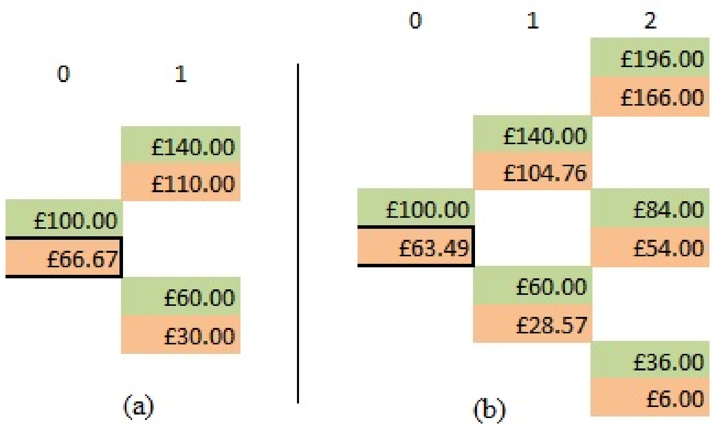


Table 6.7: TD vs. Values of the new candidate web service

Month	TD	Value
1	53.42857	0
2	41.17007	0
3	27.05086	0
4	10.88139	0
5	0	77.54535
6	0	108.4555

6.2.3 Scenario (3): Abandon-Option

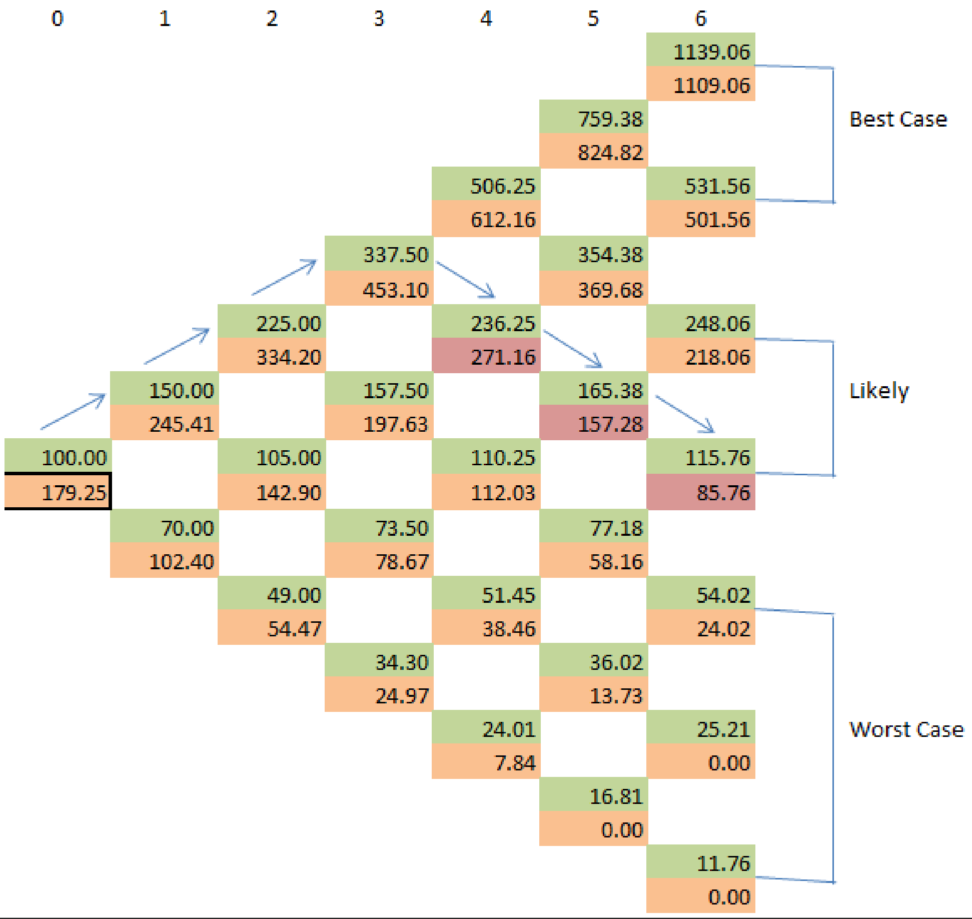
In this scenario, we demonstrate the case of unmanageable technical debt, which will lead to service abandon due to the unacceptable gap between the expected utility-level and the actual one. In this scenario, MySocialBook has added a new advertising service to its system. After adding the new service, MySocialBook is supposed to preserve its business sustainability and prevents any likely negative impact on their system. Therefore, MySocialBook needs to know whether if it is beneficial to keep, substitute or abandon the new service. In addition, MySocialBook needs their system to accommodate the variation in load (number of end-users fluctuation) with respect to time. The system should be able to scale under uncertainty, where uncertainty is attributed to the unpredictable fluctuation in load. In that sense, the analysis is carried out based on options thinking. Whether to keep the new service or not is viewed as an option. As discussed before, Real Options valuation identifies the value of the investment, which is not only in the expected direct revenues, but also in the future opportunities that flexibility creates [208, 78].

MySocialBook has different needs and requirements which may change depending on a specific time of the year, i.e. MySocialBook requirements may change depending on the number of end-users at a specific time during the year (time-sensitive). MySocialBook needs a systematic methodology that helps them in quantifying the decision of whether to keep, substitute or abandon their new service, and predicts the value-added. Moreover, MySocialBook requires a model to provide them with an analytical view of the system. The system should be flexible in which it can scale up or down to accommodate the fluctuation in QoS generally and load specifically (number of active users). Moreover, the system should be able to handle the sudden increases in load. MySocialBook new service should be scalable, which means accommodating more user requests during peak times (scaling up) and scaling down as demand falls down.

6.2.3.1 Abandon-Option Results and Discussion

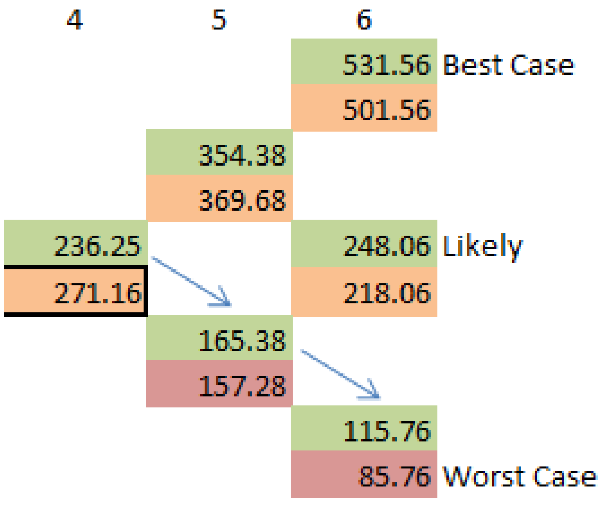
After 3 months of adding the new advertising service, MySocialBook observed that their utility "expected-level" is degrading and approaching the lowest point of their "accepted-level" (threshold). Accordingly, their business value is degrading as a result of losing customers. From CloudMTD perspective, the difference between the utility "expected-level" and "actual-level" is triggering a *technical debt* that is likely to increase if the value keeps on degrading. In the view of options thinking, on the fourth month the option value starts to degrade gradually till it reaches zero where the value is unable to cover for the cost and as a result signaling technical debt. At this point, there are three scenarios: (1) Best, if the option value goes up again by gaining and accommodating more users, (2) likely, the plateau mode: if the value stays as is, and (3) worst, if the option value keeps on degrading. At the moment, the question is whether to keep, substitute or abandon the contract of the new advertising service. Here, we view the best case scenario in Figure 6.16 as a "virtual-value", as the option is extra flexible and unlikely to pay off in future. Seeing that the capacity of the new released service is not fully utilized as expected during the first few months. The analysis showed that the service was newly released, and as a result MySocialBook gained a "short-term" benefit during the first three months. During the first three months, the option value was increasing and covering the operational cost, interests and technical debt, if any. After a few months, MySocialBook could not retain the users of the new service.

Figure 6.16 Six-month evaluation of the new released service.



0	1	2	3	4	5	6	
						1139.06	
						1109.06	
					759.38		Best Case
					824.82		
				506.25		531.56	
				612.16		501.56	
			337.50		354.38		
			453.10		369.68		
		225.00		236.25		248.06	
		334.20		271.16		218.06	
	150.00		157.50		165.38		Likely
	245.41		197.63		157.28		
100.00		105.00		110.25		115.76	
179.25		142.90		112.03		85.76	
	70.00		73.50		77.18		
	102.40		78.67		58.16		
		49.00		51.45		54.02	
		54.47		38.46		24.02	
			34.30		36.02		
			24.97		13.73		
				24.01		25.21	Worst Case
				7.84		0.00	
					16.81		
					0.00		
						11.76	
						0.00	

Figure 6.17 Evaluation of the new released service, after the third month.



4	5	6	
		531.56	Best Case
		501.56	
	354.38		
	369.68		
236.25		248.06	Likely
271.16		218.06	
	165.38		
	157.28		
		115.76	
		85.76	Worst Case

Figure 6.18 TD vs. Option value.

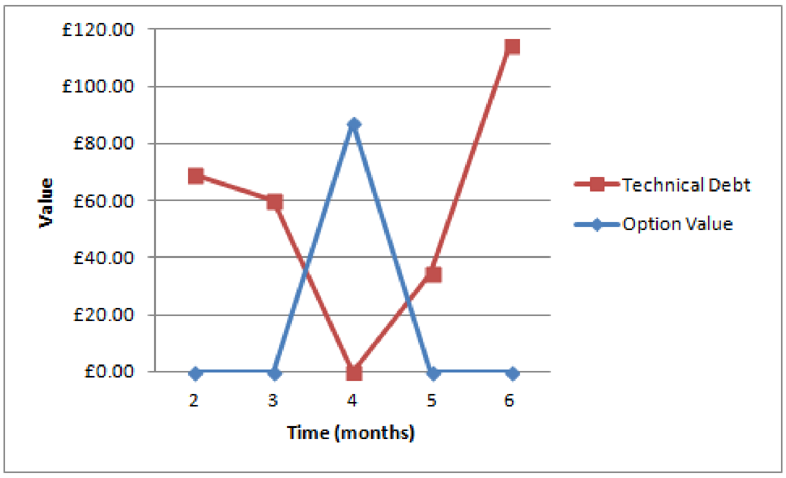


Table 6.8: Add caption

V_{t0}	£100.00
r	5%
u	1.50
d	0.70
p	0.6
Exercise Price	£30.00
Expiry Date	6 months

7 Evaluation

7.1 Overview

In the previous chapters, CloudMTD model was described and explained for the case of web service selection and composition in Cloud-Based Service-Oriented Architectures (CB-SOA). In this chapter, we report on the implementation of CloudMTD model and we report on some results related to the behavioral and structural aspects of evaluation. We also demonstrate the applicability of CloudMTD model on a real word case study, which is built on CB-SOA architecture.

In this thesis, the evaluation method is based on a combination of case study and experimentation. The evaluation procedure consists of two major stages; the first one is related to CB-SOA selection evaluation (behavioral) and the second one is related to CB-SOA composition evaluation (structural). The experiments were designed in order to verify how the CloudMTD model performs when varying the following: (1) number of web services, (2) number of binomial steps, (3) options stages and (4) technical debt stages. We are interested in tracking the performance and examine the extent to which the CloudMTD model can scale taking into consideration these sensitive variables. Here, we are testing the impact of changing these variables on the performance.

7.2 The Evaluation Process Context

For the evaluation process, we can distinguish between two different types: Qualitative Evaluation and Quantitative Evaluation.

7.2.1 Qualitative Evaluation

In this type of evaluation process, we rely on the decision makers' preferences. It may include discussing different aspects qualitatively that are related to the problem and solution. It is also used to qualitatively compare a set of scenarios. This is useful when some aspect is hard to quantify (e.g. security, value-added, etc.). We evaluate the "goodness" of selection and composition decisions by the use of the utility function, as this is the common practice in most research works in the literature. In previous chapters, we have differentiated between two levels of utility: the “expected-level” and the “actual-level”. We have also posited that in CB-SOA selection and composition, the utility is subjective in nature and depends on the level of relative satisfaction of decision makers. The decision maker is interested in achieving the desired levels of QoS attributes, which in turn imparts the utility. Accordingly, different utility levels are calculated based on QoS attributes in a given web services dataset (QoS aggregation and utility calculation was discussed in detail in the previous chapter). In this sense, we use the decision maker’s expected-level as the benchmark, which we use to compare against the actual-level. Here, we define the utility gap as the difference between expected-level of utility and the actual one. In this context, each decision maker defines their own utility threshold, from which we define the decision maker’s satisfactory levels of utility. In this sense, if the utility gap is within the decision maker’s satisfactory levels, the decision is said to be satisfied. On the other hand, this utility gap is used for defining the technical debt and its acceptable levels, as discussed in chapter 3.

7.2.2 Quantitative Evaluation

In this type of evaluation process, we are interested in different aspects: scalability, relevancy, and performance. These aspects are correlated when evaluating CloudMTD model. In this context, we say that performance is evaluated taking into consideration relevancy and scalability goals. Scalability goals are acquired from the decision maker in order to evaluate CloudMTD model based on the desired number of web services to be tested in a specific time. We evaluate scalability of the model where we are interested in testing the extent to which the CloudMTD can scale with respect to the number of web services. In this sense, the number of web services is the variable that affects CloudMTD performance.

7.3 OptiViTA Case Study

OptiViTA is an adaptation of a Travel Agency case study which was published by the department of Informatics at University of Fribourg, Switzerland [1]. OptiViTA case study was implemented using Web-Services Business Process Execution Language (WS-BPEL) and Java (Figure 7.1 and 7.2). We assume that the CB-SOA architecture has WS-BPEL business process representation, where each process in the WS-BPEL workflow represents task(s) in the corresponding architecture. OptiViTA is a Cloud-Based Service-Oriented Architecture, which consists of a group of web services that all together form a cloud computing platform. Web services include: flight booking (wsF), hotel booking (wsH), car hire (wsC). In a typical scenario of OptiViTA business process when receiving a request, it invokes the involved web services and then sends the responds. OptiViTA is a large-scale distributed system based on internet business. This means that it requires high performance, scalability and availability accordingly. In this sense, the system should be able to handle thousands of requests generated by online users. In addition, the system should response to these requests in a specific time fraction, as any lost request can affect OptiViTA's business value. OptiViTA CB-SOA architecture is illustrated in the application tier in Figure7.3.

OptiViTA case study was used as an implementation environment for applying option-based analysis in the view of addressing web service selection and composition problem. We have applied our CloudMTD model to this case study in order to review the OptiViTA core architecture and its constituent web services.

Figure 7.1 OptiViTA BPEL

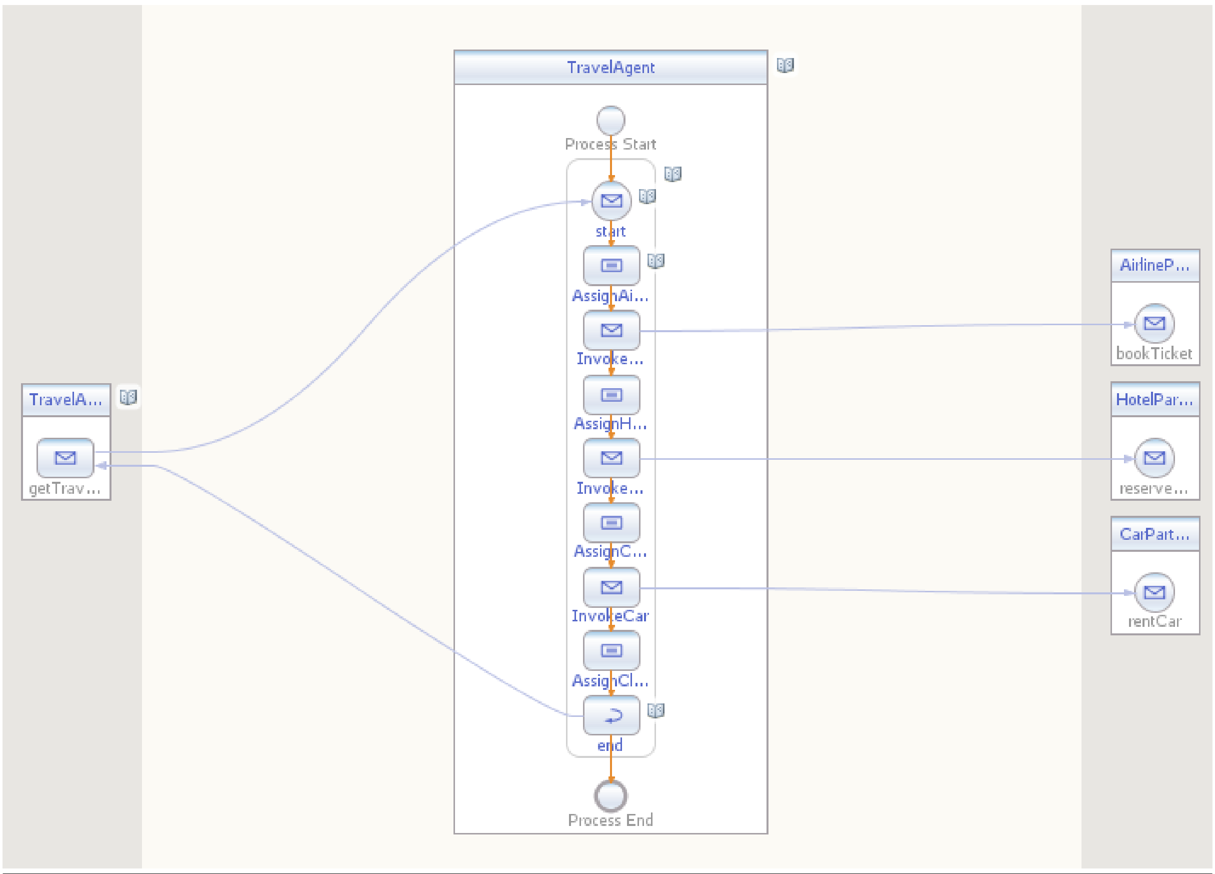
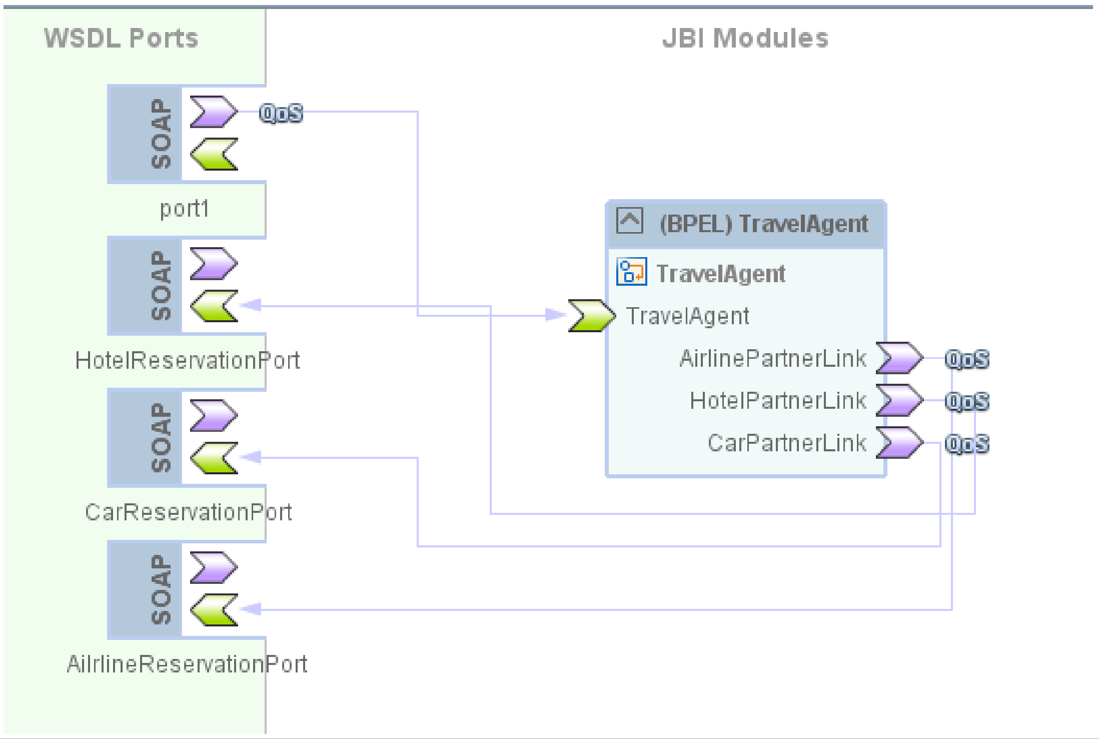


Figure 7.2 OptiViTA CASA



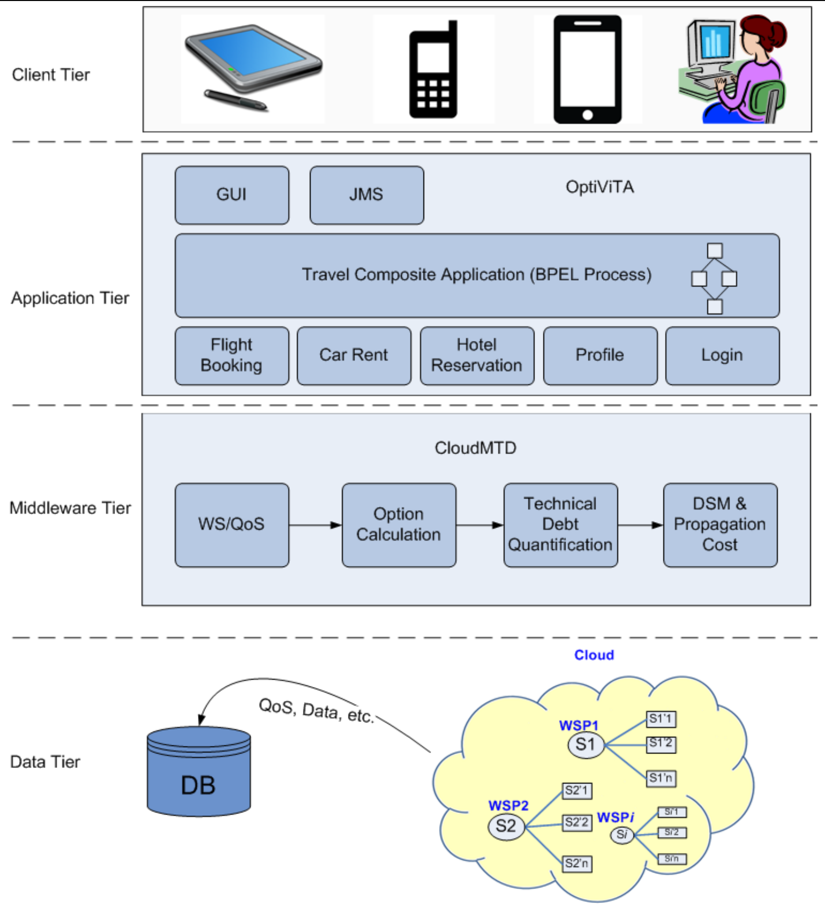
7.4 Environmental Setup and Implementation

Implementation was done as a proof of concept and the main purpose of the implementation is to fulfill interaction requirements of each phase of CloudMTD model. Implementation Architecture Tiers is shown in Figure 7.3.

Software: different softwares were used based on the needed techniques, such as OpenESB[1] for CB-SOA architecture implementation, SOAPUI[2] for simulation, performance testing and stress testing of the architecture, and NetBeans Profiling for the CloudMTD performance and stress testing. Each part will be explained subsequently.

Hardware: implementation, testing, and experiments were done using an Intel(R) Pentium(R) CPU P6100 2.00GHz machine, with 4GB of RAM, and 64-bit Operating System.

Figure 7.3 Implementation Architecture Tiers



[1]http://www.open-esb.net
[2]http://www.soapui.org

7.4.1 QWS Dataset

We have chosen a large dataset for the purpose of representing the cloud marketplace. The dataset was also used with the aim of stress testing the CloudMTD model with respect to performance and scalability metrics. Until now, there have been few publicly available QoS datasets that represent real web services, so we are limited to three possible real datasets. The first dataset is the QWS[3] [11], which is the dataset we have used. The second one is the WS-DREAM[4] [241], which consists of three attributes: Response Time, Response Data Size and Failure Probability (the website was not available for sometime). The third dataset is the tpds2012[5] [243], which consists of 500 web service and two attributes: response-time and throughput. Other research work has published their own randomly generated web services such as [224], which consists of 10,000 web services and 3 QoS attributes (Response Time, Response Data Size and Failure Probability).

As mentioned before, even though there are many ways of collecting data about web services and QoS aggregation available in the literature some of them are based on QoS metrics that are provided by the service providers themselves and the other are calculated from a broker's viewpoint. We say that QoS metrics should be calculated from an unbiased point of view, such as a trusted service agent (third party broker). For that reason, we have taken our data from [9, 11]. Furthermore, QWS dataset holds information about various web services which happen to satisfy our case in terms of the selection problem scope.

The QWS [11] consists of two Datasets; Dataset1 = 365 web services and Dataset2 = 2507 web services. Every row in the QWS dataset represents an accessible web service that was tested over the period of three days. Each web service in this dataset may satisfy the requirements of a candidate web services in our model. The dataset consists of 9 QoS, which were calculated based on some commercial benchmark tools (described in Table 7.1). Al-Masri et al. data were collected from some sources, which are publically available

[3]http://www.uoguelph.ca/ qmahmoud/qws/index.html
[4]http://www.wsdream.net
[5]http://www.zibinzheng.com/tpds2012

on web using Web Service Crawler Engine (WSCE). These datasets were extracted from [11, 9] and saved into MySQL database to be called by our Java program (CloudMTD model). Finally, we note that we have reported on four attributes of the QWS dataset: Availability, Successability, Reliability, and Compliance. However, our CloudMTD model is generic enough and flexible to deal with the rest of the attributes of the given dataset.

Table 7.1: Dataset QoS attributes [11].

QoS Attribute	Description	Units
Response Time	Time taken to send a request and receive a response	ms
Availability	Number of successful invocations/total invocations	%
Throughput	Total Number of invocations for a given period of time	invokes/second
Successability	Number of response / number of request messages	%
Reliability	Ratio of the number of error messages to total messages	%
Compliance	The extent to which a WSDL document follows WSDL specification	%
Best Practices	The extent to which a Web service follows WS-I Basic Profile	%
Latency	Time taken for the server to process a given request	ms
Documentation	Measure of documentation (i.e. description tags) in WSDL	%
WsRF	Web Service Relevancy Function: a rank for Web Service Quality	%
Service Classification	Levels representing service offering qualities (1 through 4)	Classifier
Service Name	Name of the Web service	None
WSDL Address	Location of the Web Service Definition Language (WSDL) file on the Web	None

7.4.2 WS-BPEL

WS-BPEL language was chosen as it facilitates the analysis for web service substitutability while keeping the orchestration, requirements and constraints explicit. The version that was used is WS-BPEL 2.0 as it is supported by OpenESB tool. Business Process Execution

Language (BPEL) and Web Services Business Process Execution Language (BPEL4WS 1.1 and WS-BPEL 2.0) is an orchestration language that is used for assembling a group of web services into an end-to-end process workflow. BPEL is normally used to model the behavior of executable business processes based on OASIS standards (Organization for the Advancement of Structured Information Standards)[6]. We have implemented our CB-SOA architecture (OptiViTA case study) using WS-BPEL.

7.4.3 OpenESB

OpenESB[7] was chosen as an environment for implementing WS-BPEL web service business processes. OpenESB is a cross-platform open source Enterprise Service Bus (ESB) based on Java. OpenESB relies on standard Java Business Integration (JBI), XML, XML Schema, WSDL, BPEL and Composite application that are used for building SOA applications. We have used OpenESB as it is an open source that is reliable, powerful, scalable, simple and efficient. In addition to, that OpenESB had less computational overhead than other SOA integration tools, which we have examined during the implementation process such as Oracle SOA and Business Process Management (BPM) Suite[8], Eclipse SOA Platform Project[9], and NetBeans SOA Project[10]. The version that was used is OpenESB 2.3.1, which relies on NetBeans 7.3.1 and Glassfish 2.1.1 (Glassfish is an open-source application server).

7.4.4 SoapUI

SoapUI is a cross-platform open source simulation and mocking software that is used for functional and load testing of both SOAP and REST web services. Normally, web services are either based on SOAP or REST standards. For the nature of our problem, we have chosen to implement web services using SOAP instead of REST. SOAP was chosen as it

[6]https://www.oasis-open.org
[7]www.open-esb.net
[8]http://www.oracle.com
[9]http://www.eclipse.org/soa
[10]http://soa.netbeans.org

is language- and platform-independent protocol, while REST requires the use of HTTP (note: transport protocol for information exchange includes [202]: HTTP, HTTPS, SMTP, etc.). We have also used WSDL, which is an interface description based on XML, for the purpose of describing the functionality of our workflow and web services. In addition, WS-BPEL is “wrapped” as WSDL and accordingly can be tested using SOAPUI. In our case, WSDL files were used as inputs to be tested using SoapUI. SoapUI version 5.0.0 was used in order to carry out the functional testing of CB-SOA implementation, i.e. testing how WS-BPEL workflow is working.

7.4.5 NetBeans Profiling

NetBeans Profiling[11] was used for the purpose of performance and stress testing of our CloudMTD model as well as for measuring the overhead associated with it. NetBeans Profiling is an open source tool that provides information about runtime behavior of an application. It allows monitoring of thread states, CPU performance, and memory usage. The original meaning of profiling is concluding information about something based on set of criteria. And in Software Engineering particularly, profiling is the act of analyzing dynamic programs in order to measure performance, memory usage, complexity, etc. based on some criteria at runtime.

7.5 Experiments Design

The CloudMTD model is developed to screen the cloud marketplace, which is represented as a dataset in the database, and select a service. The assumption is that the CloudMTD model is value- and debt-aware. We are interested in reporting on the performance of the CloudMTD model in selecting these web services, once we query the cloud marketplace.

[11]https://netbeans.org/kb/docs/java/profiler-intro.html

Inputs: Inputs to CloudMTD model are divided into two categories based on offline and online calculations. Some data must be calculated offline or acquired from the CB-SOA decision makers (e.g. stakeholders), such as the binomial tree coefficients. These data are stored in the database after been acquired from decision makers. Some other data are calculated online.

The choice of QoS attributes: The evaluation was performed bases on the following QoS attributes from the QWS dataset: (1) Availability, (2) Successability, (3) Reliability, and (4) Compliance. First, the choice of the QoS attributes was based on: (1) QoS attributes importance, for a given case, and (2) behavioral requirements. We assume that some attributes are more important than other ones. For example, if the service is unavailable or has a low availability, it is unlikely to be used and the business value will be affected accordingly. In addition, we are interested in the behavioral requirements, as these requirements are likely to change and fluctuate. These attributes are more dynamic and more likely to carry different types of information at different stages of the evaluation process. Accordingly, these attributes imply different values and give further insights, which inform the selection of services. This will be explained in detail in the QoS correlation section. Some other attributes in the QWS dataset tend to be static. These attributes may not carry extra information and the change is unlikely to be observed. These attributes were excluded accordingly (e.g. documentation). Second, these attributes are aggregated in order to be used as an input for the binomial model. In this sense, the QoS aggregation and ΔV_{QoS} calculation phases may be affected by the number of attributes. However, the other phases of the CloudMTD model will not be affected by the number of QoS attributes, such as options and technical debt calculation phases. Third, following similar argument of [165], where they claimed in their SLR that the majority of QoS-aware selection approaches used 1-3 QoS attributes. We went beyond this number and we tested CloudMTD model by using 4 QoS attributes. Accordingly, we have fixed the number of attributes to 4 QoS attributes.

The choice of number of stages: Stages are periods of time that are selected per scenario of interest (e.g. 6 months). Number of stages is used for calculating options and calculating the technical debt. Based on the literature, business planning cycles are normally based on quarterly intervals [172]. Ozkaya et al. used three stages evaluation using the binomial tree, we went beyond this number and we tested CloudMTD model by using different number of stages up to 6 stages.

The number of test cases: The evaluation process was performed on 200 test cases. The first category was tested using the extreme case of 2507 web services. The second category was tested based on varying number of web services. In each category, we have varied the following: (1) number of web services, (2) number of binomial steps, (3) options stages and (4) technical debt stages.

7.6 Results: Performance and Memory Usage

Profiling techniques were used in order to report on performance accepted-levels and to identify bottlenecks those are related to performance. We have used two profiling techniques: Performance and Memory Usage. Performance metric is used in order to know how much each method is contributing to the overall execution time. Table 7.2 describes the analogy and definitions of each method in CloudMTD model and its implementation. Profiling results were extracted based on the QWS dataset [11] as inputs. The following figures are samples of the results of execution time for each method in CloudMTD model. Figure 7.7 presents performance test results for CloudMTD methods those are related to connecting and disconnecting to the database to import web services data, QoS aggregation and QoS attributes improvements. Figure 7.6 illustrates the CloudMTD performance test results for CloudMTD methods those are related to the selection phase, such as technical debt and options calculations. Figure 7.8 illustrates CloudMTD performance test results for CloudMTD methods those are related to the composition phase.

Next, we present the results of 200 test cases based on different variables. The purpose is to understand how CloudMTD model can be sensitive to these variations. Table 7.3 presents three different categories of results, taking into consideration 2507 web services and varying number of stages from 1 to 6. By increasing the number of stages up to 6, we get finer results. However, the average execution time of 30 runs is 48229 (ms), and the standard deviation is 14095, when number of stages equals to 6. Table 7.4 presents three different categories of results for 616 web services and varying number of stages from 1 to 6. For example, the average execution time of 10 runs is 6025 (ms) and the standard deviation is 405, when number of stages equals to 3. The following tables (7.5, 7.6, 7.7, and 7.8) present three different categories of results, taking into consideration 940, 129, 349, and 473 web services respectively.

Generally, when Java code is started, Java Virtual Memory (JVM) reserves a part of the machines memory. And part of this memory is reserved for Java heap memory, in which objects are created. Figure 7.4 and 7.5 illustrate CloudMTD memory usage results for selection and composition respectively. Performance test was carried out according to the following setting criteria: (1) Settings Name: Analyze Performance, (2) Profiling Type: CPU Profiling (Sampling Application), (3) CPU Profiling Type: Sampled, (4) Instrumentation Filter: Profile only project classes. Memory usage test was carried out based on the following setting criteria: (1) Settings Name: Analyze Memory, (2) Profiling Type: Memory (Sampling), and (3) Run Garbage Collection When Getting Results: Yes.

Table 7.2: CloudMTD methods description.

Method	Implementation	CloudMTD
getServicesQoS	Imports services' QoS Data from the database.	Query the cloud marketplace and aggregate the QoS of each candidate service.
connect	Establishes MySQL database connection.	Establishes a connection to market data which is published.
disconnect	Terminates MySQL database connection.	Terminates the connection.
getDeltaFunction	Calculates the difference among specific QoS attributes	Calculates the improvement in specific QoS value (ΔV_{qos_n})
tdCalculation	Calculates the technical debt.	Technical Debt quantification.
myOptionPrice	Calculates options values based on Binomial Option Pricing.	Option calculation.
getTravelInfo	Connects to the WS-BPEL workflow and returns the needed information regarding travel: flight dates, hotel info, etc.	Services Composition.

Figure 7.4 CloudMTD memory usage results for selection phase based on 2507 web services and 6 stages.

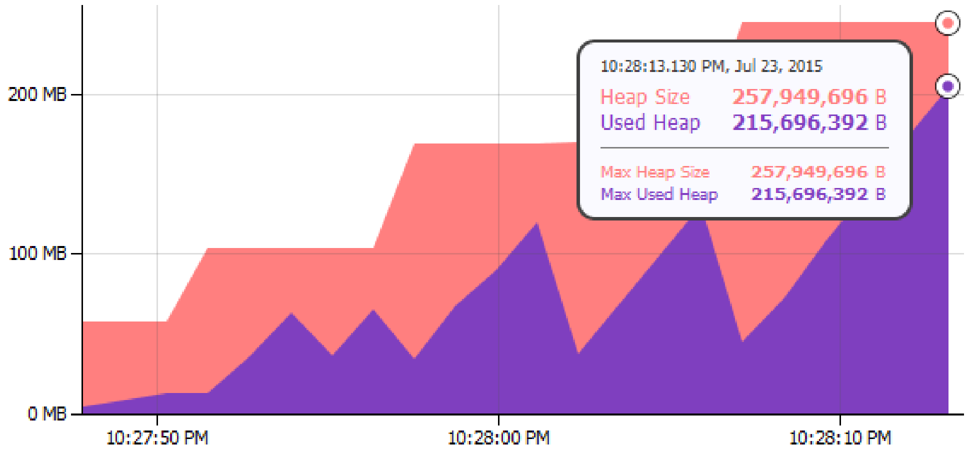


Figure 7.5 CloudMTD memory usage results for composition phase.

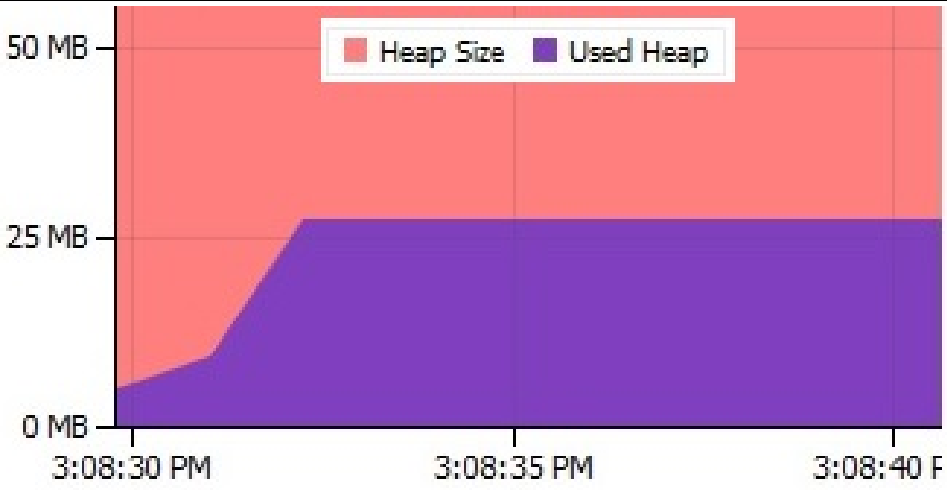
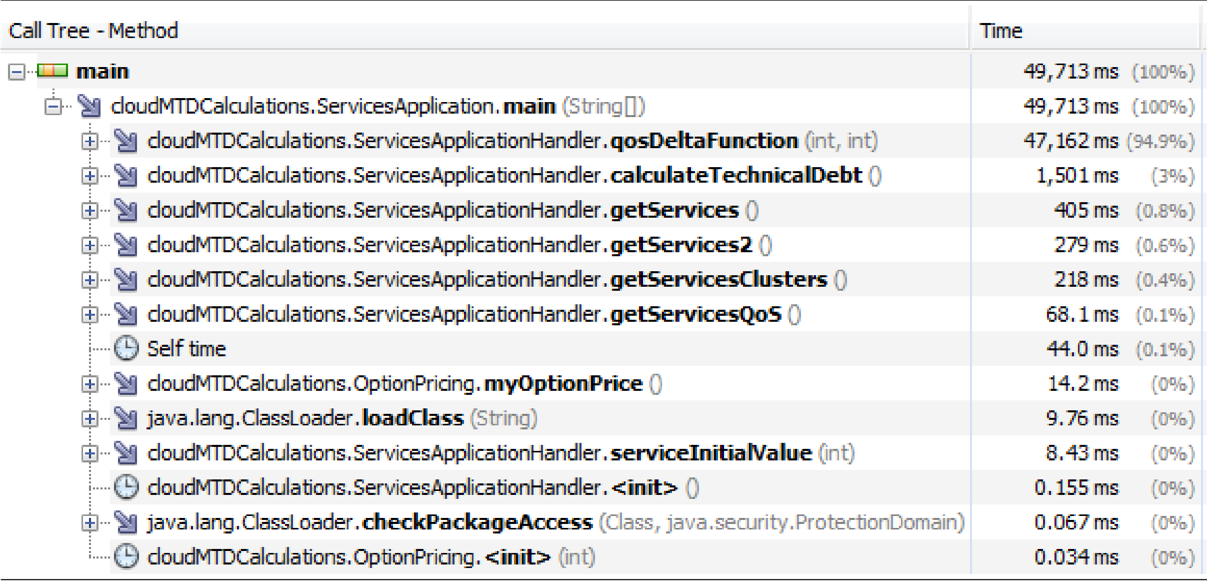
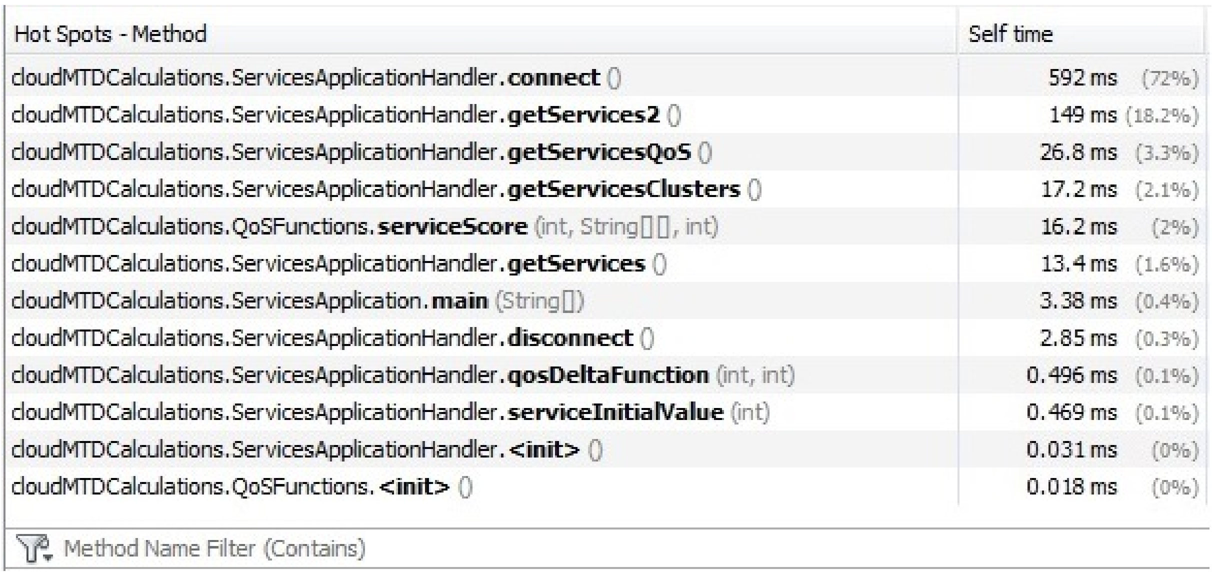


Figure 7.6 CloudMTD performance test results for methods related to the selection phase.



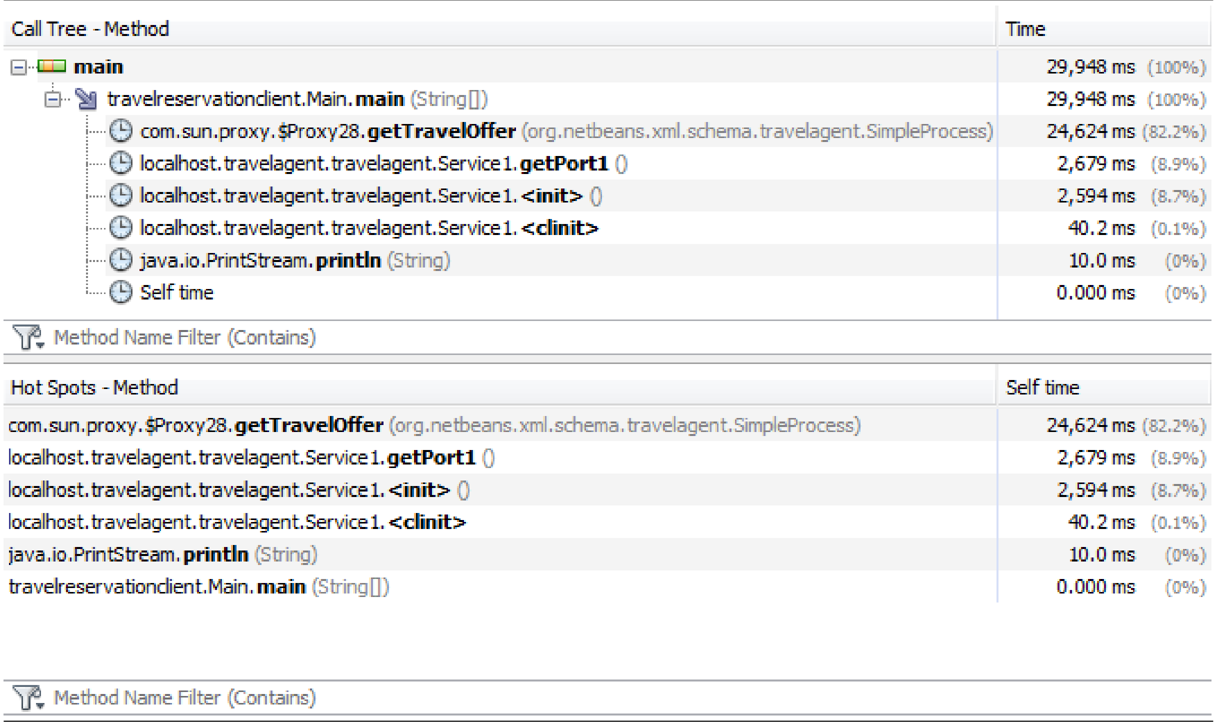
Call Tree - Method	Time
main	49,713 ms (100%)
cloudMTDCalculations.ServicesApplication.**main** (String[])	49,713 ms (100%)
cloudMTDCalculations.ServicesApplicationHandler.**qosDeltaFunction** (int, int)	47,162 ms (94.9%)
cloudMTDCalculations.ServicesApplicationHandler.**calculateTechnicalDebt** ()	1,501 ms (3%)
cloudMTDCalculations.ServicesApplicationHandler.**getServices** ()	405 ms (0.8%)
cloudMTDCalculations.ServicesApplicationHandler.**getServices2** ()	279 ms (0.6%)
cloudMTDCalculations.ServicesApplicationHandler.**getServicesClusters** ()	218 ms (0.4%)
cloudMTDCalculations.ServicesApplicationHandler.**getServicesQoS** ()	68.1 ms (0.1%)
Self time	44.0 ms (0.1%)
cloudMTDCalculations.OptionPricing.**myOptionPrice** ()	14.2 ms (0%)
java.lang.ClassLoader.**loadClass** (String)	9.76 ms (0%)
cloudMTDCalculations.ServicesApplicationHandler.**serviceInitialValue** (int)	8.43 ms (0%)
cloudMTDCalculations.ServicesApplicationHandler.**<init>** ()	0.155 ms (0%)
java.lang.ClassLoader.**checkPackageAccess** (Class, java.security.ProtectionDomain)	0.067 ms (0%)
cloudMTDCalculations.OptionPricing.**<init>** (int)	0.034 ms (0%)

Figure 7.7 CloudMTD performance test results for main methods.



Hot Spots - Method	Self time	
cloudMTDCalculations.ServicesApplicationHandler.**connect** ()	592 ms	(72%)
cloudMTDCalculations.ServicesApplicationHandler.**getServices2** ()	149 ms	(18.2%)
cloudMTDCalculations.ServicesApplicationHandler.**getServicesQoS** ()	26.8 ms	(3.3%)
cloudMTDCalculations.ServicesApplicationHandler.**getServicesClusters** ()	17.2 ms	(2.1%)
cloudMTDCalculations.QoSFunctions.**serviceScore** (int, String[][], int)	16.2 ms	(2%)
cloudMTDCalculations.ServicesApplicationHandler.**getServices** ()	13.4 ms	(1.6%)
cloudMTDCalculations.ServicesApplication.**main** (String[])	3.38 ms	(0.4%)
cloudMTDCalculations.ServicesApplicationHandler.**disconnect** ()	2.85 ms	(0.3%)
cloudMTDCalculations.ServicesApplicationHandler.**qosDeltaFunction** (int, int)	0.496 ms	(0.1%)
cloudMTDCalculations.ServicesApplicationHandler.**serviceInitialValue** (int)	0.469 ms	(0.1%)
cloudMTDCalculations.ServicesApplicationHandler.**<init>** ()	0.031 ms	(0%)
cloudMTDCalculations.QoSFunctions.**<init>** ()	0.018 ms	(0%)

Figure 7.8 CloudMTD performance test results for methods related to the composition phase.



```
Call Tree - Method                                                                              Time
⊟ main                                                                                          29,948 ms  (100%)
  ⊟ travelreservationclient.Main.main (String[])                                                29,948 ms  (100%)
     ├ com.sun.proxy.$Proxy28.getTravelOffer (org.netbeans.xml.schema.travelagent.SimpleProcess) 24,624 ms (82.2%)
     ├ localhost.travelagent.travelagent.Service1.getPort1 ()                                     2,679 ms  (8.9%)
     ├ localhost.travelagent.travelagent.Service1.<init> ()                                       2,594 ms  (8.7%)
     ├ localhost.travelagent.travelagent.Service1.<clinit>                                         40.2 ms  (0.1%)
     ├ java.io.PrintStream.println (String)                                                        10.0 ms    (0%)
     └ Self time                                                                                  0.000 ms    (0%)
```

Hot Spots - Method	Self time	
com.sun.proxy.$Proxy28.**getTravelOffer** (org.netbeans.xml.schema.travelagent.SimpleProcess)	24,624 ms	(82.2%)
localhost.travelagent.travelagent.Service1.**getPort1** ()	2,679 ms	(8.9%)
localhost.travelagent.travelagent.Service1.**<init>** ()	2,594 ms	(8.7%)
localhost.travelagent.travelagent.Service1.**<clinit>**	40.2 ms	(0.1%)
java.io.PrintStream.**println** (String)	10.0 ms	(0%)
travelreservationclient.Main.**main** (String[])	0.000 ms	(0%)

Table 7.3: CloudMTD performance test results based on different stages for calculating options and technical debt

Test Case	Run	QoS	Stages	Services	Execution Time (ms)
TC1	1	4	6	2507	38545
TC2	2	4	6	2507	35175
TC3	3	4	6	2507	32963
TC4	4	4	6	2507	56776
TC5	5	4	6	2507	42539
TC6	6	4	6	2507	64655
TC7	7	4	6	2507	62502
TC8	8	4	6	2507	51541
TC9	9	4	6	2507	79234
TC10	10	4	6	2507	77475
TC11	11	4	6	2507	46827
TC12	12	4	6	2507	37787
TC13	13	4	6	2507	40244
TC14	14	4	6	2507	37285
TC15	15	4	6	2507	57479
TC16	16	4	6	2507	65330
TC17	17	4	6	2507	63339
TC18	18	4	6	2507	39260
TC19	19	4	6	2507	49050
TC20	20	4	6	2507	49713
TC21	21	4	6	2507	38436
TC22	22	4	6	2507	25720
TC23	23	4	6	2507	55447
TC24	24	4	6	2507	40292
TC25	25	4	6	2507	64044
TC26	26	4	6	2507	68231
TC27	27	4	6	2507	44398
TC28	28	4	6	2507	29548
TC29	29	4	6	2507	53049
TC30	30	4	6	2507	41123
				Average	48229
				Std Dev	14095
TC31	1	4	3	2507	16538
TC32	2	4	3	2507	14260
TC33	3	4	3	2507	13885
TC34	4	4	3	2507	16407
TC35	5	4	3	2507	15692
TC36	6	4	3	2507	14397
TC37	7	4	3	2507	14877
TC38	8	4	3	2507	13476
TC39	9	4	3	2507	14531
TC40	10	4	3	2507	14622
				Average	14847
				Std Dev	1114
TC41	1	4	1	2507	11242
TC42	2	4	1	2507	6870
TC43	3	4	1	2507	7105
TC44	4	4	1	2507	7516
TC45	5	4	1	2507	7806
TC46	6	4	1	2507	6775
TC47	7	4	1	2507	7050
TC48	8	4	1	2507	7616
TC49	9	4	1	2507	9875
TC50	10	4	1	2507	7098
				Average	7895
				Std Dev	1477

Table 7.4: CloudMTD performance test results based on different stages for calculating options and technical debt

Test Case	Run	Stages	Services	Execution Time (ms)	Test Case	Run	Stages	Services	Execution Time (ms)	Test Case	Run	Stages	Services	Execution Time (ms)
TC51	1	6	616	13358	TC61	1	3	616	5292	TC71	1	1	616	3216
TC52	2	6	616	9791	TC62	2	3	616	6161	TC72	2	1	616	3567
TC53	3	6	616	13304	TC63	3	3	616	5974	TC73	3	1	616	3476
TC54	4	6	616	10948	TC64	4	3	616	5860	TC74	4	1	616	3961
TC55	5	6	616	9030	TC65	5	3	616	5730	TC75	5	1	616	3734
TC56	6	6	616	10946	TC66	6	3	616	6477	TC76	6	1	616	3312
TC57	7	6	616	10327	TC67	7	3	616	6509	TC77	7	1	616	3547
TC58	8	6	616	13314	TC68	8	3	616	5952	TC78	8	1	616	3342
TC59	9	6	616	11618	TC69	9	3	616	6556	TC79	9	1	616	3596
TC60	10	6	616	8655	TC70	10	3	616	5743	TC80	10	1	616	3336
			Average	11129				Average	6025.4				Average	3509
			Std Dev	1756				Std Dev	405.245				Std Dev	224

Table 7.5: CloudMTD performance test results based on different stages for calculating options and technical debt

Test Case	Run	Stages	Services	Execution Time (ms)	Test Case	Run	Stages	Services	Execution Time (ms)	Test Case	Run	Stages	Services	Execution Time (ms)
TC81	1	6	940	22573	TC91	1	3	940	8055	TC101	1	1	940	4781
TC82	2	6	940	19610	TC92	2	3	940	6788	TC102	2	1	940	4504
TC83	3	6	940	14516	TC93	3	3	940	7220	TC103	3	1	940	4143
TC84	4	6	940	14408	TC94	4	3	940	7146	TC104	4	1	940	4118
TC85	5	6	940	18512	TC95	5	3	940	7709	TC105	5	1	940	4316
TC86	6	6	940	18346	TC96	6	3	940	8276	TC106	6	1	940	3921
TC87	7	6	940	15597	TC97	7	3	940	6749	TC107	7	1	940	4496
TC88	8	6	940	12087	TC98	8	3	940	7348	TC108	8	1	940	4720
TC89	9	6	940	13992	TC99	9	3	940	8871	TC109	9	1	940	4825
TC90	10	6	940	18284	TC100	10	3	940	7574	TC110	10	1	940	4410
			Average	16793				Average	7573.6				Average	4423.4
			Std Dev	3184				Std Dev	673.514				Std Dev	302.749

Table 7.6: CloudMTD performance test results based on different stages for calculating options and technical debt

Test Case	Run	Stages	Services	Execution Time (ms)	Test Case	Run	Stages	Services	Execution Time (ms)	Test Case	Run	Stages	Services	Execution Time (ms)
TC111	1	6	129	4724	TC121	1	3	129	2780	TC131	1	1	129	1424
TC112	2	6	129	4313	TC122	2	3	129	3193	TC132	2	1	129	1445
TC113	3	6	129	4451	TC123	3	3	129	3180	TC133	3	1	129	1388
TC114	4	6	129	4415	TC124	4	3	129	2813	TC134	4	1	129	1350
TC115	5	6	129	5609	TC125	5	3	129	2482	TC135	5	1	129	1501
TC116	6	6	129	6041	TC126	6	3	129	2936	TC136	6	1	129	1509
TC117	7	6	129	5109	TC127	7	3	129	2962	TC137	7	1	129	1358
TC118	8	6	129	5063	TC128	8	3	129	2834	TC138	8	1	129	1602
TC119	9	6	129	4613	TC129	9	3	129	2773	TC139	9	1	129	1531
TC120	10	6	129	5315	TC130	10	3	129	2748	TC140	10	1	129	1282
			Average	4965				Average	2870.1				Average	1439
			Std Dev	568				Std Dev	211.011				Std Dev	97.6217

Table 7.7: CloudMTD performance test results based on different stages for calculating options and technical debt

Test Case	Run	Stages	Services	Execution Time (ms)	Test Case	Run	Stages	Services	Execution Time (ms)	Test Case	Run	Stages	Services	Execution Time (ms)
TC141	1	6	349	7787	TC151	1	3	349	4344	TC161	1	1	349	3003
TC142	2	6	349	8866	TC152	2	3	349	3735	TC162	2	1	349	2972
TC143	3	6	349	7658	TC153	3	3	349	4372	TC163	3	1	349	2844
TC144	4	6	349	7625	TC154	4	3	349	4359	TC164	4	1	349	2723
TC145	5	6	349	7624	TC155	5	3	349	4216	TC165	5	1	349	2897
TC146	6	6	349	8692	TC156	6	3	349	4065	TC166	6	1	349	2991
TC147	7	6	349	6361	TC157	7	3	349	4136	TC167	7	1	349	2830
TC148	8	6	349	8413	TC158	8	3	349	3872	TC168	8	1	349	3073
TC149	9	6	349	7894	TC159	9	3	349	4033	TC169	9	1	349	2938
TC150	10	6	349	6665	TC160	10	3	349	3660	TC170	10	1	349	2969
			Average	7759				Average	4079.2				Average	2924
			Std Dev	799				Std Dev	257.24				Std Dev	102

Table 7.8: CloudMTD performance test results based on different stages for calculating options and technical debt

Test Case	Run	Stages	Services	Execution Time (ms)	Test Case	Run	Stages	Services	Execution Time (ms)	Test Case	Run	Stages	Services	Execution Time (ms)
TC171	1	6	473	11121	TC181	1	3	473	4712	TC191	1	1	473	3249
TC172	2	6	473	9384	TC182	2	3	473	4611	TC192	2	1	473	3137
TC173	3	6	473	12029	TC183	3	3	473	4799	TC193	3	1	473	3176
TC174	4	6	473	10057	TC184	4	3	473	4524	TC194	4	1	473	3234
TC175	5	6	473	9033	TC185	5	3	473	4756	TC195	5	1	473	3215
TC176	6	6	473	7880	TC186	6	3	473	4473	TC196	6	1	473	3161
TC177	7	6	473	10842	TC187	7	3	473	4429	TC197	7	1	473	3128
TC178	8	6	473	7569	TC188	8	3	473	4257	TC198	8	1	473	3139
TC179	9	6	473	10946	TC189	9	3	473	4544	TC199	9	1	473	3319
TC180	10	6	473	10533	TC190	10	3	473	4578	TC200	10	1	473	3164
			Average	9939				Average	4568.3				Average	3192.2
			Std Dev	1451				Std Dev	162.948				Std Dev	61.1025

7.7 Evaluation from CloudMTD Perspective

7.7.1 K-means Clustering

By the use of k-means, we are evaluating the extent to which a clustering technique can improve service selection process. K-means is known to be an NP-hard problem [70]. In addition, based on the literature services selection and composition is known to be an NP-hard problem [235]. In fact, it is possible to nominate all available services for value creation; however we decided to introduce web services clustering in order to reduce the search space. Accordingly, by filtering services we get a smaller space, and consequently it will be easier to consider the context of services composition. We are interested in three aspects when using the k-means clustering:

1. Improving Scalability. The used of k-means clustering improves the scalability by reducing the search space of candidate web services. Otherwise, the selection process and the related modeling can get complex in case if we model every single web service in the marketplace.

2. Improving Relevancy. The used of k-means clustering improves the relevancy by fetching the relevant cluster only instead of searching the whole dataset (the whole marketplace).

3. QoS correlation. The use of clustering makes QoS correlation explicit by analyzing data and finding structures in the given dataset.

7.7.2 Binomial Model

As mentioned in previous chapters, because of uncertainty, we have chosen Real Options theory as it is an ultimate choice to be used as a valuation technique, since other valuation techniques has some shortages in dealing with the value of flexibility under uncertainty, i.e. do not account for the value of flexibility. The Binomial model is one of the approaches that is used in option pricing to model uncertainty [77]. There are some goals that should be fulfilled during the evaluation procedure when using the Binomial Option model in order to address the web service selection problem. These goals must be fulfilled based on a specific scenario that is linked to a suitable option type. The following summarizes the main goals:

7.7.2.1 Staging-View

One of the reasons of why we have chosen the binomial option pricing is that because it allows the "staging-view". What is meant by the "staging-view" is that we can analyze and interpret the results during different time periods (intervals) of time, for example month by month or year by year depends on needs and contracts length, which is described in SLA.

7.7.2.2 Multiple-Decision Points

Because of the dynamic nature of the selection problem that is due to different dimensions of uncertainty that were discussed previously (e.g. continuous fluctuation of QoS, needs and requirements), there is a continuous need for a step-by-step assessment and judgment that allows taking dynamic decisions (continuous assessment). Binomial analysis fits the need of a dynamic and continuous assessment by presenting many decision points during a specific time-period. In addition, there is a need for a technique that predicts and keeps track of possible values during this time-period. Each point in the tree expresses either an improvement or a degradation of the value. Here, we are evaluating the extent to which CloudMTD model is capable of facilitating the what-if analysis at different points in the decision space.

7.7.2.3 Transparency: Value and Debt Visualization

Binomial Tree allows the decision maker to observe the change in value and debt at each time-period in the tree. In this sense, CB-SOA decision makers will be able to visualize the value and debt over different time periods. On the other hand, Binomial Tree is used for technical debt lifecycle visualization: (1) when it starts, (2) when interest are incurred, if any, (3) Payback; when the technical debt is expected to be cleared out and/or when it is likely to generate future options if cleared.

7.7.3 Dependency Tracking

CloudMTD model tracks the dependencies among constituent services in CB-SOA based on: (1) Dependency Structure Matrix (DSM), (2) Visibility Matrix, and (3) Propagation-cost. Here, we evaluate the extent to which substituting a particular service has an impact on other services in a given architecture.

7.8 Evaluation from OptiViTA Perspective

We have used a case study in order to evaluate the applicability of the CloudMTD model on a real architecture. As well as to answer some research questions those are related to CloudMTD model such as, how CloudMTD model can be used? What sort of decisions can be taken using CloudMTD model? Furthermore, the case study is used to empirically evaluate and assess the CloudMTD model in dealing with the behavioral and structural aspects of a given architecture. In that sense, we state that case study is the most appropriate and suitable research method to be used for answering the research questions that are related to CloudMTD model. Next we list the objectives and aims of using OptiViTA case study in the evaluation:

7.8.1 The Applicability of CloudMTD Model

OptiViTA case study is used for demonstrating the CloudMTD model's applicability. In particular, the case study is used to proof that CloudMTD model is practically suitable and applicable to the problem of web service selection and composition in cloud-based service-oriented architectures. In addition, OptiViTA case study is used in order to demonstrate how CloudMTD model can inform the selection/substitution decision of a web service in CB-SOA for a given scenario, with respect to value-added, technical debt, and cost.

7.8.2 The Worthiness of the Decision

OptiViTA case study is used for evaluating the worthiness of the substitution/selection decision. In particular, the case study is used in order to assess the worthiness of the reconfiguration i.e. whether if it is valuable to reconfigure the architecture or not. Here we are concerned about three situations: 1. keeping the current web service, 2. substituting the current web service by another one, which is said to be more suitable in terms of value-added, technical debt and cost, and 3. abandon the current web service.

7.8.3 Decision Time-Value

OptiViTA case study is used for expressing the time-value of the substitution/selection decision with respect to future change. CloudMTD is used to quantify the time-value of the substitution/selection decision. For example, if the case is to switch, CloudMTD informs when it is beneficial to switch.

7.9 Behavioral Evaluation: Selecting a Web Service

In this part of the evaluation process, we evaluate the extent to which CloudMTD model can assist in selecting "good enough" (qualified) services which tend to meet the behavioral requirements of the system. In addition, we test the scalability of the technique on a large dataset that consists of real web services, representing the marketplace. We consider both the number of services and the QoS attributes as measure of scalability.

7.9.1 K-means clustering

The clustering hypothesis is that web services in the same cluster behave similarly with respect to QoS attributes. In this section, we present the clustering evaluation process and discuss how clustering will improve the scalability and the relevancy of the data. In addition, we present and analyze QoS correlation.

7.9.2 K-means Results and Discussion

A given web service dataset consist of more than one dimension, as each web service has many attributes. In this context, visualizing multidimensional datasets is complex. However, K-means clustering technique is capable of presenting clustered data in a structured way that enables the decision maker to investigate "groups' structure" in the given dataset.

As mentioned in the previous chapter, the Silhouette plot is used for the purpose of testing the strength of a given cluster in the dataset, i.e. Silhouette plot reveals how well is each element clustered in a given dataset. Silhouette Coefficients lies somewhere between -1 and 1, where 1 represents well-clustered data and -1 represents misclassified data (chapter 6, section 6.1.2). Figure 7.9 represents Silhouette plots for the QWS dataset from K=2 till K=8. Figure 7.10 represents a Silhouette plot for K = 5, i.e. clustering the web services into five clusters.

For more quantitative method of comparing different Silhouette plots, let us consider the Average Silhouette Values (ASV) presented in Table 7.9. ASV is calculated by computing the means for each Silhouette plot. The interpretation of the results of K-means and Silhouette values depends on the decision maker intention. For example, if we consider K = 2, the ASV is the highest, however, clustering web services into two different categories is not enough and impractical. In addition, K = 5 was the only Silhouette plot with no negative Silhouette values and accordingly with least outliers of services.

Figure 7.9 Silhouettes results from K=2 till K=8

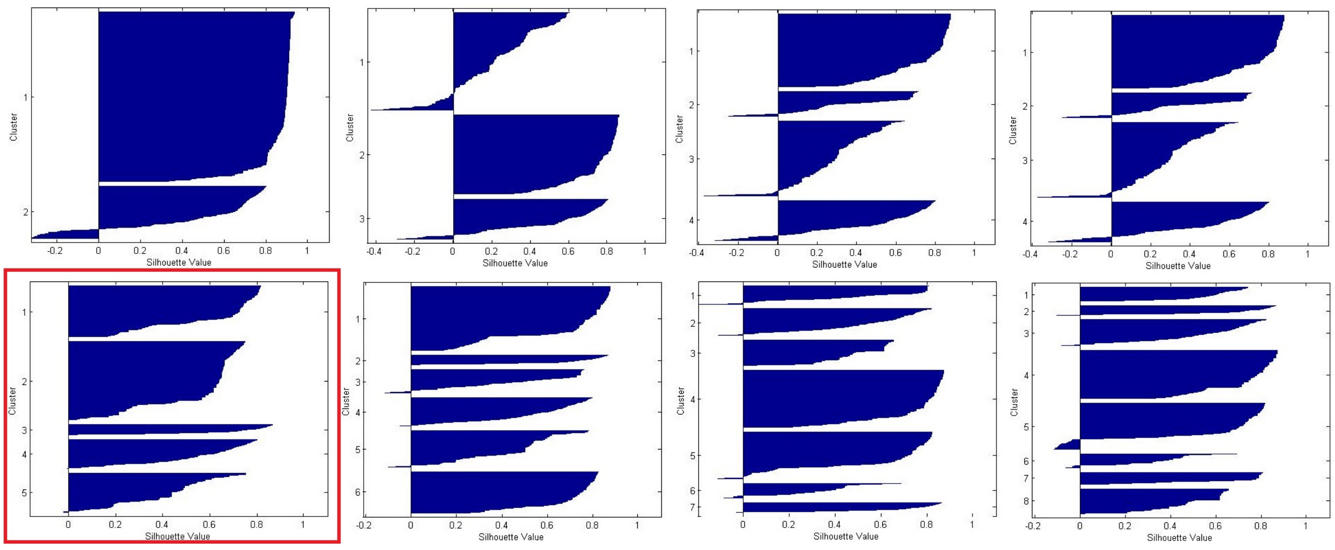
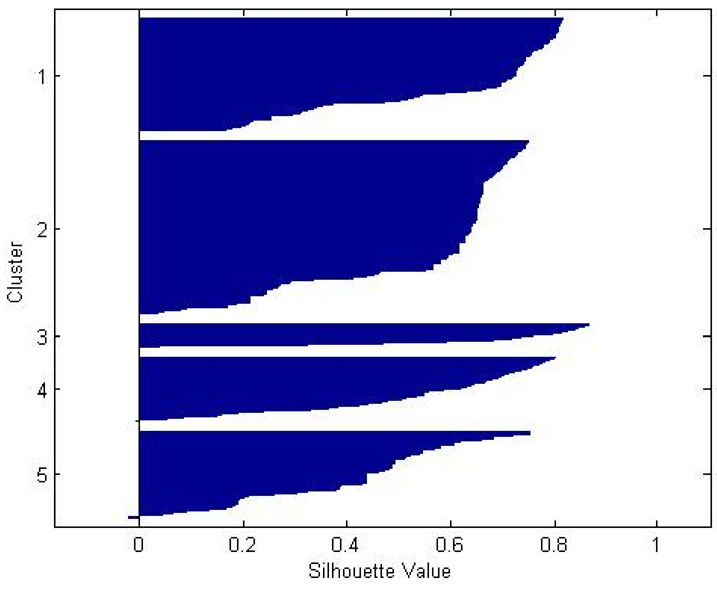


Table 7.9: Average Silhouette Values (ASV).

K	ASV
2	0.7626
3	0.4565
4	0.4758
5	0.5455
6	0.5864
7	0.5825
8	0.5713

Figure 7.10 Silhouettes results for K=5



Before analyzing the clustering results, we would like to present the original candidate dataset distribution before clustering technique is applied. This is for the purpose of presenting all available values for each attribute of a given web service in the dataset. Figure 7.11 represents the histograms of the candidate dataset. Figure 7.12 represents the pie charts of the candidate dataset.

Figure 7.11 The candidate dataset histograms of 2507 web services and 4 QoS attributes.

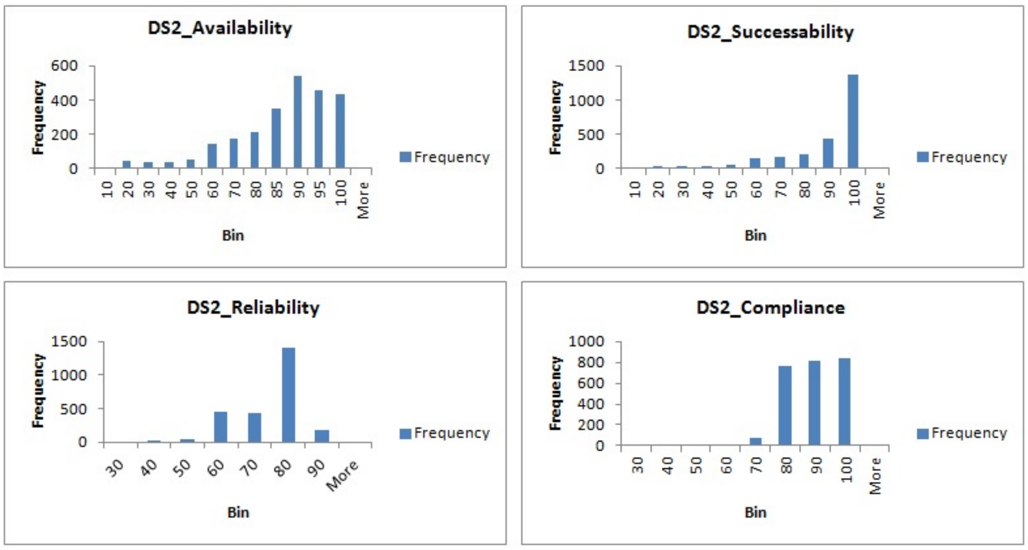


Figure 7.12 The candidate dataset pie charts of 2507 web services and 4 QoS attributes.

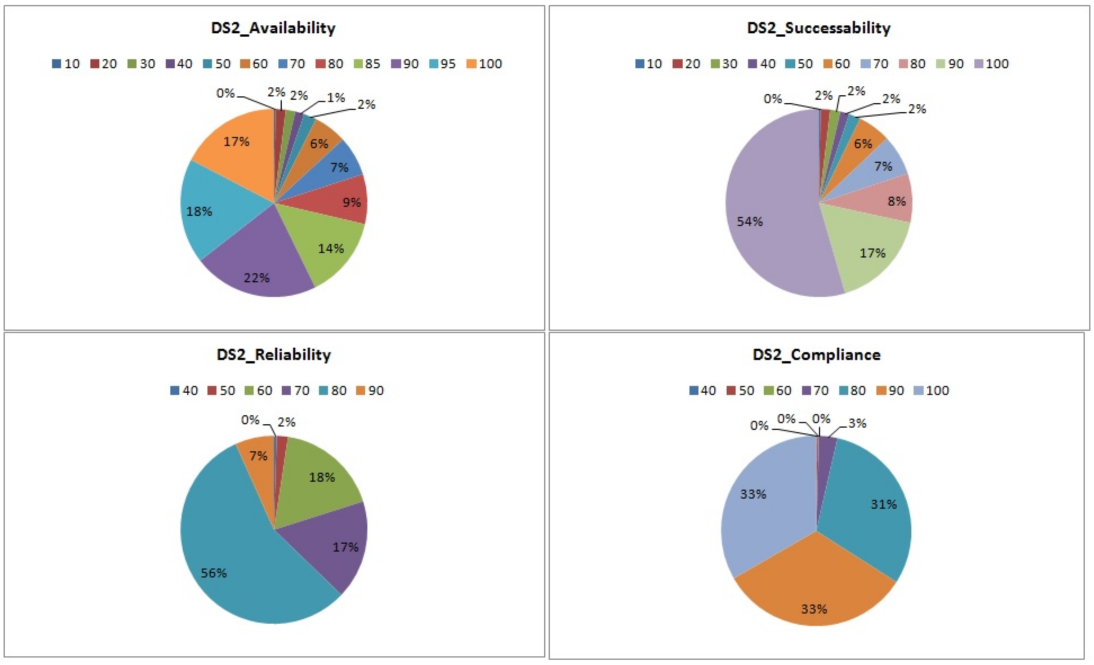
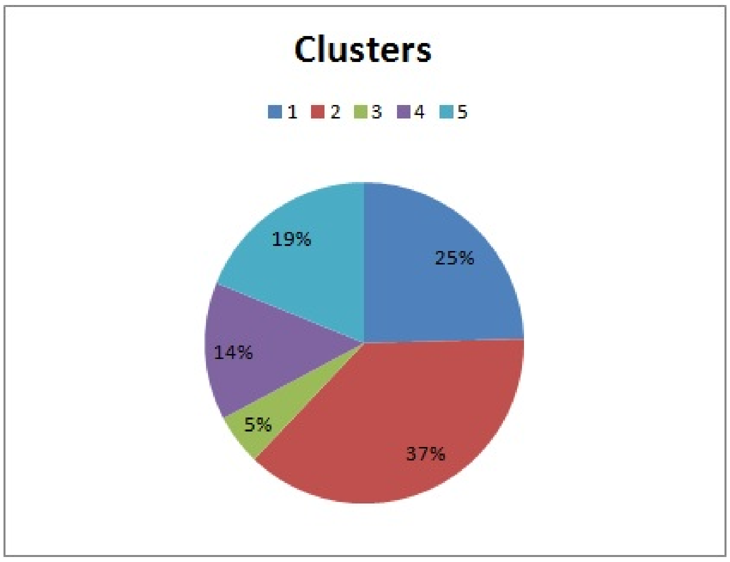


Table 7.10 represents the number of data records in each cluster when applying K-means for K = 5, five clusters. This information is needed in different aspect. For example, it will be used for calculating each cluster quality. Figure 7.13 present the pie charts of the clusters when k = 5.

Table 7.10: Number of data records in each cluster.

cluster	records
1	616
2	940
3	129
4	349
5	473
total	2507

Figure 7.13 Clusters Pie Charts (web services distribution of K = 5)



The following figures present histograms and pie charts of each QoS attributes in each cluster when k=5 (from Figure 7.14 to Figure 7.23). These figures were used for analyzing web services clusters distribution and QoS correlations in each cluster. For example, figure 7.17 and figure 7.16 represent second cluster pie charts and histograms. If we consider the availability attribute, for example, we can conclude out of these figures that 27% of web services in cluster number 2 has availability values of the range 96-100. This 27% represents $\frac{254}{435} = 58\%$ of the total number of web services with availability range 96-100 in the dataset. We found that cluster number 2 contains high values of each QoS attribute. By looking at the figures and table 7.11, we find that cluster number 2 consists of the following: (1) Availability: Excellent and Very Good values, (2) Successability: Excellent Values. (3) Reliability: Excellent and Very Good values, and (4) Compliance: Excellent values. In this sense, if the decision maker is interested in web service with high values, s/he can choose cluster number 2 instead of searching the whole marketplace. However, cluster number 3 contains either very bad values or unstructured one, and accordingly this cluster can be neglected. If the decision maker is interested in moderate data, s/he can have a look at cluster number 1, where QoS data are almost of Good values. Table 7.11 presents a full description of QoS correlations in each cluster when k=5. Tables 7.12, 7.13, and 7.14 present ranges descriptions of each QoS attributes.

Table 7.11: Clusters Themes based on QoS Correlations

Cluster	Availability	Successability	Reliability	Compliance
1	Good	Excellent	Good	Good
2	Excellent and Very Good	Excellent	Excellent and Very Good	Excellent
3	Very Bad	Very Bad	No Theme	No Theme
4	Very Low and Bad	Acceptable and Low	Good and Bad	Very Good
5	Acceptable and Low	Very Good and Good	Bad	Very Good

Table 7.12: Availability Theme

Availability	Theme
96 - 100	Excellent
91 - 95	Very Good
86 - 90	Good
81 - 85	Acceptable
71 - 80	Low
61 - 70	Very Low
51 - 60	Bad
$\leq$ 50	Very Bad

Table 7.13: Reliability Theme

Reliability	Theme
$\leq$ 60	Excellent
61 - 70	Very Good
71 - 80	Good
81 - 90	Bad

Table 7.14: Other QoSs Theme

Other QoS	Theme
91 - 100	Excellent
81 - 90	Very Good
71 - 80	Good
61 - 70	Acceptable
51- 60	Low
$\leq$ 50	Very Bad

Figure 7.14 First cluster histograms of 4 QoS attributes.

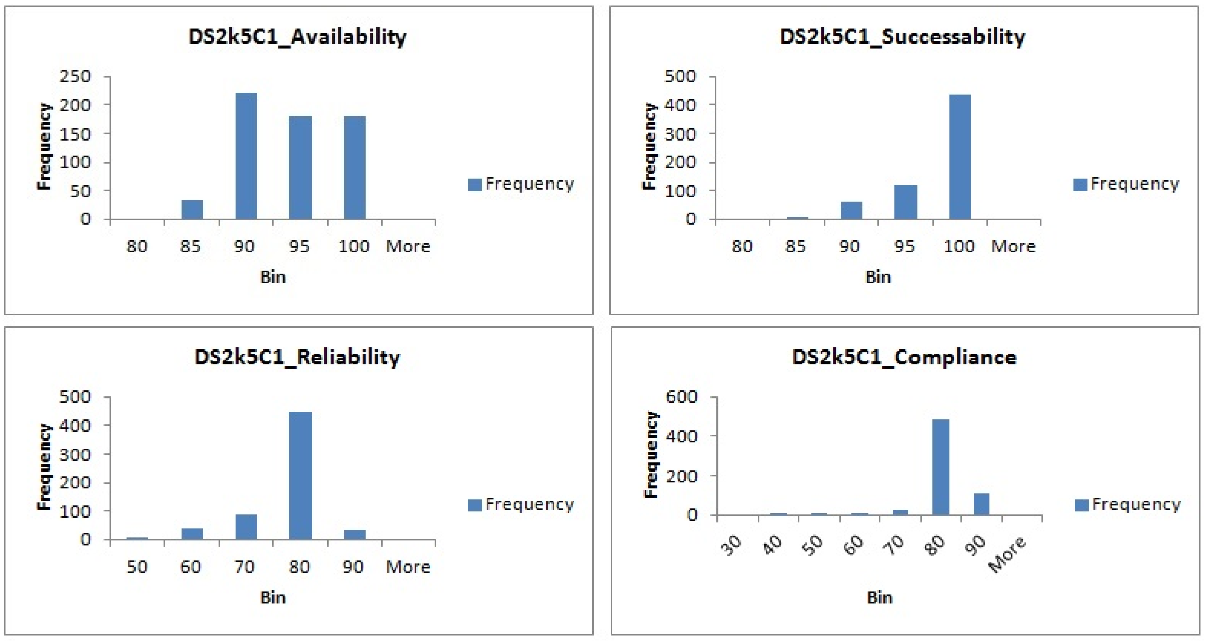


Figure 7.15 First cluster pie charts of 4 QoS attributes.

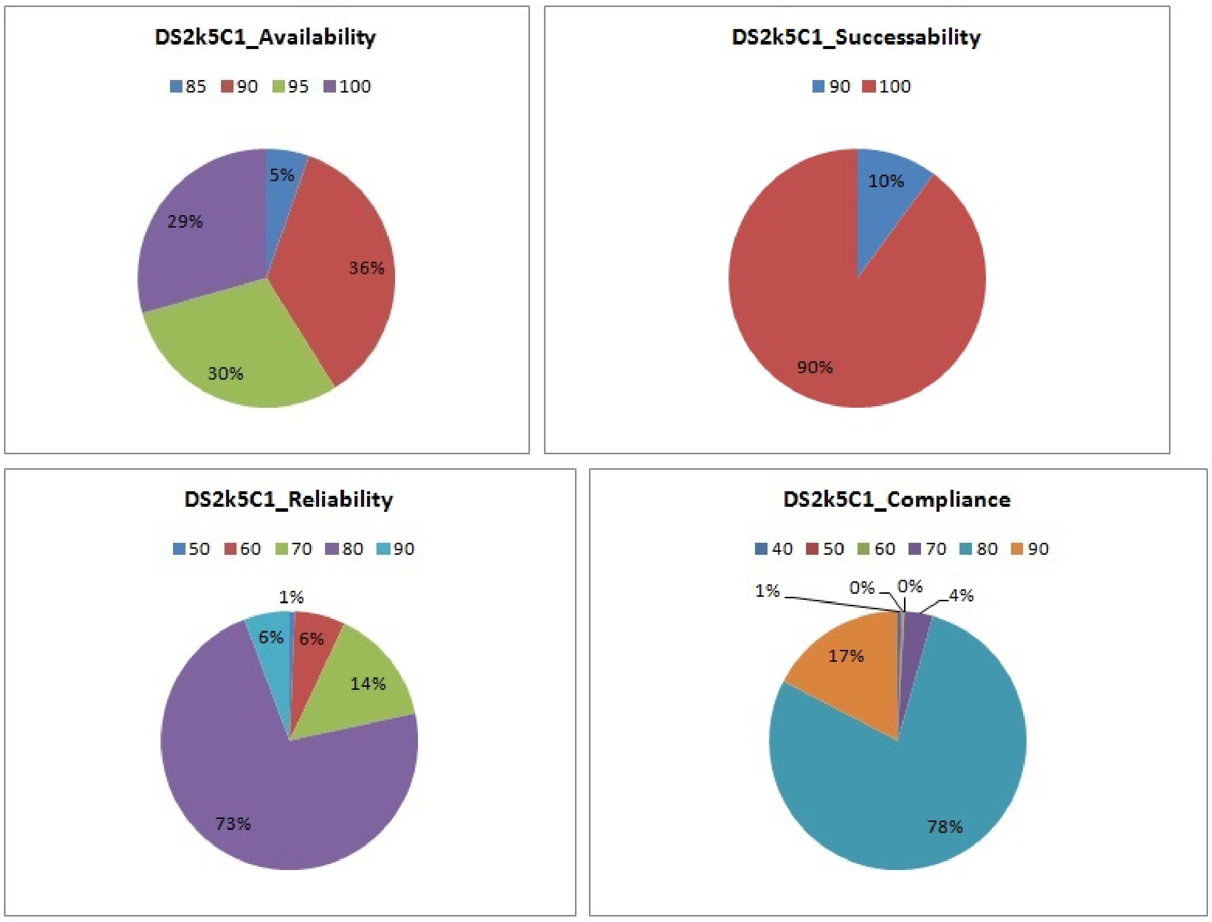


Figure 7.16 Second cluster histograms of 4 QoS attributes.

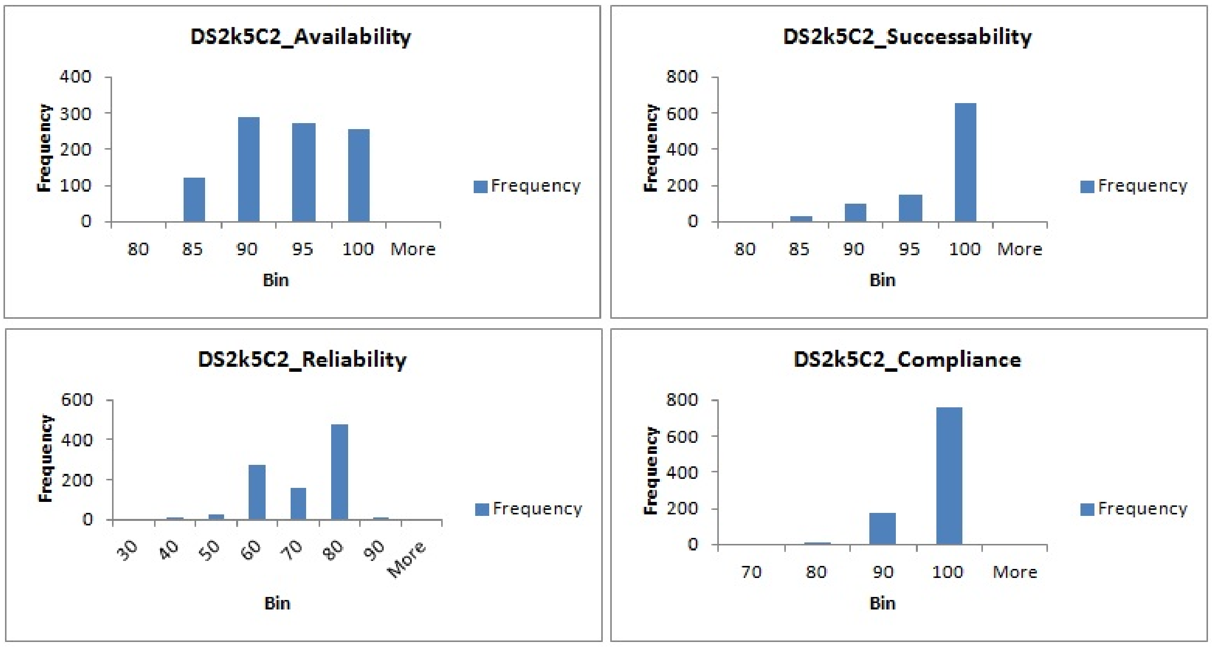


Figure 7.17 Second cluster pie charts of 4 QoS attributes.

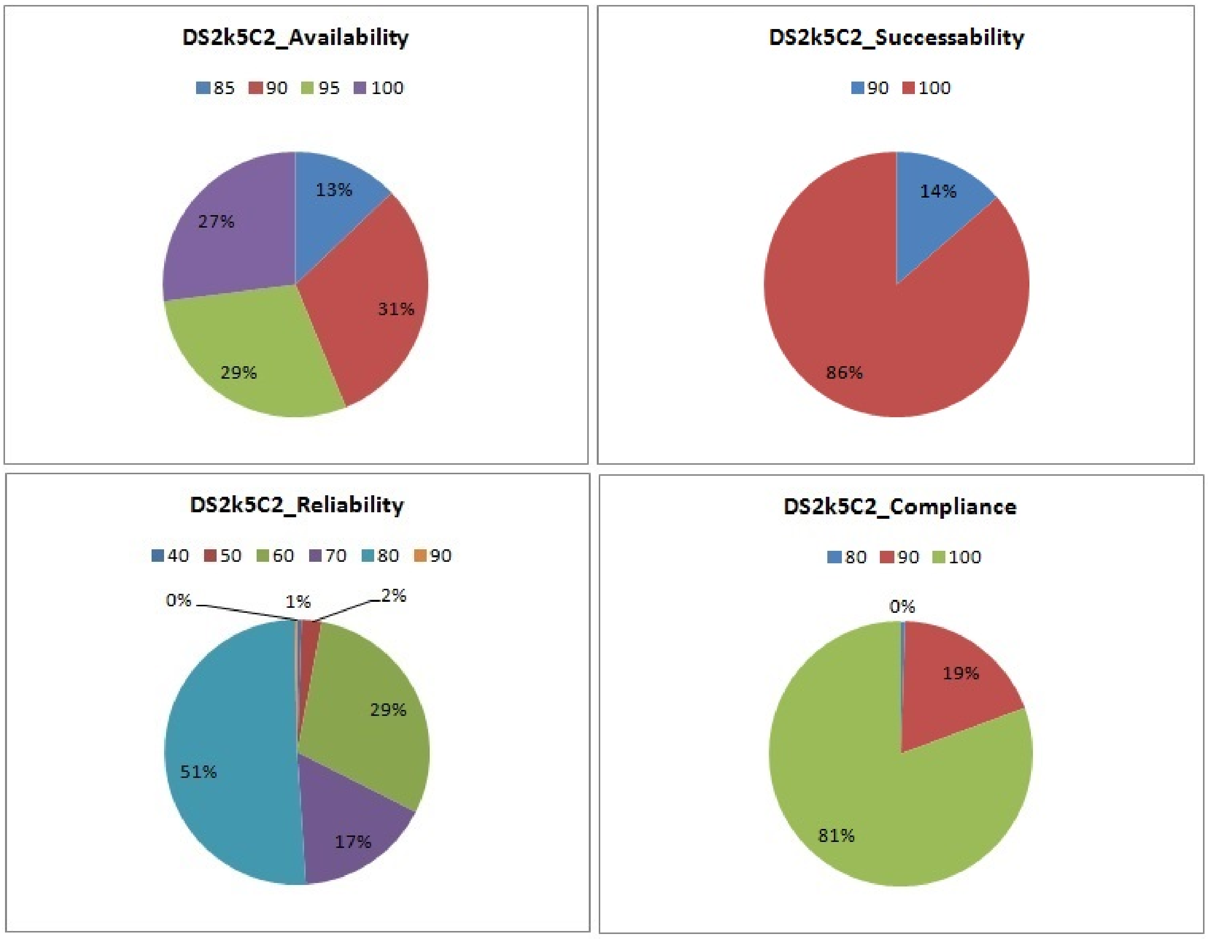


Figure 7.18 Third cluster histograms of 4 QoS attributes.

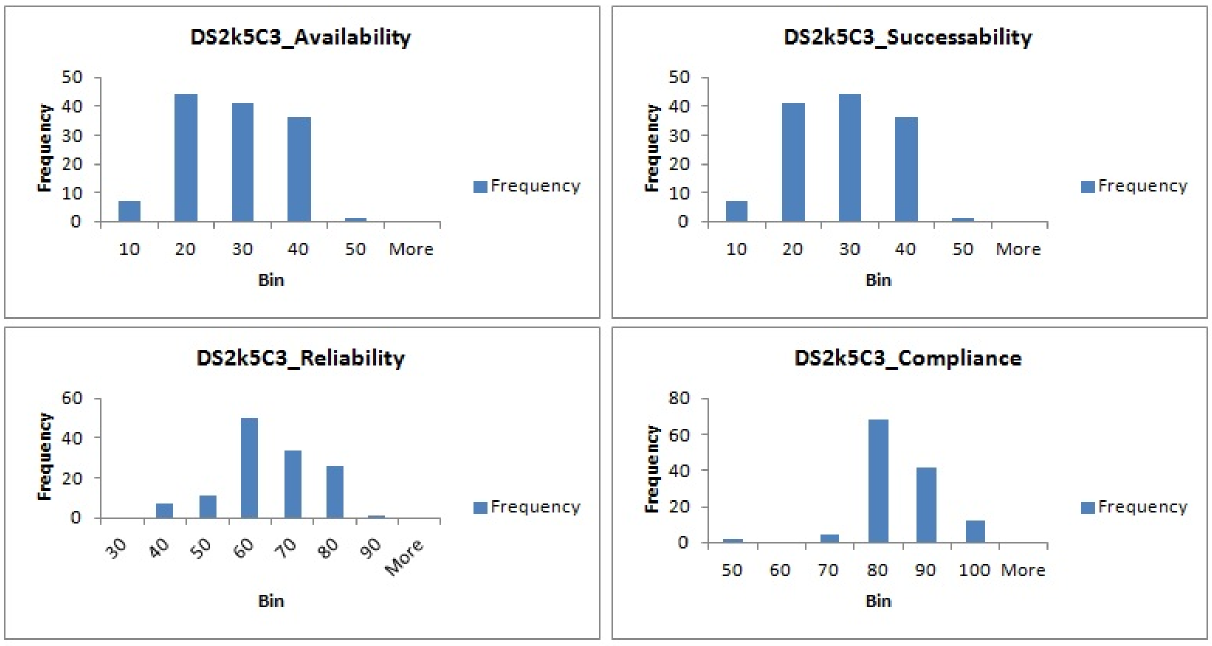


Figure 7.19 Third cluster pie charts of 4 QoS attributes.

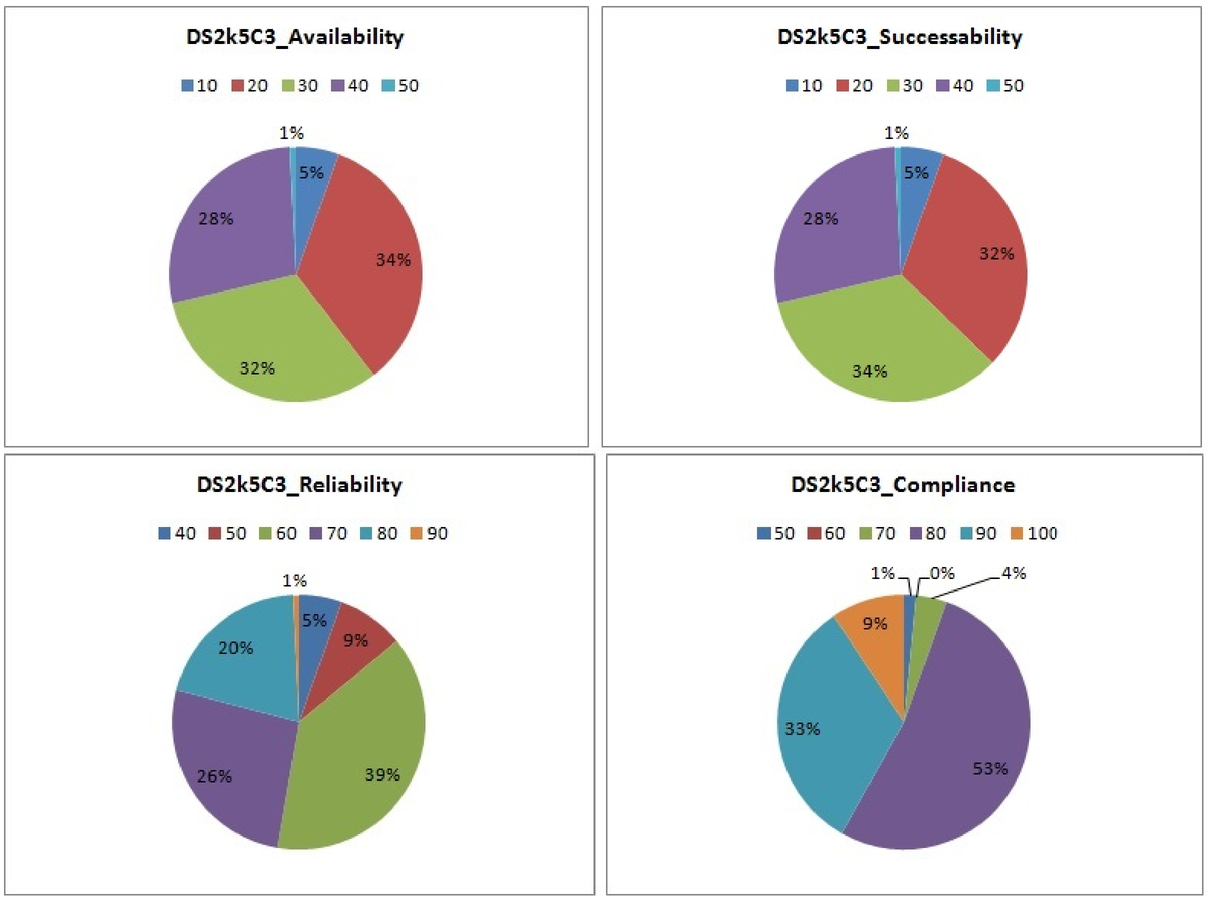


Figure 7.20 Fourth cluster histograms of 4 QoS attributes.

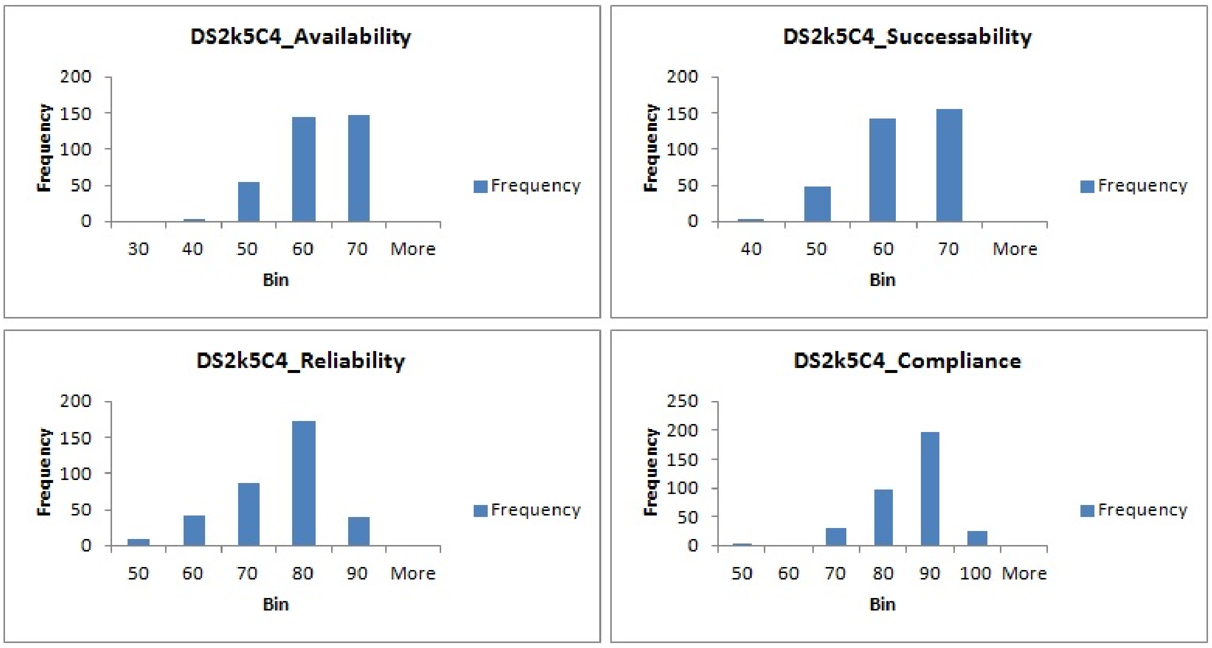


Figure 7.21 Fourth cluster pie charts of 4 QoS attributes.

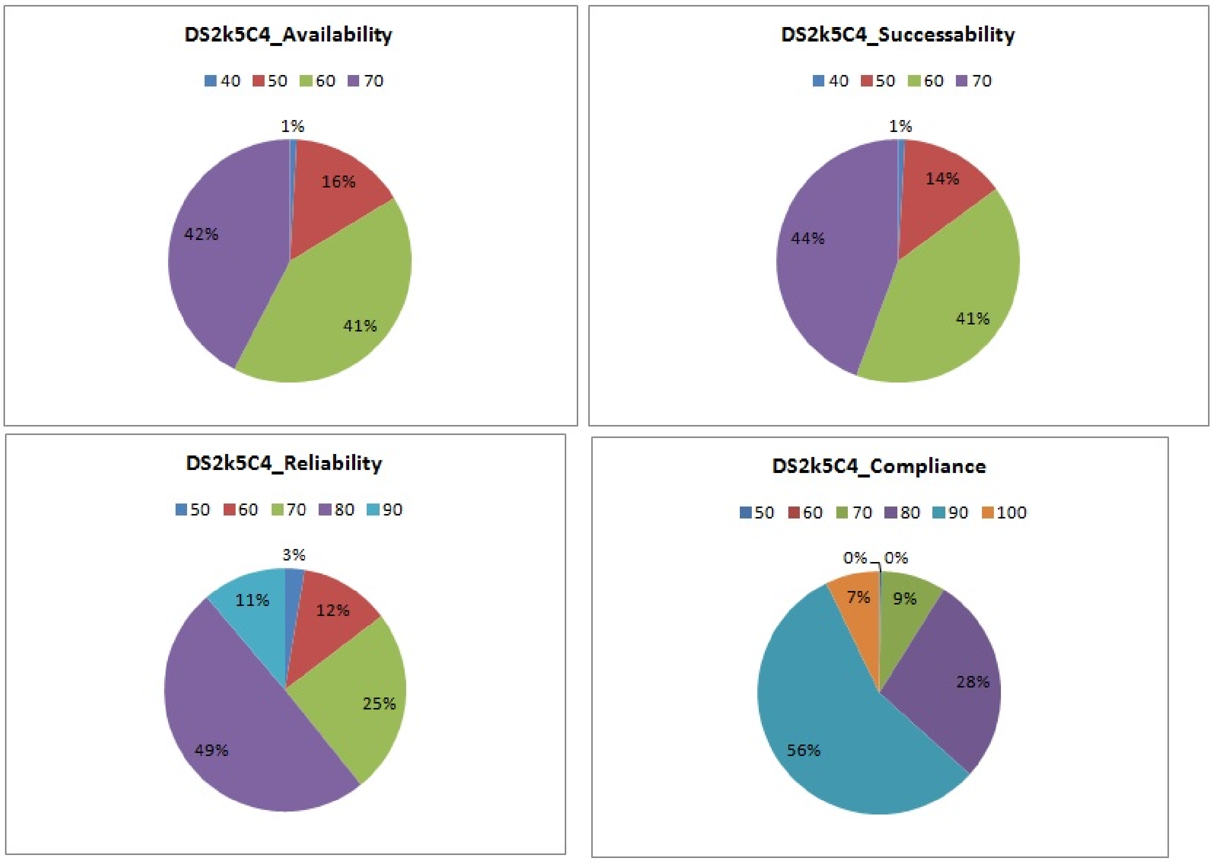


Figure 7.22 Fifth cluster histograms of 4 QoS attributes.

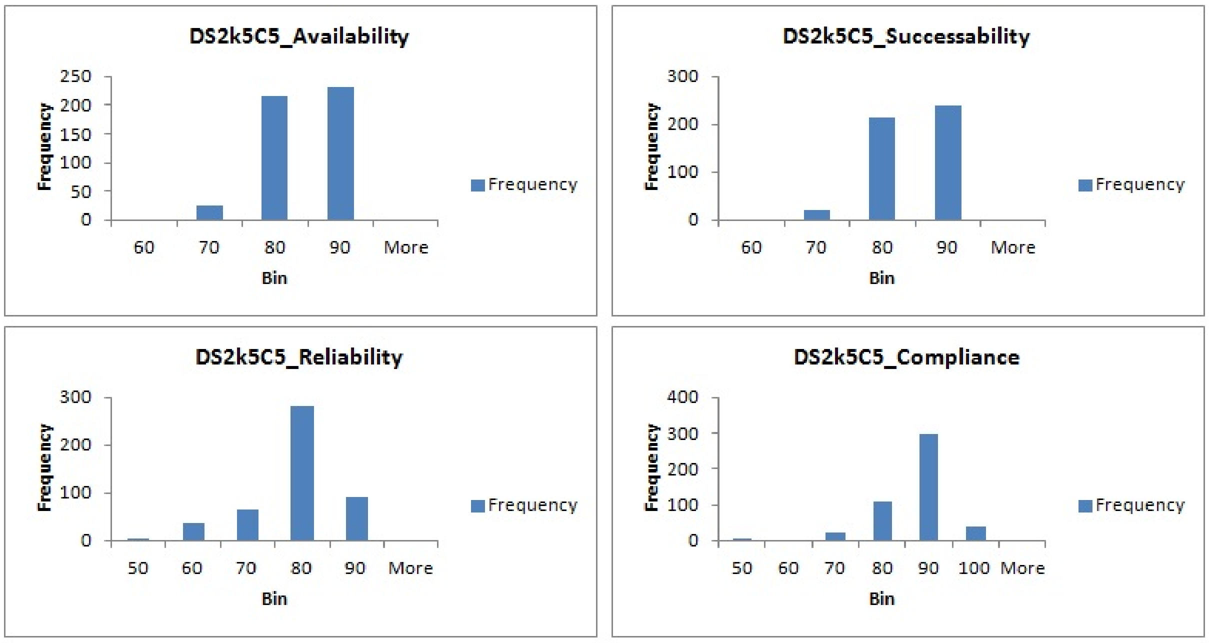
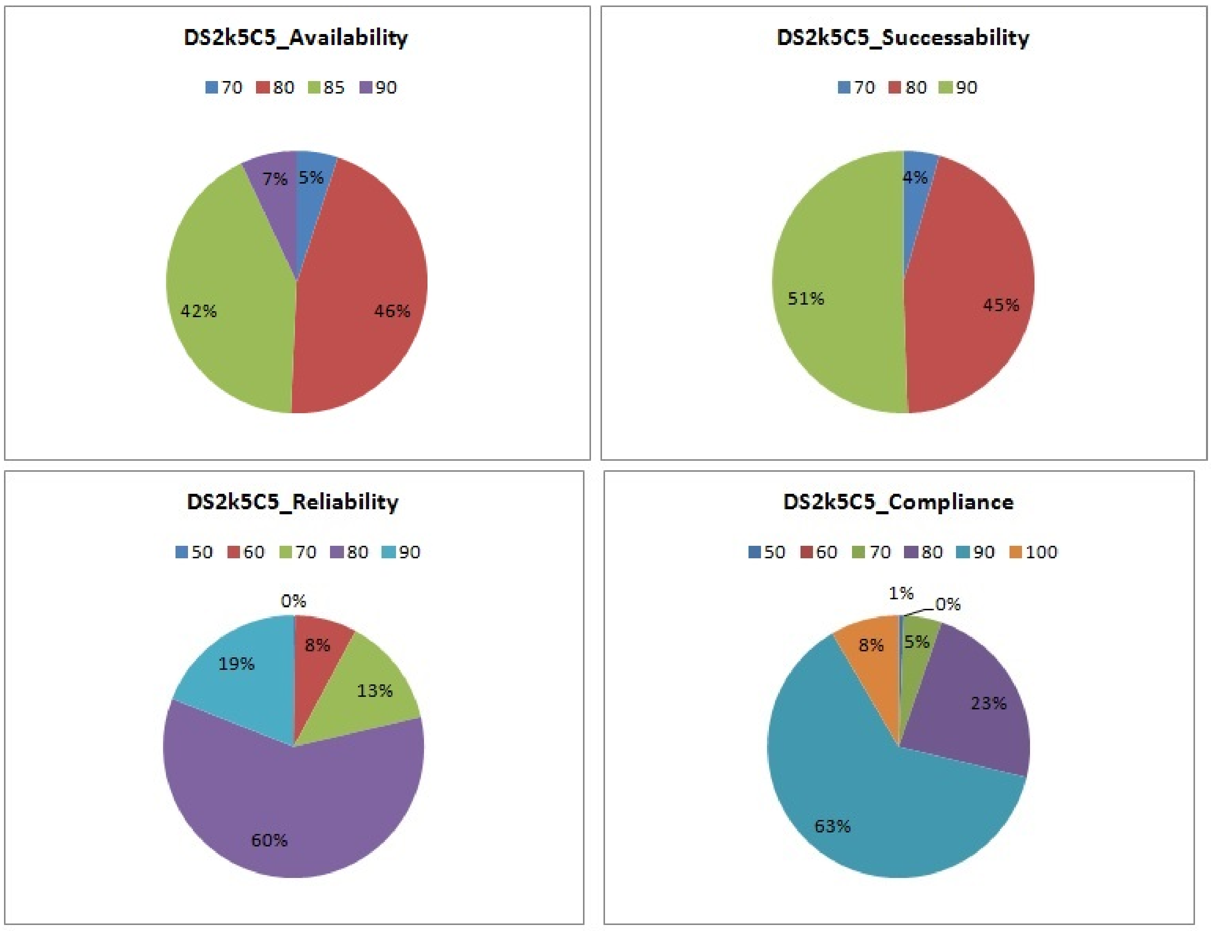


Figure 7.23 Fifth cluster pie charts of 4 QoS attributes.



7.9.3 Evaluation of the Clusters of Web Services

The next step after finding the appropriate number of clusters is to asses these clusters. Some metrics must be used in order to measure the quality of clusters (quality of services grouping and classification), i.e. how good is the technique of clustering web services. Here, we are evaluating to what extent the clustering technique is improving the service selection process. We have employed some of the commonly used clustering evaluation methods, such as Purity, Precision and Recall. These metrics are useful as they produce fixed numbers, which makes clusters comparison easier and comprehensive. We have used these metrics in order to measure the extent to which a specific cluster contains objects (web services) of only one class and the majority of these objects (web services) of the particular class. For example, analyzing if cluster number x contains only web services with high availability and at the same time it contains the majority of web services with high availability.

7.9.3.1 Purity

We use the purity evaluation method in order to measure how pure is a specific cluster with respect to one QoS attribute. Purity computes the extent to which a cluster includes one class of objects (web services) [229]. Next, we present the tables of the purity values of each QoS attribute. For example, in Table 7.15, the purity percent for the first cluster is $\frac{182}{616} = 36\%$, where (182) is the highest frequency of a given data in the first cluster and (616) represents the sum of records in cluster1.

Table 7.15: Availability Purity Table

Cluster	96 - 100	91 - 95	86 - 90	81 - 85	71 - 80	61 - 70	51 - 60	≤ 50	**Purity**
1	181	182	220	33	0	0	0	0	0.357 = 36%
2	254	274	291	121	0	0	0	0	0.310 = 31%
3	0	0	0	0	0	0	0	129	1.000 = 100%
4	0	0	0	0	0	148	144	57	0.424 = 42%
5	0	0	33	200	216	24	0	0	0.457 = 46%
Total	435	456	544	354	216	172	144	186	

Table 7.16: Successability Purity Table

Cluster	91 - 100	81 - 90	71 - 80	61 - 70	51- 60	$\leq$ 50	**Purity**
1	554	62	0	0	0	0	0.899 = 90%
2	812	128	0	0	0	0	0.864 = 86%
3	0	0	0	0	0	129	1.000 = 100%
4	0	0	0	155	142	52	0.444 = 44%
5	0	239	213	21	0	0	0.505 = 51%
Total	1366	429	213	176	142	181	

Table 7.17: Reliability Purity Table

Cluster	81 - 90	71 - 80	61 - 70	51- 60	$\leq$ 50	**Purity**
1	35	448	90	39	4	0.727 = 73%
2	3	477	155	277	28	0.507 = 51%
3	1	26	34	50	18	0.388 = 39%
4	39	173	86	42	9	0.496 = 50%
5	90	282	64	36	1	0.596 = 60%
Total	168	1406	429	444	60	

7.9.3.2 Precision

Precision metric is used in order to measure the relevancy of the retrieved web services from a given cluster [158]. Here, we are testing to what extent the number of web services in a specific cluster are relevant, i.e. how many web services, in a specific cluster, are relevant candidates to be selected? The following tables present the precision tables of each QoS attribute. For example, in Table 7.19, the precision value of web services which have the availability range 96 - 100 in the first cluster is $\frac{181}{616} = 29\%$, where (181) is the number of web services which have the availability range 96 - 100 in the first cluster and (616) represents the sum of records in the cluster1.

Table 7.18: Compliance Purity Table

Cluster	91 - 100	81 - 90	71 - 80	61 - 70	51- 60	≤ 50	**Purity**
1	0	107	482	22	1	4	0.782 = 78%
2	758	177	5	0	0	0	0.806 = 81%
3	12	42	68	5	0	2	0.527 = 53%
4	25	196	97	30	0	1	0.562 = 56%
5	40	297	111	22	0	3	0.628 = 63%
Total	835	819	763	79	1	10	

Table 7.19: Availability Precision Table

Cluster	96 - 100	91 - 95	86 - 90	81 - 85	71 - 80	61 - 70	51 - 60	≤ 50
1	0.294	0.295	0.357	0.054	0.000	0.000	0.000	0.000
2	0.270	0.291	0.310	0.129	0.000	0.000	0.000	0.000
3	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000
4	0.000	0.000	0.000	0.000	0.000	0.424	0.413	0.163
5	0.000	0.000	0.070	0.423	0.457	0.051	0.000	0.000

Table 7.20: Successability Precision Table

Cluster	91 - 100	81 - 90	71 - 80	61 - 70	51- 60	≤ 50
1	0.899	0.101	0.000	0.000	0.000	0.000
2	0.864	0.136	0.000	0.000	0.000	0.000
3	0.000	0.000	0.000	0.000	0.000	1.000
4	0.000	0.000	0.000	0.444	0.407	0.149
5	0.000	0.505	0.450	0.044	0.000	0.000

Table 7.21: Reliability Precision Table

Cluster	81 - 90	71 - 80	61 - 70	51- 60	≤ 50
1	0.057	0.727	0.146	0.063	0.006
2	0.003	0.507	0.165	0.295	0.030
3	0.008	0.202	0.264	0.388	0.140
4	0.112	0.496	0.246	0.120	0.026
5	0.190	0.596	0.135	0.076	0.002

Table 7.22: Compliance Precision Table

Cluster	91 - 100	81 - 90	71 - 80	61 - 70	51- 60	≤ 50
1	0.000	0.174	0.782	0.036	0.002	0.006
2	0.806	0.188	0.005	0.000	0.000	0.000
3	0.093	0.326	0.527	0.039	0.000	0.016
4	0.072	0.562	0.278	0.086	0.000	0.003
5	0.085	0.628	0.235	0.047	0.000	0.006

7.9.3.3 Recall

Recall metric is used in order to measure the percentage of the relevant retrieved web services in a given cluster [158]. Here we are testing to what extent the relevant web services in the dataset are grouped in a specific cluster, i.e. how many relevant web services are grouped in a specific cluster? The following tables present the recall tables of each QoS attribute. For example, in table 7.23, the recall value of web services which have the availability range 96 - 100 in the first cluster is $\frac{181}{435} = 42\%$, where (181) is the number of web services which have the availability range 96 - 100 in the first cluster and (435) is the total number of records of web services which have the availability range 96 - 100 in the dataset.

Table 7.23: Availability Recall Table

Cluster	96 - 100	91 - 95	86 - 90	81 - 85	71 - 80	61 - 70	51 - 60	≤ 50
1	0.416	0.399	0.404	0.093	0.000	0.000	0.000	0.000
2	0.584	0.601	0.535	0.342	0.000	0.000	0.000	0.000
3	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.694
4	0.000	0.000	0.000	0.000	0.000	0.860	1.000	0.306
5	0.000	0.000	0.061	0.565	1.000	0.140	0.000	0.000

Table 7.24: Successability Recall Table

Cluster	91 - 100	81 - 90	71 - 80	61 - 70	51- 60	≤ 50
1	0.406	0.145	0.000	0.000	0.000	0.000
2	0.594	0.298	0.000	0.000	0.000	0.000
3	0.000	0.000	0.000	0.000	0.000	0.713
4	0.000	0.000	0.000	0.881	1.000	0.287
5	0.000	0.557	1.000	0.119	0.000	0.000

Table 7.25: ReliabilityRecallTable

Cluster	81 - 90	71 - 80	61 - 70	51- 60	≤ 50
1	0.208	0.319	0.210	0.088	0.067
2	0.018	0.339	0.361	0.624	0.467
3	0.006	0.018	0.079	0.113	0.300
4	0.232	0.123	0.200	0.095	0.150
5	0.536	0.201	0.149	0.081	1.000

Table 7.26: Compliance Recall Table

Cluster	91 - 100	81 - 90	71 - 80	61 - 70	51- 60	≤ 50
1	0.000	0.131	0.632	0.278	1.000	0.400
2	0.908	0.216	0.007	0.000	0.000	0.000
3	0.014	0.051	0.089	0.063	0.000	0.200
4	0.030	0.239	0.127	0.380	0.000	0.100
5	0.048	0.363	0.145	0.278	0.000	0.300

7.9.4 Selection Improvement: Clustering-Aware vs. Clustering-Neutral

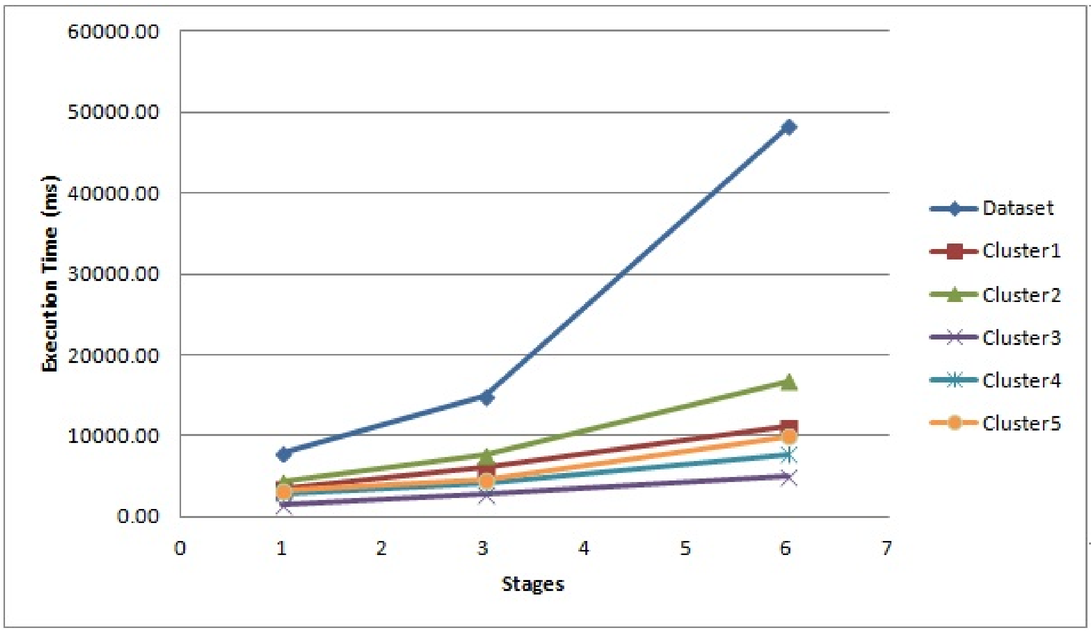
7.9.4.1 From CloudMTD Perspective

As mentioned before, Clustering phase was chosen for the purpose of improving the scalability and reducing the search space, and accordingly improving the performance of service selection process. In this context, we have tested the impact of the clustering phase on improving the performance and scalability of the selection process. Figure 7.24 illustrates CloudMTD model execution time with and without clustering phase for the whole QWS dataset and the five clusters. As seen in the figure, execution time is reduced by using the clustering technique, instead of searching the whole dataset, which represent the cloud marketplace. In this sense, decision makers can search for candidate web services in the relevant cluster instead of the whole dataset. In this context, we have evaluated to what extent the CloudMTD model is saving time and reducing computational overhead by using the clustering technique.

7.9.4.2 From Selection Decision Perspective

The clustering phase improved the selection process by informing a better selection decision in CB-SOA. This is achieved by considering the purity, precision and recall metrics for each cluster. These metrics help in getting better insights on the effectiveness of the selection process. In this sense, decision makers are directed toward the precise cluster and accordingly they reach better candidate services for the selection process. Hence, when heading toward the precise cluster, the likelihood of getting a better service is high.

Figure 7.24 CloudMTD model execution time with varying number of stages (Clustering-Aware vs. Clustering-Neutral).



7.10 Structural Evaluation: Composing Web Services

In the second part of the evaluation, we evaluate the phase of composing services in CB-SOA using CloudMTD model based on OptiViTA case study. We are interested in finding the consequences of a change, which occur due to dependencies among services. Here, we look at the extent to which substituting a service has an impact on other services in a given architecture. We also evaluate the extent of the ripple impact of a change in CB-SOA. Ripple impact takes place when a change originated from one service affects one or more services in CB-SOA.

The structural evaluation is performed on the architectural-level based on the DSM, Visibility Matrix and the Propagation-cost. These metrics provide a view of the extent and patterns of dependencies among services in CB-SOA. In this sense, CloudMTD model aids decision makers by making the architectural dependency visible. When analyzing the architecture based on the DSM analytical view, some metrics can be extracted such as the propagation-cost metric. Such metric provides some information on the likely rework cost, by investigating the associated dependencies. Here, we look at the rework cost that is associated with selection and composition decisions in CB-SOA. In this sense, the propagation-cost metric provides insights on cost and value of selection and composition decisions in CB-SOA. We particularly look at the propagation cost in order to model the impact of the selection and composition decisions in relation to both dependency and complexity. DSM and propagation-cost metrics are used for: (1) identifying dependencies, (2) tracking the impact of change, and (3) identifying rework cost.

7.11 Option to Defer Scenario: Results and Discussion

In previous chapter, we presented three types of options, which are related to different service selection scenarios. In this section, we look at another type of option, which is option to defer a decision. The current OptiViTA Architecture consists of different web services and OptiViTA decided to improve its utility by upgrading one of its constituent web services to a new one that has been released in the cloud marketplace. As the prediction showed that there will be an increase in load after some time, which will call for upgrading the current web service by switching to another one with higher capacity. OptiViTA needs to know when to invest with respect to load fluctuation, i.e. when it is suitable to substitute the web service by comparing the relative value of switching to the new web service against the value of the current one (ΔV). The suitable time for switching could be related to the nature of demand; peak or off-peak daily usage; peak or off-peak seasons, etc. This decision can be formulated using *option-to-defer*. As mentioned in chapter 5, option to defer can assess the *time-value* of waiting and it has many applications such as waiting to expand, waiting to enter a new market, etc. In this scenario, we predict the time value of waiting to defer the decision of investing in web services substitution. The question is when it is beneficial, in terms of value and technical debt, to exercise the option. Next, we will track the time value of waiting and make a comparison in two extremes; peak time - deferring the decision for one month, and off-peak time - deferring the decision for two months respectively. In either case, we assume that OptiViTA cloud-based service-oriented architecture is already in operation and delivering value. This value is used to be compared against the new web service's value. In this scenario, the hypothesis is that substituting a web service at specific time may embed flexibility to accommodate future load and generate a value. The substitution decision may incur a technical debt, which can be translated into future options, if properly managed. The decision maker needs to assess i) how switching can clear an existing debt and create future options (ii) how switching is likely to introduce an intentional debt, (iii) when it is optimal to switch, while tracking the debt against utility. Here, OptiViTA business goal is to maximize the system's availability, as more fulfilled requests may imply revenue and will improve the utility value of the structure through supporting more users. From CloudMTD perspective, OptiViTA business goal should be also debt-aware.

Let us consider the case where OptiViTA needs to substitute only one of its constituent web services to meet changes in requirements. After selecting the suitable cluster, a binomial tree will be built for each nominated web service based on inputs in table 7.27 (steps were explained in Section 6.1.4). In this scenario, we present the selection and composition phases after conducting the behavioral analysis that was presented in section 7.9. The below illustration shows the analysis for one selection.

Table 7.27: Coefficients Values for the candidate web service

Attributes	Values
V_0	Depends on the starting date (defer)
r	5%
u	40%
d	-10%
p	0.5
Expiry Date	6 months
Rework Cost	£200.00
Exercise Price	£40.00

7.11.1 Behavioral Evaluation Results

7.11.1.1 Defer for one month results

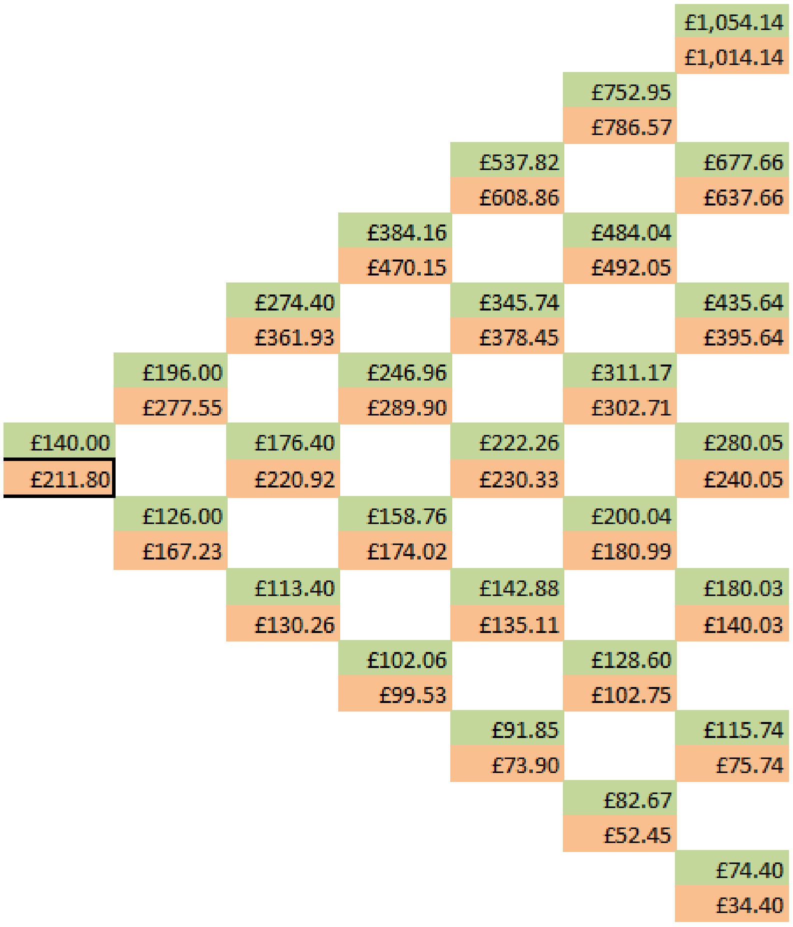
Deferring the switching decision by one month will increase the interest on the switching cost (rework). Here, the cost of acquiring the option and the interest on the loan continue to be higher than the option value as when compared to the value delivered from retaining the old service. Accordingly, the debt is not cleared during the six-month evaluation of the investment, Table 7.28 (Figure 7.25 and 7.26). Here, the scenario reveals that the new architecture capacity in supporting users is underutilized due to off-peak time. Henceforth, it is not fully unlocking the architecture potential in delivering value, which overweighs the cost and interest. As a result, it would be wise to keep the current web service as the value-added of the new investment is likely to be wasted as the current architecture potential is just fit for purpose. Moreover, the value of utilizing the new architecture potential still lags behind the switching and interest costs signaling a technical debt.

Option1 summary: Deferring the decision for one month is the *cheapest* option in term of interest, however at this point the option value continues to be zero during the evaluation period and as a result the technical debt is not cleared. Therefore, it is not suitable to exercise the option at this stage. Here, we say that the option is *out-of-the-money.*

Table 7.28: Deferring for one month Options, TD and Value.

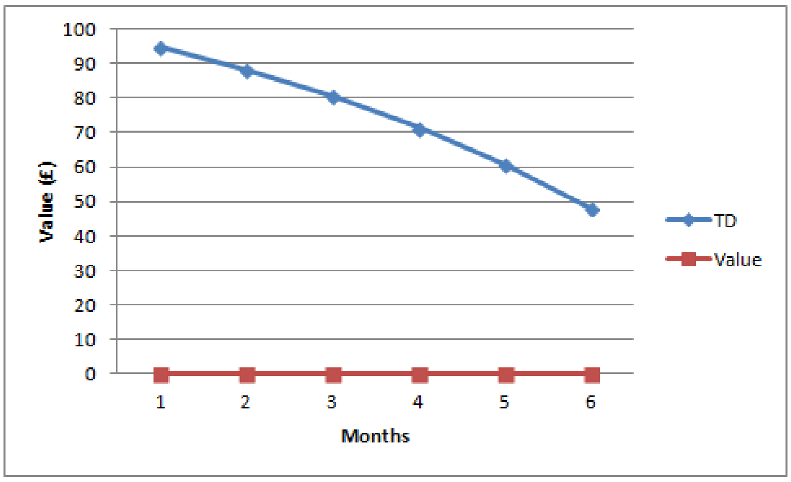
Month	Options	TD	Value
1	115.24	94.7619	0
2	131.66	88.34467	0
3	149.38	80.62304	0
4	168.54	71.46045	0
5	189.29	60.70791	0
6	211.80	48.2028	0

Figure 7.25 6-month evaluation after one month of waiting to invest



						£1,054.14
						£1,014.14
					£752.95	
					£786.57	
				£537.82		£677.66
				£608.86		£637.66
			£384.16		£484.04	
			£470.15		£492.05	
		£274.40		£345.74		£435.64
		£361.93		£378.45		£395.64
	£196.00		£246.96		£311.17	
	£277.55		£289.90		£302.71	
£140.00		£176.40		£222.26		£280.05
£211.80		£220.92		£230.33		£240.05
	£126.00		£158.76		£200.04	
	£167.23		£174.02		£180.99	
		£113.40		£142.88		£180.03
		£130.26		£135.11		£140.03
			£102.06		£128.60	
			£99.53		£102.75	
				£91.85		£115.74
				£73.90		£75.74
					£82.67	
					£52.45	
						£74.40
						£34.40

Figure 7.26 TD vs. Option Value over 6-month evaluation



7.11.1.2 Defer for two months results

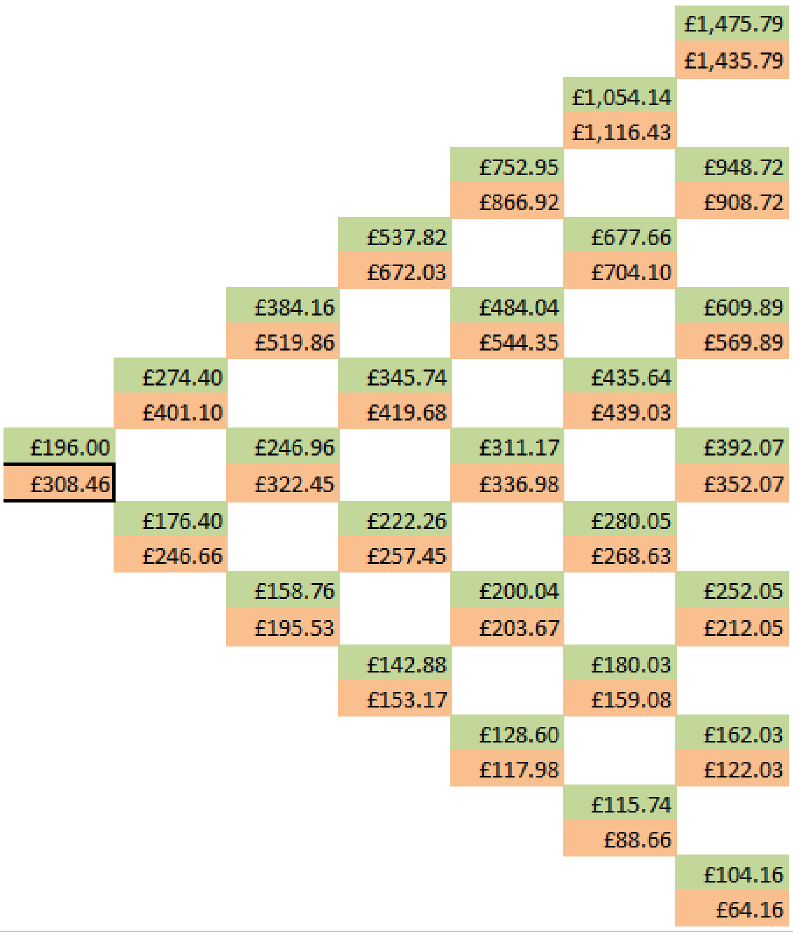
The CloudMTD analysis showed that the value of waiting to switch is higher than the value of immediate switching. Here, the switching decision will be deferred by two months. Accordingly, more interests will be incurred. We assume that the interest will be incurred each month as long as the technical debt is not cleared yet. The analysis revealed that the option-to-wait will start to pay off after the fourth month as it is a high season period. As the load fluctuates on the architecture, waiting for peak seasons could be ideal, as we expect to have more active users benefiting from the structure. This is attributed to the system being able to fully utilize the web service capacity at this time. As a result, the technical debt will be managed and cleared on the fifth month, Table 7.29 (Fig. 7.27 and 7.28). Here, the value of the option exceeds the switching cost and the flexibility of the decision relative to the change in load is likely to pay off, if the option is exercised.

Option2 summary: The option of deferring the decision for two months is *more expensive* than the first option, however this option will pay off in later stages and clear the technical debt. As a result, it is suitable to exercise the option at this period. At this point, the option is said to be *in-the-money.*

Table 7.29: Deferring for two months Options, TD and Value.

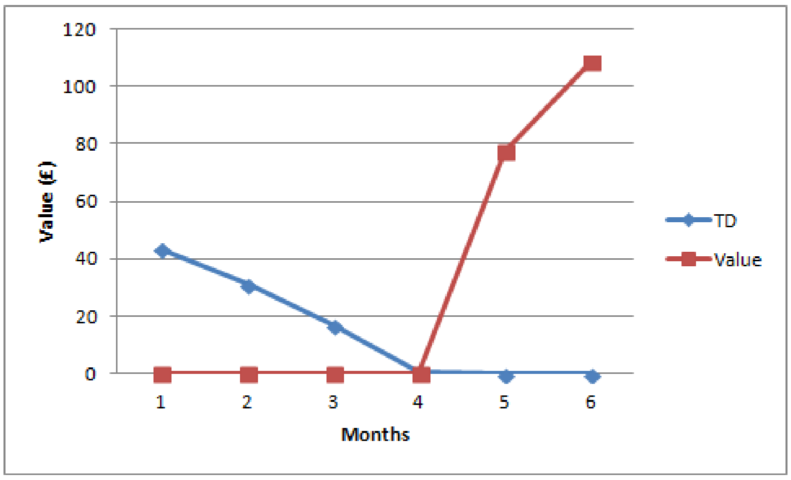
Month	Options	TD	Value
1	176.57	43.42857	0
2	198.83	31.17007	0
3	222.95	17.05086	0
4	249.12	0.881392	0
5	277.55	0	77.54535
6	308.46	0	108.4555

Figure 7.27 6-month evaluation after 2 months of waiting to invest



						£1,475.79
						£1,435.79
					£1,054.14	
					£1,116.43	
				£752.95		£948.72
				£866.92		£908.72
			£537.82		£677.66	
			£672.03		£704.10	
		£384.16		£484.04		£609.89
		£519.86		£544.35		£569.89
	£274.40		£345.74		£435.64	
	£401.10		£419.68		£439.03	
£196.00		£246.96		£311.17		£392.07
£308.46		£322.45		£336.98		£352.07
	£176.40		£222.26		£280.05	
	£246.66		£257.45		£268.63	
		£158.76		£200.04		£252.05
		£195.53		£203.67		£212.05
			£142.88		£180.03	
			£153.17		£159.08	
				£128.60		£162.03
				£117.98		£122.03
					£115.74	
					£88.66	
						£104.16
						£64.16

Figure 7.28 TD vs. Option Value over 6-month evaluation



7.11.2 Structural Evaluation Results

We now evaluate the entire architecture's propagation cost as explained in chapter 6-Step(6). The first matrix in figure 7.30 shows the DSM of OptiViTA architecture (OptiViTA architecture was explained in section 7.3). From the given DSM we can conclude which web services are affected by switching a specific web service and then we can compute the propagation cost in the entire architecture. Figure 7.30 shows the derivation of the propagation cost metric for OptiViTA Architecture based on the approach described in previous chapter. Then, we compute the final density matrix by converting all non-zero cells in V matrix to ones and applying Equation 7.1. Accordingly, the propagation cost is computed as follows: $\frac{15}{25} = 60\%$ (Equation 7.2). This means that on average 60% of the services are likely to be affected by the change. As mentioned before, this percentage does not provide a meaningful indication of the likely complexity that could affect the cost. Finally, we compute the "Complexity Density Matrix" by reflecting the dependency complexity levels (Figure 7.29). Despite the fact that two services may be dependent, the complexity of the dependency may vary. For example, higher complexity may signal higher rework cost, lower complexity may indicate otherwise. This can be due to the effort required for code development, rewriting, configuration, data migration, new/throwaway maintenance, fixing mismatches, changes to legacy code, etc. We adjust the dependency by a multiplying weight to reflect complexity.

$$V = \sum_{i=0}^{n} M^i = \sum_{i=0}^{5} M^i \tag{7.1}$$

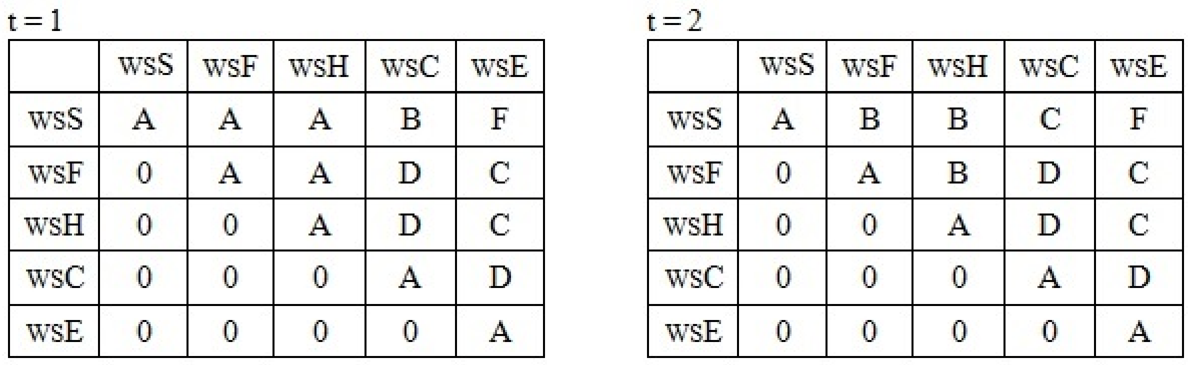
$$PropCost_{OptiViTAarch} = \frac{nzc}{nc} = \frac{15}{25} = 0.6 = 60\% \tag{7.2}$$

Table 7.31 shows the complexity levels values in the Complexity Density Matrix at t=1 and t=2 (Described previously). The dependency analysis for t_1 and t_2 corresponds to deferring the switching by one and two months respectively. Table 7.31 shows that by substituting the web service at t_1 that 60% of the affected web services by the change

Table 7.30: Density Matrix Level of Complexity

Scheme	Complexity level	Multiplication factor
A	Very high	1
B	High	0.7-0.9
C	Moderate	0.5-0.69
D	Low	0.3-0.49
F	Very low	0.1-0.29
0	No dependency	0

Figure 7.29 OptiViTA Weighted Density Matrix at t=1 and t=2



t=1

	wsS	wsF	wsH	wsC	wsE
wsS	A	A	A	B	F
wsF	0	A	A	D	C
wsH	0	0	A	D	C
wsC	0	0	0	A	D
wsE	0	0	0	0	A

t=2

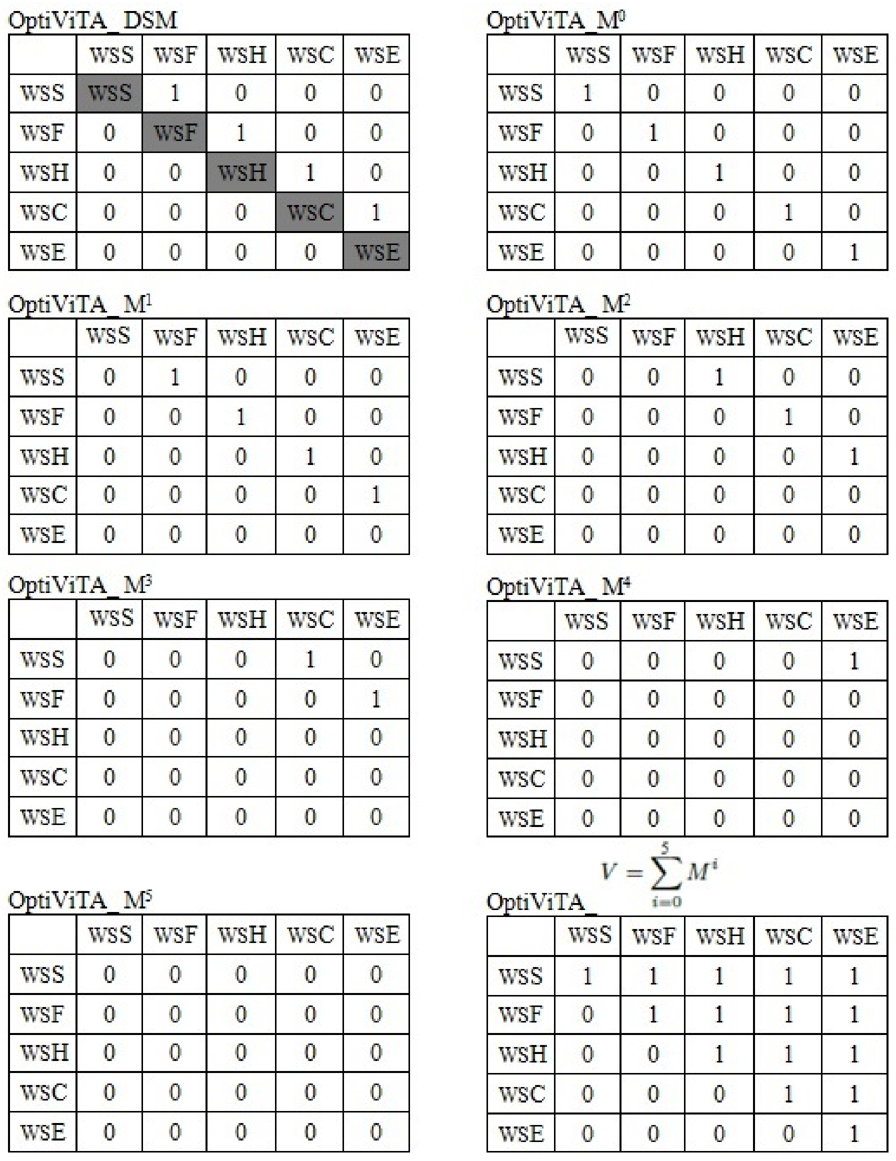
	wsS	wsF	wsH	wsC	wsE
wsS	A	B	B	C	F
wsF	0	A	B	D	C
wsH	0	0	A	D	C
wsC	0	0	0	A	D
wsE	0	0	0	0	A

are of level complexity A(53%), B(7%), C(13%), D(20%) and F(7%). If we defer the substitution decision to t_2, the percentage of elements classified under complexity level A has decreased. This can be attributed, for example, to upgrades to one of the legacy services in the composition, which has eliminated the need for designing wrappers for integration.

Table 7.31: Complexity Density Matrix values at t=1 and t=2

Level	t = 1	t = 2
A	8/15 = 53%	5/15 = 33%
B	1/15 = 7%	3/15 = 20%
C	2/15 = 13%	3/15 = 20%
D	3/15 = 20%	3/15 = 20%
F	1/15 = 7%	1/15 = 7%

Figure 7.30 The Derivation of propagation cost out of the Visibility Matrix (V) of OptiViTA structure.



OptiViTA_DSM

	WSS	WSF	WSH	WSC	WSE
WSS	WSS	1	0	0	0
WSF	0	WSF	1	0	0
WSH	0	0	WSH	1	0
WSC	0	0	0	WSC	1
WSE	0	0	0	0	WSE

OptiViTA_M^0

	WSS	WSF	WSH	WSC	WSE
WSS	1	0	0	0	0
WSF	0	1	0	0	0
WSH	0	0	1	0	0
WSC	0	0	0	1	0
WSE	0	0	0	0	1

OptiViTA_M^1

	WSS	WSF	WSH	WSC	WSE
WSS	0	1	0	0	0
WSF	0	0	1	0	0
WSH	0	0	0	1	0
WSC	0	0	0	0	1
WSE	0	0	0	0	0

OptiViTA_M^2

	WSS	WSF	WSH	WSC	WSE
WSS	0	0	1	0	0
WSF	0	0	0	1	0
WSH	0	0	0	0	1
WSC	0	0	0	0	0
WSE	0	0	0	0	0

OptiViTA_M^3

	WSS	WSF	WSH	WSC	WSE
WSS	0	0	0	1	0
WSF	0	0	0	0	1
WSH	0	0	0	0	0
WSC	0	0	0	0	0
WSE	0	0	0	0	0

OptiViTA_M^4

	WSS	WSF	WSH	WSC	WSE
WSS	0	0	0	0	1
WSF	0	0	0	0	0
WSH	0	0	0	0	0
WSC	0	0	0	0	0
WSE	0	0	0	0	0

OptiViTA_M^5

	WSS	WSF	WSH	WSC	WSE
WSS	0	0	0	0	0
WSF	0	0	0	0	0
WSH	0	0	0	0	0
WSC	0	0	0	0	0
WSE	0	0	0	0	0

$$V = \sum_{i=0}^{5} M^i$$

OptiViTA_

	WSS	WSF	WSH	WSC	WSE
WSS	1	1	1	1	1
WSF	0	1	1	1	1
WSH	0	0	1	1	1
WSC	0	0	0	1	1
WSE	0	0	0	0	1

7.11.3 Decision-Making Improvement

In the previous scenario, the analysis has provided assessment of some situations leading to underutilization and incurring technical debt. Substitution decision was primarily driven by improving availability business objective.

7.11.3.1 Options-Based vs. Options-Neutral

The option value has provided insights into the suitable switching time based on the improved utility from behavioral and structural points of view. Unlike other selection and composition approaches, those are based on short-term analysis; CloudMTD takes into consideration long-term value and opportunities that option creates.

7.11.3.2 Technical Debt-Aware vs. Technical Debt-Neutral

Technical debt and complexity analysis, when coined with option thinking can provide a powerful what-if analysis tool to inform (1) when to substitute; (2) the implication of time-value of the decision on utility and technical debt reduction; (3) opportunities for creating value through waiting for possible changes in the structure, which can better withstand the change and reduce the cost of substitution etc. On the other hand, if the selection decision is "technical debt-neutral", less attractive web services may be neglected and future value will be lost accordingly.

7.11.3.3 Decision's Staging Analysis

The use of the binomial tree has improved the decision-making in CB-SOA by providing the following criteria while analyzing the decision: (1) staging-view, (2) multiple-decision points, and (3) transparency of value and debt. In this context, we have evaluated the impact of increasing the window of staging the decision-making in CB-SOA. By the use of staging, we are able to have finer analysis. Seeing that the more stages we have, the more information we get. In addition, the binomial tree provided multiple-decision points within each stage. Accordingly, CB-SOA decision makers acquire more flexibility to make a better decision at different nodes in the tree. Furthermore, the use of the binomial tree has improved the transparency of the decision making in CB-SOA. This is achieved by visualizing the value and technical debt at each stage. Accordingly, the analysis is improved by presenting the possible ups and downs in the tree at each stage.

7.11.3.4 Dependency Tracking

The dependency and complexity analysis of two time intervals have provided means to evaluate the time-value of the decision to substitute relative to changes in the structure. Overall, the structural and behavioral analysis for utility, technical debt and complexity informed the time-value of the substitution decision.

7.12 Limitations and Threats to Validity

CloudMTD model was developed based on some scenarios and under some assumptions.

- The binomial tree coefficients were used under controlled experiments, where we assumed their values for each scenario. However, in reality, these coefficients can be captured using human-centered elicitation. In this thesis, we assume that the binomial tree coefficients are taking from the CB-SOA decision makers (e.g. stakeholders). In this sense, coefficients may vary depending on the number of stakeholders and their expertise. For example, stakeholders can decide on staging based on the type of option. In some cases, the change to any of these coefficients can affect the results of the CloudMTD model. Accordingly, options and technical debt calculations can be affected.

- The choice of the dataset and the QoS attributes. In this thesis, we have fixed the number of QoS attributes and we have chosen 4 attributes. This was done for the purpose of evaluating the CloudMTD model under an extreme case. In addition, the 4 attributes were chosen for the purpose of evaluating the QoS correlation among web services.

- The clustering overhead was equal to 31.491 (ms) for 2507 web services when k=5. However, this execution time does not include the offline processing, such as clustering analysis, which is a human-centered activity.

- Technical Debt is subjective. In this sense, technical debt can be affected by the following: (1) CB-SOA decision makers and their expertise (e.g. stakeholders), (2) binomial tree coefficients, (3) options, and (4) rework cost. The threshold of technical debt and its clearance month depend on the acceptance of stakeholders. For example, clearing technical debt on the fourth month can be accepted for some stakeholders and cannot be accepted for others.

- Value is subjective. Value depends on the acceptance-level of the CB-SOA decision makers. For example, a £100 increase on revenues can be accepted for some stakeholders and cannot be accepted for others.

- Dependencies among web services and their complexities can vary based on CB-SOA decision makers. In this sense, the time of taking the composition decision can be influenced by dependencies and complexities. We have chosen a simplistic assumption to guide the dependency calculation. Nevertheless, other techniques can be used from the literature of Software Engineering for quantifying dependencies based on structural analysis.

- The choice of the software. As mentioned before, CloudMTD model was implemented and test using the OpenESB software. This software is lighter, in terms of overhead, than other software such as NetBeans. Accordingly, performance results may vary based on the choice of the software.

7.13 Summary

The evaluation was carried out based on a combination of a case study and experimentation. The experiments were designed in order to verify how the CloudMTD model performs when varying the following: (1) number of web services, (2) number of binomial steps, (3) options stages and (4) technical debt stages.

We have accomplished the evaluation process based on two major stages; behavioral (service-level) and structural (architecture-level). The behavioral evaluation was performed based on option theory and K-means clustering. Accordingly, we have informed the following: (1) when to substitute; (2) the implication of time-value of the decision on utility and technical debt reduction; (3) opportunities for creating value. We have evaluated to what extent the CloudMTD model is saving time and reducing computational overhead by using the clustering technique. We have also evaluated the extent to which CloudMTD model is capable of facilitating the what-if analysis at different points in the decision space.

The structural evaluation was carried out using three metrics: (1) Dependency Structure Matrix (DSM), (2) Visibility Matrix (based on reachability matrix), and (3) the Propagation-cost. Information extracted from these metrics were used for: (1) identifying dependencies among services, (2) tracking the impact of change, (3) identifying the likelihood of change propagation, and (4) identifying rework cost.

7.14 Discussions and Conclusions

A distinctive feature of the CloudMTD model is that it links value of substitution decisions to both the behavior and the structure of CB-SOA. The former is through coining options analysis with improved utilities upon realizing changes, which are driven by business objectives and of non-functional nature. The latter is through tracking the "ripple" impact of the decision on the structure. This is because the composition may change over time as any of the constituent services are upgraded, modified, etc. As a result, the selection and composition decisions tend to be influenced by changes in the structure, likely technical debt, and changes in complexity. The model acknowledges the fact that CB-SOA are utility-driven and tend to be dynamic and market sensitive. The model caters for dynamism and utility-driven evaluation by introducing complexity- and time-aware propagation metrics.

By using CloudMTD model, we are not aiming to get precise results, as the aim is not for precision but to get a "good enough" and qualified selection and composition decisions based on the context of each scenario. This is also related to the domain of the problem to be solved. Here, we say that when investing in web services, we can tolerate some changes and accept good enough results. This is unlike other inflexible domains such as domains dealing with medical data, where changes are not tolerant. In this sense, utility is quantified based on scenarios of interests of CB-SOA decision makers.

8 Conclusion and Future Work

8.1 Summary

This thesis has investigated the problem of service selection and composition in Cloud-Based Service-Oriented Architectures (CB-SOA). It has also defined the concept of technical debt for Cloud-Based Service-Oriented Architectures, which covers several dimensions that are related to service substitution decisions. Technical debt in cloud-based web service selection can be attributed to different factors such as poor and swift selection decisions, mismatches in applications' requirements with that of the service provision, falsely accelerating the velocity of the integration, testing and deployment process due to budget restrictions, and/or through accidentally acquiring the debt via an untrusted provider. The problem of web service substitution and its technical debt valuation was formulated as an option problem using Binomial Options valuation.

In addition, this thesis used Dependency Structure Matrix (DSM) for representing dependencies among services in CB-SOA. It introduced time and complexity sensitive propagation cost metrics to DSM for dependencies quantification. In addition, this thesis presented the CloudMTD model that is capable of informing the time-value of the decisions under uncertainty based on behavioral and structural aspects of CB-SOA. The CloudMTD model can provide CB-SOA decision makers (e.g. architects) with insights of the value of the structure, its utilities in supporting changes and the technical debt as a result of service substitution. The model is appealing to the recent hype in looking at cloud as a marketplace for trading services. It is in line with the motivation of linking technical decisions in software to value under uncertainty. This thesis has also reported on the implementation of the CloudMTD model for selection and composition in CB-SOA. It demonstrated and evaluated the applicability of CloudMTD model using a case study that was implemented using WS-BPEL. Results showed that the analysis can link substitution decisions in CB-SOA to long term value creation and technical debt reduction.

This thesis answered the following research questions:

- How is the CloudMTD model capable of enhancing the selection and composition decisions in CB-SOA under uncertainty? How can CloudMTD model improve the execution time of the selection and composition process? The results proved that the CloudMTD model improved the selection and composition process in CB-SOA, in terms of execution time, time-value of the decision, and technical debt.

- How does the Binomial Option Analysis (option thinking) add value to selection decisions in CB-SOA taking into consideration technical debt? Is option theory capable of valuing web services investments? This thesis proved that the use of option theory is valuable when there are uncertainties associated with the decision making process. This was approached by the use of the CloudMTD model that was capable of managing such uncertainties, based on Binomial Option Analysis. CloudMTD model provides the decision maker with flexibility when dealing with such decisions.

- Does technical debt have an impact on the selection decision? Does CloudMTD model make technical debt explicit in CB-SOA? This thesis proved that technical debt, which is associated with service selection decisions, can be managed and transformed to future value.

- How is the CloudMTD model capable of giving a good recommendation about time-value of selection and composition decisions in CB-SOA taking into consideration the behavioral and structural aspects of CB-SOA? In this thesis, the CloudMTD model informed the time-value of the selection and composition decisions under uncertainty based on behavioral and structural aspects of CB-SOA.

- How can K-means clustering approach enhance the scalability of the model? Does the clustering approach make QoS correlations, recall and relevancy explicit in CB-SOA selection and composition? This thesis proved that by the use of k-means clustering technique, services are classified into different groups based on their QoS attributes' similarities. K-means clustering technique has improved the service selection process in CB-SOA in terms of scalability, relevancy, and QoS correlation.

8.2 Summary of Contributions

This thesis's contributions are summarized as follows:

- A novel model (CloudMTD) for services selection and composition in CB-SOA based on options theory, DSM and propagation-cost metrics. The novelty of CloudMTD model is based on different aspects, such as (1) a value-driven model for Managing Technical Debt in Cloud-Based Service-Oriented Architectures, (2) the use of Binomial model for modeling selection decisions in CB-SOA, and (3) CloudMTD model quantifies the time-value of selection and composition decisions in CB-SOA, technical debt and dependencies they can imply on the structure. The analysis is done taking into consideration the structural and behavioral aspects of CB-SOA. A distinctive feature of the CloudMTD model is that it links time-value of the substitution decision to both the behavior and the structure of CB-SOA.

- The use of k-means clustering in service selection in CB-SOA. K-means clustering technique is capable of presenting data in a structured way, which enables the decision maker to investigate the structure of each group (cluster) in the given web services dataset. K-means clustering technique can improve service selection in CB-SOA by improving the following:

  1. Scalability: K-means improve scalability by reducing the search-space in the cloud marketplace and accordingly will reduce searching-time. And consequently it will be easier to consider the context of services composition.

  2. Relevance and Recall: K-means improve the relevancy of the retrieved web services from a given cluster. In addition, recall metric measures the percentage

of the relevant retrieved web services in a given cluster.

3. QoS correlation: K-means is capable of making QoS correlation explicit by finding structures in data in a given web services dataset.

- A novel Service-level Technical Debt in Cloud-Based Service-Oriented Architectures: We describe the concept of technical debt for cloud-based service selection and composition and discuss its causes. We introduce a new novel dimension of technical debt explicating service-level in CB-SOA.

- Literature Review:

  1. Web service selection and composition: A representative sample of research works that have been done in the field of web service selection and composition.

  2. Technical Debt and Managing Technical Debt: This review presents the available definitions of technical debt in different fields and on different levels in the literature. The review also presents the available approaches, which have been investigating technical debt. It provides a comparison of different approaches dealing with technical debt on different levels, dimensions, causes, solutions, and evaluation methods.

  3. Real Options: A review of the available research works that investigate the option theory in software engineering, economics, SOA and IT investments.

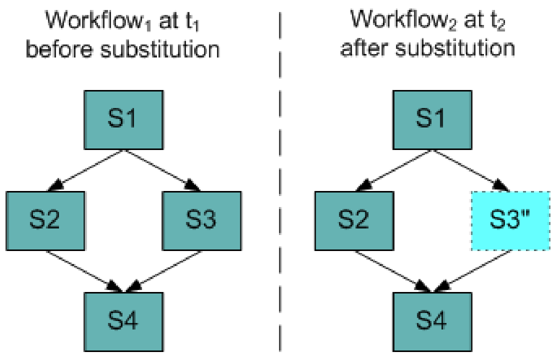
8.3 Future Work

- This thesis presented many paths those are related to service selection and composition in Cloud-Based Service-Oriented Architectures and managing the likely technical debt and dependencies among constituent services.

- Binomial coefficients estimation: in chapter 5, we presented the binomial model and its coefficients estimation methods. We aim to extend these estimation methods by presenting other possible ways that help in:
  - Performing finer what-if analysis.
  - Getting more cases out of the simulation.
  - Deduce possible values of these coefficients.

Other possible usage of SoapUI is to compare two different WS-BPEL workflows (SoapUI and WS-BPEL were explained in the previous chapter). For example, let $workflow_1$ to be the initial WS-BPEL workflow at t_2 and $workflow_2$ is the recomposed workflow following substitution at t_2, as seen in Figure 8.1. Here, we are interested in comparing and testing both workflows in terms of load and performance at different timestamps. Given a likely load, stress testing using SoapUI can reveal best and worst performance of the given compositions, which can be used to calculate u, d and p binomial coefficients. Here, binomial coefficients are extracted from SoapUI simulation as an alternative of other eliciting methods that were described previously. Such information can help in modeling the likely added-value or "sacrifices" which is expressed as utility. In this sense, we are interested in investigating the behavioral characteristics of the given CB-SOA architecture, before and after substitution.

Figure 8.1 Workflows at t_1 and t_2.



- Outliers analysis: k-means clustering has been covered in chapter 6, section 6.1.2. By using k-means clustering, some services appear to be as outliers. Outliers are services, which are not well-clustered. This can be attributed to the fact that these services have varying QoS correlations, which is different than other services in a given cluster. Accordingly, these outliers (services) have varying utility levels. In this sense, outliers can be viewed as a new dimension of technical debt in Cloud-Based Service-Oriented Architectures.

Bibliography

[1] P. F. A. Albreshne and J. Pasquier-Rocha. Web services orchestration and composition: Case study of web services composition. Department of Informatics Internal Working Paper no 09-03, University of Fribourg, Switzerland, September 2009.

[2] H. Abbas, L. Yngstrom, and A. Hemani. Option based evaluation: Security evaluation of it products based on options theory. In *The First IEEE Eastern European Conference on the Engineering of Computer Based Systems*, pages 134–141, Sept 2009.

[3] S. Abolfazli, Z. Sanaei, A. Gani, and M. Shiraz. MOMCC: market-oriented architecture for mobile˜cloud˜computing based on service˜oriented˜architecture. *CoRR*, abs/1206.6209, 2012.

[4] T. R. Adler, J. G. Leonard, and R. K. Nordgren. Improving risk management: moving from risk elimination to risk avoidance. *Information and Software Technology*, 41(1):29 – 34, 1999.

[5] S. I. Ahamed and M. Sharmin. A trust-based secure service discovery (TSSD) model for pervasive computing. *Comput. Commun.*, 31:4281–4293, December 2008.

[6] G. Alferez and V. Pelechano. Facing uncertainty in web service compositions. In *IEEE 20th International Conference on Web Services (ICWS)*, pages 219–226, June 2013.

[7] S. M. Ali Shah, M. Torchiano, A. Vetro, and M. Morisio. Exploratory testing as a source of technical debt. *IT Professional*, 16(3):44–51, May 2014.

[8] E. Allman. Managing technical debt. *Queue*, 10(3):10:10–10:17, Mar. 2012.

[9] E. AlMasri and Q. H. Mahmoud. QoS-based Discovery and Ranking of Web Services. In *ICCCN*, pages 529–534, 2007.

[10] E. AlMasri and Q. H. Mahmoud. Investigating web services on the world wide web. In *Proceeding of the 17th international conference on World Wide Web*, WWW '08, pages 795–804, New York, NY, USA, 2008. ACM.

[11] E. AlMasri and Q. H. Mahmoud. Discovering the Best Web Service: A Neural Network-based Solution. In *SMC*, pages 4250–4255, 2009.

[12] E. AlMasri and Q. H. Mahmoud. Web Service Discovery and Client Goals. *Computer*, 42:104–107, 2009.

[13] M. Alrifai and T. Risse. Combining global optimization with local selection for efficient qos-aware service composition. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, pages 881–890, New York, NY, USA, 2009. ACM.

[14] M. Alrifai, T. Risse, P. Dolog, and W. Nejdl. A scalable approach for qos-based web service selection. In G. Feuerlicht and W. Lamersdorf, editors, *Service-Oriented Computing ICSOC 2008 Workshops*, volume 5472 of *Lecture Notes in Computer Science*, pages 190–199. Springer Berlin Heidelberg, 2009.

[15] M. Alrifai, T. Risse, and W. Nejdl. A hybrid approach for efficient web service composition with end-to-end qos constraints. *ACM Trans. Web*, 6(2):7:1–7:31, June 2012.

[16] M. Alrifai, D. Skoutas, and T. Risse. Selecting skyline services for qos-based web service composition. In *WWW'10*, pages 11–20, 2010.

[17] N. Alves, L. Ribeiro, V. Caires, T. Mendes, and R. Spinola. Towards an ontology of terms on technical debt. In *Sixth International Workshop on Managing Technical Debt*, pages 1–7, Sept 2014.

[18] E. Alzaghoul and R. Bahsoon. CloudMTD: Using Real Options to Manage Technical Debt in Cloud-Based Service Selection. In *Proceeding of the Fourth International Workshop on Managing Technical Debt, in conjunction with 35th International Conference on Software Engineering (ICSE)*. ACM, 2013.

[19] E. Alzaghoul and R. Bahsoon. Economics-driven Approach for Managing Technical Debt in Cloud-Based Architectures. In *Proceeding of the 6th IEEE/ACM International Conference on Utility and Cloud Computing*. ACM, 2013.

[20] E. Alzaghoul and R. Bahsoon. Evaluating Technical Debt in Cloud-based Architectures using Real Options. In *Proceeding of the 23rd Australasian Software Engineering Conference (ASWEC) Co-located with WICSA 2104*. ACM, 2014.

[21] M. Amram and N. Kulatilaka. *Real Options: Managing Strategic Investment in an Uncertain World*. Harvard Business School Press, 1999.

[22] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. Optics: Ordering points to identify the clustering structure. In *ACM SIGMOD International Conference on Management of Data*, SIGMOD '99, pages 49–60, New York, NY, USA, 1999. ACM.

[23] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Commun. ACM*, 53:50–58, April 2010.

[24] L. Auer. Soa investment decision-making using real options analysis. *International Journal of Services, Economics and Management*, 5(1):21–40, 2013.

[25] S. Baccar, M. Rouached, and M. Abid. A goal oriented approach for web services selection and composition. In *Digital Information Management (ICDIM), 2013 Eighth International Conference on*, pages 102–107, Sept 2013.

[26] R. Bahsoon. *Evaluating architectural stability with real options theory.* PhD thesis, University College London (University of London), 2006.

[27] R. Bahsoon and W. Emmerich. ArchOptions: A Real Options-Based Model for Predicting the Stability of Software Architectures. In *Proceedings of the The ICSE 2003 Workshop on Economics-Driven Software Engineering Research*, Portland, Oregon, USA.

[28] R. Bahsoon and W. Emmerich. An Economics-Driven Approach for Valuing Scalability in Distributed Architectures. In *Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA)*, pages 9–18, Washington, DC, USA, 2008. IEEE Computer Society.

[29] R. Bahsoon, W. Emmerich, and J. Macke. Using Real Options to Select Stable Middleware-Induced Software Architectures. pages 167–186. IEEE Computer Society, 2005.

[30] C. Y. Baldwin and K. B. Clark. *Design Rules: The Power of Modularity Volume 1.* MIT Press, Cambridge, MA, USA, 1999.

[31] F. Barbon, P. Traverso, M. Pistore, and M. Trainotti. Run-time monitoring of instances and classes of web service compositions. In *ICWS*, pages 63–71. IEEE Computer Society, 2006.

[32] I. Bardhan, S. Bagchi, and R. Sougstad. A real options approach for prioritization of a portfolio of information technology projects: a case study of a utility company. In *Proceedings of the 37th Annual Hawaii International Conference on System Sciences*, Jan 2004.

[33] M. Benaroch. Managing information technology investment risk: A real options perspective. *J. Manage. Inf. Syst.*, 19(2):43–84, 2002.

[34] M. Benaroch, Q. Dai, and R. Kauffman. Should we go our own way? backsourcing flexibility in it services contracts. *J. Manage. Inf. Syst.*, 26(4):317–358, 2010.

[35] M. Benaroch and J. Goldstein. An integrative economic optimization approach to systems development risk management. *IEEE Trans. Softw. Eng.*, 35(5):638–653, 2009.

[36] M. Benaroch, M. Jeffery, R. Kauffman, and S. Shah. Option-based risk management: A field study of sequential information technology investment decisions. *J. Manage. Inf. Syst.*, 24(2):103–140, 2007.

[37] M. Benaroch and R. J. Kauffman. A case for using real options pricing analysis to evaluate information technology project investment. *Info. Sys. Research*, 10(1):70–86, Mar. 1999.

[38] M. Benaroch, Y. Lichtenstein, and K. Robinson. Real options in information technology risk management: an empirical validation of risk-option relationships. *MIS Q.*, 30(4):827–864, 2006.

[39] M. Benaroch, S. Shah, and M. Jeffery. On the valuation of multistage information technology investments embedding nested real options. *J. Manage. Inf. Syst.*, 23(1):239–261, 2006.

[40] P. Bianco, R. Kotermanski, and P. Merson. Evaluating a service-oriented architecture (cmu/sei-2007-tr-015). *Software Engineering Institute, Carnegie Mellon University*, September 2007.

[41] F. Black and M. S. Scholes. The Pricing of Options and Corporate Liabilities. *Journal of Political Economy*, 81(3):637–54, May-June 1973.

[42] S. Black, P. Boca, J. Bowen, J. Gorman, and M. Hinchey. Formal versus agile: Survival of the fittest. *Computer*, 42(9):37 –45, sept. 2009.

[43] E. Blanzieri, P. Giorgini, P. Massa, and S. Recla. Implicit culture for multi-agent interaction support. In C. Batini, F. Giunchiglia, P. Giorgini, and M. Mecella, editors, *Cooperative Information Systems*, volume 2172 of *Lecture Notes in Computer Science*, pages 27–39. Springer Berlin Heidelberg, 2001.

[44] B. Boehm. Value-based software engineering. *SIGSOFT Softw. Eng. Notes*, 28(2):4–, Mar. 2003.

[45] B. W. Boehm and K. J. Sullivan. Software Economics: a Roadmap. In *ICSE - Future of SE Track*, pages 319–343, 2000.

[46] J. Bohnet and J. Döllner. Monitoring code quality and development activity by software maps. In *Proceedings of the 2Nd Workshop on Managing Technical Debt*, MTD '11, pages 9–16, New York, NY, USA, 2011. ACM.

[47] S. Borzsony, D. Kossmann, and K. Stocker. The skyline operator. In *Data Engineering, 2001. Proceedings. 17th International Conference on*, pages 421–430, 2001.

[48] N. Brown, Y. Cai, Y. Guo, R. Kazman, M. Kim, P. Kruchten, E. Lim, A. MacCormack, R. Nord, I. Ozkaya, R. Sangwan, C. Seaman, K. Sullivan, and N. Zazworka. Managing technical debt in software-reliant systems. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, FoSER '10, pages 47–52, New York, NY, USA, 2010. ACM.

[49] N. Brown, R. Nord, and I. Ozkaya. Enabling agility through architecture. *Crosstalk ARCHITECTURE TODAY*, pages 12–17, Nov/Dec 2010.

[50] N. Brown, R. L. Nord, I. Ozkaya, and M. Pais. Analysis and management of architectural dependencies in iterative release planning. In *Proceedings of the 2011 Ninth Working IEEE/IFIP Conference on Software Architecture*, WICSA '11, pages 103–112, Washington, DC, USA, 2011. IEEE Computer Society.

[51] W. J. Brown, R. C. Malveau, H. W. S. McCormick, and T. J. Mowbray. *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis: Refactoring Software, Architecture and Projects in Crisis.* John Wiley and Sons, 1. auflage edition, 1998.

[52] R. Buyya. Market-oriented cloud computing: Vision, hype, and reality of delivering computing as the 5th utility. In *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, CCGRID '09, Washington, DC, USA, 2009. IEEE Computer Society.

[53] R. Buyya, S. Pandey, and C. Vecchiola. *Market-Oriented Cloud Computing and The Cloudbus Toolkit*, pages 319–358. John Wiley and Sons, Inc., 2013.

[54] R. Buyya, C. S. Yeo, and S. Venugopal. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In *Proceedings of the 2008 10th IEEE International Conference on High Performance Computing and Communications*, HPCC '08, pages 5–13, Washington, DC, USA, 2008. IEEE Computer Society.

[55] 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 Gener. Comput. Syst.*, 25:599–616, June 2009.

[56] Y. Cai. *Modularity in Design: Formal Modeling and Automated Analysis.* PhD thesis, Charlottesville, VA, USA, 2006. AAI3225953.

[57] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani. An approach for qos-aware service composition based on genetic algorithms. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*, GECCO '05, pages 1069–1075, New York, NY, USA, 2005. ACM.

[58] J. Carriere, R. Kazman, and I. Ozkaya. A cost-benefit framework for making architectural decisions in a business context. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2*, ICSE '10, pages 149–157, New York, NY, USA, 2010. ACM.

[59] S. Chaiworawitgul and D. Sutivong. The options approach to hardware design decision: Switching from object-oriented to aspect-oriented concepts. In *Engineering Management Conference, 2006 IEEE International*, pages 204–208, Sept 2006.

[60] X. Chen, X. Liu, Z. Huang, and H. Sun. Regionknn: A scalable hybrid collaborative filtering algorithm for personalized web service recommendation. In *IEEE International Conference on Web Services*, pages 9–16, July 2010.

[61] E. Collins, A. Dias-Neto, and V. de Lucena. Strategies for agile software testing automation: An industrial experience. In *Computer Software and Applications Conference Workshops (COMPSACW), 2012 IEEE 36th Annual*, pages 440–445, July 2012.

[62] D. Comes, H. Baraki, R. Reichle, M. Zapf, and K. Geihs. Heuristic approaches for qos-based service selection. In P. Maglio, M. Weske, J. Yang, and M. Fantinato, editors, *Service-Oriented Computing*, volume 6470 of *Lecture Notes in Computer Science*, pages 441–455. Springer Berlin Heidelberg, 2010.

[63] M. Comuzzi and B. Pernici. A framework for qos-based web service contracting. *ACM Trans. Web*, 3(3):10:1–10:52, July 2009.

[64] E. Costante, F. Paci, and N. Zannone. Privacy-aware web service composition and ranking. In *ICWS*, pages 131–138. IEEE, 2013.

[65] J. C. Cox and S. A. Ross. The valuation of options for alternative stochastic processes. *Journal of Financial Economics*, 3(12):145 – 166, 1976.

[66] J. C. Cox, S. A. Ross, and M. Rubinstein. Option pricing: A simplified approach. *Journal of Financial Economics*, 7(3):229–263, 1979.

[67] M. Crasso, A. Zunino, and M. Campo. Easy web service discovery: A query-by-example approach. *Sci. Comput. Program.*, 71:144–164, April 2008.

[68] W. Cunningham. The wycash portfolio management system. *SIGPLAN OOPS Mess.*, 4(2):29–30, Dec. 1992.

[69] B. Curtis, J. Sappidi, and A. Szynkarski. Estimating the size, cost, and types of technical debt. In *Managing Technical Debt (MTD), 2012 Third International Workshop on*, pages 49–53, June 2012.

[70] S. Dasgupta and Y. Freund. Random projection trees for vector quantization. *Information Theory, IEEE Transactions on*, 55(7):3229–3242, July 2009.

[71] J. de Groot, A. Nugroho, T. Back, and J. Visser. What is the value of your software? In *Managing Technical Debt (MTD), 2012 Third International Workshop on*, pages 37–44, June 2012.

[72] X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang. Similarity search for web services. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB '04, pages 372–383. VLDB Endowment, 2004.

[73] M. Dumas, Y. Yang, and L. Zhang. Improving web service survivability via gracefully degraded substitution. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference on*, volume 1, pages 597 –600, 31 2010-sept. 3 2010.

[74] R. J. Eisenberg. A threshold based approach to technical debt. *SIGSOFT Softw. Eng. Notes*, 37(2):1–6, Apr. 2012.

[75] J. El Hadad, M. Manouvrier, and M. Rukoz. Tqos: Transactional and qos-aware selection algorithm for automatic web service composition. *Services Computing, IEEE Transactions on*, 3(1):73–85, Jan 2010.

[76] K. Elgazzar, A. Hassan, and P. Martin. Clustering wsdl documents to bootstrap the discovery of web services. In *IEEE International Conference on Web Services*, pages 147–154, July 2010.

[77] H. Erdogmus and J. Favaro. Keep Your Options Open: Extreme Programming and Economics of Flexibility. In *In*, pages 503–552. Addison Wesley, 2002.

[78] H. Erdogmus, J. Vandergraaf, and J. V. Quantitative Approaches for Assessing the Value of COTS-centric Development. In *Sixth International Symposium on Software Metrics, Boca*, 1999.

[79] D. Falessi, M. Shaw, F. Shull, K. Mullen, and M. Keymind. Practical considerations, challenges, and requirements of tool-support for managing technical debt. In *the 4th International Workshop on Managing Technical Debt*, pages 16–19, May 2013.

[80] J. M. Favaro, K. R. Favaro, and P. F. Favaro. Value based software reuse investment. *Ann. Softw. Eng.*, 5:5–52, Jan. 1998.

[81] F. Fontana, V. Ferme, and S. Spinelli. Investigating the impact of code smells debt on quality code evaluation. In *Third International Workshop on Managing Technical Debt*, pages 15–22, June 2012.

[82] M. Fowler. Technical debt quadrant, http://martinfowler.com/bliki.

[83] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts. *Refactoring: improving the design of existing code.* Addison-Wesley Professional, 1999.

[84] X. Fu, K. Yue, L. Liu, P. Zou, and Y. Feng. Discovering admissible web services with uncertain qos. *Frontiers of Computer Science*, 9(2):265–279, 2015.

[85] J. Gai and Y. Du. An efficient algorithm of service discovery based on service clusters. *Journal of Software Engineering*, 8(2):100–107, 2014.

[86] D. Garlan. Software architecture: A travelogue. In *Proceedings of the on Future of Software Engineering*, FOSE 2014, pages 29–39, New York, NY, USA, 2014. ACM.

[87] D. Garlan, F. Bachmann, J. Ivers, J. Stafford, L. Bass, P. Clements, and P. Merson. *Documenting Software Architectures: Views and Beyond.* Addison-Wesley Professional, 2nd edition, 2010.

[88] I. Gat. Revolution in software: Using technical debt techniques to govern the software development process. Cutter Consortium, 2010.

[89] M. Gaynor and S. Bradner. A real options framework to value network, protocol, and service architecture. *SIGCOMM Comput. Commun. Rev.*, 34(5):31–38, Oct. 2004.

[90] M. Gaynor, S. Bradner, M. Iansiti, and H. Kung. The real options approach to standards for building network-based services. In *The 2nd IEEE Conference Standardization and Innovation in Information Technology*, pages 217–228, 2001.

[91] A. Goscinski and M. Brock. Toward dynamic and attribute based publication, discovery and selection for cloud computing. *Future Gener. Comput. Syst.*, 26:947–970, July 2010.

[92] I. Griffith, C. Izurieta, H. Taffahi, and D. Claudio. A simulation study of practical methods for technical debt management in agile software development. In *Proceedings of the 2014 Winter Simulation Conference*, WSC '14, pages 1014–1025, Piscataway, NJ, USA, 2014. IEEE Press.

[93] Y. Guo and C. Seaman. A portfolio approach to technical debt management. In *Proceedings of the 2nd Workshop on Managing Technical Debt*, MTD '11, pages 31–34, New York, NY, USA, 2011. ACM.

[94] Y. Guo, C. Seaman, R. Gomes, A. Cavalcanti, G. Tonin, F. da Silva, A. L. M. Santos, and C. Siebra. Tracking technical debt - an exploratory case study. In *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*, pages 528–531, 2011.

[95] H. Gustavsson and J. Axelsson. Evaluating flexibility in embedded automotive product lines using real options. In *Software Product Line Conference, 2008. SPLC '08. 12th International*, pages 235–242, Sept 2008.

[96] T. Han and K. M. Sim. An ontology-enhanced cloud service discovery system. In *International MultiConference of Engineers and Computer Scientists*, 2010.

[97] J. Heidenberg and I. Porres. Metrics functions for kanban guards. In *Engineering of Computer Based Systems (ECBS), 2010 17th IEEE International Conference and Workshops on*, pages 306–310, 2010.

[98] B. Heinrich, A. Huber, and S. Zimmermann. Make-and-sell or buy of web services a real option approach. In *ECIS*, 2011.

[99] T. N. Herzog and G. Lord. *Applications of Monte Carlo methods to finance and insurance.* Actex Publications, 2002.

[100] A. Heß and N. Kushmerick. Learning to attach semantic metadata to web services. In *The Semantic Web-ISWC 2003*, pages 258–273. Springer, 2003.

[101] J. Highsmith. *Agile Project Management: Creating Innovative Products.* Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2004.

[102] A. F. M. Huang, C.-W. Lan, and S. J. H. Yang. An optimal QoS-based Web service selection scheme. *Inf. Sci.*, 179:3309–3322, September 2009.

[103] J. C. Hull. *Options, Futures, and Other Derivatives.* Pearson, Essex, England, eighth edition, 2012.

[104] S.-Y. Hwang, H. Wang, J. Tang, and J. Srivastava. A probabilistic approach to modeling and estimating the qos of web-services-based workflows. *Inf. Sci.*, 177(23):5484–5503, Dec. 2007.

[105] M. Jaeger and H. Ladner. Improving the qos of ws compositions based on redundant services. In *Next Generation Web Services Practices, 2005. NWeSP 2005. International Conference on*, page 6, Aug 2005.

[106] M. Jaeger, G. Rojec-Goldmann, and G. Muhl. Qos aggregation for web service composition using workflow patterns. In *Enterprise Distributed Object Computing Conference, 2004. EDOC 2004. Proceedings. Eighth IEEE International*, pages 149–159, Sept 2004.

[107] M. Jaeger, G. Rojec-Goldmann, and G. Muhl. QoS aggregation in Web service compositions. In *e-Technology, e-Commerce and e-Service, 2005. EEE '05. Proceedings. The 2005 IEEE International Conference on*, pages 181 – 185, march-1 april 2005.

[108] Y. Jiang, J. Liu, M. Tang, and X. Liu. An effective web service recommendation method based on personalized collaborative filtering. In *Web Services (ICWS), 2011 IEEE International Conference on*, pages 211–218, July 2011.

[109] R. Jurca, B. Faltings, and W. Binder. Reliable qos monitoring based on client feedback. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 1003–1012, New York, NY, USA, 2007. ACM.

[110] G. Kang, J. Liu, M. Tang, X. Liu, and K. Fletcher. Web service selection for resolving conflicting service requests. In *IEEE International Conference on Web Services (ICWS)*, pages 387–394, July 2011.

[111] M. Karlesky, G. Williams, W. Bereza, and M. Fletcher. Mocking the embedded world: Test-driven development, continuous integration, and design patterns. In *Proc. Emb. Systems Conf, CA, USA*, 2007.

[112] R. Kauffman and X. Li. Technology competition and optimal investment timing: a real options perspective. *Engineering Management, IEEE Transactions on*, 52(1):15–29, Feb 2005.

[113] R. Kauffman and R. Sougstad. Risk management of contract portfolios in it services: The profit-at-risk approach. *J. Manage. Inf. Syst.*, 25(1):17–48, July 2008.

[114] R. Kazman, J. Asundi, and M. Klein. Quantifying the costs and benefits of architectural decisions. In *Proceedings of the 23rd International Conference on Software Engineering*, pages 297 – 306, May 2001.

[115] R. Kazman, Y. Cai, R. Mo, Q. Feng, L. Xiao, S. Haziyev, V. Fedak, and A. Shapochka. A case study in locating the architectural roots of technical debt. In *Proceedings of the 37th International Conference on Software Engineering*, 2015.

[116] W. C. Kester. Today's Options for Tomorrow's Growth. *Harvard Business Review*, 62:153–184, 1984.

[117] W. C. Kester. *Turning Growth Options Into Real Assets*. Working paper (Harvard University. Graduate School of Business Administration. Division of Research). Division of Research, Graduate School of Business Administration, Harvard University, 1984.

[118] J. Klein, G. Chastek, S. Cohen, R. Kazman, and J. McGregor. An early look at defining variability requirements for system of systems platforms. In *IEEE Second Workshop on Requirements Engineering for Systems, Services and Systems-of-Systems (RES4)*, pages 30–33, Sept 2012.

[119] T. Klinger, P. Tarr, P. Wagstrom, and C. Williams. An enterprise perspective on technical debt. In *Proceedings of the 2nd Workshop on Managing Technical Debt*, MTD '11, pages 35–38, New York, NY, USA, 2011. ACM.

[120] N. Kokash, A. Birukou, and V. DAndrea. Web service discovery based on past user experience. In *Business Information Systems*, pages 95–107. Springer, 2007.

[121] P. Kruchten. What colour is your backlog? In *Agile Vancouver Conference*, 2009.

[122] P. Kruchten, R. Nord, and I. Ozkaya. Technical debt: From metaphor to theory and practice. *Software, IEEE*, 29(6):18–21, 2012.

[123] O. Ktata and G. Lévesque. Designing and implementing a measurement program for scrum teams: What do agile developers really need and want? In *Proceedings of the Third C* Conference on Computer Science and Software Engineering*, C3S2E '10, pages 101–107, New York, NY, USA, 2010. ACM.

[124] R. Kumar and J. Singh. A review on improvement of qos in web services functions. *International Journal of Computer Applications*, 112(11), 2015.

[125] B. Kumara, I. Paik, and W. Chen. Web-service clustering with a hybrid of ontology learning and information-retrieval-based term similarity. In *IEEE 20th International Conference on Web Services (ICWS)*, pages 340–347, June 2013.

[126] B. Kumara, Y. Yaguchi, I. Paik, and W. Chen. Clustering and spherical visualization of web services. In *IEEE International Conference on Services Computing (SCC)*, pages 89–96, June 2013.

[127] J.-L. Letouzey. The sqale method for evaluating technical debt. In *Managing Technical Debt (MTD), 2012 Third International Workshop on*, pages 31–36, June 2012.

[128] J.-L. Letouzey and M. Ilkiewicz. Managing technical debt with the sqale method. *IEEE Software*, 29(6):44–51, 2012.

[129] X. Li, J. Wu, and S. Lu. Qos-aware service selection in geographically distributed clouds. In *Computer Communications and Networks (ICCCN), 2013 22nd International Conference on*, pages 1–5, July 2013.

[130] Z. Li, P. Avgeriou, and P. Liang. A systematic mapping study on technical debt and its management. *Journal of Systems and Software*, 101(0):193 – 220, 2015.

[131] Z. Li, Z. Bin, L. Ying, G. Yan, and Z. Zhi-Liang. A web service qos prediction approach based on collaborative filtering. In *IEEE Asia-Pacific Services Computing Conference*, pages 725–731, Dec 2010.

[132] Z. Li, P. Liang, and P. Avgeriou. *Architectural Debt Management in Value-oriented Architecting*. Elsevier, 1 edition, 2014.

[133] Z. Li, P. Liang, P. Avgeriou, N. Guelfi, and A. Ampatzoglou. An empirical investigation of modularity metrics for indicating architectural technical debt. In *Proceedings of the 10th International Conference on the Quality of Software Architectures (QoSA)*. ACM, 2014.

[134] Z. Lia, P. Liangb, and P. Avgerioua. Architecture viewpoints for documenting architectural technical debt. In *Software Quality Assurance in Large Scale and Complex Software-intensive Systems*. Elsevier, 2015.

[135] E. Lim, N. Taksande, and C. Seaman. A balancing act: What software practitioners have to say about technical debt. *Software, IEEE*, 29(6):22–27, 2012.

[136] T. Lin and H. Kang. The market entry/exit model on the free internet service firm. In *Industrial Engineering and Engineering Management, 2008. IEEM 2008. IEEE International Conference on*, pages 1605–1609, Dec 2008.

[137] W.-L. Lin, C.-C. Lo, K.-M. Chao, and N. Godwin. Fuzzy similarity clustering for consumer-centric qos-aware selection of web services. In *Complex, Intelligent and Software Intensive Systems, 2009. CISIS '09. International Conference on*, pages 904–909, March 2009.

[138] D. S. Linthicum. *Cloud Computing and SOA Convergence in Your Enterprise: A Step-by-Step Guide*. Addison-Wesley Professional, 1st edition, 2009.

[139] H. Liu, F. Zhong, B. Ouyang, and J. Wu. An approach for qos-aware web service composition based on improved genetic algorithm. In *Web Information Systems and Mining (WISM), 2010 International Conference on*, volume 1, pages 123–128, Oct 2010.

[140] J. Liu, D. Ning, K. Xing, Z. Tian, P. Liu, and F. Liu. Web service aggregation platform implementation based on join operation. In *International Conference on Advances in Mechanical Engineering and Industrial Informatics*. Atlantis Press, 2015.

[141] W. Lo, J. Yin, S. Deng, Y. Li, and Z. Wu. An extended matrix factorization approach for qos prediction in service selection. In *Services Computing (SCC), 2012 IEEE Ninth International Conference on*, pages 162–169, June 2012.

[142] J. Ma, Y. Zhang, and J. He. Efficiently finding web services using a clustering semantic approach. In *Proceedings of the 2008 international workshop on Context enabled source and service selection, integration and adaptation: organized with the 17th International World Wide Web Conference (WWW 2008)*, CSSSIA '08, pages 5:1–5:8, New York, NY, USA, 2008. ACM.

[143] J. Ma, Y. Zhang, and J. He. Web Services Discovery Based on Latent Semantic Approach. *IEEE International Conference on Web Services*, 0:740–747, 2008.

[144] A. MacCormack, C. Baldwin, and J. Rusnak. Exploring the duality between product and organizational architectures: A test of the mirroring hypothesis. *Research Policy*, 41(8):1309 – 1324, 2012.

[145] A. MacCormack, J. Rusnak, and C. Y. Baldwin. Exploring the structure of complex software designs: An empirical study of open source and proprietary code. *Management Science*, 52(7):1015–1030, 2006.

[146] J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA., 1967.

[147] M. Makhlughian, S. M. Hashemi, Y. Rastegari, and E. Pejman. Web service selection based on ranking of qos using associative classification. *CoRR*, abs/1204.1425, 2012.

[148] R. Marinescu. Assessing technical debt by identifying design flaws in software systems. *IBM Journal of Research and Development*, 56(5):9:1–9:13, Sept 2012.

[149] R. Marinescu and C. Marinescu. Are the clients of flawed classes (also) defect prone? In *Source Code Analysis and Manipulation (SCAM), 2011 11th IEEE International Working Conference on*, pages 65–74, Sept 2011.

[150] M. Matskin and J. Rao. Value-added web services composition using automatic program synthesis. In C. Bussler, R. Hull, S. McIlraith, M. Orlowska, B. Pernici, and J. Yang, editors, *Web Services, E-Business, and the Semantic Web*, volume 2512 of *Lecture Notes in Computer Science*, pages 213–224. Springer Berlin Heidelberg, 2002.

[151] A. Mavridis, A. Ampatzoglou, I. Stamelos, P. Sfetsos, and I. Deligiannis. Selecting refactorings: An option based approach. In *The Eighth International Conference on the Quality of Information and Communications Technology (QUATIC)*, pages 272–277, Sept 2012.

[152] S. McConnell. 10x softwar e development: Technical debt, http://www.construx.com. 2007.

[153] S. McConnell". Managing technical debt (talk). In *Proceeding of the Fourth International Workshop on Managing Technical Debt, in conjunction with 35th International Conference on Software Engineering (ICSE)*, ICSE '13, 2013.

[154] A. Meligy and P. El-Kafrawy. A web service discovery based on qos negotiation approach. *International Journal of Computer Applications*, 111(13), 2015.

[155] D. Menasce. Qos issues in web services. *Internet Computing, IEEE*, 6(6):72–75, Nov 2002.

[156] R. C. Merton. Theory of Rational Option Pricing. *Bell Journal of Economics*, 4(1):141–183, Spring 1973.

[157] R. C. Merton. Applications of option-pricing theory: Twenty-five years later. *American Economic Review*, 88(3):323–49, 1998.

[158] D. S. Modha and W. S. Spangler. Feature weighting in k-means clustering. *Machine learning*, 52(3):217–237, 2003.

[159] M. Moghaddam and J. Davis. Service selection in web service composition: A comparative review of existing approaches. In A. Bouguettaya, Q. Z. Sheng, and F. Daniel, editors, *Web Services Foundations*, pages 321–346. Springer New York, 2014.

[160] S. B. Mokhtar, D. Preuveneers, N. Georgantas, V. Issarny, and Y. Berbers. EASY: Efficient semAntic Service discoverY in pervasive computing environments with QoS and context support. *Journal of Systems and Software*, 81(5):785 – 808, 2008. Software Process and Product Measurement.

[161] S. C. Myers. Determinants of corporate borrowing. *Journal of Financial Economics*, 5(2):147 – 175, 1977.

[162] S. C. Myers. Finance theory and financial strategy. *Interfaces*, 14(1):126–137, 1984.

[163] S. C. Myers. Abandonment value and project life. volume 4 of *Advances in futures and options research : a research annual*, pages 1–21. JAI Press, 1990.

[164] K. Nakamura and M. Aoyama. Value-based dynamic composition of web services. In *13th Asia Pacific Software Engineering Conference APSEC*, pages 139–146, Dec 2006.

[165] V. Nallur and R. Bahsoon. A decentralized self-adaptation mechanism for service-based applications in the cloud. *IEEE Transactions on Software Engineering*, 99(PrePrints):1–1, 2012.

[166] C. Neill and P. Laplante. Paying down design debt with strategic refactoring. *Computer*, 39(12):131–134, Dec 2006.

[167] R. Nord, I. Ozkaya, P. Kruchten, and M. Gonzalez-Rojas. In search of a metric for managing architectural technical debt. In *2012 Joint Working IEEE/IFIP Conference on Software Architecture (WICSA) and 6th European Conference on Software Architecture (ECSA)*, pages 91–100, 2012.

[168] R. L. Nord, I. Ozkaya, R. S. Sangwan, J. Delange, M. Gonzalez, and P. Kruchten. Variations on using propagation cost to measure architecture modifiability properties. In *29th IEEE International Conference on Software Maintenance (ICSM)*, pages 400–403, 2013.

[169] D. North. A tutorial introduction to decision theory. *Systems Science and Cybernetics, IEEE Transactions on*, 4(3):200–210, Sept 1968.

[170] A. Nugroho, J. Visser, and T. Kuipers. An empirical model of technical debt and interest. In *Proceedings of the 2nd Workshop on Managing Technical Debt*, MTD '11, pages 1–8, New York, NY, USA, 2011. ACM.

[171] B. Ojameruaye and R. Bahsoon. Systematic elaboration of compliance requirements using compliance debt and portfolio theory. In C. Salinesi and I. van de Weerd, editors, *Requirements Engineering: Foundation for Software Quality*, volume 8396 of *Lecture Notes in Computer Science*, pages 152–167. Springer International Publishing, 2014.

[172] I. Ozkaya, R. Kazman, and M. Klein. Quality-attribute based economic valuation of architectural patterns. In *Proceedings of the 29th International Conference on Software Engineering Workshops*, ICSEW '07, pages 84–, Washington, DC, USA, 2007. IEEE Computer Society.

[173] P. Papakos, L. Capra, and D. S. Rosenblum. Volare: Context-aware adaptive cloud service discovery for mobile systems. In *Proceedings of the 9th International Workshop on Adaptive and Reflective Middleware*, ARM '10, pages 32–38, New York, NY, USA, 2010. ACM.

[174] J. Pathak, S. Basu, and V. Honavar. On Context-Specific Substitutability of Web Services. In *Web Services, 2007. ICWS 2007. IEEE International Conference on*, pages 192 –199, july 2007.

[175] R. S. Pindyck. Irreversible investment, capacity choice, and the value of the firm. *American Economic Review*, 78(5):969–85, December 1988.

[176] M. Pistore, F. Barbon, P. Bertoli, D. Shaparau, and P. Traverso. Planning and monitoring web service composition. In C. Bussler and D. Fensel, editors, *Artificial Intelligence: Methodology, Systems, and Applications*, volume 3192 of *Lecture Notes in Computer Science*, pages 106–115. Springer Berlin Heidelberg, 2004.

[177] F. Probst and M. Lutz. Giving Meaning to GI Web Service Descriptions. In *2nd International Workshop on Web Services: Modeling, Architecture and Infrastructure*, 2004.

[178] Z. Racheva and M. Daneva. Using measurements to support real-option thinking in agile software development. In *Proceedings of the 2008 International Workshop on Scrutinizing Agile Practices or Shoot-out at the Agile Corral*, APOS '08, pages 15–18, New York, NY, USA, 2008. ACM.

[179] Z. Racheva and M. Daneva. How do real options concepts fit in agile requirements engineering? In *The Eighth ACIS International Conference on Software Engineering Research, Management and Applications (SERA)*, pages 231–238, May 2010.

[180] Z. Racheva, M. Daneva, and L. Buglione. Complementing measurements and real options concepts to support inter-iteration decision-making in agile projects. In *The 34th Euromicro Conference of Software Engineering and Advanced Applications*, pages 457–464, Sept 2008.

[181] K. K. Reddy D, K. Maralla, R. K. G, and M. Thirumaran. A greedy approach with criteria factors for qos based web service discovery. In *Proceedings of the 2Nd Bangalore Annual Compute Conference*, COMPUTE '09, pages 12:1–12:5, New York, NY, USA, 2009. ACM.

[182] T. Roth. Application of real options analyses for service oriented computing projects. Faculty of Economics, University of Zurich, 2007.

[183] J. Rothman. An incremental technique to pay off testing technical debt. *stickyminds, Weekly Column*, 2006.

[184] P. J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20(0):53 – 65, 1987.

[185] F. Safy, M. El-Ramly, and A. Salah. Runtime monitoring of soa applications: Importance, implementations and challenges. In *IEEE 7th International Symposium on Service Oriented System Engineering (SOSE)*, pages 315–319, March 2013.

[186] M. Sánchez and G. Milanesi. Evaluation of software development investments: a real options approach. In *Proc. Argentine Symposium on Software Engineering, parte de JAIIO*, pages 49–60, 2011.

[187] G. R. Santhanam, S. Basu, and V. Honavar. Web Service Substitution Based on Preferences Over Non-functional Attributes. In *Proceedings of the 2009 IEEE International Conference on Services Computing*, SCC '09, pages 210–217, Washington, DC, USA, 2009. IEEE Computer Society.

[188] K. Schmid. Technical debt — from metaphor to engineering guidance: A novel approach based on cost estimation. (1/2013, SSE 1/13/E), 2013.

[189] M. Schulmerich. Real options in theory and practice. In *Real Options Valuation*, pages 21–69. Springer Berlin Heidelberg, 2010.

[190] E. Schwartz and L. Trigeorgis. *Real Options and Investment Under Uncertainty: Classical Readings and Recent Contributions.* MIT Press, 2004.

[191] C. Seaman and Y. Guo. Measuring and monitoring technical debt. *Advances in Computers*, 82:25–46, 2011.

[192] C. Seaman, Y. Guo, C. Izurieta, Y. Cai, N. Zazworka, F. Shull, and A. Vetro. Using technical debt data in decision making: Potential decision approaches. In *Third International Workshop on Managing Technical Debt (MTD)*, pages 45 –48, june 2012.

[193] A. Segev and E. Toch. Context-based matching and ranking of web services for composition. *Services Computing, IEEE Transactions on*, 2(3):210–222, July 2009.

[194] L. Shao, J. Zhang, Y. Wei, J. Zhao, B. Xie, and H. Mei. Personalized qos prediction for web services via collaborative filtering. In *IEEE International Conference on Web Services*, pages 439–446, July 2007.

[195] B. Sharma, R. K. Thulasiram, P. Thulasiraman, S. K. Garg, and R. Buyya. Pricing cloud compute commodities: A novel financial economic model. In *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, CCGRID '12, pages 451–457, Washington, DC, USA, 2012. IEEE Computer Society.

[196] T. Sharma, G. Samarthyam, and G. Suryanarayana. Applying design principles in practice. In *Proceedings of the 8th India Software Engineering Conference*, pages 200–201. ACM, 2015.

[197] D. M. Sharman and A. A. Yassine. Characterizing complex product architectures. *Syst. Eng.*, 7(1):35–60, Mar. 2004.

[198] F. Shull. Perfectionists in a world of finite resources. *Software, IEEE*, 28(2):4 –6, march-april 2011.

[199] G. Skourletopoulos, C. X. Mavromoustakis, R. Bahsoon, G. Mastorakis, and E. Pallis. Predicting and quantifying the technical debt in cloud software engineering. In *the 19th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, pages 36–40, Dec 2014.

[200] D. Skoutas, D. Sacharidis, A. Simitsis, and T. Sellis. Ranking and clustering web services using multicriteria dominance relationships. *Services Computing, IEEE Transactions on*, 3(3):163–177, July 2010.

[201] W. Snipes, B. Robinson, Y. Guo, and C. Seaman. Defining the decision factors for managing defects: A technical debt perspective. In *Managing Technical Debt (MTD), 2012 Third International Workshop on*, pages 54–60, 2012.

[202] I. Sommerville. *Software Engineering (8th Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[203] I. Sommerville. *Software Engineering*. Addison-Wesley, Harlow, England, 9 edition, 2011.

[204] D. Steward. The design structure system: A method for managing the design of complex systems. *Engineering Management, IEEE Transactions on*, EM-28(3):71–74, Aug 1981.

[205] M. Stochel, M. Wawrowski, and M. Rabiej. Value-based technical debt model and its application. In *The Seventh International Conference on Software Engineering Advances*, 2012.

[206] N. Su, R. Akkiraju, N. Nayak, and R. Goodwin. Valuating service transformation alternatives using real options. In *IEEE International Conference on Service Operations and Logistics, and Informatics*, volume 2, pages 2662–2667, Oct 2008.

[207] K. Sullivan, Y. Cai, B. Hallen, and W. G. Griswold. The Structure and Value of Modularity in Software Design. In *SIGSOFT SOFTWARE ENGINEERING NOTES*, pages 99–108. ACM Press, 2001.

[208] K. Sullivan, P. Chalasani, S. Jha, and V. Sazawal. Software Design as an Investment Activity: A Real Options Perspective. In *UNIVERSITY OF VIRGINIA DEPARTMENT OF COMPUTER SCIENCE*, 1999.

[209] K. J. Sullivan. Software design: the options approach. In *Joint proceedings of the second international software architecture workshop (ISAW-2) and international workshop on multiple perspectives in software development (Viewpoints '96) on SIGSOFT '96 workshops*, ISAW '96, pages 15–18, New York, NY, USA, 1996. ACM.

[210] D. Tamburri, P. Kruchten, P. Lago, and H. van Vliet. What is social debt in software engineering? In *Cooperative and Human Aspects of Software Engineering (CHASE), 2013 6th International Workshop on*, pages 93–96, May 2013.

[211] M. Tang, Y. Jiang, J. Liu, and X. Liu. Location-aware collaborative filtering for qos-based service recommendation. In *IEEE 19th International Conference on Web Services*, pages 202–209, June 2012.

[212] B. Tansey and E. Stroulia. Valuating software service development: Integrating cocomo ii and real options theory. In *The First International Workshop on the Economics of Software and Computation*, pages 8–8, May 2007.

[213] A. Taudes. Software growth options. *J. Manage. Inf. Syst.*, 15(1):165–185, June 1998.

[214] A. Taudes, M. Feurstein, and A. Mild. Options analysis of software platform decisions: A case study. *MIS Q.*, 24(2):227–243, June 2000.

[215] M. Tian, A. Gramm, H. Ritter, and J. Schiller. Efficient selection and monitoring of qos-aware web services with the ws-qos framework. In *IEEE/WIC/ACM International Conference on Web Intelligence*, pages 152–158, Sept 2004.

[216] E. Tom, A. Aurum, and R. Vidgen. An exploration of technical debt. *Journal of Systems and Software*, 86(6):1498 – 1516, 2013.

[217] A. Toosi, R. Thulasiram, and R. Buyya. Financial option market model for federated cloud environments. In *Utility and Cloud Computing (UCC), 2012 IEEE Fifth International Conference on*, pages 3–12, Nov 2012.

[218] R. Torkar, P. Minoves, and J. Garrigós. Adopting free/libre/open source software practices, techniques and methods for industrial use. *Journal of the Association for Information Systems*, 12(1), 2011.

[219] L. Trigeorgis. *Real Options: Managerial Flexibility and Strategy in Resource Allocation.* Mit Press, 1996.

[220] W. M. P. Van Der Aalst, A. H. M. Ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distrib. Parallel Databases*, 14(1):5–51, July 2003.

[221] C. Verhoef. How to implement the future? In *Euromicro Conference, 2000. Proceedings of the 26th*, volume 1, pages 32–47 vol.1, 2000.

[222] P. Wang. Qos-aware web services selection with intuitionistic fuzzy set under consumers vague perception. *Expert Systems with Applications*, 36(3, Part 1):4460 – 4466, 2009.

[223] S. Wang, C.-H. Hsu, Z. Liang, Q. Sun, and F. Yang. Multi-user web service selection based on multi-qos prediction. *Information Systems Frontiers*, 16(1):143–152, 2014.

[224] S. Wang, Z. Zheng, Q. Sun, H. Zou, and F. Yang. Cloud model for service selection. In *Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference on*, pages 666–671, April 2011.

[225] W. Wang, F. Yao, S. De, K. Moessner, and Z. Sun. A ranking method for sensor services based on estimation of service access cost. *Information Sciences*, (0):–, 2015.

[226] J. N. Warfield. Binary matrices in system modeling. *Systems, Man and Cybernetics, IEEE Transactions on*, (5):441–449, 1973.

[227] K. Wiklund, S. Eldh, D. Sundmark, and K. Lundqvist. Technical debt in test automation. In *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*, pages 887–892, April 2012.

[228] Y. Xia, P. Chen, L. Bao, M. Wang, and J. Yang. A qos-aware web service selection algorithm based on clustering. In *IEEE International Conference on Web Services (ICWS)*, pages 428–435, July 2011.

[229] H. Xiong, J. Wu, and J. Chen. K-means clustering versus validation measures: a data-distribution perspective. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 39(2):318–331, 2009.

[230] B. Xu, T. Li, Z. Gu, and G. Wu. Swsds: Quick web service discovery and composition in sewsip. *E-Commerce Technology, IEEE International Conference on, and Enterprise Computing, E-Commerce, and E-Services, IEEE International Conference on*, 0:71, 2006.

[231] H. Yahyaoui, M. Almulla, and H. S. Own. A novel non-functional matchmaking approach between fuzzy user queries and real world web services based on rough sets. *Future Generation Computer Systems*, 35(0):27 – 38, 2014. Special Section: Integration of Cloud Computing and Body Sensor Networks; Guest Editors: Giancarlo Fortino and Mukaddim Pathan.

[232] C.-Y. Yam, A. Baldwin, S. Shiu, and C. Ioannidis. Migration to cloud as real option: Investment decision under uncertainty. In *Proceedings of the 2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, TRUSTCOM '11, pages 940–949, Washington, DC, USA, 2011. IEEE Computer Society.

[233] B. Yoo, V. Choudhary, and T. Mukhopadhyay. Marketplaces or web services? alternate business models for electronic b2b transactions. In *The 44th Hawaii International Conference on System Sciences (HICSS)*, pages 1–10, Jan 2011.

[234] Q. Yu and A. Bouguettaya. Multi-attribute optimization in service selection. *World Wide Web*, 15(1):1–31, 2012.

[235] T. Yu, Y. Zhang, and K.-J. Lin. Efficient algorithms for web services selection with end-to-end qos constraints. *ACM Trans. Web*, 1(1), May 2007.

[236] N. Zazworka, R. O. Spínola, A. Vetro', F. Shull, and C. Seaman. A case study on effectively identifying technical debt. In *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*, EASE '13, pages 42–47, New York, NY, USA, 2013. ACM.

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

[238] L. Zeng, B. Benatallah, A. H.H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-Aware Middleware for Web Services Composition. *IEEE Trans. Softw. Eng.*, 30(5):311–327, May 2004.

[239] L.-J. Zhang, S. Cheng, C. Chang, and Q. Zhou. A pattern-recognition-based algorithm and case study for clustering and selecting business services. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 42(1):102–114, Jan 2012.

[240] X. Zheng, F. Qin, L. Wei, and X. Wang. Truthful load-aware service selection: A mechanism design method. In *International Conference on Electronics, Communications and Computers*, pages 48–54, Feb 2015.

[241] Z. Zheng and M. R. Lyu. Ws-dream: A distributed reliability assessment mechanism for web services. In *DSN*, pages 392–397, 2008.

[242] Z. Zheng, H. Ma, M. Lyu, and I. King. Wsrec: A collaborative filtering based web service recommender system. In *Web Services, 2009. ICWS 2009. IEEE International Conference on*, pages 437–444, July 2009.

[243] Z. Zheng, X. Wu, Y. Zhang, M. Lyu, and J. Wang. Qos ranking prediction for cloud services. *Parallel and Distributed Systems, IEEE Transactions on*, 24(6):1213–1222, June 2013.

[244] G. Zou, Q. Lu, Y. Chen, R. Huang, Y. Xu, and Y. Xiang. Qos-aware dynamic composition of web services using numerical temporal planning. *Services Computing, IEEE Transactions on*, 7(1):18–31, Jan 2014.