UNIVERSITY OF TECHNOLOGY SYDNEY

A FRAMEWORK FOR MANAGEMENT OF CLOUD SERVICES

Hong Thai Tran
University of Technology Sydney

A thesis submitted to Faculty of Engineering and Information Technology
University of Technology Sydney
in fulfilment of the requirements for the thesis of
Doctor of Philosophy in Information System

2018

Certificate of Original Authorship

I certify that the work in this report has not previously been submitted for a degree nor has it been submitted as part of requirements for a degree except as fully acknowledged within the text.

I also certify that the report has been written by me. Any help that I have received in my research work and the preparation of the report itself has been acknowledged. In addition, I certify that all information sources and literature used are indicated in the report.

Signature of Student: Production Note: Signature removed prior to publication.

Date: 10 May 2018

Acknowledgment

First, I would like to thank my principal supervisor Dr George Feuerlicht for his guidance, encouragement, enthusiasm, technical comments, and constructive criticism through all years of my research. I will extremely appreciate his full support during my study. I also would like to express my deep gratitude to my co-supervisor, Professor Didar Zowghi, for her valuable advice and guidance during my PhD study.

Special thanks to the University of Economics, Ho Chi Minh City (UEH), Vietnam International Education Development (VIED) under the Ministry of Education and Training (MoET) and University of Technology Sydney (UTS) for providing a scholarship and other financial support for my study.

To my friends, Thanh Nguyen Van, Son Nguyen Thanh and my colleagues at Vastbit, Ndex and IMT Solutions, many thanks for a vibrant, creative and exciting discussions and experiences that help me to initialize ideas in the early stage.

Dad, thanks for your love and everything you did for me. You are always in my heart and my memory. Without you, this work would not have been possible. Mum, thank for your love, encouragement, and patience to help me complete this study. Sisters, thanks a lot for your inspiration and supports. I have been incredibly fortunate to have love and support from such family.

Finally, to my wife, my daughter, and my son, thank you for besides me through years and encouraging, enduring and inspiring this work. Love you always.

Contents

Certificate of Original Authorship 3
Acknowledgment 5
List of Publications 11
List of Illustrations 13
List of Tables 15
List of Abbreviations 17
Abstract 19
Chapter 1. Introduction 21
1.1. Introduction 22
1.2. Problem Definitions 24
1.2.1. Challenges in cloud service management 24
1.2.2. Research Gaps 26
1.2.3. Research Problem 27
1.2.4. Research Questions 27
1.3. Key Terminologies 29
1.3.1. Enterprise Application 29
1.3.2. Cloud Service 30
1.3.3. Service Consumer 30
1.3.4. Service Incident 30
1.4. Scope of Work 31
1.4.1. Using Payment Service as Examples 31
1.4.2. Service Consumer Perspective 31
1.4.3. Using Availability and Response Time for Service Measurements 32
1.5. Research Methodology 32
1.6. Structure of the Dissertation 34
1.7. Conclusion 36
Chapter 2. Literature Review 37
2.1. Introduction 38
2.2. Background and Concepts 38

2.2.1. Software Engineering 38
2.2.2. Service Oriented Architecture 43
2.2.3. Cloud Computing 47
2.3. Review of Cloud SDLC 53
2.3.1. ITIL Lifecycle for Cloud 54
2.3.2. Extreme Cloud Programming SDLC 56
2.3.3. Cloudification Security Development Lifecycle 57
2.3.4. SOA Cloud Lifecycle Using Service Component Architecture 58
2.3.5. The Service Lifecycle on Cloud 59
2.4. Review of SDLC Phases 61
2.4.1. Requirements Specification 62
2.4.2. Cloud Service Identification 64
2.4.3. Cloud Service Integration 69
2.4.4. Cloud Service Monitoring 70
2.4.5. Cloud-based Application Optimisation 72
2.5. Evaluation of The Existing Work 73
2.6. Conclusion 74
Chapter 3. Research Methodology 76
3.1. Introduction 77
3.2. Research Paradigm 77
3.3. Design Science 78
3.4. Research Techniques and Methods 80
3.4.1. Motivation Research 81
3.4.2. Literature Survey 81
3.4.3. Action Design Research 82
3.4.4. Simulation Research 83
3.5. Conclusion 83
Chapter 4. Cloud Service Consumer SDLC 85
4.1. Introduction 86
4.2. Cloud Service Consumer System Development Lifecycle 87
4.3. Requirements Specification 88
4.3.1. Request for Service 89
4.3.2. Functional Requirements 91

4.3.3. Non-Functional Requirements 92
4.3.4. Other Requirements 92
4.4. Service Identification 94
4.5. Service Integration 96
4.6. Service Monitoring 98
4.7. Optimisation 101
4.8. Evaluation using The Case Study at Family Medical Practice 105
4.8.1. Situation at FMP 105
4.8.2. Requirements of Enterprise Application at FMP 106
4.8.3. Evaluation of SC-SDLC 107
4.8.4. Feedback of the SC-SDLC Phases 110
4.8.5. Design Cycle of SC-SDLC 113
4.9. Conclusion 115
Chapter 5. Service Consumer Framework Implementation 117
5.1. Introduction 118
5.2. SCF Architecture 119
5.3. SCF Modules 121
5.3.1. Service Repository 122
5.3.2. Service Adaptors 123
5.3.3. Workflow Engine 124
5.3.4. Monitoring Centre 125
5.4. SCF Implementation 125
5.4.1. Service Repository Implementation 126
5.4.2. Workflow Engine Implementation 130
5.4.3. Monitoring Centre Implementation 131
5.4.4. Evaluation of SCF 134
5.4.5. Design Cycle of SCF 136
5.5. Conclusion 136
Chapter 6. Failover Strategies for Improving Application Availability 138
6.1. Introduction 139
6.2. SCF Reliability Features 140
6.2.1. Retry Fault Tolerance 141

6.2.2. Recovery Block Fault Tolerance 142
6.2.3. Dynamic Sequential Fault Tolerance 143
6.3. Cloud Service Load Balancing 144
6.4. Implementation of Reliability Strategies and Load Balancing 146
6.4.1. Service Adaptors 146
6.4.2. Workflow Engine 147
6.5. Experimental Verification 148
6.5.1. Case study of payment services 148
6.6. Conclusion 152
Chapter 7. Multi-SIte Monitoring for Application OptimiSation 154
7.1. Introduction 155
7.2. Multi-Site Monitoring Model 156
7.2.1. Enterprise application optimisation strategies 158
7.3. Experimental setup for multi-site monitoring 159
7.4. Conclusion 171
Chapter 8. Conclusion and Future Work 173
8.1. Conclusion 174
8.2. Future Work 177
Bibliography 179
APPENDIX A. LIST OF QUALITY OF SERVICE ATTRIBUTES 188
APPENDIX B. THE FMP BUSINESS REQUIREMENTS 190
APPENDIX C. CLOUD SERVICE ADAPTOR INTERFACES 211
APPENDIX D. QUESTIONNAIRE 223
APPENDIX E. SOFTWARE 225

List of Publications

The following publications were made during this thesis study:

[1] Feuerlicht, G. & Tran, H. T., *"Service consumer framework: Managing Service Evolution from a Consumer Perspective."* In ICEIS-2014. 16th International Conference on Enterprise Information Systems (ICEIS), 2014, Portugal.

[2] Feuerlicht, G. & Tran, H. T., *"Enterprise Application Management in Cloud Computing Context."* In The 8th International Conference on Research and Practical Issues of Enterprise Information Systems (CONFENIS), 2014, Ha Noi, Vietnam. ACM. (Best Paper Award)

[3] Feuerlicht, G., Tran, H. T., *"Adapting Service Development Lifecycle for Cloud,"* In The 17th International Conference on Enterprise Information Systems (ICEIS), 2015, Spain

[4] Tran, H. & Feuerlicht, G., "*Service Repository for Cloud Service Consumer Lifecycle Management."* in The European Conference on Service-Oriented and Cloud Computing (ESOCC), 2015, Taormina, Italy, Springer, 171-180.

[5] Tran, H. & Feuerlicht, G., "*Service Development Life Cycle for Hybrid Cloud Environments,"* Journal of Software, 2016.

[6] Tran, H. & Feuerlicht, G., "*Improving Reliability of Cloud-based Applications."* in The European Conference on Service-Oriented and Cloud Computing (ESOCC), 2016, Vienna, Austria, Springer.

[7] Tran, H. & Feuerlicht, G., "*Optimization of Cloud Applications Using Location-Based QoS Information."* In The 10th International Conference on Research and Practical Issues of Enterprise Information Systems (CONFENIS), 2016, Vienna, Austria, Springer.

List of Illustrations

Figure 1.1. Cloud service oriented enterprise application 29
Figure 1.2. The overview of research methodology 32
Figure 1.3. Thesis structure and organisation 33
Figure 2.1. Traditional SDLC (Papazoglou, 2008) 45
Figure 2.2. Cloud service model and management levels (Walton, 2016) 49
Figure 2.3. The conceptual reference model (Liu et al., 2011) 52
Figure 2.4. The ITIL service lifecycle (ITIL, 2014) 54
Figure 2.5. Extreme cloud programming SDLC on cloud computing (Guha, 2013) 56
Figure 2.6. Cloudification security development lifecycle (Wagner et al., 2015) 57
Figure 2.7. SOA cloud lifecycle using SCA (Ruz et al., 2011) 58
Figure 2.8. SOA cloud lifecycle using SCA (Joshi et al., 2014) 60
Figure 2.9. Requirements engineering activities (Cemuturi, 2014) 62
Figure 2.10. Service selection system architecture (Tang et al., 2017) 65
Figure 3.1. Design science research cycles (Hevner, 2007) 78
Figure 3.2. Overview of FMP 82
Figure 4.1 Cloud Service Consumer SDLC and lifecycle activities 88
Figure 4.2. AWS EC2 server monitoring at the provider side 98
Figure 4.3. Airport pick-up service optimisation scenario 101
Figure 4.4. Business process optimisation example 103
Figure 4.5. Technical optimisation example 104
Figure 4.6. The deployment diagram of FMP application (Nguyen, T., 2017) 107
Figure 4.7. The first version of SC-SDLC 114
Figure 4.8. The second version of SC-SDLC 114
Figure 5.1. SCF architecture 119
Figure 5.2. SCF modules 121
Figure 5.5. Service Repository user interface for cloud service selection 127
Figure 5.6. Service Repository entity relationship diagram 128
Figure 5.7. Workflow Engine and Service Adaptor implementation 130
Figure 5.8. Monitoring Center implementation 132

Figure 5.9. QoS analysis entity relationship diagram 133
Figure 5.10. Monitoring Centre user interface 134
Figure 5.11. The first version of SCF 136
Figure 6.1. Retry Fault Tolerance 141
Figure 6.2. Recovery Block Fault Tolerance 142
Figure 6.3. Online shopping scenario using a composite payment service 143
Figure 6.4. Dynamic Sequential Strategy 144
Figure 6.5. Cloud service load balancing 144
Figure 6.6. Service Consumer Framework reliability features 146
Figure 6.7. Experimental configuration 148
Figure 6.8. Availability of reliability strategies from 15th to 31st of March 151
Figure 6.9. Availability of reliability strategies from 1st to 15th of April 2016 152
Figure 7.1. Online shopping check out optimisation scenario 156
Figure 7.2. Multi-site cloud service monitoring 157
Figure 7.3. Daily average response times of eWay and PayPal services 167
Figure 7.4. Daily average availability of eWay and PayPal services 171

List of Tables

Table 2.1. Advantages and disadvantages of the traditional SDLC 41
Table 2.2. Comparison of cloud models 51
Table 2.3. Cloud service identification approaches 68
Table 4.1. Request for Service for online payment. 90
Table 4.2. Requirement specification priority levels 93
Table 4.3. Requirement specification importance levels 93
Table 5.1. SCF modules implementation 126
Table 6.1. Suitability of reliability strategies 145
Table 6.2. Payment service transaction logs 150
Table 6.3. Availability of payment services 150
Table 6.4. Response time of payment services in seconds 151
Table 7.1. QoS data for eWay and PayPal payment services 160
Table 7.2. Response time and availability correlation coefficients 162
Table B.1. List of QoS attributes (Meier et al., 2009). 188

List of Abbreviations

CTO	Chief Technology Officer
DFST	Dynamic Sequential Fault Tolerance
ERD	Entity Relationship Diagram
ESB	Enterprise Service Bus
FT	Fault Tolerance
IaaS	Infrastructure as a Service
IoT	Internet of Things
IT	Information Technology
ITIL	Information Technology Infrastructure Library
MSSQL	Microsoft SQL Server
NIST	National Institute of Standards and Technology
PaaS	Platform as a Service
QoS	Quality of Service
RBFT	Recover Block Fault Tolerance
RE	Requirements Engineering
REST	Representational State Transfer
RFS	Request for Service
RFT	Retry Fault Tolerance
SaaS	Software as a Service
SBA	Service-based application
SCA	Service Component Architecture
SCF	Service Consumer Framework
SC-SDLC	Service Consumer System Development Life Cycle
SDLC	System Development Life Cycle
SLA	Service Level Agreement

SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SOC	Service-Oriented Computing
UML	Unified Modelling Language
XaaS	Everything as a Service
XML	Extensible Markup Language

Abstract

The rapid growth of various types of cloud services is creating new opportunities for innovative enterprise applications. Organisations are using cloud services to deliver significant parts of their enterprise system instead of on-premises implementation. However, existing traditional software engineering lifecycle models are not adequate in the cloud context, and a comprehensive framework for managing cloud services from the consumer perspective throughout all phases of the lifecycle has not been fully described in the literature. This thesis addresses the following key research questions: "What lifecycle methodology should be used by cloud service consumers?" and "What framework is required to support this lifecycle methodology?"

To address these research questions, this thesis is concerned with how to develop cloud service-oriented enterprise applications from the service consumer perspective. For the purposes of this work, a cloud system development lifecycle is proposed, and a supporting framework is developed. This research also aims to expand the understanding of the challenges of using cloud services in enterprise application management. The main contributions of this thesis are the Service Consumer System Development Lifecycle (SC-SDLC), the Service Consumer Framework (SCF), a set of failover strategies for improving application reliability and a multi-site monitoring model for managing cloud services proactively. The SC-SDLC is the cloud lifecycle that is used to develop cloud service-oriented enterprise applications. Supporting the SC-SDLC activities, the SCF is implemented to manage cloud services from the consumer perspective. The failover strategies are designed to handle the problems of service disruptions, and the multi-site monitoring model is designed to monitor cloud services for the purposes of service selection and application optimisation.

Using a research approach based on design science and action research, the SC-SDLC and SCF are the results of an iteration process between the core activities of building and evaluating using a case study. By working closely with the team members of a *real-world* project for developing a hospital management system, the lifecycle activities and the framework features are being continuously improved during the project execution phase.

Additionally, to evaluate the failover strategies, a simulation environment is set up to provide a comparison of the theoretical calculation results with the experimental measurements. A separate simulation environment is also used to demonstrate the applicability of the multi-site monitoring model in selecting cloud services and application optimisation.

CHAPTER 1.

INTRODUCTION

1.1. Introduction

Recent years have seen the rapid development of cloud computing and the corresponding desire of consumers to use cloud services. Many organisations are using cloud services to deliver a significant part of their infrastructure and business applications. They adopt cloud computing by employing various cloud models (public cloud, private cloud, or hybrid cloud) at different levels (infrastructure level, platform application level or business function level). For example, organisations may entirely use the public cloud SaaS (Software as a service) solution (e.g., Salesforces) without the usual concerns of hardware provision, deployment, installation, and maintenance. However, the public SaaS has some limitations in terms of security, customisability, control of business process and manageability. For better security and application management, the common form of cloud adoption is that organisations only use virtual private cloud IaaS/PaaS (e.g., AWS VPC, Azure Virtual Network, Google Cloud Platform VPC) but develop their own enterprise applications. The increase in the number of available cloud services enables the development process in which enterprise applications are developed by composing cloud services. For example, organisations develop an application that uses PayPal for payment, OpenExchangeRate for money exchange, Speech API for the translator, Mendix Profile for user management, AWS EC2 for the server, Azure RDS for the database, Google Drive for storage, and so on. In such a way, organisations gain many of the advantages of cloud computing but still have control in terms of the application process and customisability.

Enterprise application development in the cloud computing context requires a new method. Because service providers already take responsibility for implementing application features and delivering them in the form of cloud services, the responsibilities of service consumer organisations are mainly those of integration and the management of cloud services. Traditional development process models used in software engineering and service-oriented engineering are not adequate in the cloud context (Kashfi, 2017, Panigrahi et al., 2016). Cloud services with their characteristics (i.e., on-demand, network access, resource pooling) have a significant impact on the traditional development process and they alter all stages of the SDLC. A new SDLC for developing enterprise applications is needed in the cloud context to address the challenges of cloud services

from the service consumer perspective. Although the lifecycle methodology for the development of enterprise applications and the management of cloud services is the subject of many research projects, the research problem is still active. Most of the research is concerned with the service provider perspective. Only a small number of researchers consider the cloud lifecycle from the consumer perspective and attempt to transform the traditional SDLC for the cloud. Because the SDLCs originally focus on on-premises implementation, they have limited applicability for integrating and managing cloud services. The traditional lifecycle can be translated to the context of cloud computing, but many cloud service challenges have not been fully addressed (Babar et al., 2017). Similarly, the main focus of cloud service management studies is the methodology for service providers delivering their cloud services. Accordingly, the framework for the management of cloud services from a consumer perspective has had insufficient attention in the literature.

In this thesis, a methodology for management of cloud services in the context of enterprise applications is proposed. A Service Consumer SDLC (SC-SDLC) is designed not only for developing enterprise applications but also for managing cloud services from the consumer perspective. During the SC-SDLC phases, the challenges of cloud application development, i.e., service selection, service integration, and cloud service monitoring are addressed. To support SC-SDLC, a Service Consumer Framework (SCF) is implemented to help service consumers in the management of cloud services throughout the lifecycle phases. Using design science research methodology, a case study project that aims to develop a hospital management system is used to evaluate the SC-SDLC and the features of SCF. To address specific challenges of cloud services, a cross-provider failover strategy is proposed for handling service disruptions, and a multi-site monitoring model for cloud service selection and application optimisation is described. This thesis contributes to research areas relating to the following objectives:

- to advance understanding of the challenges of managing cloud services in enterprise applications;
- to develop a cloud System Development Lifecycle for enterprise applications;
- to propose a methodology for management of cloud services through the lifecycle;

and

- to implement a cloud framework that supports the lifecycle activities and manages cloud services.

1.2. Problem Definitions

Using cloud services in enterprise applications has many advantages such as rapid development, dynamicity, flexibility, and scalability but the management of cloud services is also associated with many challenges (Kashfi, 2017).

1.2.1. Challenges in cloud service management

In this section, I briefly discuss the challenges of managing cloud services from the consumer perspective before discussing the gaps in the literature and the research questions of this thesis.

1.2.1.1. Methodology for cloud service selection

Due to the diversity and dynamics of cloud services, selecting the most suitable cloud service has become a major challenge for cloud consumers (Qu et al., 2014). There are already thousands of cloud services and cloud APIs that are available in the cloud, and the number is increasing steadily (Gartner, 2017). For example, the ProgrammableWeb (2017) directory listed over 18,000 APIs for various types of services. This gives a wide choice for organisations using cloud services. However, a large number of available services makes identifying suitable cloud services challenging (Tripathi et al., 2017). Moreover, a very limited number of service providers publish the QoS value for their cloud services. This lack of an accurate QoS of cloud services makes the identification of cloud services more difficult. Moreover, from the consumer side, the performance of cloud services is affected by the context, e.g., the network condition. This is particularly complex for service consumers in terms of selecting the proper cloud services in their specific location.

1.2.1.2. Methodology for management of cloud service disruption

Because of shifting from the on-premises implementation of application features to the integration of cloud services, the reliability and performance of enterprise applications largely depend on cloud services. An important challenge, particularly in situations where a large number of cloud providers are involved, relates to managing application continuity. In practice, service providers try to protect service customers by deploying cloud services in multiple-zone datacentres with extensive backup, disaster recovery, and failover capabilities. Despite these efforts to ensure 'zero downtime' for cloud services, the issue remains unsolved. Service providers are still not able to guarantee an acceptable level of availability and performance, and service disruptions are difficult to avoid in practice (Joshi, 2015, Almalki and Shen, 2015). For example, Google, one of the most reputable cloud service providers, states that service reliability is their top priority, but their services have suffered a number of outages and disruptions[1]. Another example of service disruption[2] occurred on February 28th, 2017 in the Amazon Simple Storage Service that led to hundreds of thousands of websites going down (SimilarTech, 2017). *The current cloud service failover efforts on the provider side are not enough*; service consumers need to apply a client-side cloud service failover capability to protect their applications.

1.2.1.3. Methodology for managing cloud service changes

Characterised by rapid change and technology innovation, cloud services are subject to frequent modifications to eliminate defects and to introduce new functionality. Service providers continuously implement functional enhancements and rectify defects (Papazoglou, Andrikopoulos et al. 2011). There is an assumption that cloud service providers provide multiple versions of cloud services and maintain backward compatibility among the versions of cloud service. This assumption is not always true for all service providers. The lack of control over cloud services is one of the main disadvantages associated with cloud computing (Calero and Aguado, 2015). Considering a notification sent out when PayPal makes a service change[3], they recommend service

[1] Source from: https://www.google.com/appsstatus#hl=en-GB&v=status&ts=1502632799000
[2] Source from https://aws.amazon.com/message/41926/
[3] Source from https://www.paypal-status.com/bulletin/production

consumers make a subsequent update in applications to avoid potential failures. In this case, service consumers are forced to upgrade their applications to maintain compatibility with new versions of services. *The service version strategies applied on the provider side are not enough to protect the consumer from the problem of service changes.* Service changes not only create reliability risks for enterprise applications but also result in significant ongoing maintenance costs for service consumers (Zuo et al., 2014).

1.2.1.4. Methodology for cloud service monitoring from the service consumer perspective

From the service consumer perspective, cloud services monitoring is necessary to understand the status and performance of cloud services, so that service consumers can make informed decisions. However, it is a challenge because of the inherent complexity, e.g., different technologies and different interfaces among cloud providers. Because of the characteristics of cloud computing, the traditional methodologies have limitations when applied to cloud computing. From the consumer perspective, cloud service monitoring is needed for a number of reasons: for cloud service selection, the management of cloud service incidents, and optimisation decision-making.

1.2.2. Research Gaps

The existing literature that is discussed in Chapter 2 contains several shortcomings which have not yet been adequately investigated. In this section, the gaps in the literature related to the methodology for management of cloud services are summarised as follows:

1.2.2.1. Lack of lifecycle methodology for development of enterprise applications from the consumer perspective

The existing literature mostly focuses on lifecycle methodologies that assist the cloud providers in relation to the management of their cloud services. The literature has taken relatively little account of the works on the cloud lifecycle from the service consumer perspective and most of these works focus on transforming traditional methodologies for the cloud. The traditional methodologies have some advantages but they do not adequately address the challenges of cloud services. In addition, most research efforts

only focus on cloud infrastructure for deployment without considering the use of SaaS during application development.

1.2.2.2. Limited frameworks to support cloud service consumers during the entire SDLC

In the literature, there are a large number of research works on different parts of the SDLC: service selection, service integration, service monitoring, and service optimisation. However, a comprehensive framework that performs all the activities required for the management of cloud services from the consumer perspective throughout all of the SDLC phases has not been adequately covered in the literature. As the cloud service selection is a process that has several criteria on the basis of which a service consumer selects a cloud service, there is a connection between requirements specification and service selection. The existing literature mostly focuses on measuring and ranking the cloud service for service selection, without fully considering consumer expectations. Also, cloud service monitoring is a management activity that is not only for service selection at design time but also for decision making at runtime. However, most of the related literature only considers one aspect of cloud service monitoring, for example, for cloud service selection.

1.2.3. Research Problem

Traditional development process models are not adequate in the cloud context and the cloud SDLC for developing a cloud application is still an open research topic. Therefore, the research statement of this thesis is “a methodology for developing cloud service-oriented application and for managing cloud services in enterprise application”.

1.2.4. Research Questions

To address the above challenges concerning the management of cloud services from the service consumer perspective, the main research questions to be answered by this thesis are:

Q.1. What lifecycle methodology should be used by cloud service consumers?

As found in the literature review, the SDLC for the cloud has been the subject of recent

research efforts, but most of the studies have taken the perspective of service providers, and relatively little research has been done so far in relation to the service consumer perspective. Other studies on adapting the traditional SDLC for the cloud also fail to address the challenges of using cloud services. The SDLC for development of the cloud-based applications is still an open research problem (Rehman et al., 2015). In order to answer this question, the SC-SDLC for building enterprise applications is proposed. The research problems of cloud services are addressed throughout the phases of SC-SDLC. This lifecycle includes (1) design-time activities: requirements specification, service identification, and service integration, and (2) run-time management activities: service monitoring, and application optimisation. When discussing this question, a sub-question must also be asked:

> *How does the service consumer select a suitable cloud service for a given application requirement?*
>
> This research question concerns the methodology for selection of cloud services that satisfy the requirements specifications. Service selection is the key to success when developing cloud service-based enterprise applications. However, selecting suitable services from the increasing number of cloud services available in the cloud is a significant challenge. In the literature, there are several publications on cloud service selection that use various approaches such as QoS measures, user feedback benchmarks, or heuristics. However, these research efforts are only a part of the service selection problem. There is little research that considers service selection as a phase of the cloud service lifecycle which, in addition to service selection, also includes requirements specification and service monitoring (Rehman et al., 2015). In this thesis, a cloud Service Repository used for cloud service selection is proposed. The Service Repository supports the lifecycle for all phases from requirements specification to cloud service monitoring.

Q.2. What framework is required to support the SC-SDLC lifecycle methodology?

This question considers the methodology for the management of cloud services for an enterprise system from the client side. As mentioned in the previous section, the dynamic nature of cloud computing raises important management issues from the service consumer perspective. Traditional IT resource management solutions designed for

enterprise environments are unable to meet management requirements due to multi-tenancy, large scale, dynamism, on-demand, and pay-per-use of cloud environments. Furthermore, the management of cloud services viewed from the service consumer perspective is different from cloud services management which is viewed from the cloud provider side. However, management of cloud services from the perspective of service consumers has only recently received attention. In this thesis, I discuss the management of cloud services in all phases of the SC-SDLC including service portfolio management (Service Identification phase), service account management (Service Integration Phase), service QoS management, service cost management and service incident management (Service Monitoring phase).

1.3. Key Terminologies

In this section, the key terminologies used throughout this thesis are clarified, as definitions adopted by researchers are not always uniform. For the purposes of this research, frequently used terms are clarified as follows:

1.3.1. Enterprise Application

The cloud service-oriented enterprise application is an application that uses cloud services for deploying infrastructure and/or for implementing application features. This type of cloud service-oriented enterprise application, in this thesis, refers to *"enterprise application"* for short. Figure 1.1 illustrates an example of an enterprise system having three enterprise applications. These applications are deployed on the AWS Virtual Private Cloud (VPC) with the AWS RDS database server and they integrate a number of public

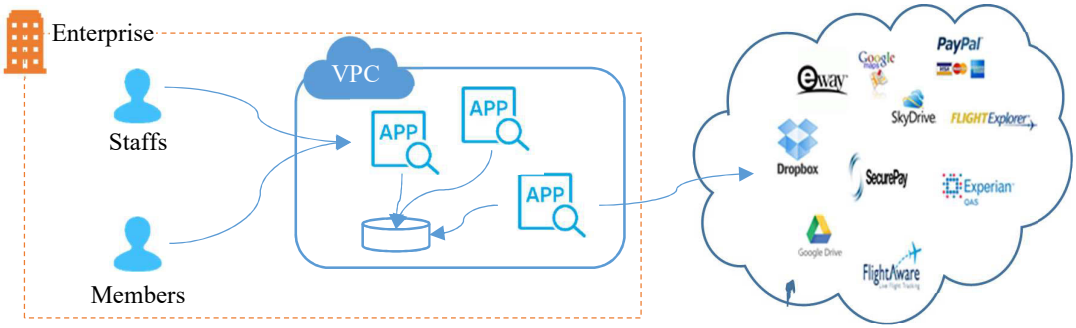


Figure 1.1. Cloud service oriented enterprise application

cloud SaaS such as Google Drive, PayPal, eWay, FlightAware and so on.

1.3.2. Cloud Service

Cloud service is popularly understood to be a resource provided over the Internet. In this thesis, cloud service is defined as a resource running on cloud infrastructure provided over the Internet by a third party service provider (Mell and Grance, 2011). Besides providing the cloud service, the service provider also provides the ability of interaction programmatically through *cloud APIs*. In the case of IaaS or PaaS, cloud APIs enable the ability to configure and monitor virtual servers. In the case of SaaS, the cloud APIs enable the rise of continuous deployment (Weber et al., 2016). For example, Salesforce provides the Salesforce Platform API that not only forms the foundation of core Salesforce products, but it also lets developers build their own application. The concept of cloud service or cloud API is related to *web service*, but they are not identical. While web service is generally associated with peer-to-peer communication, *cloud API* is defined as the application feature or business service. Normally, cloud APIs are delivered using web service (SOAP) technology or web API (REST) technology. In this thesis, the term *"service"* refers to cloud service, not web service which is only a communication technology.

1.3.3. Service Consumer

The topic of my thesis is the management of cloud services from the service consumer perspective. A cloud consumer represents an organisation that uses a cloud service in a relationship with a service provider. For example, a cloud consumer is an organisation which uses cloud servers for deploying their applications (e.g. AWS EC2, Azure), cloud storage (e.g. Google Drive, AWS S3, OneDrive, Dropbox) for storing files, cloud database (AWS RDS, Azure SQL) for data centers, or cloud business service for application features (e.g. PayPal, eWay, Flight Aware, Salesforce).

1.3.4. Service Incident

Service disruption or service outage is a period of time during which the cloud service becomes unavailable. It can be caused by a number of different factors on the provider

side including network connectivity issues between the service consumer and service provider.

The term *service evolution* that is discussed in this thesis relates to the change of service including functionalities, security policy, SLA, and performance. It can be caused by the activities of the cloud service provider such as service improvement, bug fixing, maintenance; or it can be caused by changes in context. For example, a large number of consumers who start to use cloud services can affect the performance of the cloud services or the high demand use of cloud services during a period of time (payment service during Black Friday) can have an impact on the performance of the cloud service.

In this thesis, service incident refers to an interruption to an IT service or reduction in the quality of an IT service. It relates to service disruption, service evolution, service discontinuation or any context changes that can affect the cloud service performance and availability.

1.4. Scope of Work

In this section, I define the scope of the thesis, including the types of cloud services, service consumer perspectives and types of QoS measurements.

1.4.1. Using Payment Service as Examples

The contribution of this thesis can be applied to various types of cloud services including IaaS, PaaS, and SaaS. However, payment services are selected as examples because payment services are able to fully demonstrate the challenges of cloud services in relation to the management of enterprise applications. Moreover, the business concepts and logic of payment service are clear and simple. It is also easy to get information and have a large number of providers and consumers. Although I do not provide experimental results for other types of cloud services, I describe how SC-SDLC and SCF can support other types of cloud service (i.e., IaaS and PaaS).

1.4.2. Service Consumer Perspective

In this thesis, I address the research problem from the perspective of the service consumer. In the research background, I discuss the separate roles of the cloud service consumer and

the cloud service provider. The SC-SDLC and SCF which are proposed in this research are from the viewpoint of the service consumer only, ignoring the problems on the service provider side. For example, cloud service evolution is identified as the runtime risks for an enterprise application. The change time of enterprise applications is not relevant to service evolution which happens based on the change time on the provider side.

1.4.3. Using Availability and Response Time for Service Measurements

The QoS attributes[4] which are used for cloud service selection and monitoring discussion are service availability, service response time and cost. These QoS attributes are used because they are measurable and are the key indicators of enterprise application reliability and performance. Although security is one of the common concerns in relation to the cloud model, cloud security is not the topic of this study.

1.5. Research Methodology

In this thesis, the research approach draws on the design science approach that is

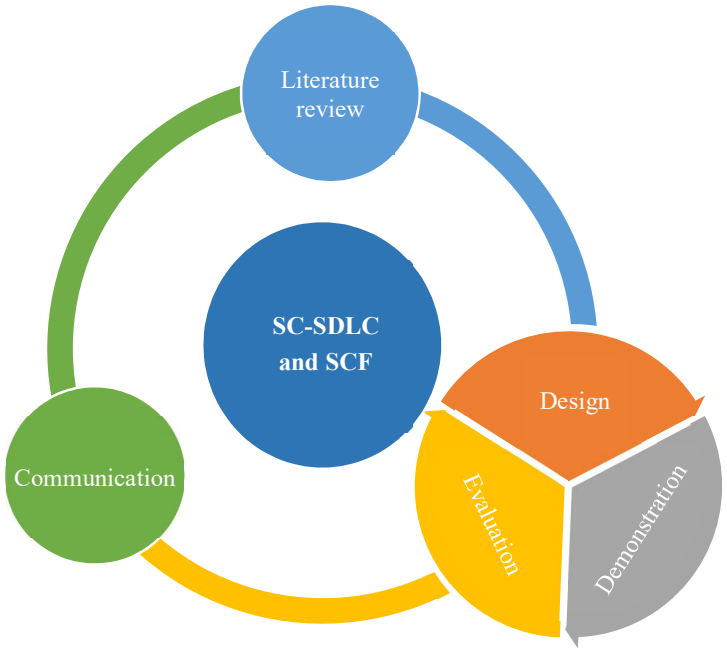


Figure 1.2. The overview of research methodology

[4] The Appendix A lists the common QoS attributes that are considered during a system development lifecycle.

concerned with practice and theory (Figure 1.2). First, a literature review is done to understand the current state of the research problems and to identify the research gaps. Then, the research objectives and contributions are inferred from the identified research problems. In this thesis, the SC-SDLC and SCF are defined as the main contributions of this study. Using the knowledge gained from the literature review, the designs of SC-SDLC and SCF are described. The SC-SDLC and SCF are applied to a real-world project of developing a hospital management system. During the demonstration of the project, the previous development approach used in the organisation is compared to the SC-SDLC. The problems encountered when using the SC-SDLC in all stages of the project are also evaluated. This process has helped to improve the designs of the SC-SDLC and SCF. The research results have been submitted to select conferences for communication

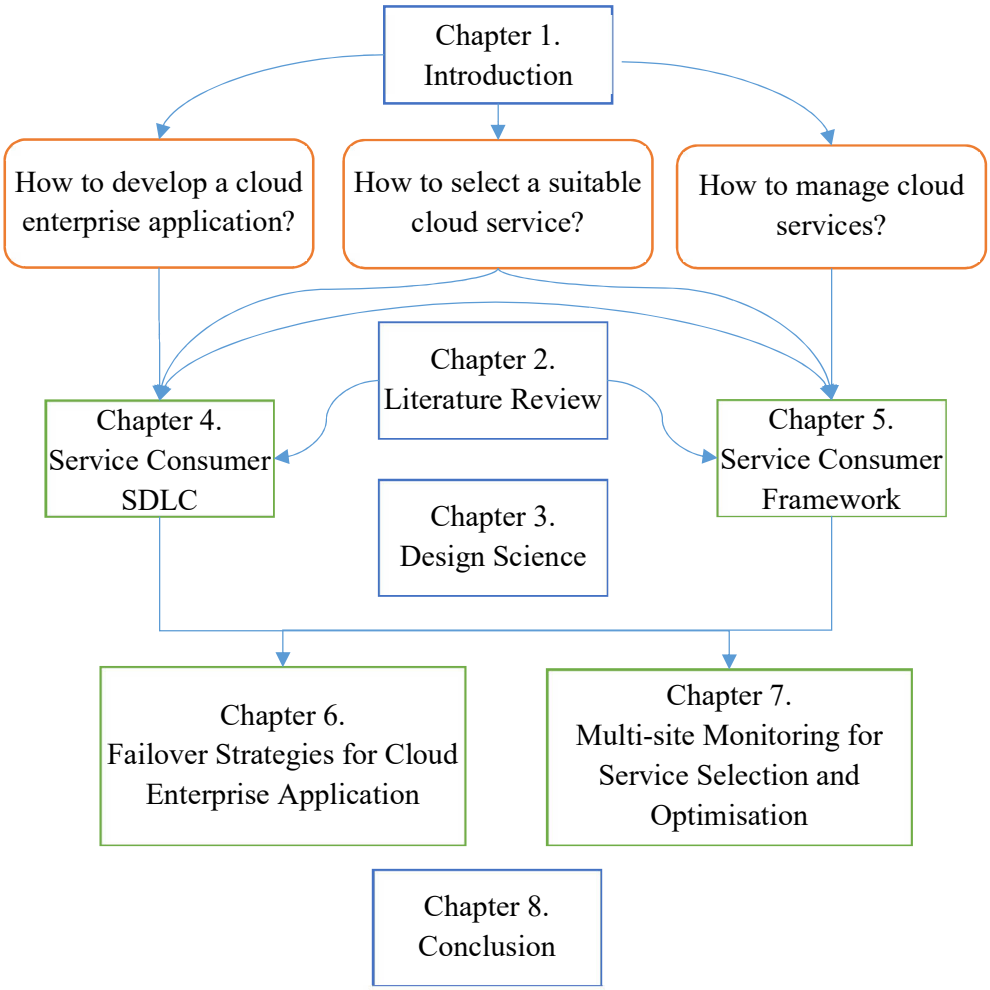


Figure 1.3. Thesis structure and organisation

with other researchers. The feedback received from the reviewers and researchers has been used to improve the design of the SC-SDLC. In Chapter 3, the details of the research methodology and techniques used in this thesis are described.

1.6. Structure of the Dissertation

Figure 1.3 shows the structure and organisation of this thesis. The following paragraphs highlight the main contents of each chapter in this thesis.

Chapter 1 - Introduction

Chapter 1 describes the context of the research and identifies research aims and research problems. It also states the research questions and my research contributions. This chapter clarifies key concepts and terms related to the context of the research, and then the proposed solutions and research methods are described.

Chapter 2 - Literature Review

This chapter presents the literature review of the Service Consumer SDLC and its phases. Predominantly, research concepts of service engineering, service-oriented computing, and cloud computing are discussed. The traditional development process, (i.e., Waterfall, Prototype, Agile, SOA SDLC) is discussed to provide a background to the topic. Then the state-of-art cloud SDLC is reviewed, and the research gap is stated. The specific phases of the SC-SDLC are also reviewed in this chapter.

Chapter 3 - Methodology

Chapter 3 describes the research approach that is applied to this study. The research methodology drawing on design science is described. In detail, the specific process model, research methods, and techniques, including the literature survey, action design research, and simulation research techniques are also described in this chapter.

Chapter 4 – Cloud Service Consumer SDLC

This chapter describes in detail the proposal for the cloud Service Consumer System Development Lifecycle (SC-SDLC) which is used for developing enterprise applications in the cloud. The phases and lifecycles are described in detail to show how the SC-SDLC addresses the research problems of the cloud model. The SC-SDLC is one of the main

contributions of this thesis because it encompasses the methodology for the development of enterprise applications. During the phases of the SC-SDLC, the activities for the management of cloud services are discussed. In this chapter, a case study of a real-world project for developing a hospital management system is described and the evaluation of the SC-SDLC in practice is provided. The feedback and suggestions from project managers are used to improve the SC-SDLC.

Chapter 5 - Service Consumer Framework Implementation

This chapter describes the proof-of-concepts of the Service Consumer Framework, which is a cloud framework for management of cloud services. The design and implementation of SCF modules are described, and a discussion of how SCF supports the SC-SDLC activities is provided.

Chapter 6 – Failover Strategies for Improving Application Availability

In this chapter, the capability of SCF for managing cloud service incidents is discussed. The failover strategies of SCF that are designed to improve the availability of enterprise applications are described. Evaluation results of failover strategies on payment services are given by a comparison of the theoretical results with the experimental measurements of availability and response time.

Chapter 7 – Multi-site Monitoring for Application Optimisation.

This chapter describes the multi-site monitoring model of SCF that is used to monitor cloud services for both the service identification phase and optimisation phase of the SC-SDLC. Experimental results using a simulation context are presented to evaluate the applicability of multi-site monitoring, and then a discussion of the use of this monitoring model is given.

Chapter 8 - Conclusion and Future Work

This chapter presents the conclusions drawn from the results discussed in earlier chapters of the thesis. I discuss the contributions of SC-SDLC and SCF in addressing cloud services challenges. Then the implications of my findings are identified and demonstrated. Finally, the limitations and future research directions based on the results of this study are discussed.

1.7. Conclusion

The topic of this study is to develop a framework for management of cloud services. In this chapter, the context for this study in which the trend of using cloud services in enterprise applications is discussed. The challenges of cloud service management, i.e., cloud service selection, cloud service monitoring, cloud service incident management are also clarified. Although there are a number of research efforts dealing with the topics of this thesis, in literature, there is a lack of SDLC for developing enterprise applications and their supporting frameworks for the management of cloud services.

The main research questions are “What lifecycle methodology should be used by cloud service consumers?” and “What framework is required to support this lifecycle methodology?” To answer these questions, the Service Consumer System Development Lifecycle and Service Consumer Framework are proposed. Although the experimental results are conducted using SaaS payment services, the contributions of this thesis (SC-SDLC and SCF) can be applied to all cloud models (IaaS, PaaS, SaaS). In the next chapter (Chapter 2), the background information of the study is provided. Also, a literature review of the current state-of-art cloud SDLC is presented to justify the contributions of this thesis.

CHAPTER 2.

LITERATURE REVIEW

2.1. Introduction

The research aims are the methodology to develop cloud enterprise applications and the management of cloud services. The main contribution of this thesis is a cloud system development lifecycle to build an enterprise application. By analysing a large number of papers mainly published between 2013 and 2017 in selected conferences and journals, a qualitative research-based survey with the common theme of the cloud SDLCs is derived. In the next section (Section 2.2), the research efforts which provide a background to the research topic are discussed. Then, a review of the current cloud SDLC models is outlined in Section 2.3. Section 2.4 reviews the research efforts which are related to the phases of the proposed lifecycle (SC-SDLC) to study the current approaches in relation to each phase. Finally, in summary (Section 2.5), the key references, research gap, and purpose of this thesis are restated.

2.2. Background and Concepts

2.2.1. Software Engineering

"Software Engineering (SE) means the application of a systematic, disciplined, quantifiable approach to development, operation and maintenance of software" (IEEE, 1990). Software engineering has, over the years, been driven by technology development and advocacy research. It implies a software process through pointing at different life cycle phases. Before discussing the traditional SDLC, I will clarify the different lifecycle terminologies that are used in this study.

- *Software life cycle.* This is the period of time that begins when a software product is conceived and ends when the software is no longer available for use. The software life cycle typically includes a concept phase, requirements phase, design phase, implementation phase, test phase, installation and checkout phase, operation and maintenance phase, and, sometimes, a retirement phase (IEEE, 1990).
- *Software development cycle.* This is the period of time that begins with the decision to develop a software product and ends when the software is delivered. This cycle typically includes a requirements phase, design phase, implementation phase, test phase, and sometimes, an installation and checkout phase (IEEE, 1990).

- *System development lifecycle.* This is the period of time that begins with the decision to develop a system and ends when the system is delivered to its end user. This term is sometimes used to mean a longer period of time, either the period that ends when the system is no longer being enhanced or the entire system life cycle (IEEE, 1990).

In this study, SDLC refers to the system development lifecycle, i.e., the various stages used in the life cycle of enterprise application development. There are some generic activities that occur in all system development processes. Major development stages include (Izzat, 2013):

- *Requirement gathering and analysis phase:* In this stage, analysts or developers are expected to analyse the problem domain or where the project is triggered or initiated to find out what is the problem that the application is trying to solve (Leau et al., 2012). They may collect these requirements from documents, similar projects or users and candidate users of the current or the new system.
- *Design stage:* In this stage, a solution is designed for the problem that was described in the requirement stage. The design is the model or the template that draws the guidelines for the actual implementation or the code. Any possible conflict that may arise in the code should be resolved to look at the application design. In most recent agile development models, there is a little focus given to this stage, and such models propose a lightweight design without putting too much effort into it especially as, in many cases, such designs may evolve and change in the future.
- *Implementation*: This is where the actual code of the project is developed. The code is the main deliverable in any development project. In some cases, where the project is about improving an existing application, some parts of the application may be affected by this project while the rest of the code is not. For example, a refactoring process can be conducted to improve the user interface of an application or improve its quality in order to solve some of the problems in the existing application. In such cases, no new major functionalities are expected to be added.
- *Testing*: Applications are tested for possible errors before their delivery to the users or the clients. They are tested first against the proposed requirements or functionality to make sure that the developed application satisfies the initially proposed

requirements. Internal or white box testing activities are also conducted by developers to make sure that the internal structure of the code, the algorithms, and the methods will not cause logical or semantic errors at the time of usage. Other testing activities include black box testing to test the high-level user-visible system functionalities. They include system testing, integration testing, user testing, and release testing. Each testing tries to expose the possible problems in a specific area of the application.

- *Deployment*: After the testing stage, the application is prepared for release and deployment to users. Packaging and operating system or environment-related issues are all considered at this stage to make sure that the system will be installed and will work as a design in the operational environment.
- *Maintenance*: Any problems that may arise after releasing the application to the users are classified under a new stage called evolution or maintenance. Changes can occur for different reasons. They may occur due to new requirements requested by users or due to errors and bugs found by users in the user environment. Other causes of changes can be responses to environmental changes and application enhancements.

The field of software development went through several evolutionary cycles. In early approaches to software development, the activities or stages of the development process are clearly defined and distinguished. Various application development approaches are defined and designed which are then used during the development process of application.

2.2.1.1. Waterfall

The Waterfall approach which is designed to ensure the success of the project was the first process model to be introduced and followed widely in application engineering. In the Waterfall approach, the process of application development is divided into separate process phases: Requirement Specifications phase, Application Design, Implementation, Testing, Deployment & Maintenance. All of these phases are cascaded into each other so that the second phase is started as and when a defined set of goals are achieved for the first phase (Tuteja and Dubey, 2012).

2.2.1.2. Prototype

The Prototype approach is to ensure that the proposed application meets the user expectations (Kumar et al., 2013). Instead of finalising the requirements before the implementation phase, the approach is to construct a "quick and dirty" partial implementation of the application during the requirements specification phase. The users can evaluate the prototype and give feedback to the developers. Then, the user feedback is used to change the requirements or to improve the design. At the implementation phase, the developers can actually implement the application with the confidence that they fully understand the user requirements and expectations.

2.2.1.3. Incremental Model

The Incremental model is the development process that partially implements the application and then slowly adds the functionalities or improves the design. This approach reduces the possibility that the user changes the requirements during the development process. It helps to release the application quicker (Scroggins, 2014).

2.2.1.4. Agile Methodology

Agile (e.g., Scrum, Crystal, and Extreme programming) represents more recent development approaches that propose new ways of dealing with requirements, communicating with customers, and progressing development activities (Leau et al., 2012). Requirements specification frequently changes through the lifecycle or the development process. Agile models are able to deal with the requirements evolution (Paetsch et al., 2003). They also help to understand the needs of the customers. The customers are closely involved in the projects in the beginning phases to give feedback. The feedback is always welcome, and the requirement changes are handled to ensure that the application meets the current needs of customers (Turk et al., 2014).

Table 2.1. Advantages and disadvantages of the traditional SDLC

	Advantages	Disadvantages
Waterfall	- Simple and easy to use. - Easy to arrange tasks and clearly defined stages. - Requirements should be clear	- Users can judge quality only at the end. - Changes in requirements pose a challenge.

	before going to the next phases. - Each phase of development proceeds in a linear order without any overlapping. - Works well for projects where requirements are well understood.	- The high amount of risk and uncertainty. - Users do not get the feel of the product before delivery. - The entire application is delivered in one shot at the end.
Prototype	- Users are actively involved in the development - When a prototype is shown to the users, they get proper clarity in terms of the requirements and they feel the functionalities. This allows them to suggest changes and modifications. - It reduces the risk of failure, as potential risks can be identified early, and steps can be taken to remove that risk. - The customer does not need to wait long to use the application.	- Wastage of time and money to build a prototype, if the client is not satisfied. - Too many changes can disturb the rhythm of the developer team. - Long-term procedure. - It follows the "quick and dirty" approach. The prototype is thrown away after it has been shown to the client.
Incremental Model	- After using the first iteration model, users can give their suggestions and make requests for change. - It is flexible in relation to user requirements and it is an easy-to-process model. - This model is used when requirements are clear to some extent, but the project scope requires a purely linear approach. - Testing and debugging during smaller iteration is easy.	- Mapping requirements to increments may not be easy, so the document management is difficult. - When application requirements change frequently, it is hard to both manage the changes and implement the new features. - More management attention is required for this model.
Agile	- Early customer involvement. - Iterative development. - Self-organising teams. - Adaptation to change.	- Hard for a large-scale project but suitable for low to medium-scaled projects (Turk et al., 2014). - Developers are required to have good social and interpersonal

		skills (Paetsch et al., 2003). - Limited support for building reusable functionalities (Turk et al., 2014).

2.2.2. Service Oriented Architecture

"Service-Oriented Computing (SOC) is the computing paradigm that utilizes services as fundamental elements for developing applications" (Papazoglou, 2003). To build a service-oriented application, Service Oriented Architecture (SOA) is the fundamental architectural model that supports the overall paradigm of Service Oriented Computing. Basically, SOA is the approach in which applications rely on a set of services to facilitate business processes. It involves componentising the enterprise and/or developing applications that use services and make functionalities available as services for other applications to use. A service is a self-contained, reusable software component provided by a service provider and consumed by service requestors.

2.2.2.1. SOA Principles

A key benefit of SOA is that it enables close alignment of information technology with the business objectives of the organisation, and at the same time, facilitates high levels of business process automation by using discrete, loosely coupled services (Consortium, 2009). Several SOA can coexist, provided they satisfy some key principles for SOC.

Loosely coupled. Coupling is a measure of the strength of the relationship between two or more services; minimisation of coupling reduces interdependencies between services facilitating service evolution (Larman 2004). The goal of loose coupling architecture is to reduce the risk that the change of a service will create unexpected changes for other services.

Service Cohesion. Cohesion refers to the level of interrelationships between the elements of a software module (Stevens 1999). The benefits of highly cohesive methods include easy maintenance and increased reuse potential (Larman 2004). Functional cohesion is achieved in situations where each service operation implements a single clearly-defined task; this leads to low coupling and high levels of reuse (Vinoski 2005). There is a close

relationship between service cohesion and coupling as poor cohesion leads to high levels of coupling.

Granularity. Service granularity is often measured in terms of the amount of data in the message payload of a service (McGovern, Tyagi, Stevens & Matthew 2003). The level of granularity (coarse-grained or fine-grained) is a statement of a service's functional richness. On the one hand, coarse-grained services typically have aggregated message payloads resulting from the execution of multiple operations before the return of the data. Fine-grained services, on the other hand, typically implement individual operations directly, resulting in excessive chattiness and complex interaction dialogues (Erl 2008).

Reusability. In SOA, services are composed to achieve good levels of reuse, i.e., service reuse is closely related to service composability. For example, a well-designed credit card verification service can be reused in a large number of payment applications, and therefore it is highly reusable. Service composition is crucial because it creates a new service from existing services crossing the organisational boundary. Following from the discussion of service granularity, the bias towards coarse-grained services makes achieving reuse difficult in practice as such services are not readily composable (Feuerlicht, 2007).

Implementation Neutrality. The interface for each component matters most because we cannot depend on the interacting components' implementation details, which can be unique. In particular, a service-based approach cannot be specific to a set of programming languages, which cuts into the freedom of different implementers and rules out the inclusion of most legacy applications.

Persistence. Services do not necessarily require a long lifetime, but they must exist long enough to detect any relevant exceptions. Further, they must take corrective action, and then respond to the corrective actions taken by others. They should also exist long enough to engender trust in their behaviour because they are engaged dynamically, and reputation might be the only means available to gauge their reliability.

2.2.2.2. Traditional SOA SDLC

SOA-based applications require a service-oriented engineering methodology that enables

modelling of the business environment, including key performance indicators of business goals and objectives; translation of the model into service design; deployment of the service system; and testing and management of the deployment (Papazoglou, 2003). While there are some differences in various approaches, most researchers include the

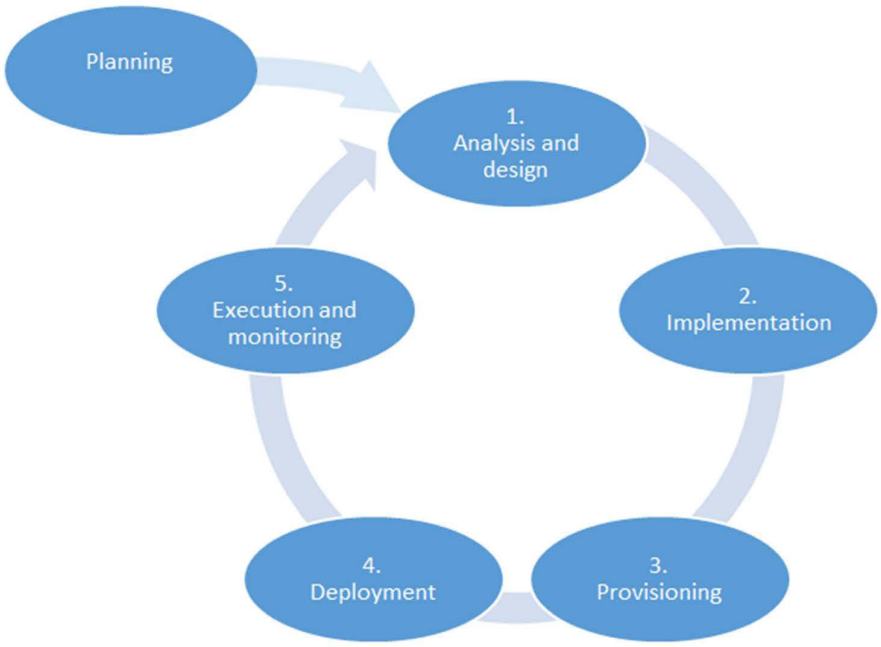


Figure 2.1. Traditional SDLC (Papazoglou, 2008)

lifecycle phases illustrated in Figure 2.1. The figure represents an iterative and incremental lifecycle process that includes a preparatory *Planning* phase and five distinct main phases: *Analysis and Design*, *Implementation*, *Provisioning*, *Deployment*, and *Execution and Monitoring* (Papazoglou, 2008, Papazoglou and Heuvel, 2007).

Planning. The planning phase comprises all activities that lay the foundation for the other phases of the SDLC. Although the planning phase is not a part of the lifecycle, it is a preparatory phase that analyses the business case for different combinations of development approaches and realisation strategies. This business case analysis includes gap analysis, and risk analysis. Gap analysis matches high-level descriptions of new services that make up a business process against the available services and includes scenario analysis that considers costs, risks, benefits, and returns on investment associated with the development of new business processes. Risk analysis compares costs

and benefits in provisioning scenarios and verifies the feasibility of specific options. The impact and probability of events that may influence the performance of services is estimated.

Analysis and Design. The analysis and design phase aims to identify and conceptualise business processes as a set of interacting services. Hence, this includes all activities related to the identification and description of the processes and services in a business problem domain (Rosemann et al., 2009). The proposal for a new service has to be analysed and decomposed to identify which services should be realised and what kind of logic should be encapsulated by each service (Kohlborn et al., 2009). After the services are analysed, the technical details of service interfaces are specified using design principles that include minimisation of coupling and maximisation of service cohesion. This requires strong enforcement of design by reuse and design for reuse principles. Importantly, this phase involves decisions about service granularity and the grouping of service operations, therefore, ensuring that the overlap between the functionality of different services is minimised.

*Implementation.*The implementation phase takes the input from the service design and analysis phase and includes all activities related to the realisation of the services. The implementation phase comprises activities that are very much aligned with traditional activities related to application implementation. Prior to development, a decision has to be made regarding the hosting environment of the application (Rosemann et al., 2009) and the programming language (Kohlborn et al., 2009). As services are reused in different application scenarios, services need to be exhaustively tested. Testing is to ensure that requirements which are as specified in the previous SDLC phase have been met and to guarantee that the deliverables are of acceptable quality (Kohlborn et al., 2009).

Provisioning. The provisioning phase involves implementing service governance, certification, auditing, metering, and billing, and controlling the behaviour of services during their use. Service provisioning can be local or over a network and can involve a complex mixture of technical and business aspects that support various client activities (Farrell, 2011). This process is to prepare for the deployment of application services on accessible servers that expose their capabilities as a network of virtualised services (hardware, storage, and database). It involves instantiation of servers that match the

specific hardware characteristics and application requirements of the services to be hosted.

Deployment. Service deployment is the process of making a service instance available for users. Service deployment is a complex process that is composed of the following deployment tasks: installation, configuration, activation, versioning and registering. During SDLC, services may have several versions and subversions. The diversity of their different installations means that the service providers have to support an unforeseen number of resources. This versioning configuration is to support multiple versions or to maintain backward compatibility of services.

Execution and Monitoring. The service execution and monitoring phase cover activities during service runtime when services are operational and their progress can be monitored. Service monitoring includes outage management with the objective of maximising the availability of services. It is an essential part of service SDLC and it is required for meeting regulatory requirements, verifying compliance with service-level agreements, optimising system performance, and minimising the cost of hosting services. However, service monitoring comes with a cost, including the performance impact on the monitored services and systems. Therefore, it is important to deploy the right level of monitoring at the appropriate time and location in order to achieve the objectives of monitoring whilst minimising its impact on services and systems.

2.2.3. Cloud Computing

As identified in the NIST definition of Cloud Computing (Mell and Grance, 2011), Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. It is a novel approach for implementing enterprise information technology solutions and promises many benefits for enterprise applications. Cloud computing is widely accepted as being associated with a number of benefits that include cost savings, elasticity, scalability, and competitiveness (Armbrust et al., 2009). This cloud model promotes the availability and is composed of five characteristics, three service models, and four deployment models.

2.2.3.1. Essential Characteristics

On-demand Self-service. A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed, This is automatic and does not require human interaction with each service provider.

Broad Network Access. Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, tablets, laptops, and workstations).

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

Rapid Elasticity. Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.

Measured Service. Cloud systems automatically control and optimise resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilised service.

2.2.3.2. Service Models

Cloud Software as a Service (SaaS). The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through a thin client interface such as a Web browser (e.g., Web-based email). The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or

even individual application capabilities, with the possible exception of limited user-specific application configuration settings.

Cloud Platform as a Service (PaaS). The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly, application hosting environment configurations.

Cloud Infrastructure as a Service (IaaS). The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include

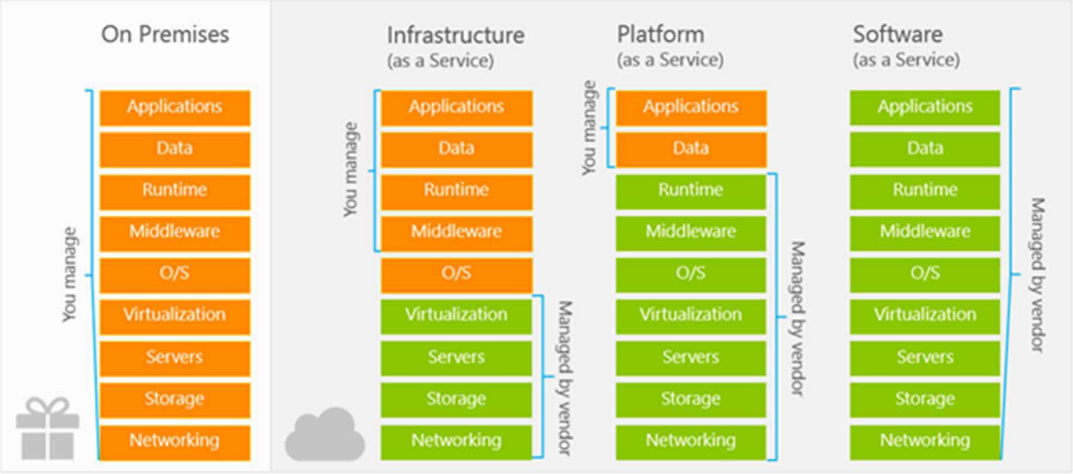


Figure 2.2. Cloud service model and management levels (Walton, 2016)

operating systems and applications. As shown in Figure 2.2, the consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, deployed applications, and possibly has limited control of select networking components (e.g., host firewalls).

2.2.3.3. Deployment Models

Public clouds are the most common way of deploying cloud computing. The cloud resources are owned and operated by a third-party cloud service provider and delivered over the Internet (Microsoft, 2017). AWS, Google Cloud Platform, Microsoft Azure,

Dropbox, and Google Drive are examples of public clouds. In a public cloud, all hardware, software and other supporting infrastructure are owned and managed by the cloud provider.

Private Cloud. A private cloud consists of computing resources used exclusively by one business or organisation. The private cloud can be physically located in your organisation's on-site data centre, or it can be hosted by a third-party service provider. In a private cloud, the services and infrastructure are always maintained on a private network, and the hardware and software are dedicated solely to your organisation (Mell and Grance, 2011). In this way, a private cloud can make it easier for an organisation to customise its resources to meet specific IT requirements. Private clouds are often used by government agencies, financial institutions, and any other medium to large-sized organisations with business-critical operations seeking enhanced control over their environment.

Virtual private cloud (VPC) is a private cloud located inside a public cloud that enables you to experience the benefits of a virtualised network while using public cloud resources. A VPC isolates your data from that of other companies helping to create a more secure environment. A VPC connects to remote networks via a virtual private network (VPN) connection. A VPC is ideal for companies seeking high levels of security, privacy, and control, such as healthcare and financial organisations dealing with regulatory compliance. Businesses also find VPC ideal for running mission-critical applications. (RACKSPACE, 2017)

Hybrid Clouds combine on-premises infrastructure, or private clouds, with public clouds so that organisations can reap the advantages of both. In a hybrid cloud, data and applications can move between private and public clouds for greater flexibility and more deployment options. For instance, the public cloud can be used for high-volume, lower-security needs such as web-based email, and the private cloud can be used (or other on-premises infrastructure) for sensitive, business-critical operations like financial reporting.

Community Cloud. A community cloud falls between public and private clouds with respect to the target set of consumers. It is somewhat similar to a private cloud, but the infrastructure and computational resources are exclusive to two or more organisations that have common privacy, security, and regulatory considerations, rather than a single

organisation (Jansen and Grance, 2011).

Table 2.2 is the comparison of cloud models in different characteristics such as scalability, cost, flexibility, customizability, performance, security and control. In general, the hybrid model is the most advantaged model that service consumers can choose the balancing point between public model and private model.

Table 2.2. Comparison of cloud models

	Public cloud	Private cloud	Virtual Private Cloud	Hybrid Cloud
Physical hardware	Shared	Dedicated	Shared	Dedicated and Shared
Best for	Non-sensitive and public-facing operations	Sensitive, business-critical operations	Higher levels of security, privacy, and control than public cloud	Combined for the best of each cloud model
Scalable	✓	✓	✓	✓
Low cost, utility billing	✓	✗	✓	✓
Flexible	✓	✗	✓	✓
Customisable	✗	✓	✓	✓
Performance	✗	✓	✓	✓
Security and control	✗	✓	✓	✓
Predictable cost	✗	✓	✗	✓

2.2.3.4. The Conceptual Reference Model

Figure 2.3 presents an overview of the NIST cloud computing reference architecture, which identifies the major actors, their activities, and the functions in cloud computing. The diagram depicts a generic high-level architecture and is intended to facilitate the understanding of the requirements, uses, characteristics, and standards of cloud computing. The NIST cloud computing reference architecture (Liu et al., 2011) defines five major actors: cloud consumer, cloud provider, cloud carrier, cloud auditor, and cloud

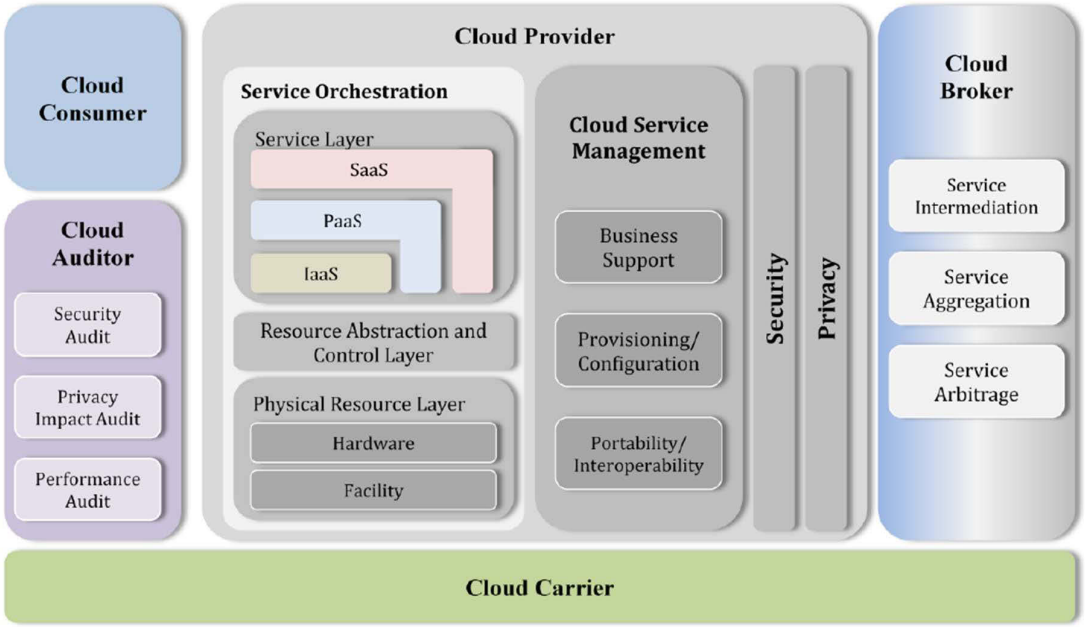


Figure 2.3. The conceptual reference model (Liu et al., 2011)

broker. Each actor is an entity (a person or an organisation) that participates in a transaction or process and/or performs tasks in cloud computing.

Cloud Provider. A cloud provider is a person, an organisation; it is the entity responsible for making a service available to interested parties. A cloud provider acquires and manages the computing infrastructure required for providing the services, runs the cloud software that provides the services, and makes arrangements to deliver the cloud services to the cloud consumers through network access.

Cloud Consumer. The cloud consumer is the principal stakeholder in the cloud computing service. A cloud consumer represents a person or organisation that maintains a business relationship with and uses the service of a cloud provider. A cloud consumer browses the service catalogue of a cloud provider, requests the appropriate service, sets up service contracts with the cloud provider, and uses the service. The cloud consumer may be billed for the service provisioned and needs to arrange payments accordingly.

Cloud Broker. As cloud computing evolves, the integration of cloud services can be too complex for cloud consumers to manage. A cloud consumer may request cloud services from a cloud broker, instead of contacting a cloud provider directly. A cloud broker is an entity that manages the use, performance, and delivery of cloud services and negotiates

relationships between cloud providers and cloud consumers.

Cloud Auditor. A cloud auditor is a party that can perform an independent examination of cloud service controls with the intent to express an opinion therein. Audits are performed to verify conformance to standards through a review of the objective evidence. A cloud auditor can evaluate the services provided by a cloud provider in terms of, for example, security controls, privacy impact, and performance.

Cloud Carrier. A cloud carrier acts as an intermediary that provides connectivity and transport of cloud services between cloud consumers and cloud providers. Cloud carriers provide access to consumers through network, telecommunication and other access devices. Cloud providers will set up SLAs with a cloud carrier to provide services consistent with the level of SLAs offered to cloud consumers and may require the cloud carrier to provide dedicated and secure connections between cloud consumers and cloud providers.

2.3. Review of Cloud SDLC

Recently, cloud computing is reshaping the ways in which information technology is being used by businesses. Cloud serves the technological needs of organisations by offering Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). It offers the *"pay per use"* service model to charge cloud users (Ahmad et al., 2015). The organisations use the computational services offered by the cloud service providers to perform the computations (Ograph and Morgens, 2008). This provides the businesses with a chance to focus more on their core capabilities rather than worrying about or investing in computing infrastructure and software (Marston et al., 2011). Consequently, services offered by the cloud are gaining more success. The increased use of cloud services has led to large-scale cloud deployments that rely on complex and distributed software and hardware systems (Zissis and Lekkas, 2012). Although cloud computing brings with it a number of key benefits, it also creates a number of challenges to the service consumer in managing their applications (Bakshi and Beser, 2016). This section is a comprehensive review of state-of-the-art cloud application SDLC approaches.

2.3.1. ITIL Lifecycle for Cloud

ITIL, formerly known as the Information Technology Infrastructure Library, is a set of practices for IT Service Management (ITSM) that focus on aligning services with the needs of the business. The objective of IT Service Management (ITSM) is the coordination of organisational capabilities to deliver IT services effectively and efficiently, resulting in cost savings and risk reduction (Brenner, 2006, Gama et al., 2013, Hochstein et al., 2005). The Information Technology Library (ITIL) is a widely-used framework for the implementation of service management in organisations. Using ITIL to seize the opportunities of the cloud, Nieves (2014) identified the challenges of

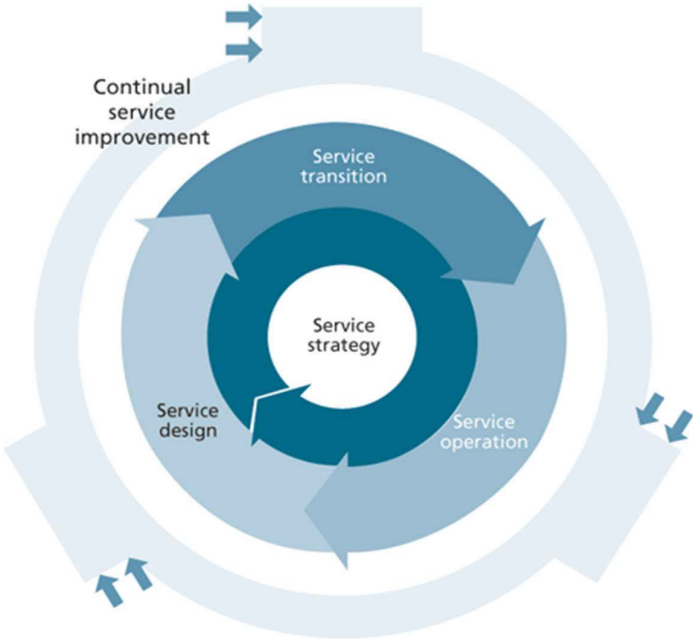


Figure 2.4. The ITIL service lifecycle (ITIL, 2014)

managing an enterprise application in a cloud setting and proposed the best practice across the ITIL service lifecycle (Figure 2.4) for cloud computing. The author proposed a methodology of management of cloud services through the ITIL service lifecycle. To address the challenges of cloud services, he adapted the phases of the ITIL lifecycle that include a range of management activities such as Service Information Management, Service Category Management, Availability Management, Change Management, and Incident management.

Service portfolio management is the process responsible for managing the service

portfolio. The service portfolio describes the provider's services in terms of business value and articulated business needs and the provider's response to those needs. A stronger and redefined cloud service portfolio will be required since some process responsibilities might remain under the control of the consumer. The purpose of capacity management is to provide a point of focus and management for all capacity and performance issues relating to services. While a cloud-based service may, in theory, be infinitely elastic, there remains a need for the cloud consumer to ensure optimum resource utilisation.

Service catalogue management. A service catalogue is often the point of entry or 'acquisition portal' for requesting supporting service offerings available to the organisation. In such cases, a clearly defined service catalogue process is required for interfacing with cloud suppliers, and for determining the organisation's cloud interfaces and organisational infrastructure dependencies related to the cloud services.

Availability management. The traditional model of availability management is based on the premise that a deployed application is dependent on the underlying infrastructure for fulfilling assurances of service availability. Increasingly, cloud consumers are applying a 'design for failure' model whereby infrastructure availability is irrelevant to application availability; and applications adapt to changes in infrastructure without downtime. Developers for cloud-based applications assume infrastructure will indeed fail, other applications will fail, and disasters will happen. Availability management applies an approach for automated recovery from failure.

Change management. Change management is responsible for controlling the lifecycle of all changes, enabling beneficial changes to be made with minimum disruption to services. One of the reasons why organisations adopt a cloud model is to free themselves of the burden of planning, testing and executing upgrades, patches, and new features although cloud consumers will still need to carry out testing, particularly when integrating cloud services. There is also an expectation that all changes will occur seamlessly, with no disruption to the service provided. A cloud provider may not understand the customer's business needs and expectations from a service quality perspective. This knowledge gap may increase the risk of the customer transitioning to a service that does not meet business requirements. It is important to avoid this type of misalignment of expectations and

objectives between the cloud provider and customer – especially since testing is a complex and multifaceted process, and deeply connected to the quality of service, and customer and business value.

Incident management. The significant activity of service operation is incident management. An incident is an unplanned interruption to an IT service or a reduction in the quality of an IT service. Incident management is the process responsible for managing the lifecycle of all incidents. The purpose of incident management is to restore normal service operation as quickly as possible and minimise the adverse impact on business operations, thus ensuring that agreed levels of service quality are maintained.

2.3.2. Extreme Cloud Programming SDLC

To realise all the advantages of these new business models of distributed, shared, and self-provisioning environments for cloud services and cloud computing resources, the traditional form of software engineering has to change as well. Guha (2013) analyses how cloud computing is going to impact the software engineering processes to develop quality software. The need for changes in the software development and deployment framework activities is also analysed to facilitate the adoption of the cloud computing platform. The author proposes an extended version of the Extreme Programming (XP) lifecycle named Extreme Cloud Programming, which is an agile process model for the cloud computing

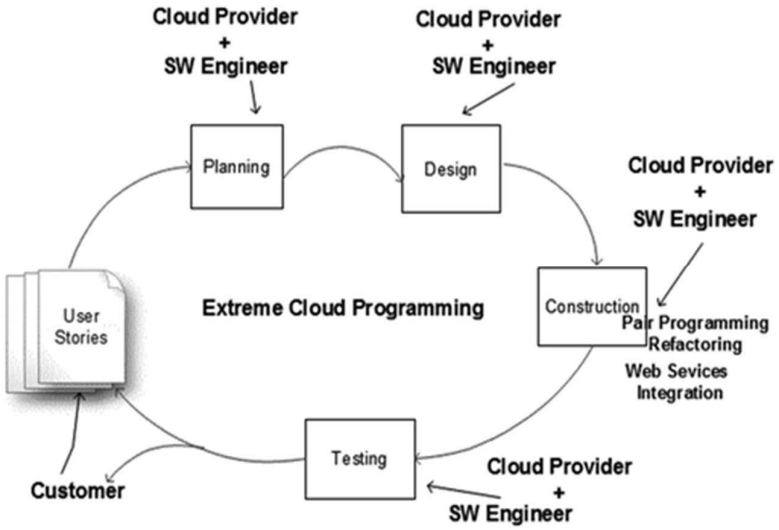


Figure 2.5. Extreme cloud programming SDLC on cloud computing (Guha, 2013)

platform (Figure 2.5). All the phases, i.e., requirements gathering, planning, design, construction, testing, and deployment need interaction with the representatives from the cloud provider. For example, the requirements gathering phase so far has included customers, users, and software engineers. However, in the cloud, it has to include the cloud providers as they will be supplying the computing infrastructure and maintaining it as well. Because the cloud providers will only know the size, architectural details, virtualisation strategy, and resource utilisation percentage of the infrastructure, the planning and design phases of software development also have to include the cloud providers. Only the coding and testing phases can be done independently by the software engineers. Coding and testing can be done on the cloud platform which is a huge benefit as everybody will have easy access to the software being built. This will reduce the cost and time of testing and validation.

2.3.3. Cloudification Security Development Lifecycle

Wagner et al. (2015) proposed a novel approach for the Cloudification Security Development Lifecycle (CloudSDL) of cloud services. CloudSDL, shown in Figure 2.6, considers both the case of application development for the cloud environment from

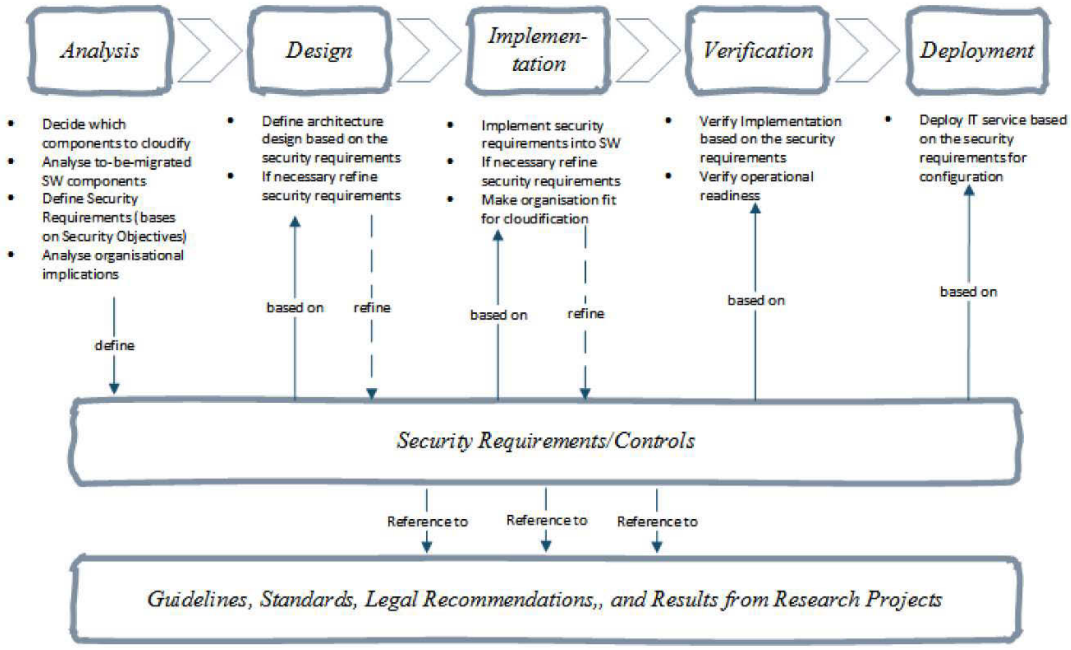


Figure 2.6. Cloudification security development lifecycle (Wagner et al., 2015)

scratch and the case of migration. It comprises five phases:

Analysis. In this phase, a decision is made about which service or which part of a service is to be migrated to the cloud. The cloud service is analysed for the initial set of security requirements. The authors suggest the need for analysing service implications on the organisation and the business. They have to be converted into security requirements.

Design. In the design phase, the architecture for the cloud service is constructed on the basis of the requirements specified in the analysis phase.

Implementation. Based on the design, the application is implemented, and the organisation is prepared for the use of the cloud service.

Verification. In the verification phase, the software is tested against the specified requirements. Also, the readiness of the organisation for the cloud service is verified.

Deployment. In this final phase of the CloudSDL, the application is deployed in the cloud environment, taking into account the security requirements related to platform configuration.

2.3.4. SOA Cloud Lifecycle Using Service Component Architecture

Using a simple travel service scenario, Ruz et al. (2011) describe a flexible SOA cloud

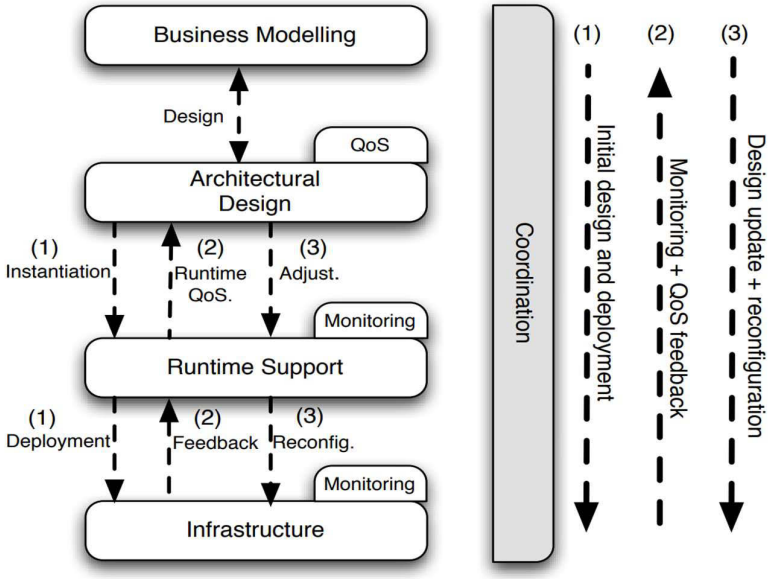


Figure 2.7. SOA cloud lifecycle using SCA (Ruz et al., 2011)

lifecycle using Service Component Architecture (SCA) as a model for managing the lifecycle of service-based applications. The authors present an integrated approach to design, monitor and manage the lifecycle of applications based on the SOA principles and take advantage of the cloud computing environment. The authors also introduce an integrated and open framework for supporting flexible cloud service management based on SOA principles. The lifecycle (Figure 2.7) consists of three phases: initial design and deployment, runtime monitoring, and design modification and reconfiguration.

Design and Deployment Phase. The business model of the application is the starting point for the SOA lifecycle. In the initial design phase, the business model is converted to an equivalent architectural description. The architectural description, possibly improved, is translated to an appropriate runtime support, which deploys and instantiates the application on the infrastructure.

Monitoring and QoS Feedback Phase. At each level, monitoring artifacts are added to be able to extract meaningful monitoring information at runtime. The goal is to provide feedback from the infrastructure level to the runtime and architectural design levels, allowing the manager of the application to obtain a QoS characterisation. The information at each level can be used by the person in charge of that level i.e. the system administrator or application designer, to take decisions in order to improve the performance of the application from its particular point of view, i.e., modifying the allocation of nodes (runtime level), or modifying the composition of the application (design level). However, the authors claim that the decision making can be improved if synthetic monitoring information to the design level is provided so that the manager of the SOA application can consider the information collected from all levels to make a better decision.

Design Update and Reconfiguration Phase. The decisions are taken into changes either to the design of service or to the associated deployment, which may require a reconfiguration phase. In this phase, the design decisions are propagated to the appropriate level, thereby keeping a consistent view. After this step, the monitoring phase now provides QoS characterisation from the updated infrastructure. Through these phases, it is possible to obtain a continuous lifecycle management of the application.

2.3.5. The Service Lifecycle on Cloud

Another approach to managing an integrated lifecycle of IT services in the cloud environment is proposed by Joshi et al. (2014). The authors propose a cloud service

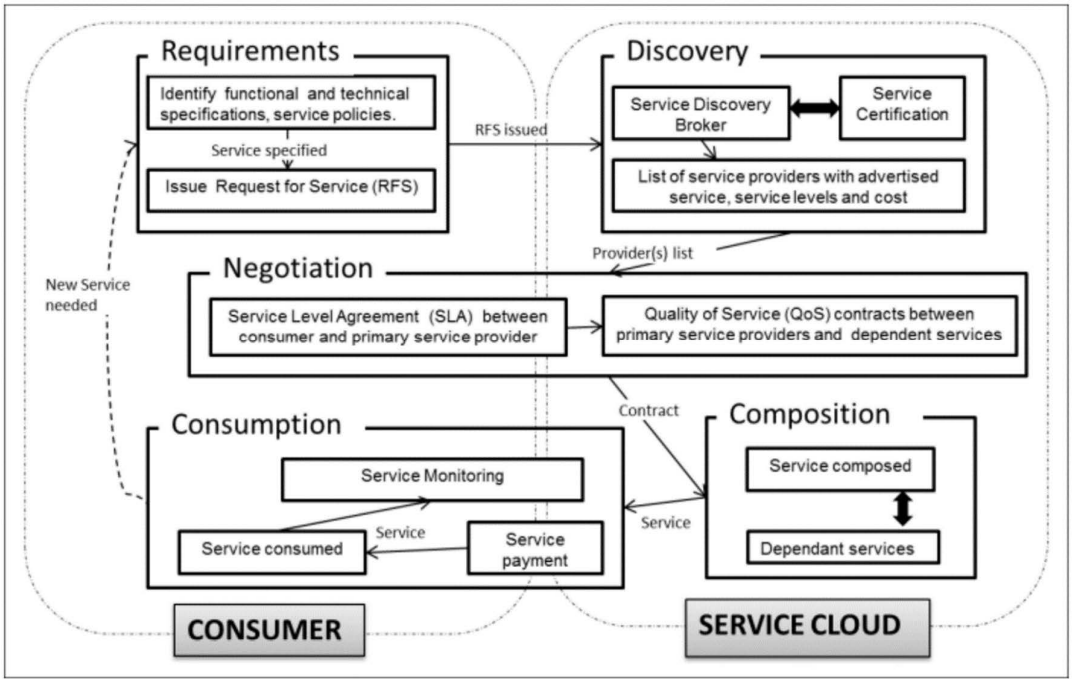


Figure 2.8. SOA cloud lifecycle using SCA (Joshi et al., 2014)

lifecycle (Figure 2.8) that consists of five sequential phases: requirements, discovery, negotiation, composition, and consumption. They have identified performance metrics associated with each phase, i.e., data quality, cost, security, service gap, SLA (Service Level Agreement), QoS, consumer satisfaction. The authors develop a cloud storage prototype to show how this lifecycle can automate the usage of cloud services,

Service Requirements Phase. In the service requirements phase, the consumer details the technical and functional specifications that a service needs to fulfil. While defining the service requirements, the consumer also specifies non-functional attributes like the characteristics of the human agent providing the service, constraints, and preferences on data quality and the required security policies for the service.

Service Discovery Phase. In the Service Discovery phase, providers are discovered by comparing the specifications listed in the requirements with service descriptions. The discovery is constrained by functional and technical attributes, and also by the budgetary, security, compliance, data quality and agent policies of the consumer. An organisation can release the requirements for a limited pre-approved set of providers. Alternatively, it

can search for all possible vendors on the Internet. If the cloud consumers find the exact service within their budgets, they can begin consuming the service immediately upon payment. However, often the consumers will get a list of providers who will need to compose a service to meet the consumer's specifications. The cloud consumer will have to begin negotiation with the service providers which is the next phase of the lifecycle.

Service Negotiation Phase. The service negotiation phase covers the discussion and agreement that the service provider and consumer have regarding the service to be delivered and its acceptance criteria. The authors found that the negotiation of SLA for the cloud services procured is the most time-consuming portion of the cloud service procurement process. While negotiating the service levels with potential service providers, consumers can explicitly specify the service quality constraints (e.g., data quality, cost, security, response time) that they require.

Service Composition Phase. In this phase, one or more components provided by one or more providers are combined and delivered as a single service to the service consumer. Service orchestration determines the sequence of the service components.

Service Consumption/Monitoring Phase. After the service is delivered to the consumer, payment is made for the same based on the pricing model agreed to in the SLA. The consumer then begins consuming the service. In this phase, the consumer will require tools that enable service quality monitoring and service termination if needed. This phase spans both the consumer and cloud areas as performance monitoring is a joint responsibility. If the consumer is not satisfied with the service quality, they should have the option to terminate the service and stop service payment.

Besides the above cloud SDLC approaches, I also review a range of cloud SDLC research from various aspects such as ITIL for cloud (Breiter and Behrendt, 2009, Karkošková and Feuerlicht, 2014); adapting traditional SDLC for cloud (Jagli and Yeddu, 2017, Krishna and Jayakrishnan, 2013, Mwansa and Mnkandla, 2014, Schmidt, 2012); green cloud SDLC (Chauhan and Saxena, 2013, Kashfi, 2017), cloud infrastructure (Baryannis et al., 2013, Rocha, 2013, Muppalla et al., 2013, Pot'vin et al., 2013, Farrell, 2011); or cloud security (Butterfield et al., 2016, Aljawarneh et al., 2017).

2.4. Review of SDLC Phases

Based on the literature review of the cloud SDLC, in this thesis, I develop a cloud SDLC for development of the cloud-based application (Chapter 4). The cloud SDLC contains five phases: Requirement Specification, Service Identification, Service Integration, Service Monitoring, and Optimisation. This section reviews the related works of these lifecycle phases.

2.4.1. Requirements Specification

The requirements specification phase aims to produce a document that specifies the requirements for a system or component. Typically, it includes functional requirements, performance requirements, interface requirements, design requirements, and

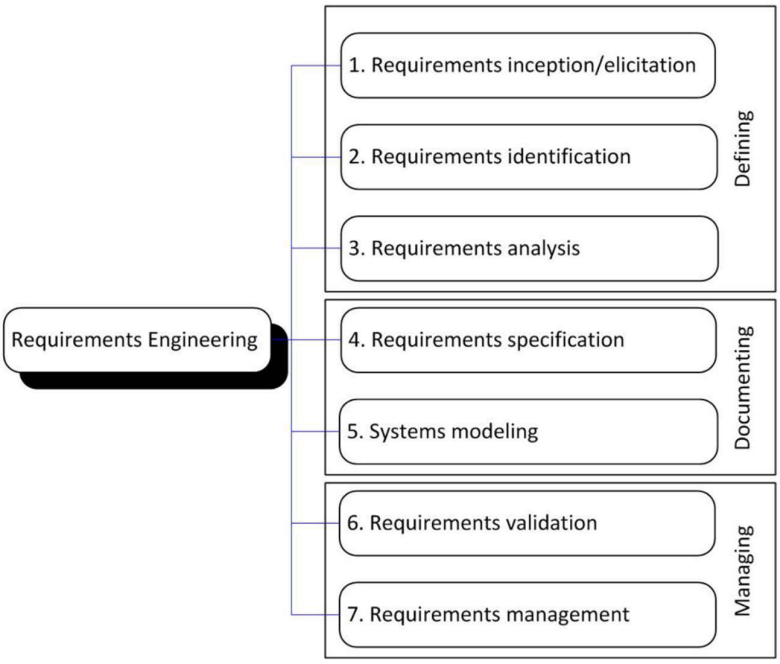


Figure 2.9. Requirements engineering activities (Cemuturi, 2014)

development standards (Figure 2.9) (IEEE, 1990). RE activities are divided into defining, documenting and managing requirements (Cemuturi, 2014). Defining requirements is concerned with gathering and analysing the requirements. Documenting activities relates to specifying and structuring the requirements. Managing requirements involves management within the project and the lifecycle.

Using a case study of a healthcare project, Vos (2017) develops a Requirements Engineering framework that contains four activities: project preparation, on-site analyses,

external expertise and requirements definition. After initiating the project, the project preparation is a phase where designers can estimate expenses for their proceedings in the project. This will be combined with information collection of the most concerning areas. The on-site analysis is most important, considering the engagement that it aims for. The onsite analysis is crucial for a good understanding of the context that the system must operate in. Also, this understanding is crucial for starting a project in these contexts as they provide social capital in the form of trust by the local stakeholders. While the on-site analyses give a better understanding of the system, during the external expertise phase, best practices are gathered and then verified on site to check whether they match the specific context. Then, the obtained requirements must be further documented, prioritised and validated. The requirement engineering process ends when a set of requirements are produced.

In an analysis of how the requirements documents should be written, Dick et al. (2017) discuss requirements engineering principles, i.e., ensuring consistency across requirements and granularity of requirements. They suggest certain criteria that every statement of requirement should meet, as follows:

- Atomic: each statement carries a single traceable element.
- Unique: each statement can be uniquely identified.
- Feasible: technically possible within cost and schedule.
- Legal: legally possible.
- Clear: each statement is clearly understandable.
- Precise: each statement is precise and concise.
- Verifiable: each statement is verifiable, and it is known.
- Abstract: does not impose a solution of design specific to the layer below.

In addition, the authors discuss other criteria that is applicable to the set of requirements as a whole:

- Complete: all requirements are present.
- Consistent: no two requirements are in conflict.

- Non-redundant: each requirement is expressed once.
- Modular: requirements statements that belong together are close to one another.
- Structured: there is a clear structure to the requirements document.
- Satisfied: the appropriate degree of traceability coverage has been achieved.
- Qualified: the appropriate degree of traceability.
- Computing coverage has been achieved.

Considering Hospital as a case study of the complex system, Chakraborty et al. (2012) present the capturing procedures of requirement specification for healthcare software. In a guideline for the establishment of requirements, Chemuturi (2013) discusses the User Requirements and System Requirements. He proposes various templates of requirements specification for different types of projects. Analysing cloud requirements and services, Zalazar et al. (2015) propose a framework for handling system requirements and supporting cloud service adoption. The authors aim to provide the complete RE process for Cloud Computing, by offering a way to manage requirements in all dimensions and also supporting cloud adoption, RE processes, and negotiation with cloud providers. Similarly, Repschlaeger et al. (2012) develop a Cloud Requirement Framework (CRF) that concentrates on the relevant requirements for adopting cloud services targeting all three service models.

2.4.2. Cloud Service Identification

Currently, service identification and framework for cloud service selection are the subjects of many research studies (Arun et al., 2017, Ghamry et al., 2017, Hajlaoui et al., 2017, Rotem et al., 2016, Yang et al., 2006, Zisman et al., 2013). They are focused on various approaches (Table 2.1), i.e., cloud service trustworthiness (Arun et al., 2017), QoS matching and ranking (Hajlaoui et al., 2017, Garg et al., 2013, Tripathi et al., 2017, ur Rehman et al., 2013, Zheng et al., 2013); or user feedback ranking (Rotem et al., 2016, Yang et al., 2006, Qu et al., 2013) (Navinkumar and Raghul, 2016). For example, Garg et al. (2013) propose the Service Measurement Index Cloud framework (SMICloud) which helps cloud customers to find the most suitable cloud provider which can initiate SLAs. The SMICloud framework provides features such as service selection based on

QoS requirements and the ranking of services based on previous user experiences and performance of services. It is a decision-making tool, designed to provide an assessment of infrastructure cloud services in terms of KPIs and user requirements.

Qu et al. (2013) propose a framework of service selection based on the aggregation of the feedback from service consumers and the objective performance measurement from a trusted third party's testing. The framework consists of four components, namely, (1) cloud selection service, (2) benchmark testing service, (3) user feedback management service, and (4) assessment aggregation service.

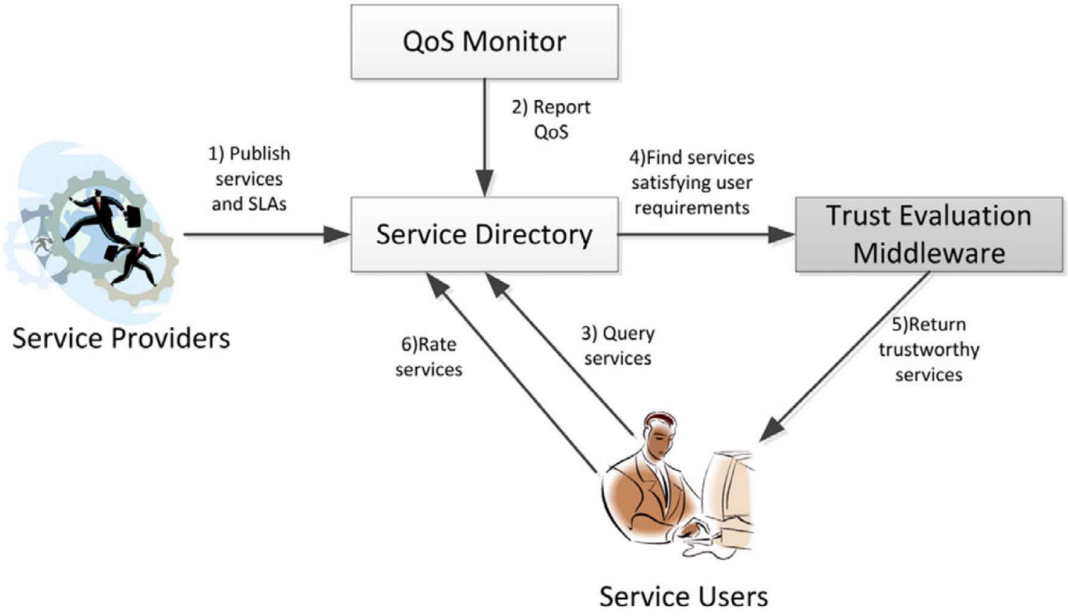


Figure 2.10. Service selection system architecture (Tang et al., 2017)

Tang et al. (2017) propose a trustworthy selection framework for cloud service selection, named TRUSS (Figure 2.10). The authors propose an integrated trust evaluation method via combining objective trust assessment and subjective trust assessment. It is based on QoS monitoring, while the subjective trust assessment is based on user feedback ratings.

Arun et al. (2017) propose a Cloud Service Identification System (CSIS) to help customers in the identification of cloud services and suggest other relevant services to suit their needs. The system has five different components: crawler, identifier, indexer, search engine, and recommender. The crawler is a bot which automatically browses the Internet, generally for the purpose of Web indexing. The identifier is used to identify if a web page is a genuine cloud service based on its service score. The indexer is used for

registering a cloud service which is identified. Search Engine is used to browse for cloud computing products. Recommender is used to recommend both content-based and collaborative recommendations. Hajlaoui et al. (2017) develop a Configurable Cloud Service Discovery and Selection System (C2SDS2) that aims to guide service consumers in retrieving configurations of IaaS Cloud resources over the Internet. The C2SDS2 takes into account both user functional and non-functional requirements in the selection process. The authors focus deeply on the design, discovery, and selection of configurable IaaS services in single Cloud computing and use QoS ranking for cloud service selection.

In early research of service repository, Vitharana and Jain (2012) introduced a Knowledge-Based Component Repository (KBCR) for enabling requirements analysis. The repository includes basic information about services (name, version, functionalities, and QoS attributes), facet information, business process templates, relationships among components, and provides support for a search capability. Yu et al. (2012) proposed a semantically enhanced Service Repository for user-centric service discovery and management. The repository consists of two main components: a service registry for storing and managing service metadata (i.e., service name, service version, provider and service descriptions) and a service discovery component that enables searching of services. Lakshmi and Mohanty (2012) described the design of a scalable Service Repository which was implemented using relational database supporting algebraic operators for service composition which employed Composition Search Trees. The database service includes five tables: Providers, Services, Parameters, Service Input and Service Output. Service providers are categorised by reputation (using categories Best, Good, Average and Below Average), and services are classified using QoS attributes. This information is used to search for services in the registry and to compose business processes based on the identified services.

Shetty and D'Mello (2013) reviewed the Service Repository strategies and service discovery techniques that aim to support the diversity of cloud services. The cloud service discovery feature supports search and browsing of services based on functional and non-functional properties. The authors classified discovery methods according to different architectures of the cloud Service Repository into centralised architectures and distributed architectures. They also described the various service discovery algorithms used in the

literature for cloud service discovery such as functional description-based methods (keyword (syntactic) based discovery, semantic-based discovery and hybrid matching), non-functional description methods (static QoS based method and dynamic QoS based method).

A method for automating the usage of cloud services that support the identification of cloud services was proposed by Joshi et al. (2014). The authors identified the performance metrics associated with each lifecycle phase including data quality, cost, and security metrics based on SLA (Service Level Agreement) and consumer satisfaction. They also proposed a Service Repository with a discovery capability for managing cloud services lifecycle. The authors divided the cloud services lifecycle into five phases: requirements specification, discovery, negotiation, composition, and consumption. During the service discovery phase, the service consumers search for services using service description and provider policies in a simple service database. Service information is stored as a Request for Service (RFS) that contains functional specifications, technical specifications, human agent policy, security policy, and data quality policy. Field et al. (2014) presented a European Middleware Initiative (EMI) Registry that uses a decentralised architecture to support service discovery for both hierarchical and peering topologies. The objective of the EMI Registry is to provide robust and scalable service discovery that contains two components: Domain Service Registry (DSR) and Global Service Registry (GSR). Service discovery is based on service information stored in service records that contain mandatory attributes such as service name, type of service, service endpoint, service interface, and service expiry date.

Vukojevic-Haupt et al. (2014) proposed a service selection method for on-demand provisioned services. Services are provided by a third-party provider and service consumers have no knowledge about the implementation and the underlying infrastructure that supports the delivery of services. The authors developed an entity relationship diagram of the service registry that contains service information and metadata, including functional and non-functional properties, service configuration parameters, service provider information, the functional description of the service, and QoS attributes. Bauer et al. (2015) presented the design of an advanced SOA repository enriched with analysis capabilities. The repository contains various types of services and

their relationships. The authors proposed a meta-model for repositories to analyse service dependency and the impact of changes. Table 2.3 shows the summary of the main research efforts of cloud service selection in which the techniques are used in each area of research.

Table 2.3. Cloud service identification approaches

	User feedback	QoS ranking	Trustworthiness	Third-parties	Repository Design
Qu et al. (2013)	✓	✗	✗	✓	✗
Garg et al. (2013)	✓	✓	✗	✗	✗
Tang et al. (2017)	✗	✓	✓	✗	✗
Arun et al. (2017)	✗	✓	✓	✗	✗
Hajlaoui et al. (2017)	✗	✓	✗	✗	✗
Vitharana and Jain (2012)	✗	✓	✗	✗	✓
Yu et al. (2012)	✗	✓	✗	✗	✓
Lakshmi and Mohanty (2012)	✗	✓	✗	✗	✓
Joshi et al. (2014)	✗	✓	✗	✗	✓
Field et al. (2014)	✗	✗	✗	✗	✓
Vukojevic-Haupt et al. (2014)	✗	✓	✗	✗	✓
ur Rehman et al. (2013)	✗	✓	✗	✗	✗
Navinkumar and Raghul (2016)	✓	✓	✗	✗	✗

Zheng et al. (2013)	✗	✓	✗	✗	✗

2.4.3. Cloud Service Integration

Integrating applications and cloud services in the cloud computing environment has been a challenge for enterprises (Bernstein and Haas, 2008). The IT industry introduced a concept of Enterprise Service Bus (ESB) which is bus-architectural technology to provide an infrastructure for SOA implementation in enterprise applications as well as integration (Chappell, 2004). In the market, major vendors (e.g IBM, Oracle, MuleSoft, Progress Software, Software AG, and Red Hat) enhanced the usage of SOA with a virtual bus to integrate many applications and services together (Schmidt et al., 2005, Menge, 2007, Bygstad and Aanby, 2010, Chappell, 2004) . Most of these ESBs contain workflow engine and monitoring modules to make sure all services and systems are connected reliably and securely. ESBs also contains various modules such as an adapter module, mediation module, message routing module, security module, and a management module. Each module is responsible for specific tasks.

Besides the commercial ESB frameworks above, there are other research approaches on cloud service integration. For example, Chen (2012) proposed a comprehensive high-level architecture for system and cloud service integration. The architecture introduces the rule-based BPM engine to automate and streamline business process management in organisations. The proposed architecture is fully implemented with the integration solutions currently available on the market. The author also discussed the demands on the out-of-box integration of business processes across organisational boundaries even though most companies still run their business process management on their on-premises systems. With more business processes migrating to cloud computing, enterprises need highly integrated infrastructure to manage their business processes globally.

Tang et al. (2010) extended the enterprise service-oriented architecture (ESOA) style to a new hybrid architectural style, Enterprise Cloud Service Architecture (ECSA) that specifies the vocabulary of ECSA architectural elements; encapsulates important decisions about the ECSA architectural elements; and emphasises important constraints on the elements and their relationships. Baude et al. (2010) presented the SOA4All

Distributed Service Bus, as a core infrastructural enabler of cloud services. First, the authors discussed how the merger of Petals Enterprise Service Bus (PEtALS) and ProActive Cloud Automation provides scalable message routings and technical registries and then proposed the fully P2P-based semantic space implementation on top of ProActive that yields SOA4All's shared semantic memory and coordination platform.

Zhang and Zhou (2009) presented seven architectural principles and derived ten interconnected architectural modules to form a reusable and customisable Cloud Computing Open Architecture (CCOA). This architecture is a cloud computing-centric service-oriented architecture framework bridging the power of SOA and virtualisation in the context of cloud computing ecosystem. It includes basic architectural elements, service orientation, and cloud principles.

Similarly, Rimal et al. (2011) provided key guidelines to software architects and Cloud Computing application developers that will be helpful for software architects and developers while designing cloud-based applications. The authors considered the system requirements for cloud computing systems from the point of view of enterprise users, such as infrastructure, storage, scalability, compliance, and security. They also addressed the cloud service reliability challenge as most cloud vendors today do not provide high availability assurances. This challenge is particularly an issue with enterprises, which use a set of cloud services hosted in various cloud computing environments. Although cloud service providers have made huge investments to make their systems as reliable as they can, the cloud service may stop working at any time due to a service incident.

2.4.4. Cloud Service Monitoring

In cloud computing, monitoring has become a necessity to fulfil the user's requirements and to meet the Service Level Agreement (SLA). The aspect and perspectives of monitoring are different for the cloud service provider and the cloud consumer. Cloud service providers seek to monitor cloud services for efficient resource utilisation and to ensure compliance with the SLA. Studying cloud provider monitoring, i.e., Anwar (2015); Lin et al. (2016); Povedano-Molina et al. (2013), and Tovarnak and Pitner (2012) is to understand the methodologies to ensure the quality and performance of cloud services. However, this section only discusses the related works on the service consumer

perspective as the proposed SDLC is from this perspective. In the cloud environment, monitoring the cloud servers can affect the speed and performance of the system. To handle this issue, (Suneja et al., 2016) have proposed a brand-new paradigm of cloud monitoring called Near Field Monitoring (NFM). Operating outside the context of the target systems, NFM enables always-on monitoring independently without modifying or accessing the enterprise system.

Zhao et al. (2015) present a framework for cloud-based database management. The authors monitor two metrics: (i) data freshness and ii) transaction response time. The framework has three modules: (i) the monitor module, (ii) the control module, and (iii) the action module. The model monitors database services and performs adaptive actions to avoid any violation of SLA, which can be costly in terms of financial penalty. It monitors application-defined SLA violations. It also reduces the cost of usage by updating the resources required. After that, Zhao et al. (2017) develop a toolkit to simulate and evaluate monitoring mechanisms in the cloud computing environment. SimMon provides a controllable and repeatable way to evaluate the mechanisms used in cloud monitoring systems. It is used in two main usage scenarios: (1) to test whether a monitoring strategy will work well in a data centre before it serves the data centre; and (2) to compare the results of different strategies and decide which one is the most appropriate strategy for a specific data centre.

A general cloud monitoring infrastructure which can work with all clouds is not available. Most of the existing paradigms address a particular cloud. Therefore, it becomes essential to have a general infrastructure for the cloud that can be implemented on all clouds. In (Yazdanov and Fetzer, 2014), the authors discuss a monitoring model for enterprise applications deployed in multiple clouds from the customer perspective. They describe a fine-grained monitoring framework which manages, profiles and keeps track of enterprise applications in clouds. The authors validate the framework by simulating a scenario in which Virtual Machines (VMs) running Hadoop applications are Denial-of-Service (DoS) attacked by compromised VMs on the same tenant network. The framework shows the ability to detect the attacks in near real-time.

Ciuffoletti (2016) addresses this issue by proposing an extension to the Open Cloud Computing Interface API which is an on-demand monitoring as-a-service model. OCCI

API is an open source IAAS service API that provides some standards and protocols. This work has focused on building a monitoring platform of OCCI and introduces a monitoring agent. In this study, performance is not a major concern. Another aspect of this work is that it has only focused on user level monitoring. Service provider level monitoring has not been considered in this study.

2.4.5. Cloud-based Application Optimisation

Optimisation techniques to improve the reliability and performance of enterprise applications that include fault prevention and forecasting have been the subject of research interest for years (Tsai et al., 2008). Using redundancy-based fault tolerance strategies, Zibin and Lyu (2008) proposed a distributed replication strategy evaluation and selection framework with fault tolerant strategies to optimise cloud service utilisation. The authors compared various replication strategies and proposed a replication strategy selection algorithm. Other authors focused on QoS optimisation, for example, Deng and Xing (2009) proposed a QoS-oriented optimisation model for service selection. This approach involves developing a lightweight QoS model, which defines functionality, performance, cost, and trust as QoS parameters of a service. The authors verified the validity of the model by simulation of cases that show the effectiveness of service selection based on these QoS parameters. They formalised the problem of finding an optimal set of adaptations, which minimises the total cost arising from Service Level Agreement (SLA) violations and the cost of preventing violations. The authors presented possible algorithms to solve this complex optimisation problem and described an end-to-end approach based on the PREvent (Prediction and Prevention based on Event monitoring) framework. They discussed experimental results that show how the application of their approach leads to reducing service provider costs. They also explained the circumstances in which different algorithms lead to satisfactory results.

The QoS that is calculated from service performance history records, associated runtime performance in the past can be used to predict future QoS values. Wenmin et al. (2011) presented a history record-based service optimisation method, called HireSome, that aims at enhancing the reliability of the service composition plan. The method takes advantage of service QoS history records collected by the consumer, avoiding the use of QoS values

recorded by the service provider. The authors used a case study of a multimedia delivery application to validate their method. Lee et al. (1999) presented a QoS management framework that is used to measure QoS quantitatively and to plan and allocate resources analytically. In this model, end-user quality preferences are considered when system resources are apportioned across multiple applications, ensuring that the end-user benefits are maximised. Using semantically based techniques to optimise service delivery automatically, Fallon and O'Sullivan (2014) introduced the Semantic Service Analysis and Optimisation (AESOP) approach and a Service Experience and Context Collection (SECCO) framework. The AESOP knowledge base models the end-user service management domain in a manner that is aware of the temporal properties of the services. The autonomic AESOP Engine runs efficient semantic algorithms that implement the Monitor, Analyse, Plan, and Execute (MAPE) functions using the temporal properties to operate on small partitioned subsets of the knowledge base. A case study is used to demonstrate that AESOP is also applicable in the Mobile Broadband Access domain.

2.5. Evaluation of The Existing Work

Lack of methodology for management of cloud service from the cloud service consumer perspective

The existing literature mostly focuses on techniques and methodologies designed to assist the cloud providers in management in their cloud service or data centres, and there is very little work on the methodology to assist the service consumer regarding the management of cloud services. These efforts take the approach of adapting traditional methodologies for the cloud. Notwithstanding these efforts to transform ITIL to cloud computing or apply software engineering processes for the cloud, they are primarily focused on on-premises solutions and are of limited application for managing cloud-based services. Traditional SDLC may be translated to the context of cloud computing, but there are many challenges that have not been addressed, i.e., cloud service selection, cloud service failover, cloud service monitoring, and cloud service optimisation (Cardoso and Simões, 2012). The main focus of service management related research (i.e., ITIL) is in the management of on-premises services (Gama et al., 2013, McNaughton et al., 2010, Brenner, 2006, Hochstein et al., 2005), and this makes it difficult to apply the results to

situations that involve large-scale use of cloud services. While cloud computing does not change the basic principles of service management, it requires a shift in perspective so that specific characteristics of cloud services are taken into account. This includes the separation of the role of service provider and service consumer and the high-level of autonomy associated with cloud services. Moreover, most research efforts only focus on cloud infrastructure for deployment without considering the use of SaaS during application development. They also neglect to address important issues that service consumers must face in cloud computing environments.

Lack of an approach that supports the cloud service consumer in all the tasks of SDLC

In the literature, there are a large number of research works on different parts of SDLC: service selection, service integration, service monitoring, and service optimisation. However, a comprehensive framework that performs all of these activities that are required for the management of cloud services from the consumer perspective throughout all phases of SDLC is missing in the literature. For example, the cloud service selection is a process that has several criteria on the basis of which a service consumer would like to select a cloud service. There is a gap in terms of the connection between the requirements specification and service selection.

Lack of proactive monitoring for management of cloud services.

The existing literature does not provide an approach to automatically warn the cloud service users of significant changes and trends in the QoS of their services, so they can take pre-emptive measures in time to avoid service disruption and degradation. There is a need for such a mechanism to assist the user to effectively use the available QoS monitoring information to detect such changes in advance or with as little delay as possible by automatically processing this information to generate an early warning.

2.6. Conclusion

In this chapter, I have reviewed the works that are related to the methodology for management of cloud services. These related works focus on cloud service SDLC and the proposed lifecycle phases. This chapter begins by discussing the research background and

then looking at the current state of cloud service SDLC. As examined in the literature, although the investigation of methodology for management of cloud services has been the subject of recent research interest and efforts, most of the work takes the perspective of service providers. Relatively little research has been done so far on the service consumer perspective, and most of the studies focus on the IaaS services that provide a platform for deployment of enterprise applications. Thus, very limited attention has been paid to using location-based QoS information for the optimisation of cloud-based enterprise applications. In the next chapter, I will describe the research approach and research methods that have been undertaken in this study.

CHAPTER 3.

RESEARCH METHODOLOGY

3.1. Introduction

The main goals and objectives of this practice-based research are described in Chapter 1. These are:

- to advance understanding of the challenges of managing cloud services in enterprise applications;
- to develop a System Development Lifecycle for enterprise applications;
- to propose a methodology for management of cloud services through the lifecycle; and
- to implement a cloud framework that supports the lifecycle activities and manages cloud services.

In this chapter, the overall approach to this research project is outlined to achieve the above research goals. In the following sections, I describe an approach to interpretive research which draws on the Design Science Research Methodology Process Model. Details of the specific research methods, including motivation and case studies, are also presented in Section 3.4.

3.2. Research Paradigm

Before describing the research methodology and research methods, I will outline the research paradigm that is applied in this thesis. Crotty (1998) contends that the terms and concepts of the research process - methods, methodology - should be considered and well understood by researchers. He defines these terms as follows:

- The methodology is the strategy, approach, action plan and process that involves the selection of research methods. Research methodology is the process of how the research project is done or the way to address the research problems.
- Methods are the techniques used for conducting research and the procedures used to collect and analyse data. During the development of a research plan and the selection of research methods, researchers need to consider two questions (Johnston, 2009):

- Which methods should be used?
- How can these methods be justified?

In the following sections, I describe and justify the specific research methods that have been employed in the thesis. The overall structure of this research has been informed by design science. The methods of Design Science share a common emphasis on intervening in the situation to build understanding.

3.3. Design Science

Because this thesis is a practical research project applied to solve the problems of managing cloud services in enterprise applications, the primary research methodology which has been used is *design science*. Design science, as conceptualised by Simon (1996), is well-known in the Engineering, Computer Science and Information System disciplines because it supports a pragmatic research paradigm to solve real-world problems. This methodology is "a framework for IT research [which] lies in the interaction of design and natural sciences. IT research should be concerned both with utility, as a design science and with theory, as a natural science. The theories must explain how and why IT systems work within their operating environments." (March and Smith, 1995)

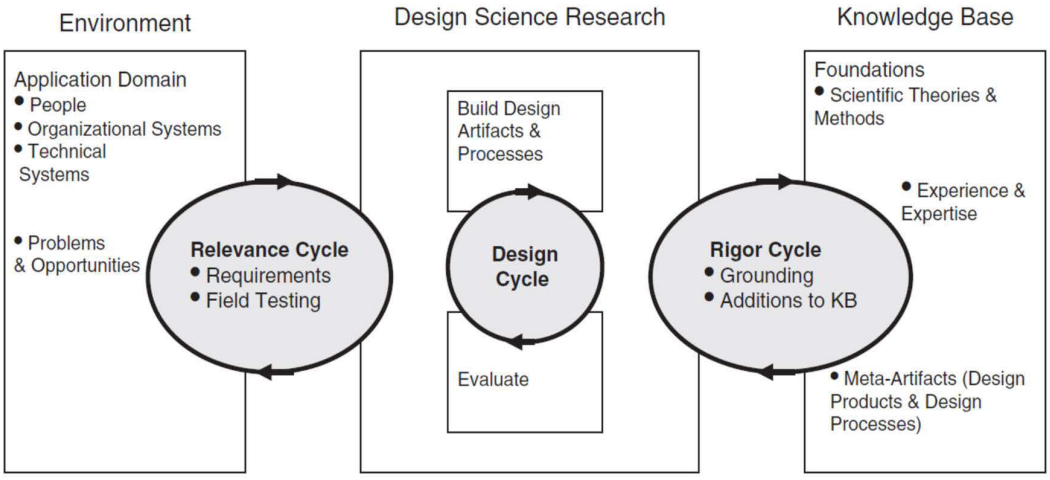


Figure 3.1. Design science research cycles (Hevner, 2007)

Design science has been an important paradigm of Information Systems (IS) research since the inception of the field, and its general acceptance as a legitimate approach to IS research is increasing (Hevner and Chatterjee, 2010, Kuechler and Vaishnavi, 2008). In the IS discipline, there are a number of researchers (Adams and Courtney, 2004, Cole et al., 2005, Rossi and Sein, 2003, Hevner and Chatterjee, 2010, Peffers et al., 2007, Gregor and Hevner, 2013) who have sought to provide guidance to define the design science research process. This rigorous research process involves creating artifacts to solve observed problems, to make research contributions, to evaluate the designs, and to communicate the results to appropriate audiences (Von Alan et al., 2004). The artifacts may include constructs, models, methods, and instantiations. The artifacts and body of knowledge associated with this research will be developed and understood through three design science research cycles in Figure 3.1 (Hevner, 2007). "The development of the artifact should be a search process that draws from existing theories and knowledge to come up with a solution to a defined problem." (Peffers et al., 2007). The aim of this practice-based research project is to develop a methodology for management of cloud services in the hybrid cloud context. To achieve this research aim, I apply the Design Science Research Methodology (DSRM) process model of Peffers et al. (2007). This model originates and summarises the ideas of Archer (1964), Walls et al. (1992), Nunamaker Jr et al. (1990), Eekels and Roozenburg (1991), Rossi and Sein (2003), Takeda et al. (1990) and Hevner (2007). It consists of six activities in sequence:

Activity 1: Problem identification and motivation. This is the first activity in the process model that aims to identify specific research problems and research motivation. Because an artifact will be developed to provide a solution for research problems efficiently, this activity not only helps the researcher to understand the theories and concepts that are related to research problems but also shows the research significance. Additionally, the problem identification process can draw initial ideas for the design of an artifact.

Activity 2: Define the objectives for a solution. This activity relates to inferring the research objectives from the identified research problems. The objectives involve the solutions for problems and include the body of knowledge of the research area. To define the research objectives, a literature review needs to be done to understand the current state of the research problems and solutions. This literature review shows the research gap that

reveals the novelty of the study. It also includes discussion in terms of how the proposed artifacts support solutions for problems.

Activity 3: Design and development. This activity involves a range of techniques, i.e., design, analysis, modelling, and coding to create the artifact. “Conceptually, a design research artifact can be any designed object in which a research contribution is embedded in the design” Peffers et al. (2007). This activity also includes determining the features, functionality, and architecture of the artifact.

Activity 4: Demonstration. This activity relates to setting up the experimental environments to demonstrate the use of the artifact to solve the research problems. There are a number of research methods that can be used such as experimentation, simulation, case studies, and participated research. During the demonstration activity, the results will be gained to prove the effectiveness of the artifact in solving the research problems.

Activity 5: Evaluation. This activity involves discussion on the experimental results collected from the demonstration in order to measure how well the artifact can address the research problems. It requires data analysis or statistical techniques depending on the nature of the research problems and the artifact. The evaluation can be a comparison of the artifact’s features with the other solutions, quantitative performance measures, the results of satisfaction surveys, client feedback, or simulations. At the end of this activity, the next step can iterate back to activity 3 to improve the artifact.

Activity 6: Communication. Open communication on the research problems, findings, the design of the artifact and other research contributions to researchers and other audiences is critical. The research results are non-orally described in the scholarly research publications. Based on the guidance on presenting the design science research knowledge contributions of Hevner (2007), Hevner and Chatterjee (2010), Sørensen (2002), Kuechler and Vaishnavi (2008), (Zobel, 2004), and Sein et al. (2011), Gregor and Hevner (2013) proposed a pattern schema or template of publication for a design science study that contains seven sections: (1) Introduction, (2) Literature review, (3) Research method, (4) Artifact Description, (5) Evaluation, (6) Discussion and (7) Conclusions.

3.4. Research Techniques and Methods

In the previous section, the overall shape of the research approach drawing on design science has been outlined. In this section, I describe the specific research techniques and methods which have been applied in this thesis. In detail, I show how the process of motivating, developing, designing, demonstrating, evaluating, and communicating the artifact is consistent with the Design Science Research Methodology (DSRM).

3.4.1. Motivation Research

Motivation research, which is a form of qualitative research, is used to identify the research problems through the means of a business case study. The research using this method aims to find a solution for the practical issues that are faced by a society or industrial organisation. My industrial experience in an IT company provides the motivation for this research. Working in the industrial projects of that company, I had to deal with the challenges of managing cloud services in applications. Those experiences helped me to identify the research problems for this thesis.

3.4.2. Literature Survey

Once the problems are identified in the motivation example, the literature is examined to propose solutions for research problems. Two types of literature need to be reviewed: (1) the conceptual literature concerning the concepts and theories, and (2) the empirical literature consisting of current studies which are similar to the proposal. The primary outcome of this review will be the knowledge of what data and other materials are available for operational purposes and this will formulate the research problems in a meaningful context. For this purpose, the abstracting and indexing journals and published or unpublished bibliographies are the first place to go. Academic journals, conference proceedings, government reports, and books must be tapped into depending on the nature of the problem. In this process, it should be remembered that one publication will lead to another one that is related to the same topic or research trend. The earlier studies, if any, which are similar to the research topic, should be carefully evaluated.

During the problem identification activity of the DSRM model, all available literature that concerns the research problems must be investigated. It helps to understand the terms, definitions, concepts, and theories in the field that are related to the research topic. The

literature survey "helps [us] to know if there are certain gaps in the theories, or whether the existing theories applicable to the problem under study are inconsistent with each other, or whether the findings of the different studies do not follow a pattern consistent with the theoretical expectations and so on" (Kothari, 2004). In this thesis, a review of the research on related problems is also undertaken in Chapter 2. This is useful for suggesting initial ideas that can be used to address research problems.

3.4.3. Action Design Research

Action Design Research is

> "a research method for generating prescriptive design knowledge through the building and evaluating ensemble IT artifacts in an organizational setting. It deals with two seemingly disparate challenges: (1) addressing a problem situation encountered in a specific organizational setting by intervening and evaluating; and

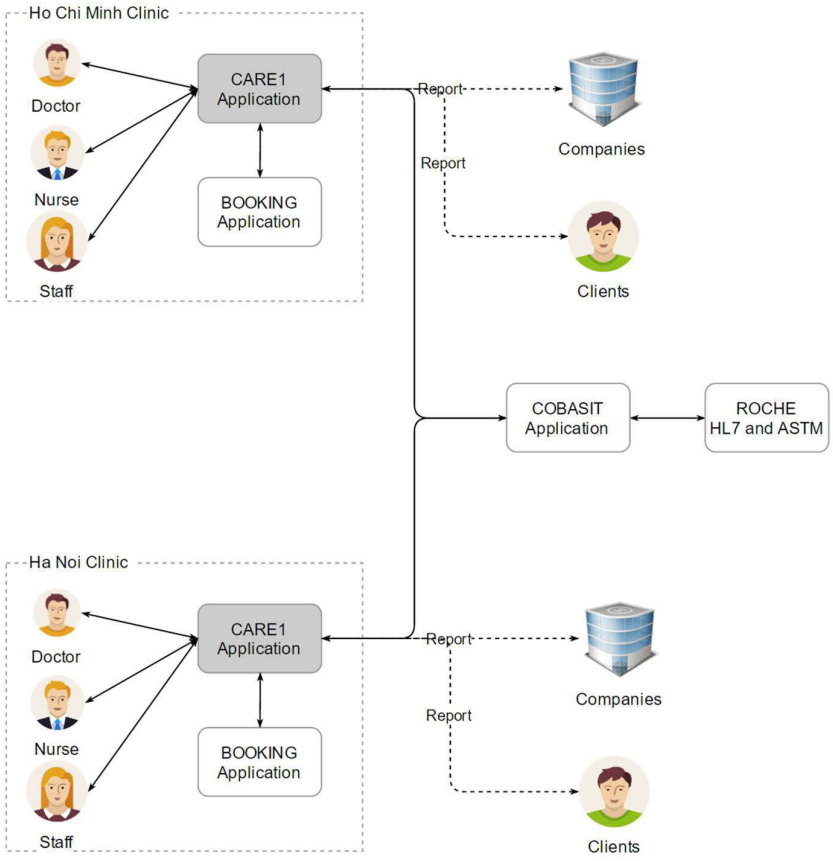


Figure 3.2. Overview of FMP

(2) constructing and evaluating an IT artifact that addresses the class of problems typified by the encountered situation." (Sein et al., 2011)

In response to the above challenges, this method aims to put theory into practice and to link thinking with doing (Susman, 1983). In this thesis, I verify the proposed cloud SDLC using an industrial project which developed a hospital management system called FMP. FMP aims to implement an enterprise application for the clinics of Family Medical Practice (FMP) which is Vietnam's leading international primary health care provider with a high standard of client care. The main functionality of FMP is the management of company health check-ups, private health check-ups, and medical records. Figure 3.2 shows the overview of the logical system architecture of FMP. Appendix B is a part of the FMP business requirements and specifications. Using SOA, this application is required to integrate with other on-premises applications (e.g., Booking Application and CobastIT application) and cloud services (e.g., payment services and storage services).

3.4.4. Simulation Research

In the context of business and social sciences, the concept of simulation refers to "the operation of a numerical model that represents the structure of a dynamic process. Given the values of initial conditions, parameters, and exogenous variables, a simulation is run to represent the behavior of the process over time" (Meler et al., 1969). The simulation approach which is a research method in quantitative research involves the building of an artificial context which is related to generating data. This method is used to evaluate the design of artifacts (Hevner, 2007). This permits an observation of the features of enterprise applications or artifacts under controlled conditions. The simulation approach is also valuable in testing models for understanding forthcoming conditions (Gregor and Hevner, 2013). In Chapter 6, a simulation environment is set up to gain experimental data to test the reliability feature of the Service Consumer Framework. Then, I simulate a multi-site monitoring environment (Chapter 7) to evaluate the proposed location-based QoS approach to optimise cloud-based enterprise applications.

3.5. Conclusion

In this section, I have described and justified the research approach and specific methods

used in this thesis. The research approach is drawing on design science, and the DSRM Process Model that is proposed by Peffers et al. (2007) is applied. Following the process model, the Design Science Research Study Schema is used to format this thesis. Then in Section 3.4, I describe the research methods and techniques that are utilised in this research and how they are applied in each activity of the design science process. The next chapters provide descriptions of the research artifacts: the cloud Service Consumer SDLC (SC-SDLC) in Chapter 4 and the Service Consumer Framework (SCF) in Chapter 5.

CHAPTER 4.

CLOUD SERVICE CONSUMER SDLC

4.1. Introduction

SOA is a successful enterprise application architecture that is implemented based upon the use of services. The SOA development process focuses on design and implementation of on-premises services. In the traditional SDLC, the organisation normally plays both the service provider role and service consumer role. It does not explicitly differentiate between the service provider and service consumer. The on-premises service is designed, developed and provisioned by the same organisation which consumes the service. This situation is valid in the traditional SOA environment, but it is not suitable for the cloud context. The movement to cloud computing creates a trend of enterprise application development by consuming a number of cloud services (Li et al., 2017). Organisations consume a large number of cloud services provided by third-party organisations. In the cloud context, the organisation only plays the role of service consumer in the SDLC. Existing application development process models applying for traditional SOA are not adequate for this type of cloud service-oriented enterprise application (Chhabi Rani et al., 2017).The service consumer needs a new cloud SDLC that focuses on selecting, integrating and managing cloud services instead of on-premises service design.

The increase in the number of cloud services and the ability to manage and control them programmatically through APIs have enabled the rise of cloud-based enterprise application development (Weber et al., 2016). The primary role of the organisation is shifting from implementation of on-premises services to integration and management of cloud services. In many cases, similar services are available from various cloud providers with different interfaces, protocols, and Quality of Service (QoS) attributes (Shetty and D'Mello, 2013). At design time, service consumers need to select suitable services for their applications among a large number of available cloud services. Due to the vast diversity of cloud services, it has become difficult to decide which services they should use and what is the basis for their selection (Garg et al., 2013). In addition, when the number of cloud services that are used inside the enterprise application becomes significant, the service consumer needs a new approach to manage these cloud services. At runtime, the service consumer needs to ensure the continuity of the enterprise application that is impacted by service disruptions, service evolution, and service maintenance activities. They also need a method to monitor the service performance and

availability so that they can proactively rectify these service incidents.

In the literature review in Chapter 2, I reviewed the SOA concepts, principles and SOA SDLC (Section 2.3). I discussed the related works on the cloud service SDLCs of Panigrahi et al. (2016), Weber et al. (2016), Kumar et al. (2013), Guha (2013), Papazoglou et al. (2011), Ruz et al. (2011) and Joshi et al. (2016). Although the cloud SDLC has been the subject of recent research interest, most of the work takes the service provider perspective in order to guide service providers in relation to developing and delivering their cloud services. Accordingly, relatively little research has been done so far on the SDLC for development of enterprise applications using cloud services. In this chapter, I propose a Cloud Service Consumer SDLC (SC-SDLC) (Feuerlicht and Thai Tran, 2015, Tran and Feuerlicht, 2016c) for developing a cloud enterprise application (Section 4.2). The lifecycle contains five phases which are classified into design-time activities: Requirements Specification (Section 4.3), Service Identification (Section 4.4), and Service Integration (Section 4.5); and run-time activities: Service Monitoring (Section 4.6), and Optimisation (Section 4.7). In each phase, the life cycle activities are specified in detail to develop an enterprise application for the hybrid cloud. The development processes of acquiring cloud services are largely independent of the type of cloud service (IaaS, SaaS, PaaS), cloud deployment (private, public, hybrid), or service domain (for example, computing services, healthcare and financial services). In Section 4.8, the case study of FMP is discussed to evaluate the SC-SDLC in a real work project. In summary (Section 4.9), I sum up the significance of the SC-SDLC from the service consumer perspective for the development and management of enterprise applications.

4.2. Cloud Service Consumer System Development Lifecycle

In this chapter, a Service Consumer SDLC named SC-SDLC that is used to develop a cloud-based application and manage cloud services is proposed. This lifecycle considers the challenges of using cloud services in an enterprise application from the service consumer perspective. In the cloud context, the methodology for the development of an enterprise application has changed. The service consumer builds the application by integrating cloud services that satisfy the application requirements. SC-SDLC (Figure

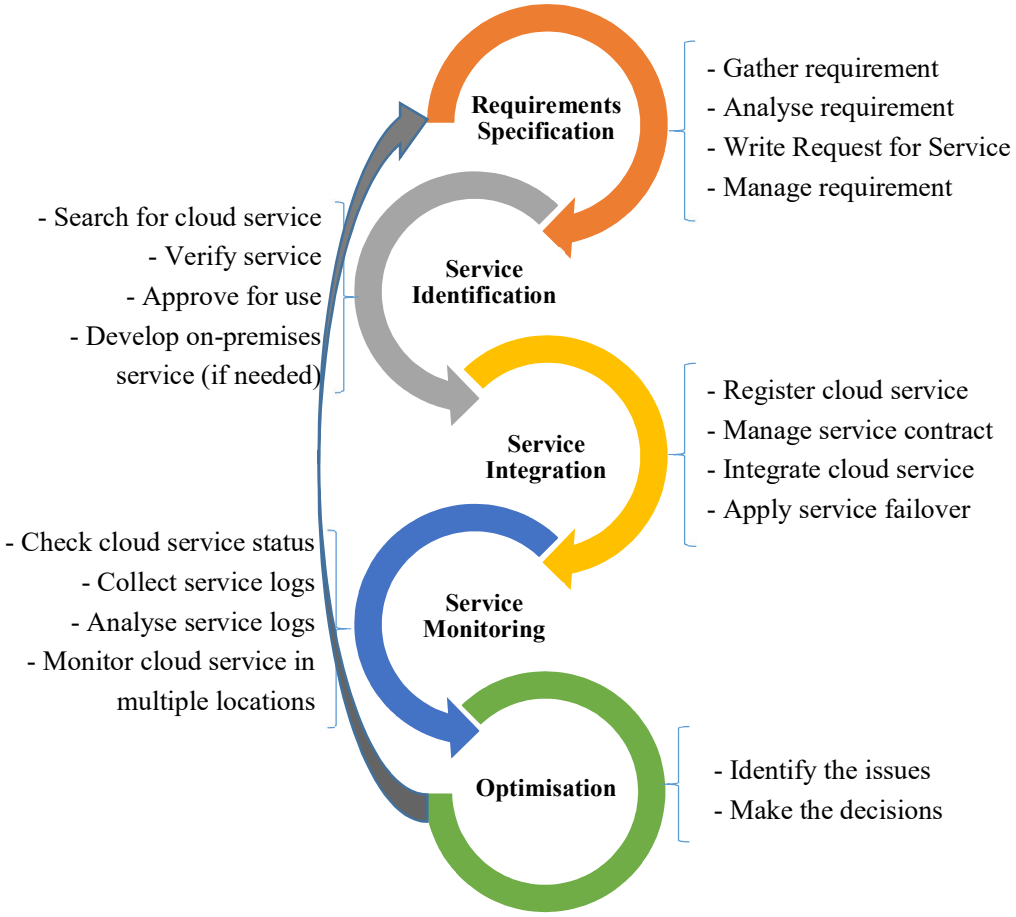


Figure 4.1 Cloud Service Consumer SDLC and lifecycle activities

4.1) contains five phases that involve both design-time activities (requirement specification, service identification, and service integration) and runtime activities (monitoring, and optimisation).

4.3. Requirements Specification

Requirements Specification is the first phase of SC-SDLC that includes analysis of system requirements. It is in the early phases of system development after kicking off the projects (Vos, 2017). Mistakes introduced during the requirements specification are difficult and expensive to correct (Ammann and Offutt, 2008, Chakraborty et al., 2012) because they are propagated into further phases of the SC-SDLC. This phase comprises activities that are very much aligned with traditional activities related to requirements engineering such as requirements gathering, requirements analysis, requirements

specification, requirement approval, and requirement management. At this stage, the decision on using cloud services or development of the feature on-premises has not been made as it is based on whether suitable cloud services are found or not. There are two principles of requirements specification:

Service neutrality. The business analyst should focus on gathering and analysing requirements without considering specific cloud service features. If the business analysts think about a specific cloud service during requirements analysis, they will try to make the requirement fit the cloud service naturally. This may result in failing to satisfy the actual needs of the users.

The non-functional requirements are specified at the application feature level. In traditional requirements engineering, the non-functional requirements are normally considered at the application level, but in the SC-SDLC, the non-functional requirements should be considered at the feature level. The non-functional requirements (i.e., security, cost, availability, and performance) should be specified in each application feature to be used as the criteria for cloud service selection in the next phase (service identification phase). For example, because of the pay-per-use model of cloud service, the accepted cost for each application feature should be estimated so that developers can use the cost as criteria to select cloud services.

4.3.1. Request for Service

When analysing a requirement, the business analyst describes features and non-functional attributes that the given application needs to fulfil. Once a requirement is fully described and classified, the business analyst creates a Request for Service (RFS) (Joshi et al., 2009). Studying the guidelines to write the specification and RFS template proposed by (Chakraborty et al., 2012, Chemuturi, 2013, Vos, 2017), the RFS template should contain general information (description, role, and preconditions), functional requirements, non-functional requirements, and constraints. The RFS specifies what functions are to be performed on what data to produce what results at what location for whom. It must contain all the details that developers need to know for the purposes of selecting the cloud service. Table 1 is a sample of RFS that is used to describe the Online Payment requirement.

Table 4.1. Request for Service for online payment.

Service Description: When users use the Create New Subscription, Create New Gift Subscription, and the Renew Subscription functions, they choose the credit card payment type to process the payment online.

Role: Application User

Precondition:

1. Users have logged in.
2. Users choose the online payment method.

Functional requirements

Minimal Guarantee: The system rolls back the payment transaction in case of failure.

Success Guarantee: The payment is processed successfully. All transactions are recorded both in the database and in the payment service gateway.

Main Scenario:

1. Users choose to process the online payment method out of payment options (Cash, Money Transfer, and Online Payment).
2. The system populates the subscriber's information (First Name, Last Name, Address 1, Address 2, City, State/Region, Zip/Postal Code, and Country) and allows users to identify the card information (Card Type, Card Number, Expiration Month and Year, Card Verification Number).
3. Users review the information and submit the payment.
4. The system validates the card information.
5. The system processes the payment
6. The system saves the payment.

If the card information is invalid, the system sends an error message to users.

Non-functional requirements:

1. Service timeout should be 10 seconds. It should be configurable based on the Payment Gateway.
2. Service can serve at least 100 concurrent payment requests from 2000 staff and 100,000

members.

QoS Attributes:

1. Availability should be higher than 99.99%
2. 90% of requests should be responded to in less than 3 seconds.
3. Security: The system does not store credit card information such as Card Number and Security Code.
4. The service fee is under 3% of the transaction amount.

Constraints:

The system must support Master Card and Visa Card.

When the users accidentally close or are forced to close the payment window for any reason, the transaction may not be recorded in the database, but the payment transaction has already been sent to the Payment Gateway. The integrity constraint is that all payment transactions sent to the Payment Gateway have to be recorded in the database.

Runtime Attributes:

1. Priority: Urgent
2. Severity: Critical

4.3.2. Functional Requirements

Functional requirements specify how service inputs should be transformed into outputs (Chakraborty et al., 2012). It describes the fundamental features that must take place in the service. Functional requirements understanding can be viewed from two perspectives: functional characteristics and technical characteristics:

- *The functional characteristics* include features which need to be fulfilled in the service (Zalazar et al., 2015). This information is related to cloud service functional selection and cloud service composition. In situations where there is not a single feature of the cloud service that can satisfy the requirement, a composite of multiple functions from multiple services is used. During analysis of the functional characteristics, non-functional properties should also be considered such as maintainability, expandability, interoperability, responsiveness, performance, security, and availability (Chemuturi, 2013).

- *The technical characteristics* of the target environment include service technology (e.g., .Net, Java, PHP, etc.), messaging technologies, communication protocols, service description languages, and service identification mechanisms. The technical characteristics that are used in the service identification phase to search for the cloud service can be technically compatible with enterprise applications.

4.3.3. Non-Functional Requirements

Non-functional requirement specifies the capacity of the service, for example, the number of transactions and the amount of data to be processed within certain time periods for both normal and peak workload conditions. The non-functional requirement is based either on minimum acceptable performance or it is derived from the need to interact with external systems (Dick et al., 2017)**.** It should be specified in measurable terms, for example, ninety-five percent (95%) of the transactions are processed in less than one-second. Among non-functional requirements, QoS[5] values are the most important information for cloud service selection in the next phase.

4.3.4. Other Requirements

(a) Requirement Type shows the category of the requirement such as *new feature, enhancement, classification or bug*. The development lifecycle can be run multiple times or cycles corresponding to the versions of the application. During the system lifetime, the requirements specification can be changed by the need for business or the innovation of technologies. Therefore, a requirements specification contains a set of development cases in different types that relate to functional changes or non-functional improvements.

(b) Constraints can be imposed by other standards or hardware limitations. For example, this could specify the requirement for the application to log user activities or service transactions. Such logs are needed in enterprise applications to monitor and optimise the utilisation of the cloud service in the enterprise application. For example, a log audit requirement states that all changes to a payroll database must be recorded in a log file, storing before and after values.

[5] The Appendix A is the list of common QoS attributes that are used during the development of applications or services.

(c) Runtime Attributes including Priority (Table 2) and Importance (Table 3) is used to manage requirements. It ensures that the requirement and related services are maintained at the same level in the Service Monitoring phase. During Service Identification, the more important services are careful in terms of testing and verification.

Table 4.2. Requirement specification priority levels

Priority Level	Description	Actions
Urgent	For a new feature or enhancement, it is urgently needed by end-users. For a bug, it is a serious security issue or functional problem that can crash the application.	Implement and release immediately.
High	This requirement is related to the feature that end-users need or the bug that leads to functionality failure.	Implement or rectify as soon as possible.
Medium	The feature of the application that is going to be implemented and released in the next version. In the case of a bug, it is a functionality bug, but it is not necessary to fix it urgently.	Implement or rectify as planned in the development cycle.
Low	The optional feature of applications, minor bugs or non-logical mistakes. This type of development case mostly relates to User Interface Optimisation or bugs.	Implement or rectify if possible.

Table 4.3. Requirement specification importance levels

Importance	Description
Critical	This requirement involves the backbone feature of the application that significantly affects the continuity of business. The functional bugs that are related to this type of requirement are normally urgent cases. However, the non-functional enhancements or UI bugs can be set at the lower level of priority.
Important	This requirement relates to the dependent function of other features. The defect is capable of halting other application functions. The performance of this feature will influence the performance of the application.

Normal	This level is set for most of the requirement in which the failures can be acceptable for a limited period.
Optional	This type of requirement is unnecessary to implement at this time. It will be developed in successive versions.

4.4. Service Identification

Service Identification (Phase 2) is driven by the requirements documented in the RFS. The main purpose of this phase is to select the suitable cloud services for RFS. The errors in this phase can significantly affect the quality of enterprise applications. In other words, the success of the development project critically depends on the correct identification of the key services. Service identification and the framework for cloud service selection are the subjects of many research studies that use various approaches, i.e., cloud service trustworthiness (Arun et al., 2017), QoS matching and ranking (Hajlaoui et al., 2017, Garg et al., 2013), using user feedback ranking (Rotem et al., 2016, Yang et al., 2006, Qu et al., 2013). However, these research efforts are only a part of the service selection problem. There is a lack of research that considers service selection as a phase of the cloud service lifecycle which, in addition to service selection, also includes requirements specification and service monitoring. In this thesis, I propose a selection methodology for a service consumer searching cloud service using historical based QoS values. The service identification process includes various lifecycle activities:

1. *Search for cloud service.* With the functional requirement in RFS, the service consumer looks for a cloud service that provides these application features. This activity involves matching application features with service functionalities. Based on QoS requirements, the developer also considers selecting the suitable service with acceptable QoS values. However, QoS of cloud services varies in different locations. For example, the availability and response time of PayPal in the US is different to that in Australia. From the service consumer perspective, network connection also impacts on the QoS of the cloud service. In some cases, local service providers have more advantages than international service providers. To search for a cloud service, developers can use a different source.

    a. *Search for verified cloud services.* Developer considers selecting the services that

are approved for use in the previous project. They compare the QoS requirements and the accurate service QoS from the previous projects.

b. *Search for alternative cloud services.* If no verified service is matched, developers search for cloud services that are recorded in the service repository.

c. *Search for cloud service over the Internet.* The developer also looks for new release cloud service over the Internet using some technical forums, and the open service directory.

2. *Verify cloud service.* Service verification includes the testing and demonstrating of service functionalities. Most of the service providers provide a pilot version of the service or a trial period for service consumers to verify their service functions. After verifying cloud service functionalities, non-functional requirements are verified. During this process, the list of suitable cloud services is produced for the purposes of getting approval. The selected services which are the output of this phase will be used in the service integration phase.

3. *Approve for use.* The selected cloud service is verified by the developer and needs to be approved by project owners, so that they are aware of the use of cloud services in the applications. Project owners can use different priorities when approving the use of cloud services. They can either choose the best performing service or the lowest price service. They can also select more than one service to design the cloud service failover.

4. *Manage cloud service portfolio.* The candidate service information will be recorded in the service repository for further use. Given a large number of available cloud services, the verification of cloud services can be time-consuming, in particular, if this task is performed multiple times in the context of different projects that require similar services (Sundareswaran et al., 2012). Using a service repository to store verification information about cloud services ensures that services are shared among different projects, and the service selection and approval process are not unnecessarily repeated.

5. *Develop on-premises service.* In the case where no cloud service that satisfies the requirements specification is found, there are two different options: contact a

preferred service provider directly to locate a suitable cloud service or develop an on-premises service. The on-premises service implementation process is very much aligned with the traditional SDLC which includes four different phases: Analysis, Design, Implementation, and Deployment.

a. *Analysis* captures all activities required for the identification and contextualisation of a service and helps to prioritise business processes that offer potential improvements in business value. In this phase, the scope of service is also defined. Services can be the aggregation of simple services or decomposition of complex services.

b. *Design* specifies the technical details of service interfaces using design principles that include minimisation of coupling and maximisation of service cohesion. Importantly, it involves decisions about service granularity and grouping of service operations ensuring that the overlap between the functionalities of different services is minimised. The design activities are dependent on the specific type of service and may vary according to a delivery strategy.

c. *Implementation* includes all activities that are related to the realisation of the services based on the detailed design. As services are reused in different applications of the enterprise system, services need to be exhaustively tested to ensure that the specified requirements have been met.

d. *Deployment* involves installation of the service instance in servers, versioning services, determination of access rights, and the specification of the details concerning security settings.

4.5. Service Integration

Following the service identification phase, cloud services need to be integrated into consumer enterprise applications. The service integration process consists of the registration of the application and/or design of failover capability using cloud services. Service integration involves four development activities:

1. *Register cloud service.* This activity is a business activity that involves the cloud

service consumption process such as service registration, account creation, pricing plan selection, and service SLAs negotiation. In the case of the SaaS service, it aims to gather the information of the cloud service (i.e., service account, service APIs, security setting) that is used for implementation of the integration code. In some instances that depend on the type and volume of the services involved; and the service consumer is able to negotiate details of the SLA with the service providers.

2. *Manage service account*, i.e., information about cloud services that includes service account information, service interface description, endpoint address, service contract, SLA. With a large number of cloud services being used in an enterprise application, a supporting tool is needed to manage the cloud service information. This activity also involves management of service logs which are the historical records of service transactions.

3. *Integrate cloud service.* This activity involves technical activities to integrate the cloud service into an enterprise system. Depending on the type of cloud services; this activity can vary from application installation, system configuration, to service adaptor development. For example, with IaaS and PaaS services, the developer needs to set up the server or configure the database server. With the SaaS services, the developer needs to implement the service adaptor or integration code to involve the cloud API. In this phase, the developer needs to implement the log system in the application, or configure notification alerts in the cloud service for later service monitoring,

4. *Implement service failover*. Cloud service is subject to changes and is out of the control of service consumers. The service disruptions and service changes can cause serious problems in terms of application reliability. During Service Integration, service failover strategies need to be applied to eliminate the impacts these cloud service incidents have in an enterprise application. The common assumption is that cloud service failover is already supported by the cloud service provider. In fact, the cloud services have experienced a number of serious outages. In Chapter 6, the failover strategies that are used to improve the continuity of operation in the face of cloud service incidents are discussed.

5. *Test and deploy*. The final activity of the cloud service integration phase comprises integration testing, provisioning, and deployment that is very much aligned with traditional activities related to application implementation.

4.6. Service Monitoring

Cloud monitoring plays a crucial role in the efficient management of cloud services as a means of gathering the required information to make decisions (Fatemaetal.,2014). Although the service consumer selects suitable cloud services ensuring that both the functional and non-functional requirements are satisfied at design time, the management of service utilisation is still required at runtime. It aims to proactively track all service

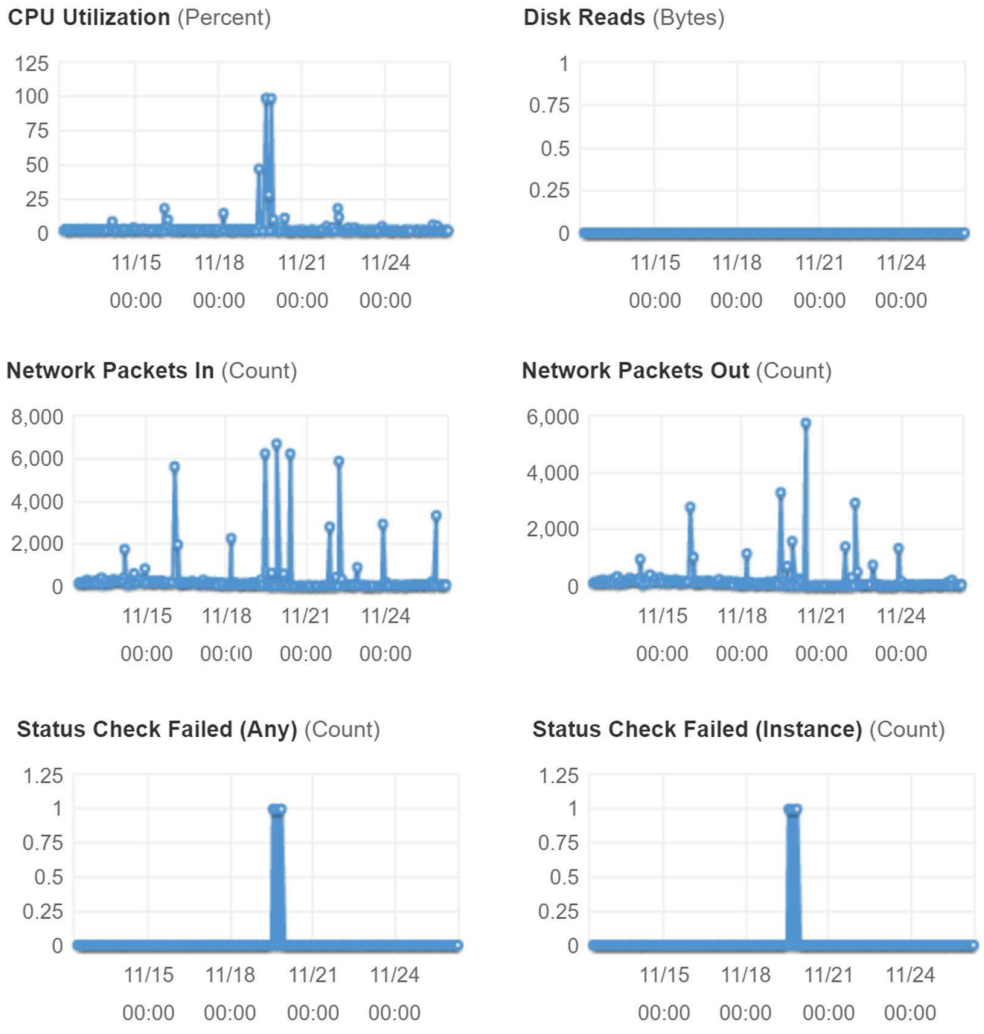


Figure 4.2. AWS EC2 server monitoring at the provider side

operations to manage service events effectively, and rectify problems before service customers become aware of them. Currently, there are a number of research studies on cloud service monitoring (Anwar et al., 2015, Lin et al., 2016, Povedano-Molina et al., 2013, Ciuffoletti, 2016, Suneja et al., 2016, Zhao et al., 2017), Alboghdady et al. (2017), (Montes et al., 2013). However, very little research has been published which monitors cloud services from the consumer side where network conditions have a significant impact. There are two monitoring sides: the service provider side and the service consumer side.

1. *Service provider side monitoring.* Service provider side monitoring does not mean cloud service monitoring from a service perspective. From the consumer perspective, provider side monitoring is used in the transaction reports or the dashboard that is provided by the service provider to monitor cloud service utilisation and status. For example, Figure 4.2 shows the monitoring metrics of an EC2 server from the service provider side. Although the provider report is not helpful for proactive management of cloud service incidents, it provides useful information about cloud service statuses (e.g., CPU overload, low memory, or overcoming cloud service requests and cost). Using monitoring data from the provider side, the service consumer has evidence to adjust and re-configure cloud services or to negotiate the service contract with the service provider.

2. *Service consumer side monitoring.* Consumer side monitoring aims to detect cloud service performance in the client environment including connection conditions. Although the QoS values measured at the consumer site are impacted by connectivity issues and do not fully reflect the quality of service, they show the actual service performance in the client location. For example, measuring from the server side, a US server may have better performance than a Sydney server. However, when measuring on the client side, the Sydney server may have lower latency than the US server. Traditionally, there are two common methods of client-side monitoring: proactively monitoring and reactively monitoring. *Proactively monitoring* is the technique that sends a testing message to the cloud service continuously to detect service status or performance issues. This method helps service consumers to be immediately notified of a service failure, but it has a negative impact on cloud service

performance. Thus, the pay-per-use characteristic of cloud service restricts the applicability of this method because service consumers must pay for the monitoring transactions which are sent to the cloud service. *Reactively monitoring* is based on the analysis of service logs during the enterprise application operation. Though this method does not create any problems for service performance and cost, it cannot help service consumers to detect the service performance issues proactively before the end-users are impacted.

3. *Multi-site monitoring.* During the service identification phase, service consumers want to select a cloud service for the enterprise application. Then, they monitor the selected services to judge whether or not they are maintaining the same level of QoS as when they were selected. In addition to monitoring selected services, service consumers need to monitor the other services that are available in the cloud to consider service change if the other services are available with better QoS at a lower price. Service monitoring data is not only used for (1) service selection but also used for (2) management of cloud service incidents, and for (3) optimisation. For these monitoring purposes, service consumers need a strategy of cooperative multi-client monitoring. A full set of QoS of cloud services that are measured across different geographical regions will help to select the best service for an application. Service logs from applications of different consumers of a cloud service can help each other in early detection of potential incidents. Organisations may have a different choice of cloud services based on their expectations, and the shared QoS of cloud services will help to compare their selection with others in order to make decisions in terms of changes. In Chapter 7, I describe the multi-site monitoring model in detail and discuss how it supports the phases of SC-SDLC such as service selection and optimisation.

4. *Context monitoring*: The cloud service performance is also impacted by the context such as location and network connection. For example, the breakage of the Asia-America Gateway submarine cable affected the performance of many cloud services in Southeast Asia. In addition to cloud service monitoring, service consumers should monitor the context as it impacts on cloud service performance and availability.

Although cloud service monitoring works to maintain application continuity, it is not the technology for "zero downtime." Monitoring the performance and uptime of the cloud service helps the service consumer to make decisions and to take actions. For example, when the AWS service fails, the service consumer can either execute a disaster recovery process or just keep tracking notifications from AWS until the problem is resolved. The earlier problem detection capability and the quicker reaction time help service consumers to manage the issues. During the monitoring phase, the administrator should define an "action plan" for the cloud service performance problems. The action plan is initially created with the most concerning issues in mind and it is continuously updated during the service monitoring phase.

4.7. Optimisation

In SC-SDLC, there are two phases that do not have "service" in the name: the requirement specification phase and the optimisation phase. Cloud service optimisation is not considered in this phase as it is the responsibility of service providers. In an enterprise application, organisations in the role of service consumers need to optimise applications or *the use of the cloud service*. Therefore, the optimisation phase (Phase 5) is concerned with continuous application improvement and aims to optimise application performance and to reduce operation failure and cost. During this phase, runtime analysis data, which is collected during the service monitoring phase, is reviewed and compared to non-

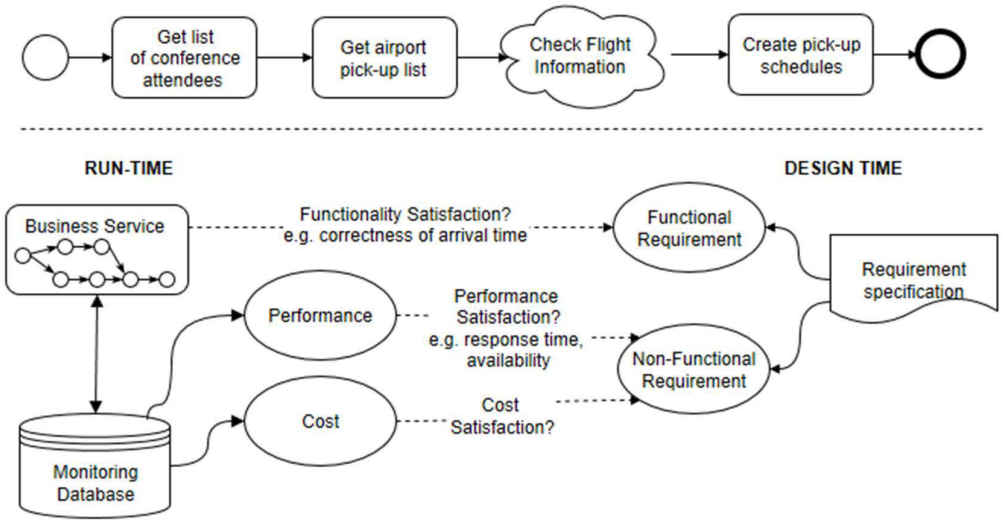


Figure 4.3. Airport pick-up service optimisation scenario

functional attributes and constraints defined in the requirement specification phase. The service consumer identifies the cloud service issues to enhance the use of the cloud service. Consider another example of the Airport Pickup service (Figure 4.3) that uses an external (cloud) service to check the up-to-date flight schedule. Three principal factors need to be considered to identify cloud service problems:

1. *Functionality requirements*: for evaluating the functional correctness of a cloud service. Organisations evaluate the features of the cloud service by user feedback and execution reports of service. They review if the cloud service performs to the level that the service consumer expected or to the level of what was advertised by the cloud service provider.

2. *Performance requirements*: the parameter for measuring the runtime service presentation. Performance can be evaluated using response time, availability, and throughput.

3. *Service Cost*: the parameter for measuring the cost of a service.

Service optimisation can be done using different methods, such as changing the business process or replacing cloud services with other services from different service providers. There are a number of optimisation approaches to enhance and improve the use of cloud services:

1. *Changing business process* involves the actions that change the workflow in the requirement specification to improve the performance or to optimise the service cost. Consider the example of the Airport Pick-up service illustrated in Figure 4.3, this service invokes a cloud service to get up-to-date flight information (for example, delay, cancellation, change estimated arrival time). When users view the pickup schedule, the application will send a number of requests for checking flight information for all attendees. Whenever a user views the Pick-up Schedule, the application sends several requests to Check Flight Information to update flight information. Users may accidentally open this function or misuse this function to view conference attendees. The monitoring data of the flight information service shows a large number of requests sent to the cloud service. In response to this issue, the

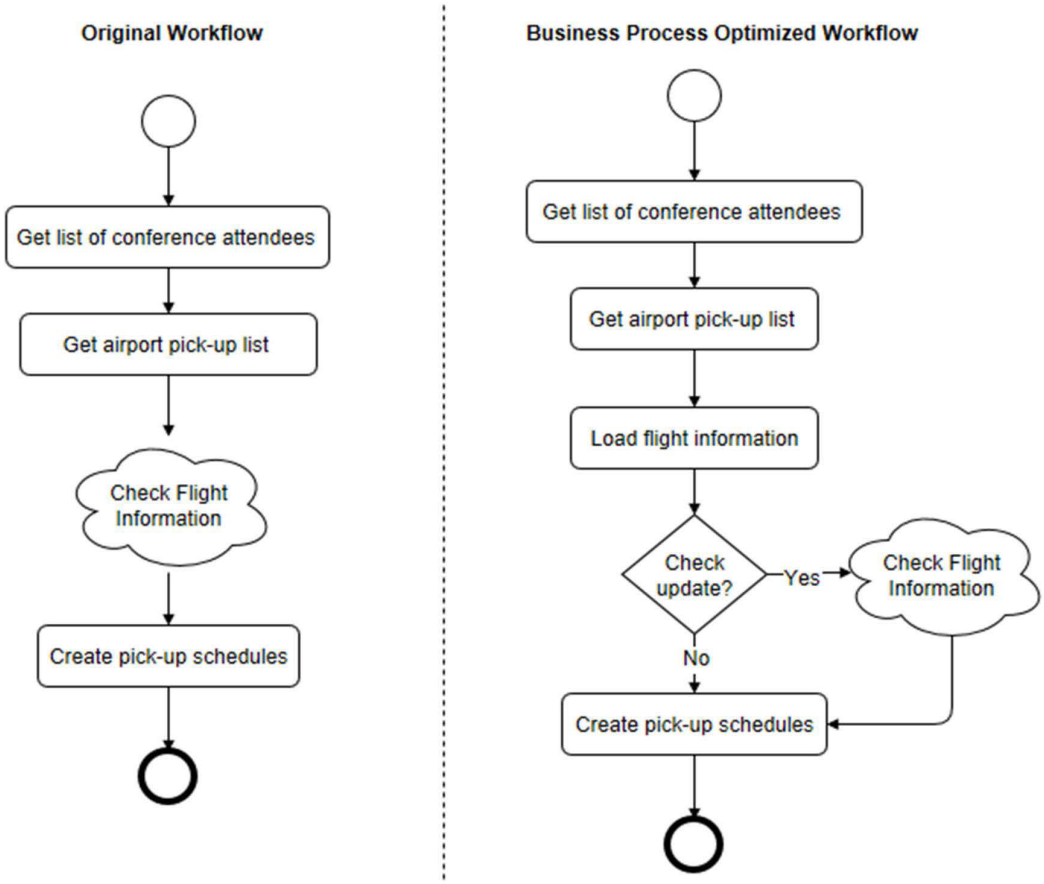


Figure 4.4. Business process optimisation example

workflow is changed to allow the optimised workflow in Figure 4.4. The flight information service is only invoked when the user wants to update it. This change helps to reduce the service by avoiding unnecessary cloud service invocations. By limiting redundant invocations of the cloud service, the optimised workflow also improves the overall performance of the service.

2. *Improving cloud service integration* involves technical activity during service implementation and integration. Considering an example illustrated in Figure 4.5, the monitoring data shows that there are different users using the airport pickup schedule function at the same time so that a number of identical requests are sent to Check Flight Information and receive the same results. Until a day before travelling, the flight information usually does not change. So, checking flight information earlier will always return the same data. Implementing the service request cache when

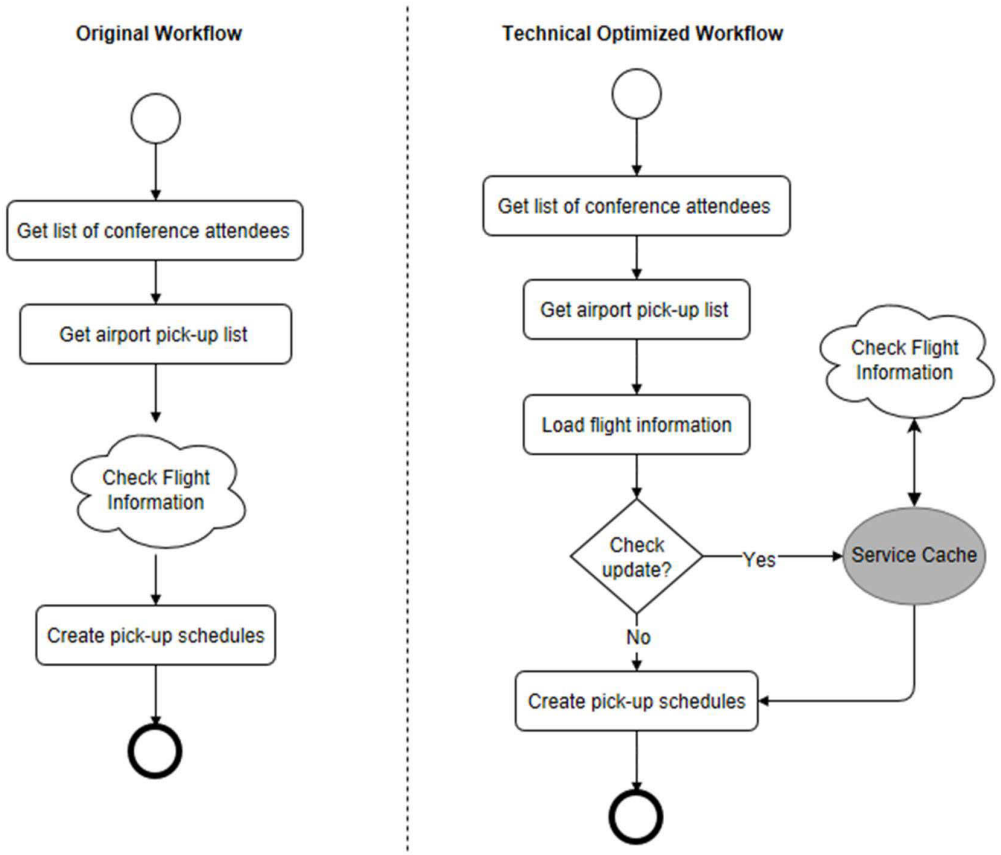


Figure 4.5. Technical optimisation example

integrating the Flight Information service will limit the number of the same requests which have been sent to cloud services at the same time. It also helps to improve application performance as well as reduce cost.

3. *Adjust cloud service configuration*. This activity is related to change the cloud service contract or SLA to improve cloud service performance or optimise the service cost. For example, the monitoring data shows the low memory of the cloud server that causes the low performance of the application. The service consumer needs to configure the cloud server by adding additional memory to the server. Another example, service consumers can adjust the Google Drive price plan based on the monitoring of data increments to optimise service cost.
4. *Replace cloud service*. In practice, optimisation of cloud enterprise applications may involve using alternative cloud services, e.g., migrating the servers that run the application to a different cloud infrastructure. With an increasing number of

alternative cloud services with equivalent functionality, service consumers can choose services to use in their enterprise applications based on the cost and QoS characteristics. Service consumers can also optimise application performance by relocating the application to a different cloud infrastructure selecting a more suitable geographic location and taking into account both end-user connectivity and connectivity to third-party cloud services.

With the increasing use of cloud computing, a large number of services are continuously introduced in the cloud service market. Moreover, the performance of a cloud service may evolve relatively frequently with its internal changes or the changes in the environment and service providers do not always ensure their anticipated service quality. Although services are carefully evaluated during the Service Identification phase, selected services can become obsolete with the rapid improvement of cloud computing. In such cases, service consumers need to change to another service provider to obtain a new cloud service to improve application functionality and performance or to reduce cost. The optimisation activities will trigger a new SC-SDLC by reviewing the requirement specification and then identifying new cloud services to integrate into the enterprise application.

4.8. Evaluation using The Case Study at Family Medical Practice

This section describes the case study at Family Medical Practice (FMP). SC-SDLC is applied to implement a healthcare application for FMP. Using a research approach based on design science and action research, the SC-SDLC is the result of a rigorous process between the core activities of building and evaluating using the case study. During the stages of the project, the SC-SDLC is evaluated, and then, it is improved to address the comments and feedback of team members. The situation at FMP will be described then the evaluation of the proposed lifecycle will be discussed.

4.8.1. Situation at FMP

Family Medical Practice (FMP) is Vietnam's leading international primary health care provider with a high standard of client care and a friendly atmosphere. FMP's clinics and hospitals are strategically located across Vietnam: Hanoi in the north, Da Nang in the

centre, and Ho Chi Minh City (Saigon) in the south. With nationwide operational advantage, FMP provides international standard medical services in a safe, professional, and welcoming environment for all customers. They also offer a direct billing service with over 50 global insurance companies, on-site medical support, and a full-service Executive Health Care Centre for health checks and visa requirements.

The FMP uses the 'Medics' application written and maintained by the UBQ company based in India. The UBQ application is implemented using Java and the MySQL Community Edition. It also requires the installation of various software to make the UBQ application function properly. The existing UBQ database contains around 350 tables with more than 40 million rows, 15 GB data and 2 GB index (the system has used up all index space allowed in the MySQL Community Edition). The size of current data is 2 TB of files and images (X-ray, Scanning) and 500GB of other documents. The system starts slowing down, impacting the users while serving the FMP clients. Technology dependency also causes difficulty in terms of adjusting the system to implement new business requirements and extending it to the next level. With the business continuing to grow, there is an urgent need to have a solid system to prepare for future growth.

4.8.2. Requirements of Enterprise Application at FMP

This project relates to many sub-systems or applications that are divided into two phases:

Phase 1: Implementation of an application for FMP clinics.

The FMP application is used by the FMP medical staff in FMP clinics during daily operations. It includes a number of features such as check-up management, company and package management, doctor's examination (i.e., vital examination, bone examination), Lab tests (i.e., X-ray and Scanning, Urine test, Blood test), Electronic Health Record, Billing, and Invoices. The timeline for this phase is six months. The QA test version of the project was deployed at http://care1.vastbit.com/[6]

Phase 2: UBQ replacement

[6] Username: dat+123@vastbit.com – Password: 1234567

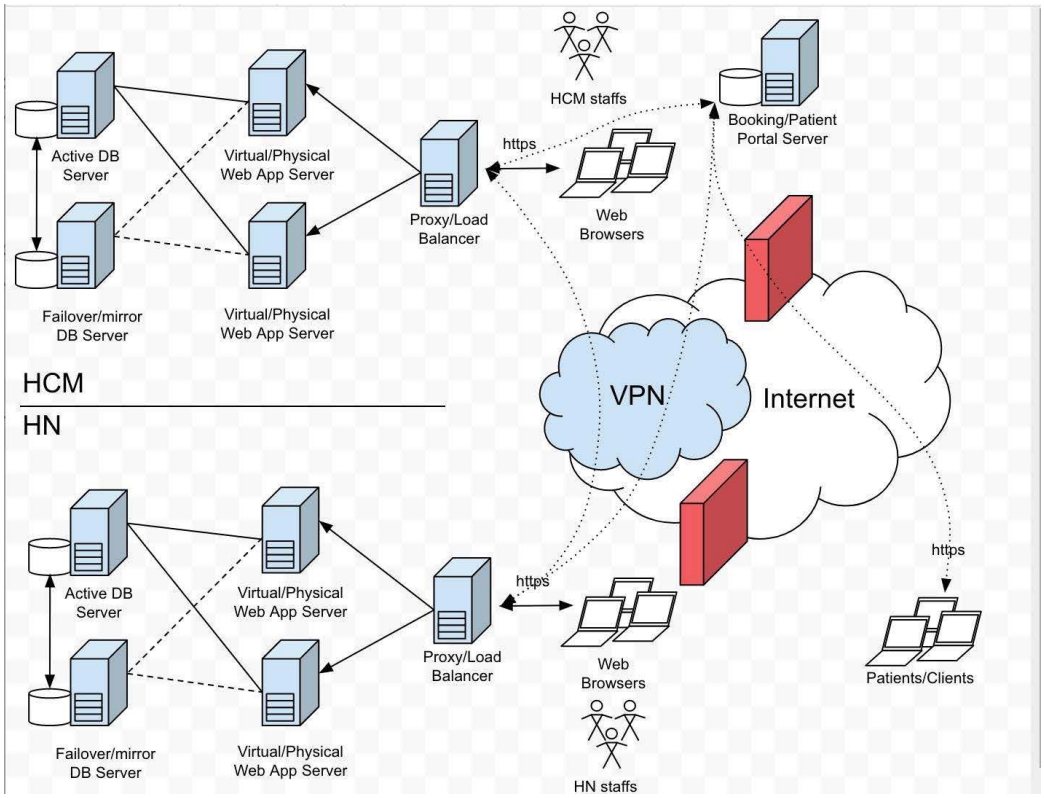


Figure 4.6. The deployment diagram of FMP application (Nguyen, T., 2017)

FMP uses the UBQ system – 'Medics' for managing the operation of the clinics and hospitals. This phase of the project works to build up a new application to replace the UBQ application. The significant non-functional requirement of the new information system is to improve performance as the major concerns with the UBQ system is its slow performance. System reliability is also a top priority of healthcare applications as this is a 24x7 healthcare service. FMP-IS is developed to fully replace the current system provided by the UBQ and fully migrate all Databases. The FMP information system has a number of modules such as: Financial and Accounting, Pharmacy, Doctor, Vaccination, Patient Electronic Medical Report, Self-Treatment for Patient, Nurse, and Schedule, Front Desk, Roster, Issuance. Figure 4.6 shows the deployment architecture of the FMP system in which application servers and database servers for the Hanoi site and the Ho Chi Minh site are separated. The timeline of Phase 2 is eighteen months.

4.8.3. Evaluation of SC-SDLC

The project was run by Vastbit which is an IT company that has a long-term contract with FMP to implement, maintain and manage the FMP information system. To evaluate the lifecycle phase and activities, the SC-SDLC was applied to the project and used by the Vastbit team.

4.8.3.1. Evaluation methods

The SC-SDLC was evaluated during the development process of the project. At the kick-off meeting, the SC-SDLC activities and SCF was presented to the Vastbit team. On the weekly meetings, we have a session to get the feedback and discuss about the SC-SDLC. During the meeting, each team member needed to answer three questions.

a. What did you do last week? (e.g. Requirements specification, Service Identification…)

b. How did you do these tasks? (e.g. How did you write the specifications)

c. Do you have any feedback about the lifecycle activities?

During the development, the team members can discuss about the SC-SDLC activities that related to their tasks. For example, during the Requirement Specification phase, we discussed about the RFS template and how to determine the non-functional attributes for the requirement. After the application was deployed, a questionnaire (Appendix D) was sent to Mr Thanh Nguyen, CTO of Vastbit, to gain the overall feedback about the SC-SDLC. Below is the feedback of the Vastbit team on each phase of SC-SDLC and their suggestion to improve the lifecycle.

4.8.3.2. Summary of the main feedback

In general, the SC-SDLC provides a good methodology to develop the FMP application. The development and management activities are clearly described to assist team members in performing their tasks during the project execution.

Feedback[7]: Before using SC-SDLC, Vastbit used Prototype and Scrum as the

[7] This is the responses of Question 2 and 3 in Appendix D: Questionnaire

development methods[8]. When the Vastbit team adopts the cloud services to develop the cloud-based enterprise application, they realize that the current development methodology is not adequate to deal with the cloud service challenges. With the SC-SDLC, the activities, strategies, methods that are described in each phase of SC-SDLC guide the developers to know how to complete the tasks. Developers know what they need to do to select, integrate and manage cloud services. The management of cloud services using Service Consumer Framework improves the collaboration among team members. As a result, the development process is shorter than before. This can be caused by both the use of SC-SDLC in application development and the natural benefits of using cloud services. The other positive feedback is for the cross-provider failover, application managers believe they have a methodology to handle the cloud service disruptions and changes. They also think that the monitoring system can help them to handle the application issues during runtime.

Suggestions[9]: The concern is that the specification in the form of RFS is relatively hard for the FMP stakeholder to understand and this makes it hard to gain approval. In the Prototype method, stakeholders get proper clarity about the requirements and feel the functionality of the application, so they can suggest changes and modifications. The requirement specification in form of the RFS is understandable for developers but it is hard for the customer to understand and approve the requirements specification. Another suggestion is the lifecycle which is the methodology for the development of cloud application needs to have a set of best practices and principles. For the SCF, they suggest the workflow design feature so that the can visualize the workflow in a graphical interface.

4.8.3.3. Response to Feedback

During the development process, after receiving the feedback from the team members, we discuss and then adjust or improve the lifecycle to address the feedback if it is necessary. For example, we discussed about the service portfolio and then add more

[8] This is the responses of Question 1 in Appendix D: Questionnaire
[9] This is the responses of Question 4 in Appendix D: Questionnaire

information for service ranking and service provider reputation. This information is used for selecting services when we do not have the information about performance of cloud services. We also discussed about Context monitoring (Section 4.6) after receiving the feedback of team members about the management of network connection and environment. To response to these above remained suggestion, we added a set of principles i.e. *Service neutrality* and *The non-functional requirements are specified at the application feature level* (Section 4.3). The remained suggestion will be addressed in the future works of the thesis (Section 8.2).

4.8.4. Feedback of the SC-SDLC Phases[10]

4.8.4.1. Requirements Specification

During this phase, there were a number of meetings between the Vastbit team and FMP, user interviews, stakeholder interviews, UBQ application investigation, and document analysis (sample EMR record, sample patient profile, check-up, medical test list). The activities aimed to gather the application requirements. The Vastbit team analysed the requirements and documents to write the specification using the RFS template. During the stakeholder meeting, the RFS was presented and explained, so as to get approval.

Feedback: The activities in this phase are what Vastbit are doing in many projects. The advantage is the template of the RFS that is used to record the specification. In our previous projects, we did not clearly specify the non-functional requirements and constraints. Now, non-functional requirements that are specified in RFS provide us with the KPIs for application monitoring and management.

However, the stakeholders found it hard to review the requirements in the form of RFS. During the meeting, the business analyst took a long time and went to a lot of effort to explain the RFS. In the guidelines of the requirements specification phase, the estimated cost should be defined for each feature; it is very difficult for the business analyst to do this because they need to review the service market in order to check the price. For this reason, we leave this until the service identification phase.

[10] This section is the responses of Question 5 – 9 in Appendix D: Questionnaire

The Vastbit team also suggested that the RFS template should include the mockups. More templates are needed to write different types of requirements. The RFS should somehow combine the mock-ups and the RFS so that the stakeholder can gain a feeling for the application feature before the application is deployed.

4.8.4.2. Service Identification

The Vastbit team verified cloud services that can be used to deliver the application infrastructures and features.

- Cloud Server: AWS EC2, Azure, FPT HI GIO Cloud.
- Payment service: PayPal, Stripe, NganLuong.
- Cloud Database: AWS RDS, Azure MS SQL.
- Storage service: Google Drive, Dropbox, OneDrive, AWS S3.
- Other services: eInvoice, CloudWare.vn, Salesforce Platform.

The Vastbit team searched for the QoS value of the cloud benchmark websites such as CloudHarmony[11], AzureSpeedTest[12], G Suite Status Dashboard[13] and so on. They also created a number of free trial accounts to test and investigate these cloud services. For example, EMR of patients (i.e., Scanning, X-Ray image, Ultrasound…) must be synced or 'replicated' among the application server. These files/documents must be versioned and tracked changes. The Vastbit team decided to use the Google Drive service for cloud storage and to replicate the files across the servers. In the AWS server, the files are stored using AWS S3.

Feedback: The advantage is that there are clear guidelines of what activities should be done in this phase. The drawback is still not having enough evidence for service selection. The developers are quite subjective when selecting the cloud service and there was a

[11] https://cloudharmony.com/
[12] http://www.azurespeed.com/
[13] https://www.google.com.au/appsstatus#hl=en-GB&v=status

discussion on the selection between AWS or Azure.

The Vastbit team suggested that some attributes should be added to the cloud service information such as the type of service (e.g., regional or worldwide), service provider reputation and user (developer) feedback. These attributes are also used for service selection when the QoS information of the cloud service is missing.

4.8.4.3. Service Integration

The Vastbit team implemented the service adaptors and configuration cross provider failover for cloud services. For example, they set up a server farm between the Azure Southeast server and the AWS EC2 Singapore server. They set up database replication between AWS RDS and Azure MS SQL. They implemented the service adaptor for PayPal and the NganLuong service and configured the failover workflow between these services. Then, they set up the synchronise folder for Google Drive and implemented the service adaptor for the Google Drive API.

The team also implemented log systems and notification rules to monitor cloud service availability. For example: if the payment fails, an email will be sent to the system administrator for inspection. When the response time of an application function is more than 20 seconds, an email with the logs will be sent to system administrators.

Feedback: The unified interface of cloud APIs is an advantage. Vastbit team implemented a set of Payment and Storage services inherited from the generic interfaces. Then, they use the Workflow Engine for configuring the failover strategies. In fact, they are always concerned about the capabilities of failover in the enterprise system, but they only focus on the infrastructure failover. Further, we did not realise that we also needed to consider the failover techniques for the software service.

The log system to collect the QoS is also another advantage. In the previous projects, Vastbit team used to write the logs only to debug and trace the failure; and they did not measure the QoS of the cloud service.

However, the Vastbit team raises a concern about the provision activities when using cloud services. The on-premises applications are often deployed in environment which has a high-speed connection (LAN or WAN). For using cloud services, the Internet

connection is a significant factor that affects the application performance. There is a need of provision activities in relation to what cloud service consumers need to use the cloud services. For example, when service consumer uses a local storage, the file can upload or retrieve the file in LAN is high performance. If service consumer uses a cloud storage, they need to have a high-speed internet connection to assure the performance of file upload and download. There are some questions: How fast can we upload and download files using cloud storage? How much does the Internet connection cost?

4.8.4.4. Service Monitoring

Using the Notification Centre, Vastbit team configured and implemented the log system to monitor the application and cloud services. The guide for an action plan is the part that Vastbit team can apply immediately in this project. They defined a list of potential risks of the application related to the use of cloud services including the context risk (electric outages, Internet disconnections) and they defined an action plan (or the backup plan) for these risks.

Feedback:

The monitoring models are an advantage of this phase. They have the tool that will be able to manage the application better than previously. However, a multi-site monitoring model needs to be applied widely with the involvement of many service consumers in order to show its efficiency.

Because the lifecycle may only focus on the cloud services; the application maintenance has not been properly discussed. As Vastbit has a long-term maintenance and support contract with FMP, maintenance, bug fixing, and end-user support are important activities of the monitoring phase. However, this phase has not paid enough attention to discuss these maintenance activities. The monitoring centre of SCF should also be improved to provide the dashboard or Business Intelligent report.

4.8.5. Design Cycle of SC-SDLC

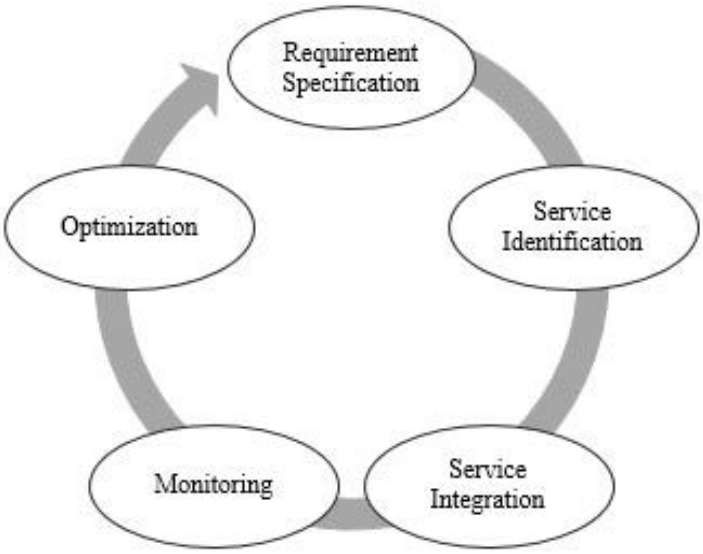


Figure 4.7. The first version of SC-SDLC

By working closely with the team members of this real-world project for developing a hospital management system, the lifecycle activities and the framework features are being continuously improved during the project execution phase. At the first stage of the project, we apply the first version of the SC-SDLC in Figure 4.7. In this version of SC-

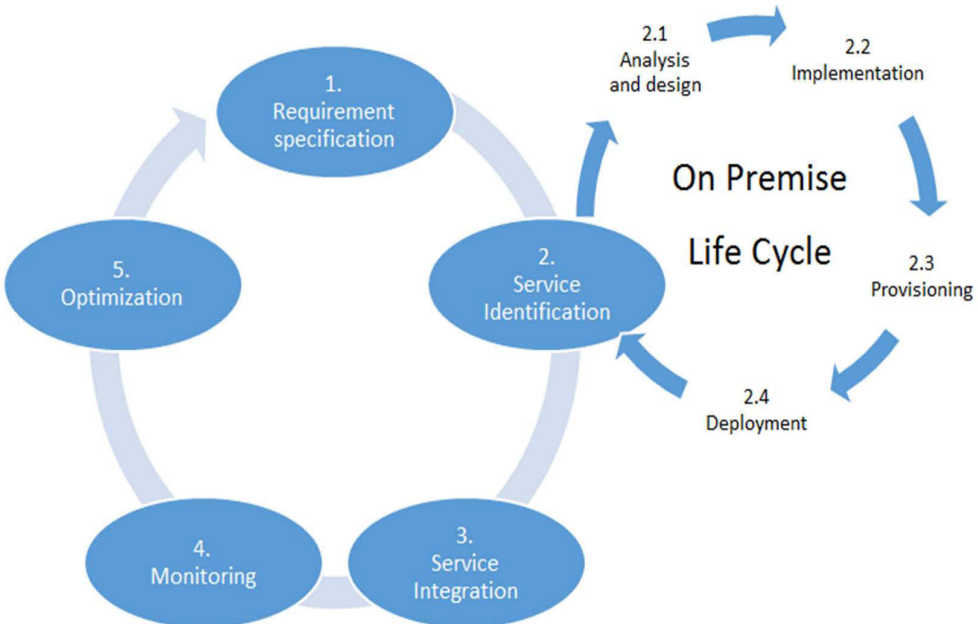


Figure 4.8. The second version of SC-SDLC

SDLC, we specified the SC-SDLC phases and describe architectural components required to support the lifecycle activities. But, we only discussed on identification and management of cloud services without concerning about the on-premise service development. After that, the SC-SDLC was improved to second version (Figure 4.8) in which the On-Premise cycle was added to Service Identification phase. The On-Premise cycle is very much aligned with the traditional SDLC which includes four different phases: Analysis, Design, Implementation, and Deployment. We also discussed the Requirement Specification and Service Identification phase by defining the structure and properties of the RFS. In this thesis, we present the third version of SC-SDLC (Figure 4.1) that includes in detail the activities of each phase. We also discussed a set of design principles that were concerned in the feedback of the Vastbit team (Section 4.3).

4.9. Conclusion

With the rapid development of cloud computing, more and more enterprises adopt third-party cloud services to implement their critical business functions. The traditional SOA model that focuses on on-premises application services is no longer applicable in situations where a significant part of the enterprise infrastructure and applications is delivered in the form of cloud services with a large number of cloud service providers involved. In this chapter, I propose a cloud Service Consumer System Development Lifecycle (SC-SDLC) for implementing cloud-based enterprise applications. The lifecycle is designed not only to support enterprise application development but also to manage services from the service consumer perspective. The SC-SDLC has five phases: Requirements Specifications, Service Identification, Service Integration, Service Monitoring, and Optimisation that contains a number of management activities. Evaluating the SC-SDLC, a case study of a hospital system development project is used in practice that helps to adjust and optimise the SC-SDLC. For future works, the concerns in the above case study will be addressed to improve the lifecycle. The future research direction is also to introduce a set of best practices for different types of projects. In the next chapter, the Service Consumer Framework (SCF) which is a supporting tool for SC-SDLC activities is described. The details of the SCF module and how the SCF is

implemented are discussed.

CHAPTER 5.

SERVICE CONSUMER FRAMEWORK IMPLEMENTATION

5.1. Introduction

Consistent with the growth of using cloud services in enterprise applications, service consumers need a comprehensive framework for management of cloud services. Because of the dynamic nature of the cloud services, in each phase of the system lifecycle, the management of cloud services from the cloud consumer perspective remains a significant challenge. For example, cloud service consumers need to deal with the challenges of service selection and service integration during design time; and the management of service performance issues or service incidents at runtime. Currently, many commercial and academic types of research have been conducted on different phases of cloud service management such as cloud service selection (Arun et al., 2017, Ghamry et al., 2017, Hajlaoui et al., 2017, Rotem et al., 2016, Yang et al., 2006, Zisman et al., 2013), cloud service integration (e.g., Informatica, Mulesoft, IBM Application Integration Suite, or Fujitsu RunMyProcess), and cloud service monitoring (Ciuffoletti, 2016, Qu et al., 2014, Qu et al., 2013, Montes et al., 2013). However, these efforts have not fully addressed the challenges of the management of cloud services and they mainly deal with the problems of cloud infrastructure. The comprehensive framework that performs the activities required for the management of cloud services from the consumer perspective throughout the entire lifecycle has rarely been discussed in the literature (Rehman et al., 2015).

In the previous chapter (Chapter 4), I developed a five-phase cloud Service Consumer System Development Lifecycle (SC-SDLC) for managing cloud services from the service consumer perspective. In this chapter, I describe the Service Consumer Framework (SCF) that was discussed in our earlier publications (Feuerlicht and Tran, 2014a, Tran and Feuerlicht, 2015). The SCF is a framework which is a supporting framework for management of cloud services throughout the SC-SDLC. The SCF supports the implementation of the cloud service-oriented enterprise application by providing a number of features: requirements management, service portfolio management, and service integration. The SCF provides the capability of failover and cloud service runtime monitoring to address the problem of cloud service incidents and to optimise the use of cloud services in the enterprise application. In the next section (Section 5.2), the overall architecture of the SCF and its high-level functionalities and characteristics are described. The main modules of the SCF are discussed in Section 5.3 to explain how the SCF

supports the development and management activities of the SC-SDLC phases. Next, database design and implementation of the SCF modules are presented in Section 5.4. Section 5.5 is the conclusion of the work that is discussed in this chapter.

5.2. SCF Architecture

The SCF is designed to manage both the design-time activities and runtime management activities of the cloud enterprise application lifecycle. The SCF has four main

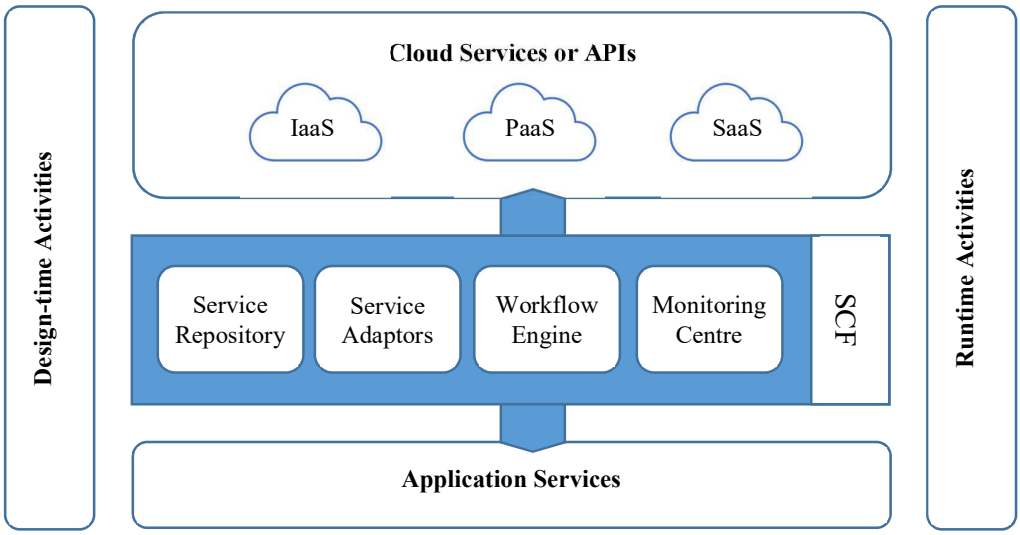


Figure 5.1. SCF architecture

components: service repository, service adaptor, workflow engine, and monitoring centre (Figure 5.1). The service repository is to maintain information about application and cloud services that are used for cloud service selection. Service adaptors are designed to unify cloud APIs and work as the cloud service connectors. The workflow engine is the module that provides failover capabilities to handle cloud service disruptions and service changes. Using the log data collected from the service adaptor and the workflow engine, the monitoring centre monitors cloud services and to analyses their runtime QoS attributes. The monitoring centre gives service consumers the ability to manage cloud service performance and the information for making the optimisation decision. Aligning with the SC-SDLC phases, the SCF provides a set of functionalities and capabilities for management of cloud services:

1. **Requirements management.** For management of the cloud service, the SCF is used to capture all the application requirements which are the user's needs and expectations. Besides managing the functional requirements, the SCF records the expected QoS values of the application features that are needed for cloud service selection.
2. **Cloud service management.** The SCF is used to manage the information about cloud services and documents their reliability and performance metrics. The SCF is used to store the information that is needed for cloud service selection such as cloud services features, QoS values (availability and response time), user's ratings as well as provider reputation ranking. After cloud services are selected for use, the service information such as service accounts, service location, SLA and service cost is also managed using the SCF.
3. **Cloud API unification.** Cloud service providers do not have any agreement about how to implement the interfaces of cloud services even though they provide the same type of services. Most cloud service providers have their own specifications for their cloud APIs. For example, PayPal uses a message format for their online payment service while SecurePay uses another format for the same functionality. Similarly, Google Drive defined a method for file downloading that is different from the method defined by Dropbox. For some types of cloud services, the SCF provides the capability of reconciling the differences of cloud APIs through the implementation of unified interfaces.
4. **Cloud service incident management.** SCF provides functionality to manage service incidents and to eliminate the impact of service changes. It can be done by managing the notification of changes from service providers or by using fault tolerance configurations for cloud services. Moreover, from the service consumer perspective, because context information such as the network condition is also a factor that impacts the cloud service performance, the SCF provides the managing capability for the context information that is related to cloud services.
5. **Cloud service monitoring.** For runtime management of cloud services, SCF offers the capability of provider side monitoring, consumer side monitoring, multi-site monitoring and context monitoring that are discussed in Section 4.6 of Chapter 4. With the multi-site monitoring model, the SCF provides the capability

of service failure detection by cooperative models of cloud service consumers. In Chapter 7, I will discuss more details about this multi-site monitoring model and its applicability.

5.3. SCF Modules

The main purpose of the SCF is to support the SC-SDLC activities. There are four

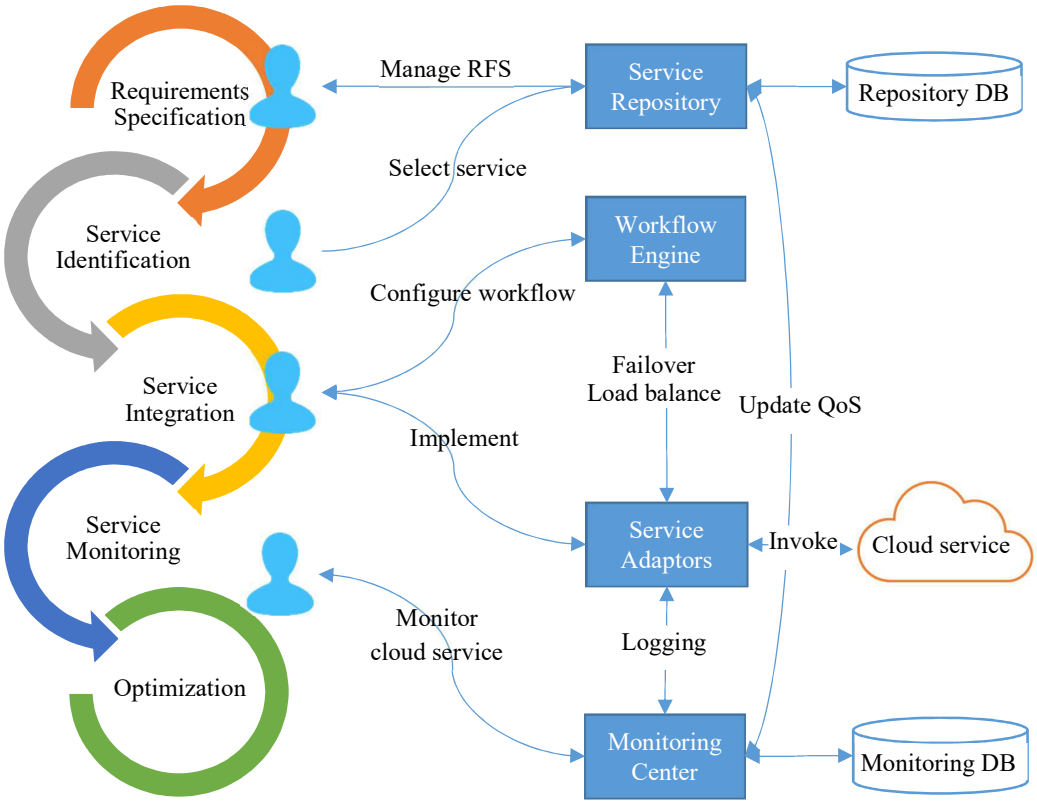


Figure 5.2. SCF modules

modules of the SCF: (1) service repository, (2) service adaptors, (3) workflow engine, and (4) notification centre. Figure 5.2 illustrates how the SCF modules involve the lifecycle tasks at both design time and runtime. During the requirements specification phase, the business analysists use the SCF to manage the requirements of the application. The functional and non-functional requirements are stored in the service repository in the form of RFS. During the service identification phase, developers use the RFS as criteria to select suitable cloud services for the application. During the service integration phase, developers use the service adaptors of the selected services to configure a workflow in

the workflow engine. During runtime, the application executes the workflow and sends runtime logs to the monitoring centre. Then, the monitoring centre analyses the runtime logs and calculates the QoS of cloud services. During service monitoring, administrators use the monitoring centre to detect the performance issues of cloud services. In case of service failure, the monitoring centre alerts the administrators so that they can execute the action plan to recover. Using the runtime logs and the performance indicators of the selected cloud service, administrators make a decision for optimisation such as replacing the cloud service or re-configuring the cloud service (e.g., upgrade the servers, change the pricing plan). The next sections describe the features of each module of the SCF.

5.3.1. Service Repository

Service Repository is a service directory implemented to maintain information about cloud services. Developers use the service repository to manage the cloud service information, to identify cloud services during design time, and to monitor cloud services at runtime. The service repository provides four main features (Tran and Feuerlicht, 2015):

1. **Manage cloud service information.** The service repository provides a feature that allows developers to populate cloud service information into the service repository. Cloud service information includes the basic information (i.e., service name, service description, service provider, version), features and nonfunctional attributes of the cloud service. The runtime QoS of cloud services on different applications and locations is also stored in the service repository.
2. **Manage application requirements.** This feature allows developers to record the Request for Service (RFS) of the enterprise application. The RFS is used to verify the suitable cloud services. During runtime, service consumers review cloud service performance using non-functional attributes in the RFS.
3. **Verify cloud service.** This feature allows developers to select the cloud service by comparing cloud service functionalities and QoS values to the requirements specification. Service repository provides the detailed information about cloud services including service features in different versions, technology features, service incidents, and QoS values in different locations, user feedback and ratings.

The developers can query cloud services using service category, keyword, availability or response time to make a list of candidate services for verifying. During the service verification process, developers can input their verification notes about the services so that the knowledge can be shared inside the organisation across different projects.

4. **Manage selected cloud service.** After services are selected and integrated, the service repository maintains the information of cloud services such as service account, SLA, pricing plan, locations, runtime QoS, notifications from providers, and service context.

5.3.2. Service Adaptors

Similar to a hardware or software driver, the service adaptor is a module to interact with the cloud API and enterprise application. Service adaptor provides two features: service integration and runtime logging.

1. **Cloud API integration.** For each type of cloud service (for example, payment service, hotel booking service, air flight service, or storage service), the generic messages and methods which are the common functionalities of that type of cloud service are defined. A service adaptor transforms the request in a generic interface

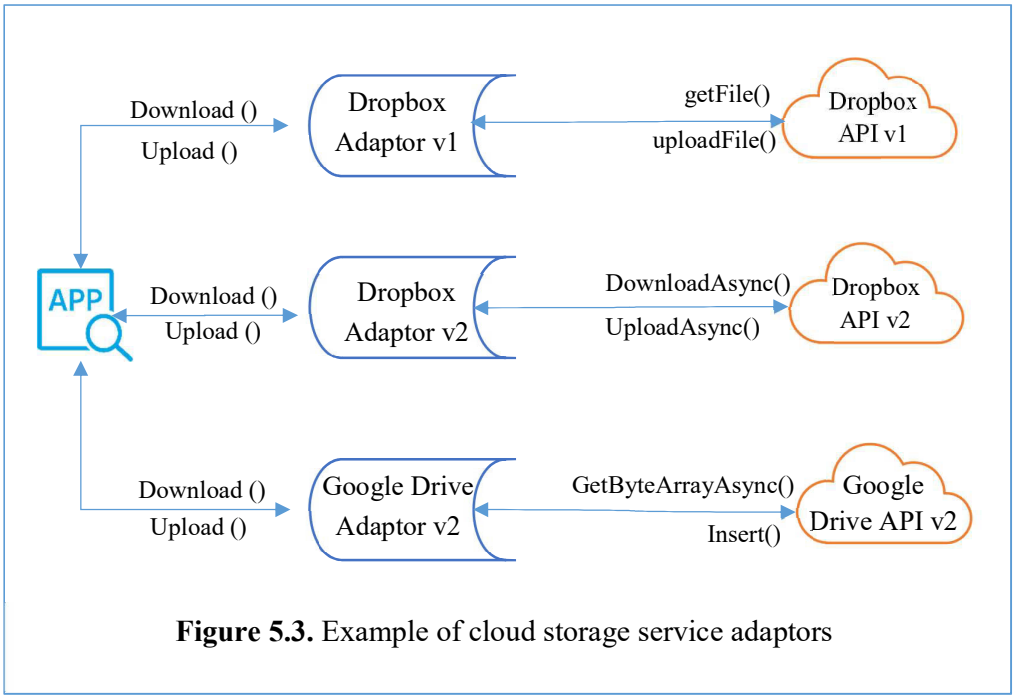


Figure 5.3. Example of cloud storage service adaptors

to the request that is defined by the cloud service provider. Considering the example in Figure 5.3, the Dropbox Adaptor v2 is implemented to transform the generic request (Download) to the Dropbox API version 2 request (DownloadAsync) while the Google Drive Adaptor is for transforming the request to the Google Drive API request (GetByteArrayAsync). The service adaptor is the basic unit of cloud API integration. The service adaptor has multiple versions aligning to the version of a particular cloud service. Each version of the service adaptor is associated with a specific version of the cloud API. When a cloud API is changed, a service adaptor version is implemented, and it is ready for use in various applications that are using this service. It helps developers to avoid duplicated efforts when reacting to the changes of cloud services in different applications.

2. **Runtime logging.** During runtime, the service adaptor writes logs of service transactions so that administrators can monitor the performance and issues of a cloud service. The runtime log is used by the monitoring centre to calculate the availability and average response time of the cloud service.

5.3.3. Workflow Engine

As a module of the SCF that provides the capabilities of cloud service failover, the workflow engine uses service adaptors to configure a number of fault tolerance strategies. Figure 5.4 shows an example of a failover workflow using Dropbox and Google Drive.

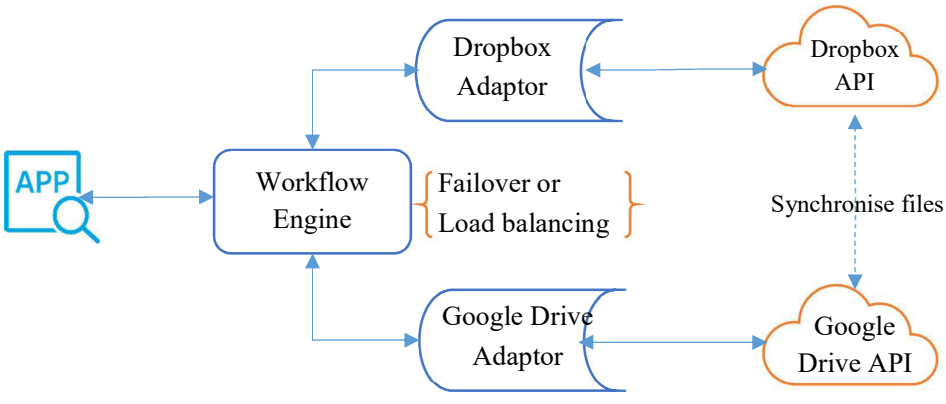


Figure 5.4. Example of a cloud storage workflow

The workflow engine enables the capability to switch from a cloud service to another cloud service during the migration process. It is also the module of SCF that addresses the problem of cloud service disruptions and cloud service changes. In Chapter 6, I discuss the failover strategies implemented in the workflow engine and service adaptors to improve the reliability of enterprise applications.

5.3.4. Monitoring Centre

For runtime management of cloud services, the monitoring centre is implemented to monitor cloud service performance, to calculate the QoS of cloud services, and to provide the information for optimisation. The monitoring centre provides three features as follows:

1. **Log collecting.** This function is implemented to collect service logs from various applications. A log collector is invoked by service adaptors to record the cloud service transaction logs of an enterprise application. The log collector can also be invoked by third-party applications to contribute their logs. During runtime, the log collector works as a notification centre that alerts the administrators about the cloud service performance issues or service disruptions.
2. **QoS calculation.** The monitoring centre is implemented to calculate the response time and availability of cloud services from the transaction logs. These historical-based QoS values are used for cloud service selection during the service identification phase.
3. **Cloud service monitoring.** The main feature of the monitoring centre is that it monitors cloud service uptime and performance. It provides a service monitor tool for service consumers to keep track of cloud services' performance. The availability and performance of cloud services will be compared to expected QoS values specified in the RFS. Service consumers can decide to change to other cloud services in cases where the current service does not satisfy the user expectations.

5.4. SCF Implementation

The SCF is developed using .Net technology. ASP.Net MVC 5 is used to build the service

repository and service monitoring tool of the monitoring center. Windows Communication Foundation (WCF) is used to implement the APIs of the SCF such as the service information API and the log collector. Microsoft SQL Server is used to implement databases for the service repository and monitoring center. The tools and APIs of the SCF are deployed on an AWS EC2 server while the databases are deployed in an AWS RDS. Table 5.1 lists the main modules of SCF and the technologies that are used.

Table 5.1. SCF modules implementation

No	Modules/Components	Technologies	Deployment
1	Service Repository	ASP.NET MVC Microsoft SQL Server	AWS EC2 AWS RDS
2	Service Adaptors	Class Library (.dll)	Nuget
3	Workflow Engine	Class Library (.dll)	Nuget
4	Monitoring Centre	Windows Service Application Microsoft SQL Server Window Foundation Communication	AWS EC2 AWS RDS

5.4.1. Service Repository Implementation

The service repository is a web-based application which is used to manage the information of cloud services. The service repository is developed using .Net technologies, i.e., ASP.Net MVC5, AngularJS, and Entity Framework. It provides a feature for developers populating the cloud service information during the service identification phase. Developers are also able to select the suitable cloud service based on service QoS values stored in the service repository. Figure 5.5 shows the user interface for searching cloud services of the service repository.

5.4.1.1. Service Repository Data Model

Service Repository uses an MSSQL database to manage cloud service information. Service Repository Database is deployed using AWS RDS. Figure 5.6 shows the Entity Relationship Diagram of Service Discovery.

The *service* is the central entity of the database diagram that includes attributes to describe

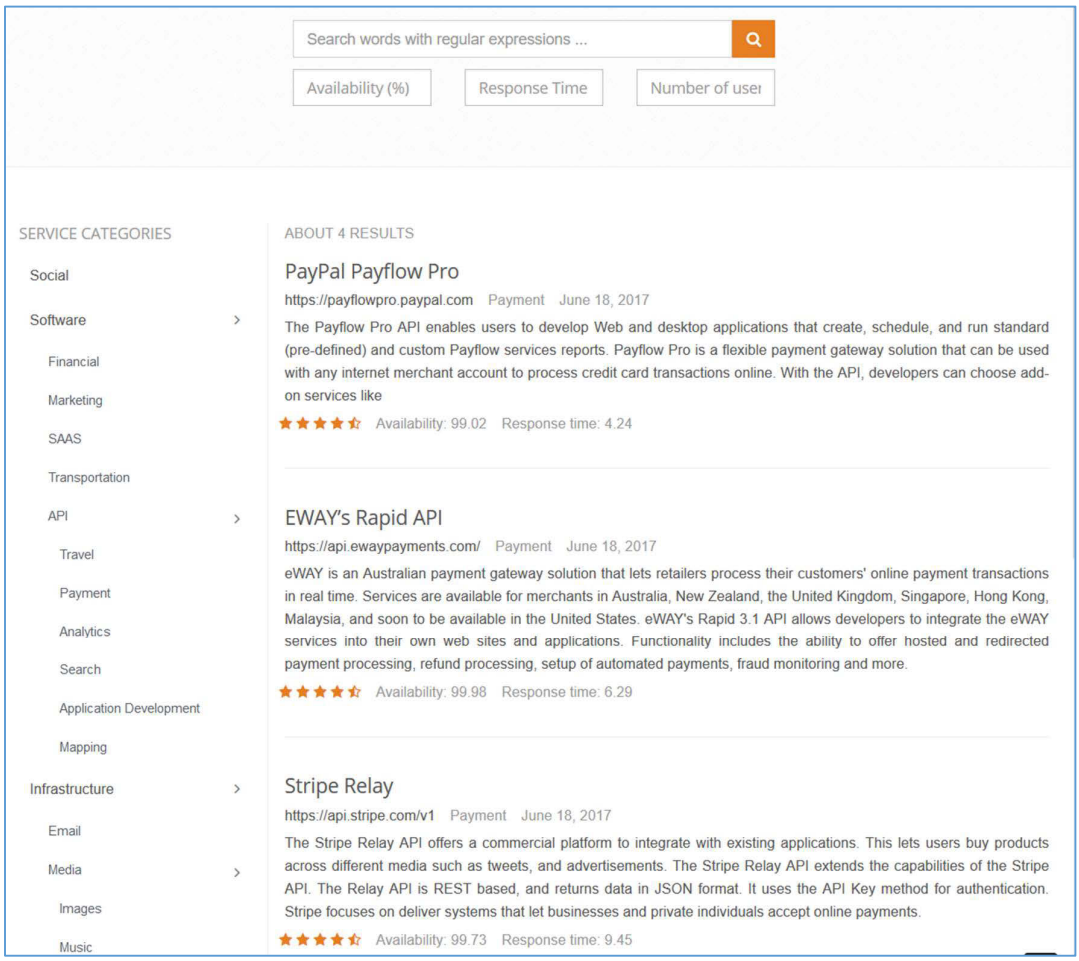


Figure 5.5. Service Repository user interface for cloud service selection

the cloud service. The service entity also contains the general information of Service Level Agreement (SLA). The information about *service version*s is also stored in the service repository in order to manage the changes of the service functionality. The availability and response time of a cloud service version are either entered by developers or the values are calculated by the QoS analysis tool of the monitoring center. A *service QoS entity* stores the QoS values of a cloud service version in a specific location. This information is the result of the analysis process of the QoS analysis tool and cannot be changed by developers.

Other complementary information of a cloud service is the service category and the service provider information. The *service category* is used to classify cloud services according to service models (IaaS, PaaS, SaaS) and service types (e.g., computing,

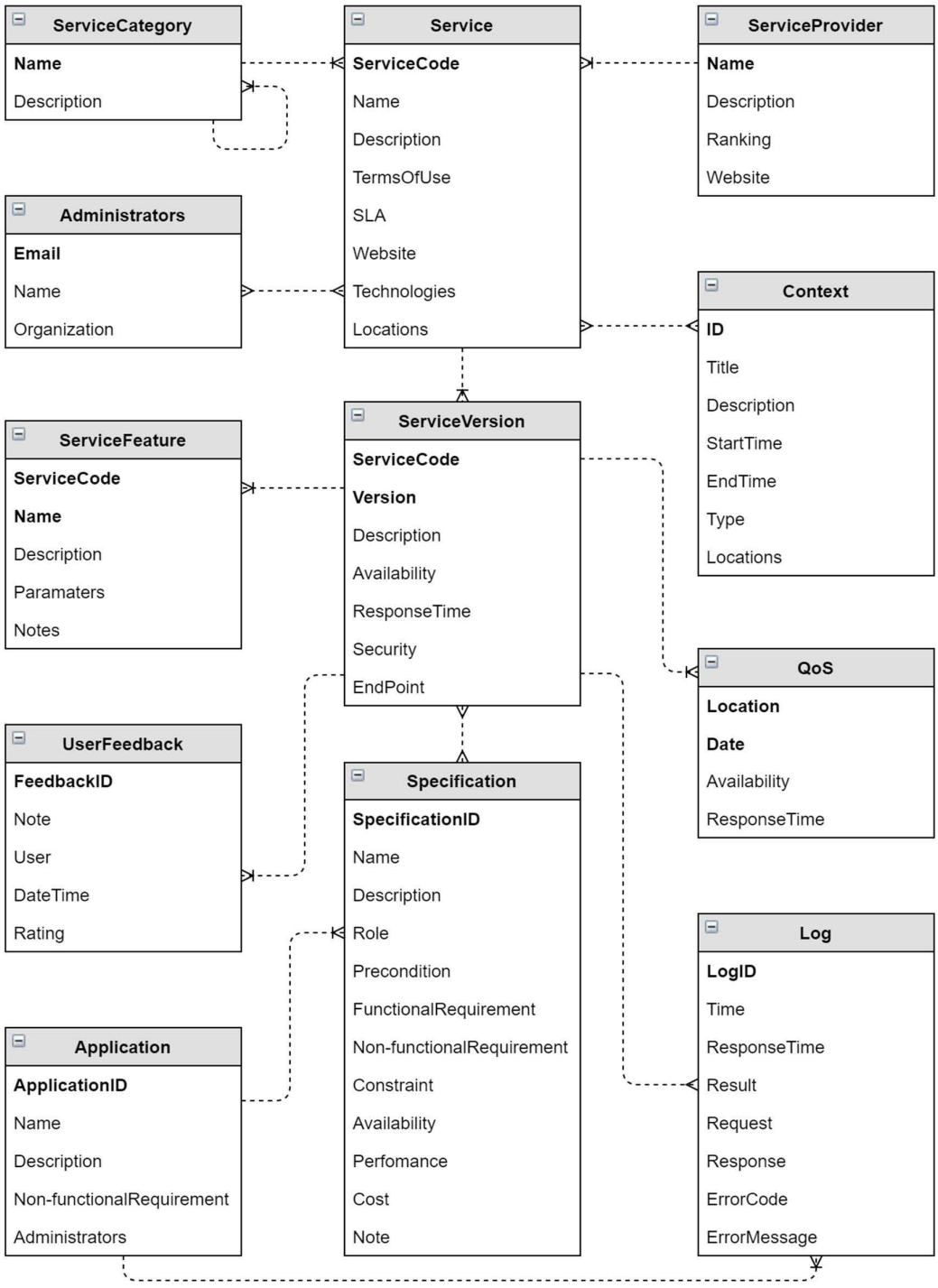


Figure 5.6. Service Repository entity relationship diagram

storage, database, payment) using a service hierarchy. The concept of service substitution is to identify the services with the same category (e.g., two payment services with identical functionality). The cloud services in the same category can be used to implement

the failover strategy for improving the availability. The service category groups the cloud services which have similar features together so that when the administrators decide to change, they can select another cloud service in the same category as the existing service. The *service provider* entity contains the information about the service provider, for example, ranking the reputation which is used for service selection.

For the managing enterprise application, the information of *applications* is stored in the service repository. The requirement statements of application reliability, performance or cost are used as the directives for service selection. An application has a set of Request for Service (RFS) which are stored as specifications. The availability and performance attributes of the specification are used as the criteria for querying cloud services. When cloud services are selected for use or observation, service entity is associated with the related specification. This relationship indicates that the cloud service is being used in the application or the cloud service is being monitored so that the administrator can make the decision of service migration. During the monitoring phase, if the quality of a service is under the values which are specified in the related specification, the monitoring centre will alert the *administrators* about this performance violation.

5.4.1.2. Service Info API

Service Info API is a service that is developed to provide service information. Service Info API is invoked by Service Repository to query the cloud service information. QoS Analysis also invokes the Service Info API to update the QoS value of the services. Service Info API is developed using a Window Communication Foundation (WCF), and it is deployed on AWS EC2. Service Info API has three methods:

- Search: This method is used to query cloud services with three criteria service types, availabilities and response times.
- GetInfo: This method is used to obtain the information about a cloud service. The information includes basic information, service features, versions, events, and location-based QoS values.
- UpdateQoS: This method is used to insert or update the QoS value of a cloud service into the QoS table. If date and location are empty, Service Availability and Response Time in Service Version will be updated.

5.4.2. Workflow Engine Implementation

As the main feature which provides the capabilities of service integration, the workflow engine is an SCF module that uses service adaptors to build up workflows. The workflow engine and service adaptors are both Class Library (dll) that is developed using C#

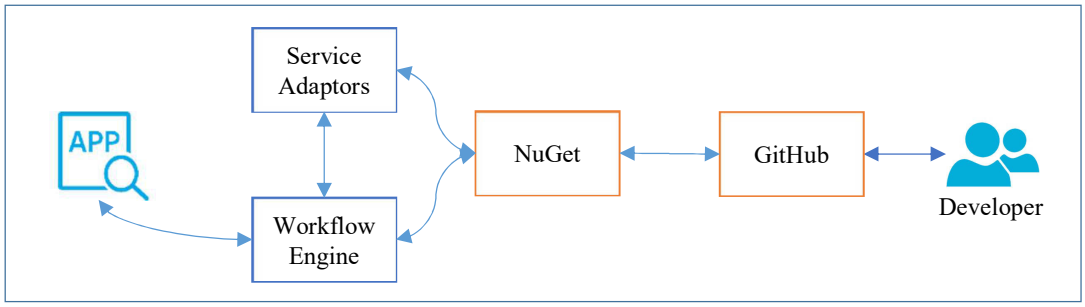


Figure 5.7. Workflow Engine and Service Adaptor implementation

programming language. As illustrated in Figure 5.7, the source code of workflow engine and service adaptor is proposed as an open source project uploaded on GitHub[14] so that developers are able to contribute to upgrade workflow engine or implement new service adaptors for cloud services. The source code project will be built and released in Microsoft NuGet[15] which is the package manager for .NET. The NuGet Gallery is the central package repository used by all package authors and developers in .Net programming.

5.4.2.1. Service Adaptor Library

The service adaptor is a connector to invoke cloud service APIs and write a runtime log of the transaction between the enterprise application and the cloud service. For a type of service, I implemented a generic interface that includes the common methods of that type of cloud service. For example, a generic payment service interface contains three common methods of payment such as Pay, Refund, and CheckBalance. The service adaptors inherit the generic interface and implement the body of the methods for interaction with specific cloud services. Currently, I have implemented a number of service adaptors that are: PayPalAdaptor, eWayAdaptor, StripeAdaptor,

[14] GitHub: https://github.com/tranhongthai/SCF
[15] NuGet https://www.nuget.org/packages

DropboxAdaptor, and GoogleDriveAdaptor. In Appendix C, the codes of payment and storage generic interface and payment adaptors are described to clarify how to unify the cloud API for payment services. Thus, to implement the generic interface, generic messages such as PaymentRequest, PaymentResponse, RefundRequest, and RefundResponse are also defined. The use of the generic interface and common messages enable the ability to switch from a cloud service to an alternative cloud service.

5.4.2.2. Workflow Engine Library

The workflow engine is a Class Library developed using C# programming language. A workflow contains a list of service adaptors or sub-workflows. When executing a workflow, adaptors and sub-workflows are executed in order to invoke the cloud services. In Chapter 6, I describe the configuration of the workflow engine to implement the failover and load balancing strategies among cloud services. Below is a sample code of workflow.

```
public class PaymentWorkflow : IPaymentService
{
     private int _active = 0;
     public List<IPaymentService> Services { get; set; }
     public WorkflowMode Mode { get; set; }

     public PaymentResponse Pay(PaymentRequest request)
     {
          var n = Services.Count;
          var response = new PaymentResponse();

          if (n == 0)
               throw new Exception("There is no service adaptors");
          if (Mode == WorkflowMode.LoadBalance)
          {
               response = Services[_active].Pay(request);
               active = (_active + 1) % n;
               return response;
          }
          // Failover Mode
          foreach (var service in Services)
          {
               response = service.Pay(request);
               if (response.Result != ServiceResult.Error)
                    break;
          }
          return response;
     }
}
```

5.4.3. Monitoring Centre Implementation

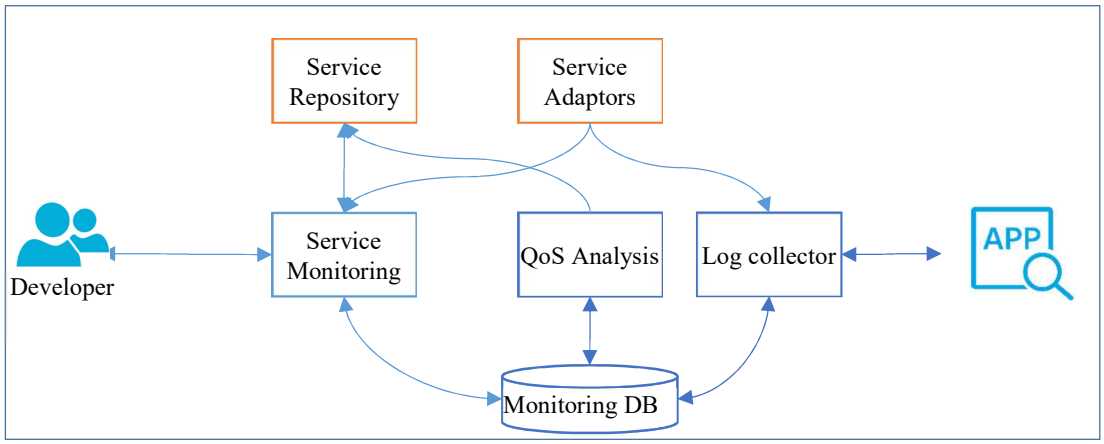


Figure 5.8. Monitoring Center implementation

Monitoring centre is the runtime module of the SCF to support service consumers tracking cloud service events and performance. The Monitoring contains three components to collect service logs, and analyse the service performance from the logs. Illustrated in Figure 5.8, monitoring centre contains three components: QoS Analysis, log collector, and service monitoring.

5.4.3.1. Monitoring Data Model

The monitoring centre uses a MSSQL to store service log and calculate cloud service QoS values. Figure 5.9 shows the Entity Relationship Diagram of QoS Analysis. A *log* is an entity that contains the runtime logs of cloud services. The runtime logs that are collected by service adaptors record the transaction time and location of enterprise application so that the QoS analysis component can calculate the service availability and average response time of cloud services in the specific locations. The reports, i.e., hourly report, daily report, monthly report and general report, are used by the QoS analysis module to analyse the QoS values.

5.4.3.2. QoS Analysis

QoS analysis is a console application that is developed using C# programming language. It is deployed on an AWS EC2 server and is configured to execute as a Window Service. Every hour, QoS analysis is executed to calculate the QoS values of the cloud services for that hour. The values of accurate QoS for each location are calculated separately.

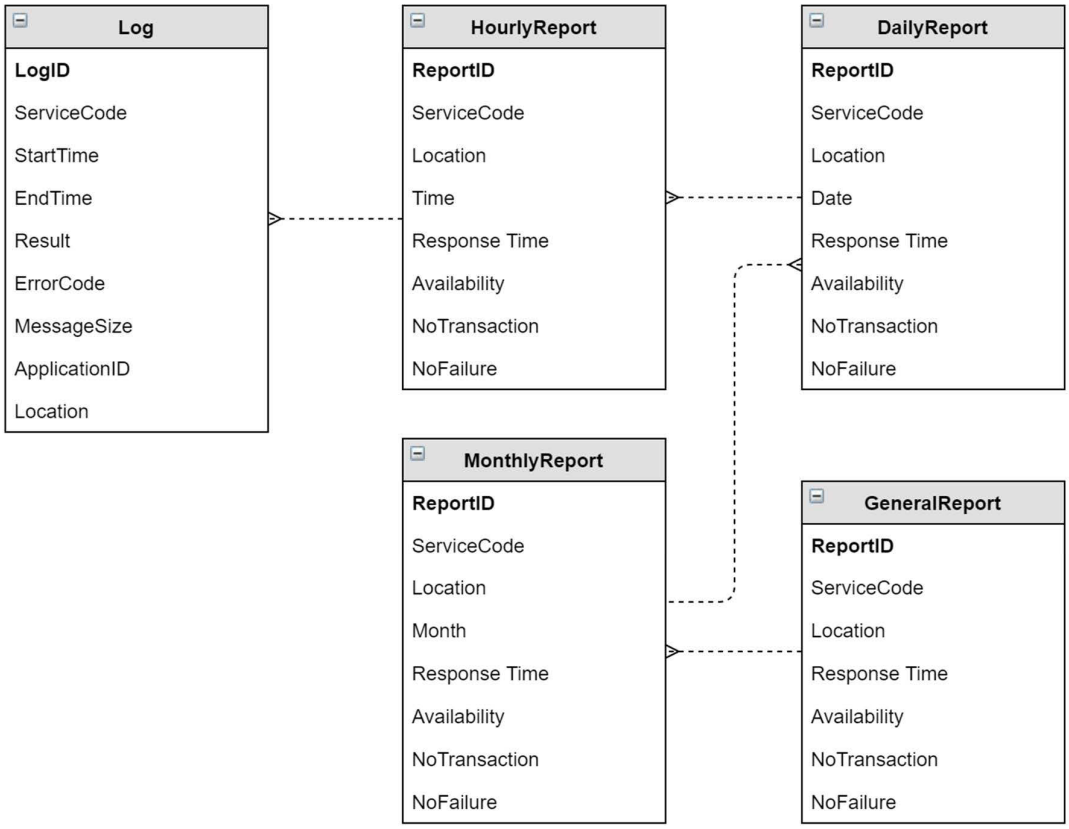


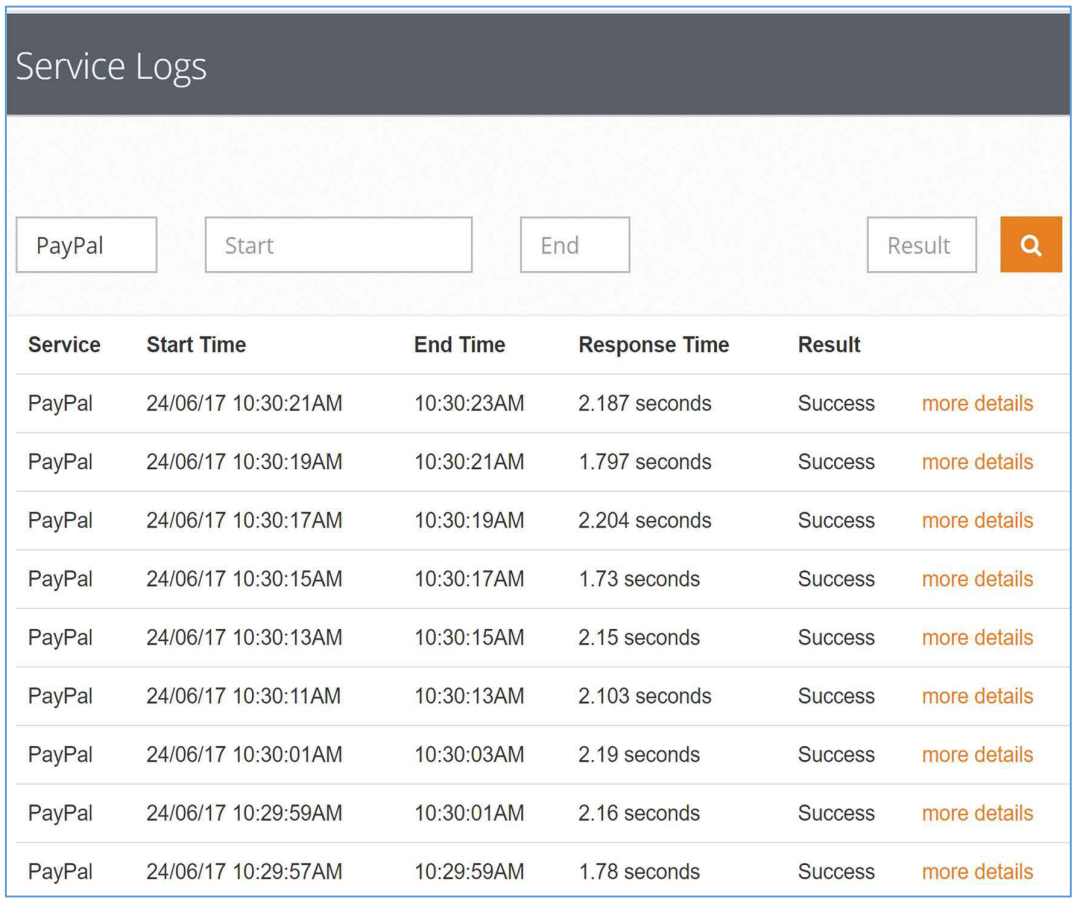
Figure 5.9. QoS analysis entity relationship diagram

5.4.3.3. Log Collector

Log collector is a Window Communication Foundation (WCF) service that is developed to receive runtime logs from applications. Service adaptors send the runtime logs to the monitoring database via log collector. The log collector can also be invoked by third-party applications to contribute their runtime logs. Whenever the log collector receives a service failure that is the log with an error in the result field, the monitoring centre sends a notification to the application administrators who are using the cloud service in their applications.

5.4.3.4. Service Monitor

Service monitor is developed using ASP.Net MVC 5 and is deployed as a client tool for monitoring the cloud services. Figure 5.10 shows the User Interface of the Application Monitor to view service logs of PayPal services. The service monitor module also shows



Service	Start Time	End Time	Response Time	Result	
PayPal	24/06/17 10:30:21AM	10:30:23AM	2.187 seconds	Success	more details
PayPal	24/06/17 10:30:19AM	10:30:21AM	1.797 seconds	Success	more details
PayPal	24/06/17 10:30:17AM	10:30:19AM	2.204 seconds	Success	more details
PayPal	24/06/17 10:30:15AM	10:30:17AM	1.73 seconds	Success	more details
PayPal	24/06/17 10:30:13AM	10:30:15AM	2.15 seconds	Success	more details
PayPal	24/06/17 10:30:11AM	10:30:13AM	2.103 seconds	Success	more details
PayPal	24/06/17 10:30:01AM	10:30:03AM	2.19 seconds	Success	more details
PayPal	24/06/17 10:29:59AM	10:30:01AM	2.16 seconds	Success	more details
PayPal	24/06/17 10:29:57AM	10:29:59AM	1.78 seconds	Success	more details

Figure 5.10. Monitoring Centre user interface

the runtime QoS of cloud service in the comparison with the QoS values that are defined for the requirements specification. The administrators use the information in the service monitor to make a decision in case the service performance is not at the same level as the user expectations.

5.4.4. Evaluation of SCF

The SCF is used to implement a healthcare application for FMP that was discussed in Section 4.8 (Chapter 4). This section discusses the feedback and suggestions of the project team members about the design and features of SCF. Similar to SC-SDLC, the SCF is the results of a rigorous process between the core activities of building and evaluating using this case study. During the stages of the project, The SCF is evaluated and improved to address the comments and feedbacks.

5.4.4.1. Requirements Management and Service Identification

As mentioned in Section 4.8, after the Vastbit team analysed the requirements and documents to write the specification using the RFS template. The RFSs were versioned and stored in the Service Repository. The management of cloud services using Service Consumer Framework improves the collaboration among team members. The non-functional requirements that are recorded in Service Repository is used as the KPIs for application monitoring and management of cloud services.

Feedback: As this is the first project Service Consumer Framework, there is a lack of QoS values of cloud services. The Vastbit team had to enter a number of cloud services and collect the QoS to input into the Service Repository. The advantage is that there is a tool that is used for identifying cloud services.

5.4.4.2. Cloud Service Integration and Management

The Vastbit team implemented the service adaptors for cloud service such as PayPal and the NganLuong, Google Drive API service. The invocation logs of cloud services were collected by service adaptors and they were stored in Service Repository. The Vastbit team also configured cross provider failover for these cloud services using the Workflow Engine. They also set up a server farm between the Azure Southeast server and the AWS EC2 Singapore server and database replication between AWS RDS and Azure MS SQL to implement the failover for infrastructure services.

Feedback: The unified interface of cloud service adaptors is an advantage. The Vastbit team are always concerned about the capabilities of failover. With the service adaptors and workflows engine, they can configure the failover capacities for both infrastructure services (e.g. server, database) and software services (e.g. Payment, Travel). The log system to collect the QoS is also another advantage. In the previous projects, Vastbit team used to write the logs only to debug and trace the failure; and they did not measure the QoS of the cloud service. Using SCF, they have a tool for management of cloud services during runtime.

However, the tool for management of infrastructure services has some limitations. The SCF need to be improved to manage the performance and problems of cloud servers and

cloud databases. Another suggestion that is related to the Service Consumer Framework SCF is the virtualisation system for workflow engines and the monitoring centre.

5.4.5. Design Cycle of SCF

The SCF is the results of an iterative process between evaluation and implementation. The framework features are being continuously improved during the project timeline. The first version of the (Figure 5.11) was developed that focused on management of workflow design and service selection. The main feature of this version of SCF is to manage the service evolution. Then, the second version of SCF was implemented that includes the design of Service Repository to manage RFS and cloud services information. In the third

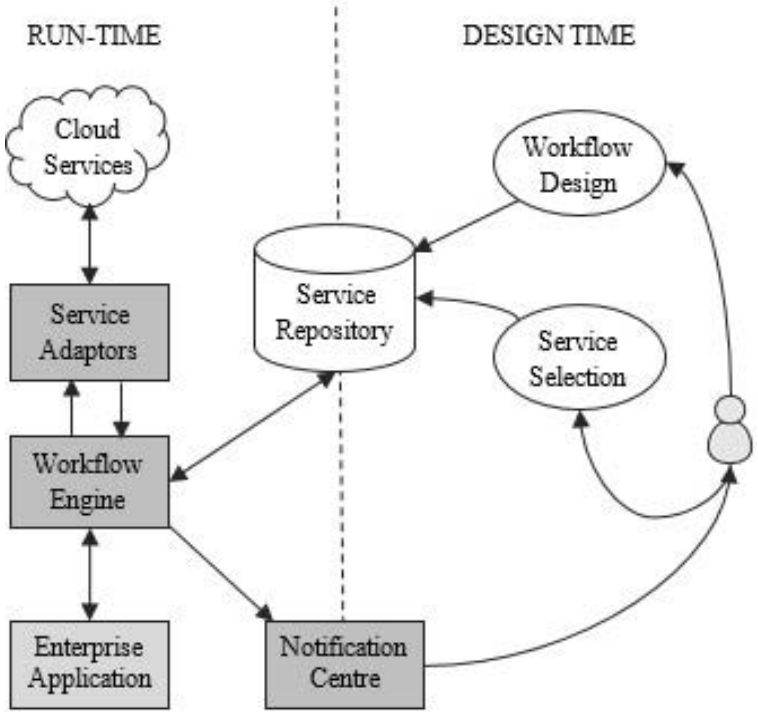


Figure 5.11. The first version of SCF

version, the failover feature of SCF (Chapter 6) was designed to handle the problems of service disruptions. Then in Figure 5.2, the service monitoring module of SCF (Chapter 7) was designed to monitor cloud services for the purposes of service selection and application optimisation.

5.5. Conclusion

Besides the lifecycle methodology for development of the cloud service-oriented application, service consumers also need a comprehensive framework to support the entire lifecycle activities and to manage the use of the cloud service in an enterprise application. In this chapter, the implementation of SCF is described. The SCF is developed to support the Service Consumer SDLC that is proposed in Chapter 4. The SCF has four main modules: service repository, service adaptors, workflow engine, and monitoring centre. Each module has a number of components that provide the capabilities for service consumers to select cloud service at design-time and to manage cloud service information at runtime. As to further directions, the SCF will be deployed as an open framework so that the developers from the IT community or even the cloud service providers can contribute to populate the cloud service information and develop the cloud service adaptors. In the next chapters, I will discuss the capability of SCF in handling the cloud service incidents (Chapter 6) and in monitoring cloud service at runtime (Chapter 7).

CHAPTER 6.

FAILOVER STRATEGIES FOR IMPROVING APPLICATION AVAILABILITY

6.1. Introduction

In the traditional SOA environment, an enterprise application is developed by using the on-premises services within the boundary of one organisation. By taking advantage of Cloud computing, the service consumer begins composing cloud services to application features. The reliability of the enterprise application becomes dependent on the reliability of the consumed cloud services. In the cloud context, service consumers do not have control over externally provided cloud services and therefore cannot guarantee the levels of security and availability that they are typically expected to provide to their users. While most cloud service providers make considerable efforts to ensure the reliability of their services, cloud service consumers cannot assume the continuous availability of cloud services and are ultimately responsible for the reliable operation of their enterprise applications. In this chapter, I describe the various strategies used during the Service Integration phase of SC-SDLC to optimise the reliability and performance of the enterprise application.

In the competitive environment, cloud services are subject to changes. On the one hand, a lot of cloud services were discontinued or retired even though they were products of reputable companies, e.g., Yahoo, Google, Microsoft. On the other hand, many new cloud services are continuing to be introduced in the cloud service market. Thus, cloud services are often changed as service providers implement functional enhancements and rectify defects (Andrikopoulos et al., 2012). Service consumers are unable to predict when or how services will change. Service consumers suffer service disruptions and are forced to upgrade their applications to maintain compatibility with new versions of cloud services, often without any notification. This cloud service evolution, i.e., changes in the functional attributes of services, is an integration challenge that has a significant impact on the reliability of applications. It is becoming imperative to develop effective methods to manage service evolution and to ensure that service consumers are protected from service changes.

In Chapter 5, I described the SCF which is the supporting tool for SC-SDLC. In this chapter, I discuss the capabilities of the *Failover* of the SCF that are used to handle cloud service outages and changes. I describe the reliability features of the SCF which is designed to implement the failover strategies that was discussed in our earlier work (Tran

and Feuerlicht, 2016a). Then, I discuss how the SCF helps to improve the reliability and performance of the cloud-based enterprise application by managing service outages, service changes, and service overload. I discuss how to configure the workflow engine and service adaptor to apply the failover strategies. In Section 6.2, I describe three reliability strategies (Retry Fault Tolerance, Recovery Block Fault Tolerance, and Dynamic Sequential Fault Tolerance) and calculate their expected theoretical impact on the probability of failure and response time. In section 6.3, I describe a load balancing strategy for improving application performance. In section 6.4, I discuss how these reliability strategies and load balancing are implemented using the SCF framework. Section 6.5 describes the experimental setup and gives a comparison of the theoretical results calculated in section 6.2 with the experimental measurements of availability and response time. In Section 6.5, I sum up the functionalities of the SCF.

6.2. SCF Reliability Features

The SCF framework is designed to manage cloud services and aims to address the main issues that impact on the reliability of enterprise applications. In this chapter, I discuss three reliability strategies that are implemented in the SCF framework: Retry Fault Tolerance (RFT), Recovery Block Fault Tolerance (RBFT), and Dynamic Sequential Fault Tolerance (DFST). As discussed in the literature review (Chapter 2), similar strategies were presented recently by Zibin et al. (2012), Reddy and Nalini (2014), and Zheng et al. (2015). However, these works mainly focus on the reliability problems of the on-premises infrastructure. Analysing the different characteristics of cloud services, the SCF uses these reliability strategies to implement the capability of failover for cloud services. I have adapted the RFT, RBFT and DFST reliability strategies for cloud services to address short-term and long-term service outages, and issues arising from service changes. Service adaptors and the workflow engine are configured to implement the various reliability strategies. These reliability strategies can be used to improve the availability of cloud-based enterprise applications by addressing service outages, service changes, and performance problem arising from overloading services.

- *Services outages* can be classified as short-term and long-term outages. Short-term outages are situations where services become temporarily inaccessible, for example

as a result of the loss of network connectivity; automatic recovery typically restores the service following a short delay. Long-term service outages are typically caused by scheduled (or unscheduled) maintenance or system crashes that require service provider intervention to recover the service. Appendix A is a list of PayPal service outages that can cause failures in consumer applications.

- *Service change* involves the change in functional characteristics of services associated with functionality enhancements or the change aimed at improving service performance. It may also involve the change of service interfaces, service endpoints, or security policy, or it is a service discontinuation. During the service change time, most cloud service providers maintain multiple versions of services to limit the impact of such changes on service consumers and attempt to ensure backward compatibility between service versions. However, in practice, service versioning is not possible to avoid breaking consumer applications, resulting in a situation where service consumers are forced to modify their applications to ensure compatibility with the new version of the service.

- *Service overload* occurs when the number of service requests in a given time period exceeds the limits. In general, cloud service providers assign a "bandwidth" to each service consumer. This feature of the service provider system helps to avoid the situation where a service consumer application affects the others. From the service consumer perspective, the significant increase of service use in the short-term can overload the assigned bandwidth for the enterprise application. In other cases, the service provider system itself can be overloaded by the rapid increases in the number of service consumers.

6.2.1. Retry Fault Tolerance

Retry Fault Tolerance (RFT) (Figure 6.1) is a relatively simple strategy commonly used in an enterprise application. Using this strategy, cloud services are repeatedly invoked following a

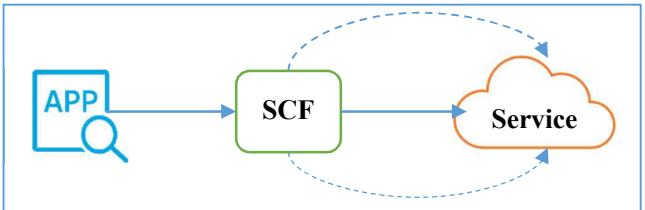


Figure 6.1. Retry Fault Tolerance

delay period until the service invocation succeeds. RFT helps to improve reliability, in particular in situations characterised by short-term outages. The overall probability of failure (PF_{RFT}) can be calculated by:

$$PF_{RFT} = PF^m \tag{6.1}$$

where PF is the probability of the failure of the service and m is the number of retry attempts. While RFT reduces the probability of failure, it may increase the overall response time T_{RFT} due to delays between consecutive service invocations. The total delay can be estimated as:

$$T_{RFT} = \sum_{i=1}^{m}(T_{(i)} + D \times (i-1)) \times (PF)^{i-1} \tag{6.2}$$

where D is the delay between retry attempts and T_i is the response time of the i^{th} invocation. The above calculations assume independent modes of failure of subsequent invocations; this assumption only holds in situations where the delay D is much greater than the duration of the outage, e.g., for long duration outages the invocation will fail repeatedly, invalidating the assumption of the independence of failures of subsequent invocations.

6.2.2. Recovery Block Fault Tolerance

Recovery Block Fault Tolerance (Figure 6.2) is a widely used strategy that relies on service substitution using alternative services invoked in a specified sequence. It is used to improve the availability of critical applications. The failover configuration includes a primary cloud service used as a default (active) service, and stand-by services that are deployed in the event of the failure of the primary service, or when the primary service becomes unavailable because of scheduled/unscheduled

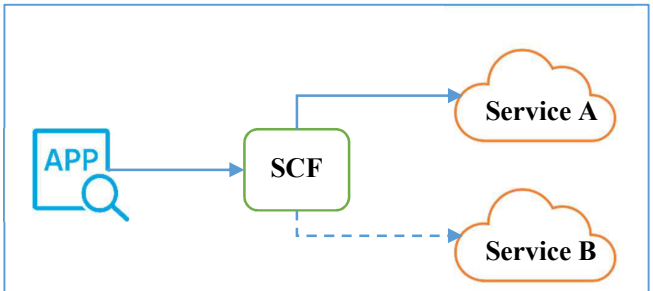


Figure 6.2. Recovery Block Fault Tolerance

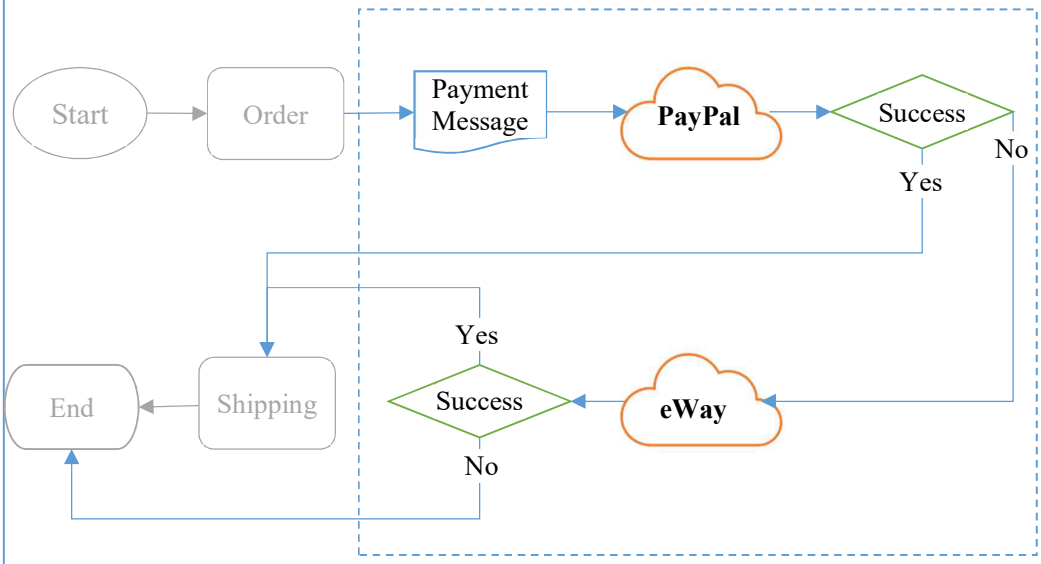


Figure 6.3. Online shopping scenario using a composite payment service

maintenance. Now assuming independent modes of failure, the overall probability of failure for n services combined can be computed by:

$$PF_{RBFT} = \prod_{i=1}^{n} PF_{(i)}\ ;\ A_{RBFT} = 1 - PF_{RBFT} \tag{6.3}$$

where n is the total number of services and PF_i is the probability of failure of the i[th] alternative service. The overall response time $T(s)$ can be calculated by:

$$T_{RBFT} = T_{(1)} + \sum_{i=2}^{n}(T_{(i)} \times \prod_{k=1}^{i-1} PF_{(k)}) \tag{6.4}$$

where T_1 is the response time of the first service invocation and T_i is the response time of the i[th] alternative service invocation. In the online shopping scenario illustrated in Figure 6.3, the composite payment service uses the eWay payment service as an alternative (stand-by) service for the PayPal (primary) service. Assuming that the availability of both the PayPal and eWay services is 99.9% (corresponding to an outage of approximately 9 hours per year), and that the probability of failure PF = 0.01 for each service, the overall RBFT probability of failure $PF = 10^{-6}$, and the overall availability A_{RBFT}s = 99.9999% (this corresponds to an outage of approximately 5 minutes per year).

6.2.3. Dynamic Sequential Fault Tolerance

The Dynamic Sequential Strategy (Figure 6.4) is a combination of the RFT and RBFT strategies. When the primary service fails following RFT retries, the dynamic sequential strategy will deploy an alternative service. The overall probability of failure for the *n* services combined is given by:

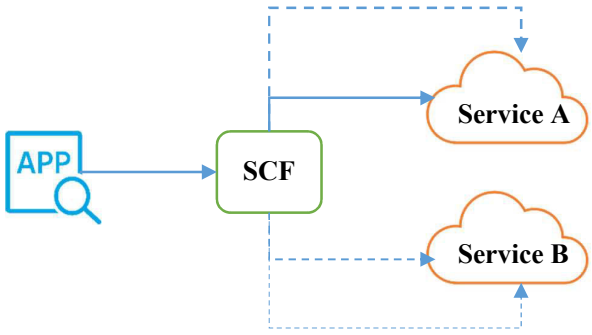


Figure 6.4. Dynamic Sequential Strategy

$$PF_{DSFT} = \prod_{i=1}^{n} PF_{RFT(i)} \; ; \; A_{DSFT} = 1 - PF_{DSRF} \tag{6.5}$$

where $PF_{RFT(i)}$ is the probability of failure of the i[th] alternative service using the RFT strategy calculated in equation (6.1), and $A(s)$ is the overall availability of the composite service using the DSFT strategy. The overall response time $T(s)$ can be calculated by:

$$T_{DSFT} = \mathrm{T}_{RFT(1)} + \sum_{i=2}^{n} (\mathrm{T}_{RFT(i)} \times \prod_{k=1}^{i-1} \mathrm{PF}_{RFT(k)}) \tag{6.6}$$

where $\mathrm{T}_{RFT(1)}$ is the response time of the first service using the RFT strategy in equation (6.2), $\mathrm{T}_{RFT(i)}$ is the response time of the i[th] alternative service calculated in equation (6.2), and $\mathrm{PF}_{RFT(k)}$ is the probability of failure of the k[th] alternative service using the RFT strategy calculated using equation (6.1).

6.3. Cloud Service Load Balancing

Similar to the failover configuration, the Workflow Engine uses multiple services redundantly to provide Load Balancing (LB). The request message is routed to cloud services using a configuration based on the capacity of services. Considering an example of the

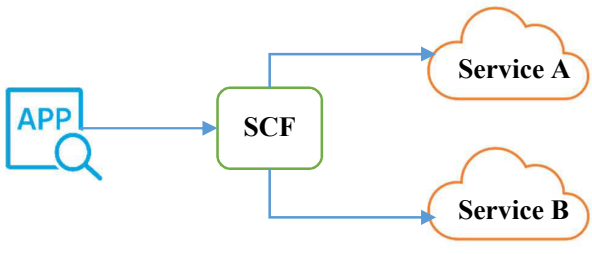


Figure 6.5. Cloud service load balancing

storage services, the storage capacity of Dropbox and OneDrive is 2GB and 25GB respectively. In Figure 6.5, the load balancing configuration shares storage between Dropbox and OneDrive to avoid the overloading of cloud services and to increase the overall storage capacity. In different types of cloud services, load balancing can be configured based on throughput or service loads. For example, the workflow engine which can be configured to use the PayPal service, shares the service load for the eWay service. It helps to improve the service response time as the request processors are shared by multiple services. The overall probability of failure for *n* services combined is given by:

$$\mathrm{PF_{LB}} = \frac{\sum_{i=1}^{n} \mathrm{PF_{RFT(i)}}}{n} \tag{6.7}$$

$$\mathrm{A_{LB}} = 1 - \mathrm{PF_{LB}} \tag{6.8}$$

where n is the number of the load balancing services; $PF_{RFT(i)}$ is the probability of failure of the ith alternative service using the RFT strategy calculated in equation (6.1); and $\mathrm{A_{LB}}$ is the overall availability of the composite service using the load balancing. The overall response time $\mathrm{T_{LB}}$ can be calculated by:

$$\mathrm{T_{LB}} = \frac{\sum_{i=1}^{n} \mathrm{T_{RFT(i)}}}{n} \tag{6.9}$$

where n is the number of load balancing services; $T_{RFT(i)}$ is the response time of the ith alternative service calculated in equation (6.3).

Table 6.1. Suitability of reliability strategies

Method	Short Outages	Long Outages	Service Evolution	Service Overload
RFT	Yes	No	No	No
RBFT	Yes	Yes	Yes	Yes
DSFT	Yes	Yes	Yes	Yes
LB	No	No	No	Yes

Table 6.1 indicates the suitability of the RFT, RBFT, DSFT and LB strategies for different types of reliability and performance challenges. Because RFT relay the same cloud

service with a short delay. RFT is used to provide failover capacity for the short-term outage of the cloud service. With the use of alternative cloud services, RBFT and DSFT provide the cross-provider failover features that address the short-term outages, long-term outages and service evolution. Because Load Balancing (LB) is not a reliability strategy, LB only processes the cloud service overload problem and does not provide the failover features. However, the developer is able to configure both failover and load balancing in a workflow.

6.4. Implementation of Reliability Strategies and Load Balancing

The SCF framework is designed to address the challenges of cloud services incidents that impact on the reliability of enterprise applications. The SCF framework implements RFT,

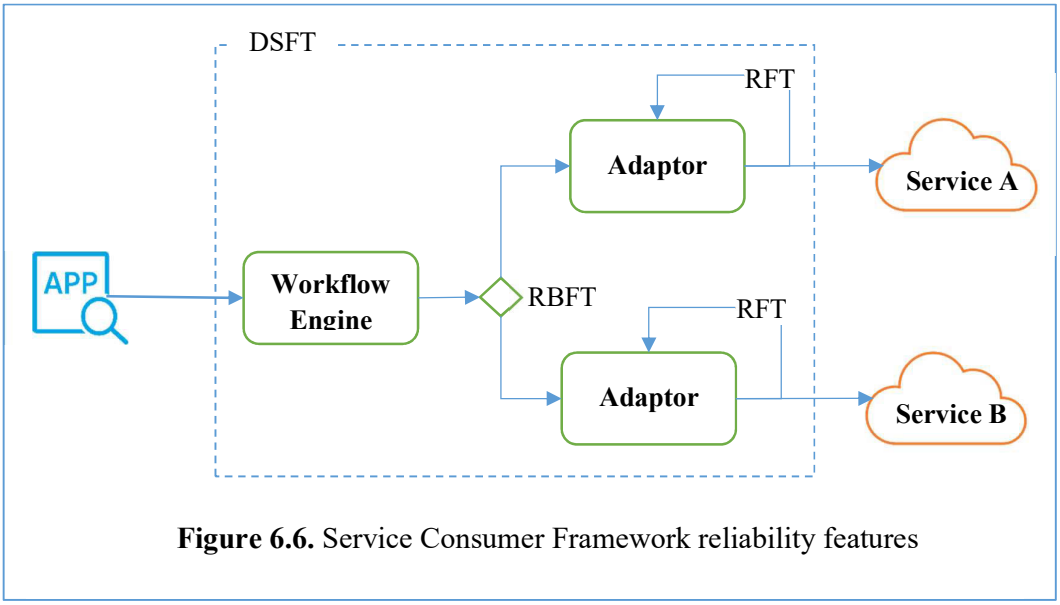


Figure 6.6. Service Consumer Framework reliability features

RBFT and DSFT strategies and is briefly described in the following sections. Figure 6.6 illustrates how Service Adaptors and the Workflow Engine can be configured to implement the various reliability strategies.

6.4.1. Service Adaptors

Service Adaptors are connectors that integrate cloud services with enterprise applications. Each cloud service recorded in the repository is associated with a corresponding service adaptor. Service adaptors use a native interface to transform service requests to a request that is compatible with the current version of the corresponding cloud service,

maintaining compatibility between enterprise applications and external services. The function of a service adaptor is to invoke a service, keep track of service status, and record service execution information in the service repository. Service adaptors implement the RFT reliability strategy by configuring the number of retry attempts and the delay period.

6.4.2. Workflow Engine

The workflow engine is used to implement a workflow and it facilitates cross provider service failover and cloud service load balancing. The workflow engine executes workflows and routes requests to corresponding cloud services. Workflows can be configured to implement the RBFT strategy by using a number of alternative services redundantly. Service adaptors can be configured as active or standby. By default, active service adaptors are used to process the requests and stand by adaptors are deployed in a situation when the primary (active) adaptor requests fail or when the primary adaptor becomes overloaded. Considering the example of cross-provider failover workflow using PayPal and eWay, workflow configuration can be implemented as:

```
workflow.Mode = WorkflowMode.Failover;
```

There are two workflow configuration modes: the load balancing mode or failover mode. After configuration of the application mode, the main cloud service adaptors are configured and added to the workflow. For example, the PayPal service adaptor with RFT configuration of 4 times relay and 6 seconds delay (R=4; D=6) is implemented as:

```
var paypalAdaptor = new PayPalAdaptor();
paypalAdaptor.Attempts = 4;
paypalAdaptor.Delay = 6000;
workflow.Services.Add(paypalAdaptor);
```

The eWayAdaptor is added to the workflow as the standby service. The RFT configuration of the eWay adaptor is 3 times relay and 5 seconds delay (R=3; D=5)

```
var ewayAdaptor = new eWayAdaptor();
ewayAdaptor.Attempts = 3;
ewayAdaptor.Delay = 5000;
workflow.Services.Add(ewayAdaptor);
```

During runtime when the workflow is executed, PayPal is used as the active service and

eWay is used as the stand-by cloud service. In case of PayPal service failure, the eWay service becomes the active service to process the payment request.

6.5. Experimental Verification

This section describes four case studies and the experimental results of failover strategies using various types of cloud services. The case studies are used to accommodate a proof of concept of the developed framework.

6.5.1. Case study of payment services

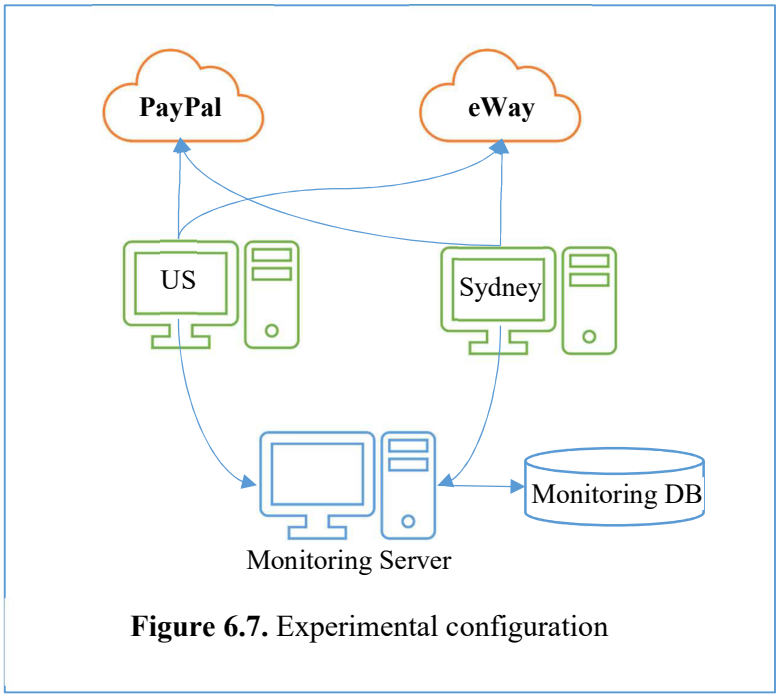


Figure 6.7. Experimental configuration

Figure 6.7 illustrates the experimental configuration that was used to verify the theoretical calculations in Section 6.2. The experimental setup consists of two servers that host the SCF framework and a separate monitoring server. Both SCF servers implement identical payment scenarios using the PayPal Pilot[16] and eWay Sandbox[17] services. The payment requests are randomly generated and sent to the PayPal and eWay payment servers from two different locations. The US west server uses Amazon Web Services (AWS) cloud-based infrastructure located on the west coast of the United States and has a high-quality

[16] PayPal Pilot service: pilot-payflowpro.paypal.com
[17] eWay Sandbox service: https://api.sandbox.ewaypayments.com

server with a reliable network connection. The Sydney server is a local server in Sydney, Australia with a less reliable Internet connection. Because I measure QoS from the client side that is affected by the network connection, I use the Bandwidth Controller[18] application in the simulation environment to create a network disconnect for some seconds randomly that simulate short-term service outages.

6.5.1.1. Experimental Setup

I have collected experimental results from both servers for a period of thirty days, storing the data in the logs on the monitoring server deployed on AWS. The log data records were analysed computing the experimental values of availability and response time for the composite payment service using different reliability strategies. Both SCF servers generate payment requests randomly between 5 and 10 seconds, and use the following four strategies:

Strategy 1: Payment requests are sent directly to the payment service without applying any reliability strategy.

Strategy 2: Payment requests are sent to the payment service using the RFT strategy with three retry attempts (R=3) and a delay of five seconds (D=5).

Strategy 3: Payment requests are sent to a composite payment service using the RBFT strategy.

Strategy 4: Payment requests are sent to a composite payment service using the DSFT strategy which is combination of RBFT and RFT with PayPal (R=3, D=5) and eWay (R=3, D=5).

6.5.1.2. Experimental Results

I have collected the payment transaction data independently of the values available from the cloud service providers, storing this information in the log files on the monitoring server. Table 6.2 shows a fragment of the response time measurements. The use of two separate servers in two different locations enables the comparison of availability and

[18] Bandwidth Controller: http://bandwidthcontroller.com/features.html

response time information collected under different connection conditions.

Table 6.2. Payment service transaction logs

Service	Start Time	Response Time	Result
eWay	31/03/2016 12:51	3.59 seconds	Success
PayPal	31/03/2016 12:50	1.48 seconds	Success
PayPal	31/03/2016 12:50	1.39 seconds	Success
PayPal	31/03/2016 12:49	1.39 seconds	Success
eWay	31/03/2016 12:49	2.50 seconds	Success
PayPal	31/03/2016 12:48	1.44 seconds	Success
eWay	31/03/2016 12:48	1.51 seconds	Success
eWay	31/03/2016 12:47	1.72 seconds	Success
PayPal	31/03/2016 12:47	1.41 seconds	Success
PayPal	31/03/2016 12:46	1.39 seconds	Success
eWay	31/03/2016 12:46	2.17 seconds	Success
PayPal	31/03/2016 12:45	1.39 seconds	Success
PayPal	31/03/2016 12:45	1.00 seconds	Error
eWay	31/03/2016 12:44	2.14 seconds	Success

Table 6.3. Availability of payment services

Server	PayPal	eWay	PayPal RFT	eWay RFT	PayPal-eWay RBFT	PayPal-eWay DSFT
US West	90.48%	93.86%	97.90%	97.26%	99.80%	99.95%
Sydney	90.18%	93.53%	97.28%	97.03%	99.29%	99.82%

As shown in Table 6.3, using Strategy 1 (i.e., without deploying any reliability strategy) the availability of the PayPal and eWay services on the US west server is 90.4815% and 93.8654%, respectively. Deploying the RFT strategy (Strategy 2) the availability increases to 97.9033% and 97.2607%, for the PayPal and eWay services, respectively.

Using the RBFT strategy (Strategy 3), the availability of the composite service (PayPal and eWay) increases to 99.8091%, and finally, using the DSFT strategy (Strategy 4) the availability of the composite service (PayPal + eWay) increases to 99.9508%. The theoretical values obtained in section 6.2 are slightly higher than the experimental values; this can be explained by noting that connection issues may affect both the PayPal and eWay services concurrently thereby invalidating the assumption of independent modes of failure.

Table 6.4. Response time of payment services in seconds

Server	PayPal	eWay	PayPal RFT	eWay RFT	PayPal-eWay RBFT	PayPal-eWay DSFT
US West	1.35	1.84	2.33	2.66	2.06	2.44
Sydney	3.70	1.75	5.67	3.02	4.98	5.43

Table 6.4 shows the average response time of the PayPal and eWay services using different reliability strategies during the period from March 15th to April 15th, 2016. The average response time of the US west server is considerably lower than the Sydney server when connecting to the PayPal service in the US. However, for the eWay service (https://www.eway.com.au/), which is located in Australia, the response time of the

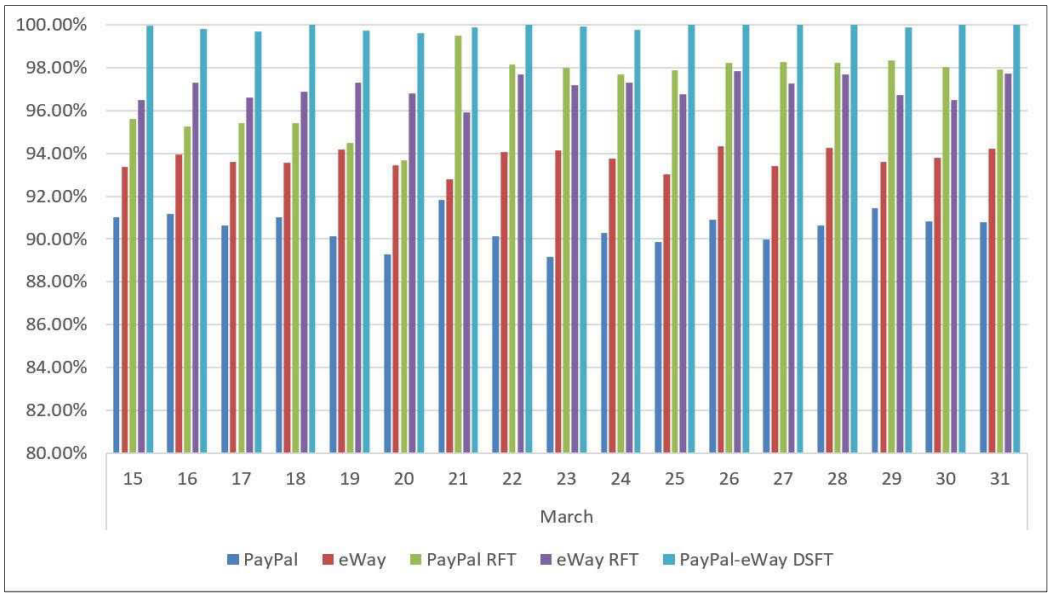


Figure 6.8. Availability of reliability strategies from 15th to 31st of March

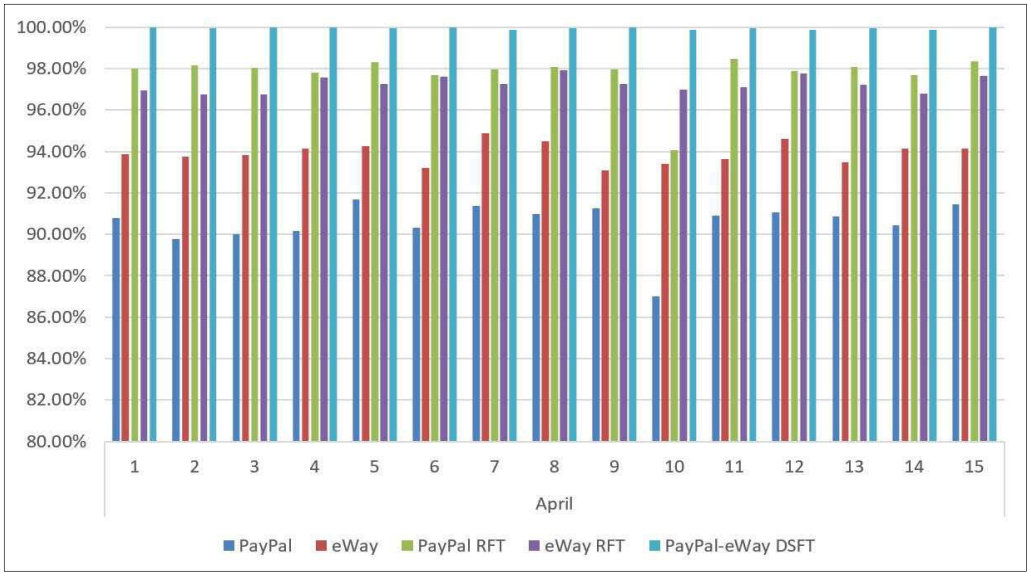


Figure 6.9. Availability of reliability strategies from 1st to 15th of April 2016

Sydney server is slightly better than for the US west server. The bar charts in Figure 6.8 and 6.9 give a comparison of the available values for various reliability strategies for the period of March 15th to April 15th, 2016. As the figures illustrate, the availability of the PayPal and eWay services using any of the reliability strategies is significantly higher than they are without deploying a reliability strategy. During the measurement period, the availability of the PayPal service varied between 88% and 92%, but the availability of the combined PayPal-eWay services using the DSFT strategy remained above 99.9%.

6.6. Conclusion

In this chapter, the cross-provider failover approach has been proposed and implemented using SCF. The different fault tolerance strategies (RFT, RBFT, DSFT) that are used to configure the cloud service failover are discussed. In this chapter, I have estimated the theoretical improvements in service availability that can be achieved using the Retry Fault Tolerance, Recovery Block Fault Tolerance, and Dynamic Sequential Fault Tolerance strategies and compared these values to experimentally obtained results. The experimental results obtained using the SCF framework are consistent with theoretical predictions, and indicate significant improvements in service availability when compared to invoking cloud services directly (i.e., without deploying any reliability strategy). In the

specific case of payment services, the availability for PayPal and eWay services increased from 90.4815% and 93.8654% respectively for direct payment service invocation, and to 97.9033% and 97.2607%, for PayPal and eWay services respectively, when the RFT strategy was used. Using the RBFT strategy, the availability of the composite service (PayPal + eWay) increased to 99.8091% and using the DSFT strategy, the availability of the composite service (PayPal + eWay) increased further to 99.9508%. The experimental result is used to demonstrate how the SCF improves the availability of cloud-based enterprise applications by addressing service outages, service evolution, and failures arising from overloaded services. In future works, I am going to implement a machine learning that dynamically adjusts the reliability configuration in service adaptors and the workflow engine to optimise the availability and performance. For example, the workflow engine will learn from historical logs to investigate the best configuration of delay times and the number of attempts for specific cloud services so that RFT configuration can be dynamically adapted at runtime to optimise application reliability and performance.

CHAPTER 7.

MULTI-SITE MONITORING FOR APPLICATION OPTIMISATION

7.1. Introduction

Cloud services are sourced from different cloud providers and their QoS (Quality of Service) characteristics can substantially differ depending on the geographical location and on the provider cloud infrastructure. While most cloud service providers publish QoS information on their websites, it often does not accurately reflect the values measured at the consumer site as the performance of cloud services is impacted by numerous factors that include dynamic changes in network bandwidth and transmission channel interference (Wenmin et al., 2011). Additionally, changes in provider internal architecture and method of service delivery can significantly impact on the QoS characteristics of cloud services. Consequently, consumer monitoring and optimisation of the runtime behaviour of cloud services has become critically important for the management of enterprise applications (Safy et al., 2013).

Service monitoring is a run-time activity that involves recording the values of response time, availability and other non-functional service parameters in order to enable predictive analysis and proactive service management. Service monitoring and service management in cloud computing environments present a particular challenge to application administrators as the enterprise application is dependent on the performance and availability of third-party cloud services. The traditional approach to QoS monitoring is based on continuously sending test messages to critical services to check their availability and performance. This approach is not suitable for the monitoring of cloud services as it increases service costs and generates unnecessary data traffic.

Monitoring and optimisation of the QoS of cloud services present an important and challenging research problem. Although some research work on monitoring of the QoS characteristics of cloud services is available in the literature, there is currently a lack of detailed information about the assessment of the run-time behaviour of cloud services that includes location-based QoS information (Aceto et al., 2013). Accordingly, it is difficult to make informed decisions about the selection and composition of cloud services (Lu et al., 2014, Noor et al., 2014). In this chapter, I discussed a multi-site monitoring model that was discussed in our earlier publication (Tran and Feuerlicht, 2016b). The model focuses on improving the estimates of the availability and response time of cloud services by introducing location-based QoS information. The QoS of eWay

and PayPal services across eleven locations in four geographical regions was monitored to obtain a more accurate estimate of the response time and availability for service selection based on specific deployment locations of the consumer enterprise application. The QoS information was collected independently of the information published by cloud service providers by collecting payment transaction logs from various applications. In the next section (Section 7.2), I discuss service optimisation using multi-site monitoring. Section 7.3 describes the experimental setup for multi-site monitoring of cloud services and gives experimental results of the availability and response time for eWay and PayPal payment services measured at eleven geographic locations. Section 7.4 contains the conclusions and proposals for future work.

7.2. Multi-Site Monitoring Model

Service optimisation is concerned with continuous service improvement and aims to optimise the performance and cost of business services. Consider, for example, the situation illustrated in Figure 7.1. This shows an Online Shopping Check Out service

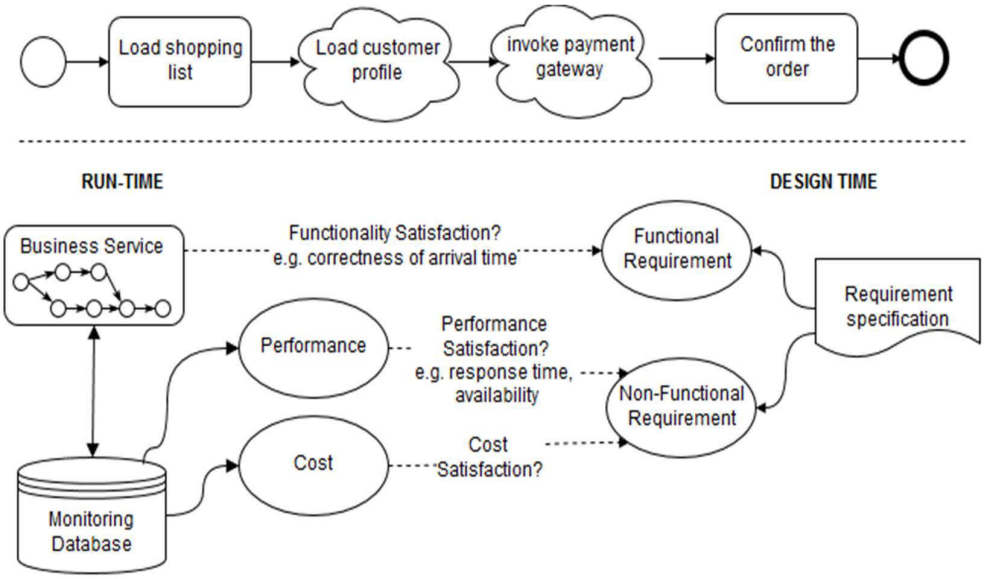


Figure 7.1. Online shopping check out optimisation scenario

which includes a cloud-based payment gateway. At design time, the service consumer needs to select a suitable payment service to integrate into the business workflow ensuring that both the functional and non-functional requirements are satisfied. Making this

selection decision requires the knowledge of the QoS parameters at the site where the enterprise application is deployed.

Typically, both the service provider and service consumer perform service monitoring independently, and both parties are responsible for resolving service quality issues that may arise. Service providers maintain transactions logs and make these logs available to service consumers who can use this information to calculate service costs and to estimate the QoS. Provider QoS data is collected continuously at the provider site irrespective of any connectivity issues and includes information about planned and unplanned outages. However, the QoS values published by service providers may not accurately reflect the values measured at the service consumer site as QoS depends on the deployment location of the enterprise application and is affected by the quality of the network connection, provider location, and service configuration. With some global cloud service providers,

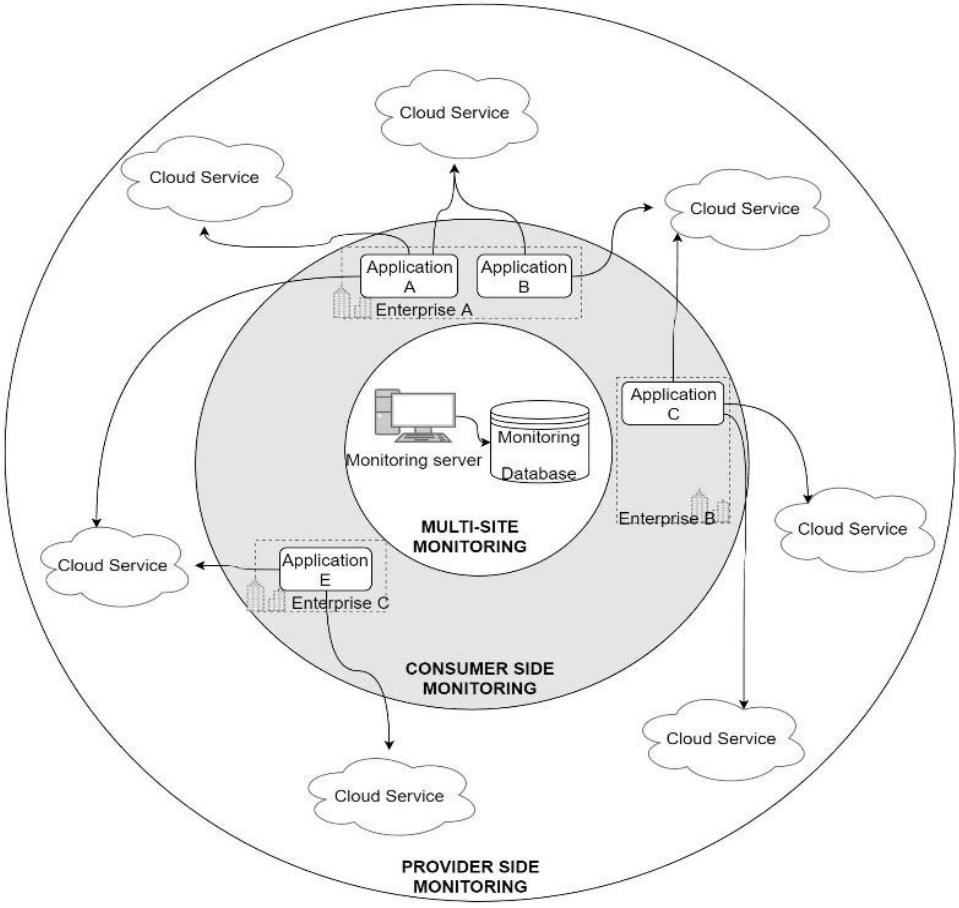


Figure 7.2. Multi-site cloud service monitoring

the actual location from which the service is delivered may not be known to service consumers, making it difficult to optimise the performance of the enterprise application based on QoS values published by the provider. The QoS values measured at the consumer site are impacted by connectivity issues, and while these values may not fully reflect provider site QoS measurements, they are important indicators of enterprise application performance. Multi-site monitoring can be used to overcome the limitations of single-site (provider or consumer) QoS monitoring by mapping the behaviour of cloud services across different sites and geographical regions. I argue that in order to fully optimise cloud service selection and deployment and to ensure that the non-functional requirements are met at run-time, the service consumer needs to know the runtime QoS values of cloud services as measured in different geographic locations. To accomplish this, I propose a model that uses a centralised monitoring database to collect service QoS data from multiple service consumer locations and make this data available for analysis by service consumers (Figure 7.2). This can be achieved by collaboration among different service consumers who record their local monitoring data in a global QoS database and share this information with other consumers of cloud services. The implementation of such a shared QoS monitoring database would enable accurate real-time QoS analysis and real-time notifications of QoS issues. Runtime performance information (i.e., response time, availability and various types of error messages) recorded in the database can be used by application administrators to monitor service utilisation, plan maintenance activities, and to perform statistical analysis of response time and throughput for individual cloud services.

7.2.1. Enterprise application optimisation strategies

Optimisation of enterprise applications that use cloud services may involve a number of different strategies that range from using alternative cloud services to migrating the servers that run the application to a different cloud infrastructure. With increasing availability of alternative cloud services with equivalent functionality, service consumers can choose services to use in their enterprise applications based on the cost and QoS characteristics. This may involve deployment of a new version of an existing service or replacement of the service with an alternative from a different provider if the original service becomes obsolete or too costly. Service consumers can also optimise application

performance by re-locating the application to a different cloud infrastructure selecting a more suitable geographic location, taking into account both end-user connectivity and connectivity to third-party cloud services. Finally, QoS characteristics of cloud-based enterprise applications can be improved by using various reliability strategies, re-configuring cloud services to provide higher levels of fault tolerance (Tran and Feuerlicht, 2016a). These fault tolerance strategies include Retry Fault Tolerance (RFT), Recovery Block Fault Tolerance (RBFT) and Dynamic Sequential Fault Tolerance (DFST) strategies. Using RFT strategy, cloud services are repeatedly invoked following a delay period until the service invocation succeeds. RFT helps to improve reliability, in particular in situations characterised by short-term outages. The RBFT strategy relies on service substitution using alternative services invoked in a specified sequence. This *failover* configuration includes a primary cloud service used as a default (active) service, and stand-by services that are deployed in the event of the failure of the primary service, or when the primary service becomes unavailable because of scheduled/unscheduled maintenance. The DFT strategy is a combination of the RFT and RBFT strategies that deploy an alternative service when the primary service fails following RFT retries (Zheng and Lyu, 2015). The choice of an optimal strategy for the deployment of cloud services must be based on in-depth knowledge of QoS characteristics including their dependence on the geographical location.

7.3. Experimental setup for multi-site monitoring

In order to evaluate the proposed location-based QoS approach to optimisation of cloud-based enterprise applications, I have implemented an experimental multi-site monitoring environment for two payment services: PayPal Pilot service (pilot-payflowpro.paypal.com) and eWay Sandbox (https://api.sandbox.ewaypayments.com). The QoS data was collected using Amazon Elastic Compute Cloud (AWS EC2) servers deployed at eleven sites (Mumbai, Seoul, Singapore, Sydney, Tokyo, Frankfurt, Ireland, Sao Paulo, California, Oregon and Virginia) across four different geographic regions (Asia Pacific, Europe, South America and the US). The monitoring database was implemented using Microsoft SQL Server Amazon Relational Database (AWS RDB). The QoS data was collected in each site by monitoring payment transactions and removing private data such as customer information before recording the information in

the monitoring database. Simulating over 400,000 payment transactions initiated by 300 users, payment services were invoked using the SCF (Service Consumer Framework) payment service adaptor that logs the service name, location, start time, end time, result, and error code for each payment transaction.

The payment service response time for a transaction (T_T) was calculated as:

$$T_T = \mathrm{T}_E - \mathrm{T}_s \quad (1)$$

where T_E is the *end time* of transaction and T_S is the *start time* of a transaction, and the average response time (T_S) of a service was calculated as:

$$T_s = \frac{\sum_1^n \mathrm{T}_T}{n} \quad (2)$$

where n is the number of transactions, and T_T is the response time of a transaction in equation (1). Similarly, an inactive time or *downtime* of a service (T_I) is calculated as:

$$T_I = \mathrm{T}_{IS} - \mathrm{T}_{As} \quad (3)$$

where T_{IS} is the *start time* of a failed transaction and T_{As} is the start time of the next successful transaction. Then, the availability of a service (A_s) is calculated as:

$$D = \mathrm{T}_{LE} - \mathrm{T}_{FS} \quad (4)$$

$$PF_s = \frac{\sum T_I}{D} \quad (5)$$

$$A_S = 1 - PF_S \quad (6)$$

where D is the duration of the test period that is calculated using the *end time* of the last transaction (T_{LE}) and the *start time* of first transaction (T_{FS}). PF_s is the probability of the failure of a service, and T_I is the downtime in equation (3) and A_S is the availability of a service.

Table 7.1. QoS data for eWay and PayPal payment services

Region	Location	Service	Attempts	Number of Fails	Response Time	Availability
Asia Pacific	Mumbai	eWay	15779	85	1.42 seconds	99.46%
		PayPal	15771	73	3.35 seconds	99.54%
	Seoul	eWay	18025	95	1.27 seconds	99.47%

		PayPal	18013	80	2.70 seconds	99.56%
	Singapore	eWay	16923	86	1.36 seconds	99.49%
		PayPal	16911	66	2.92 seconds	99.61%
	Sydney	eWay	23781	126	1.23 seconds	99.47%
		PayPal	23779	110	2.72 seconds	99.54%
	Tokyo	eWay	18324	107	1.27 seconds	99.42%
		PayPal	18310	72	2.75 seconds	99.61%
Europe	Frankfurt	eWay	17035	71	1.56 seconds	99.58%
		PayPal	17031	95	2.73 seconds	99.44%
	Ireland	eWay	17020	101	1.58 seconds	99.41%
		PayPal	17020	60	2.86 seconds	99.65%
South America	Sao Paulo	eWay	16198	76	1.55 seconds	99.53%
		PayPal	16189	61	2.91 seconds	99.62%
US	California	eWay	20665	118	1.28 seconds	99.43%
		PayPal	20653	82	2.02 seconds	99.60%
	Oregon	eWay	25763	127	1.28 seconds	99.51%
		PayPal	25759	156	2.27 seconds	99.39%
	Virginia	eWay	18892	89	1.49 seconds	99.53%
		PayPal	18876	105	2.47 seconds	99.44%

Table 7.1 shows the response time of eWay and Paypal payment services as measured in different geographical locations over the monitored period 1st May to 30th June 2017. The table shows that the response time of the eWay service is better (in most cases less than half) than the response time of the PayPal service, while the availability of both services is approximately the same. Both response time and availability are influenced by two

major factors: provider QoS characteristics and the reliability of the network connection. In order to optimise the consumer side QoS characteristics, it is important to identify which of these factors plays a dominant role. If network connectivity is the dominant factor that impacts on service quality, then using the RFT fault tolerance strategy described in section 7.2.1 above may improve consumer side QoS, but only for situations characterised by short-term outages or latency fluctuations. When network connectivity suffers from long-term outages, the solution may involve migrating the service to a different cloud infrastructure in a different geographical location. However, if network connectivity is not a dominant factor and QoS degradation is caused by provider related issues, then RBFT and DFST fault tolerant strategies may provide a solution by substituting alternative services at runtime. In order to differentiate between network connectivity and cloud service provider issues, I analyse the level of dependence between QoS parameters for the two payment services (eWay and PayPal) at each location by calculating the correlation coefficient for response time and availability. High level of dependence indicates that both payment services fail or suffer from increased response time at the same time, identifying network connectivity as the main source of the problem. Low levels of correlation indicate independent modes of failure for the two payment services, pointing to the service provider as the cause of QoS fluctuations.

Table 7.2. Response time and availability correlation coefficients

Region	Location	Response Time	Availability
Asia Pacific	Mumbai	-0.0751	-0.0589
	Seoul	0.1394	-0.0285
	Singapore	0.1015	0.0162
	Sydney	0.0561	0.0923
	Tokyo	-0.0531	0.0728
Europe	Frankfurt	-0.1855	-0.3184
	Ireland	-0.0659	-0.1902
South America	Sao Paulo	0.1799	0.0743
US	California	0.0468	0.0699

	Oregon	-0.1442	-0.0122
	Virginia	-0.0559	0.1099

Table 7.2 shows the values of correlation coefficients of eWay and PayPal payment services calculated for different locations. The correlation coefficient $C_{(T_e,T_p)}$ (Microsoft, 2016) of response time between eWay and PayPal services is calculated as:

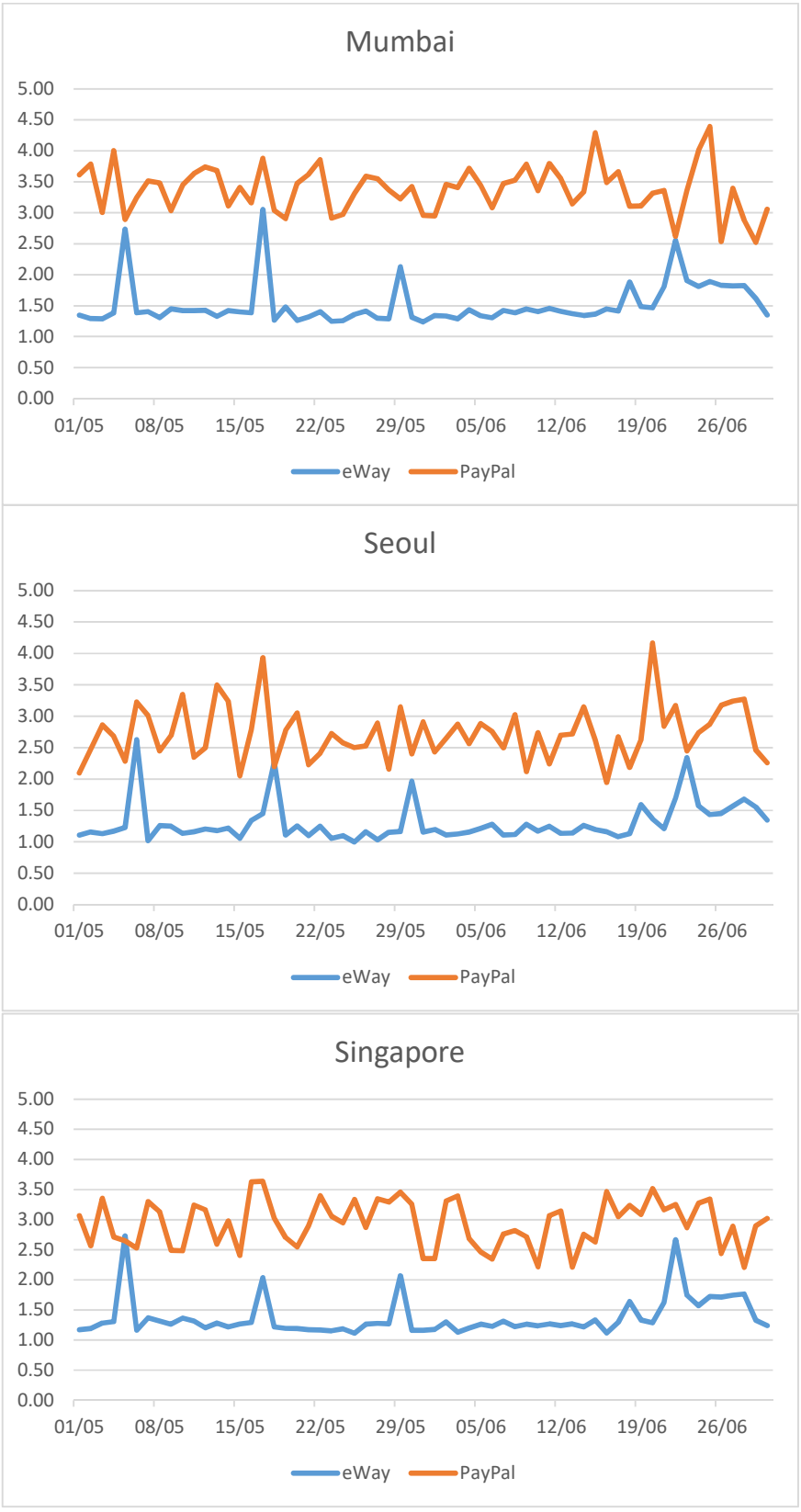
$$C_{(T_e,T_p)} = \frac{\sum(T_e - \overline{T_e})(T_p - \overline{T_p})}{\sqrt{\sum(T_e - \overline{T_e})^2 \sum(T_p - \overline{T_p})^2}} \quad (7)$$

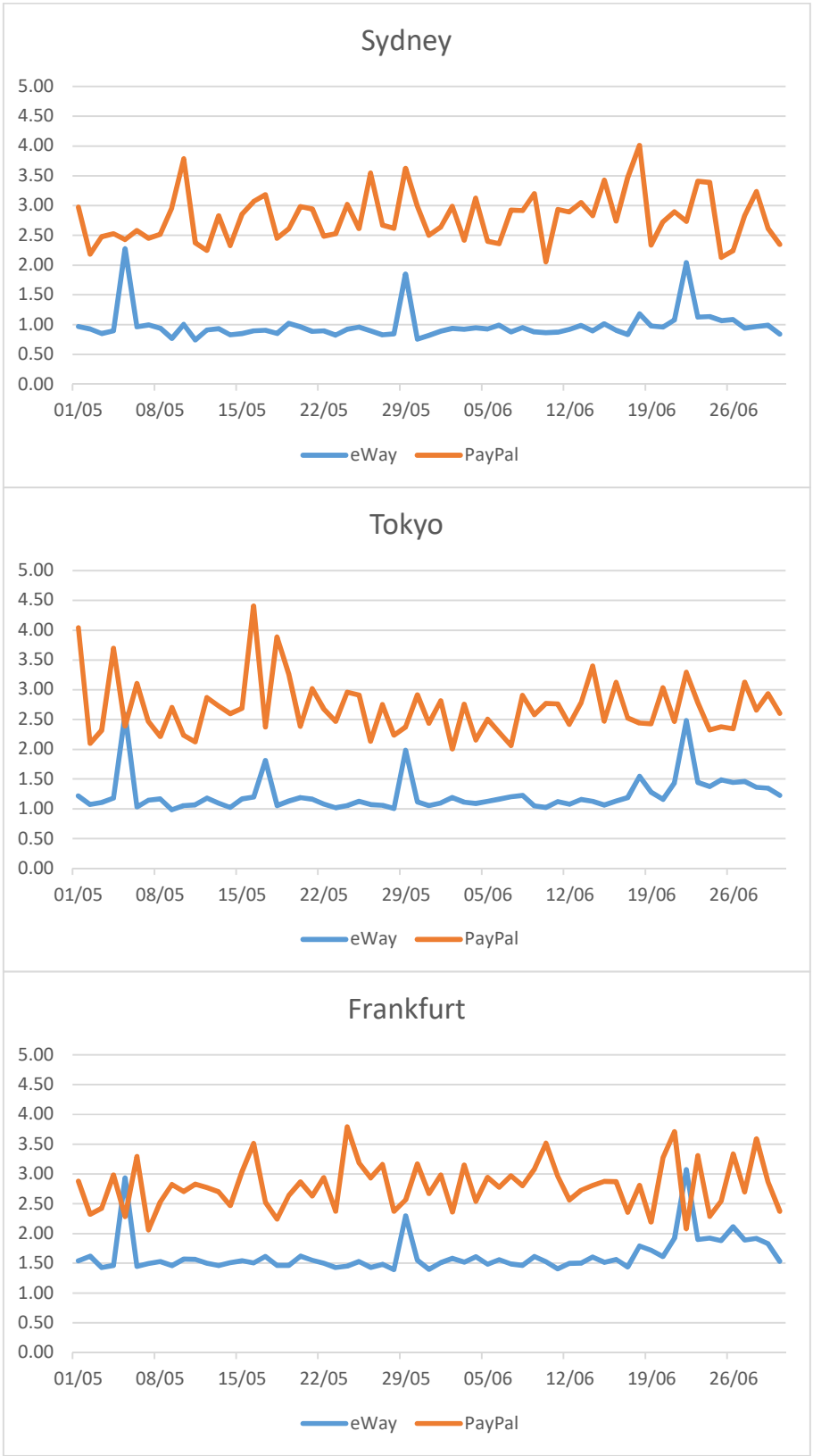
where T_e is the response time for eWay transaction, T_p is the response time for a concurrent PayPal transaction, $\overline{T_e}$ is the average response time of the eWay service and $\overline{T_p}$ is the average response time of the PayPal service. The correlation coefficient $C_{(T_e,T_p)}$ for the availability of eWay and PayPal is calculated as:

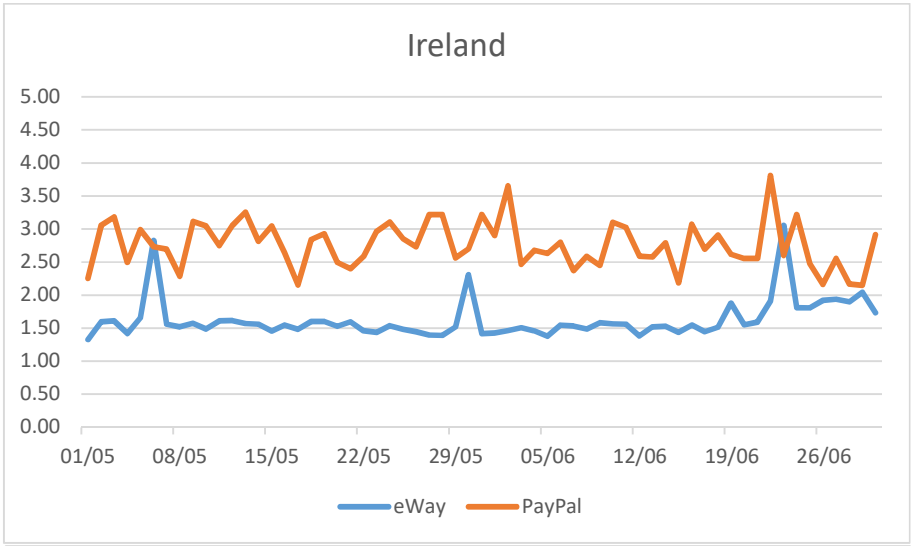
$$C_{(Ae,Ap)} = \frac{\sum(A_e - \overline{A_e})(A_p - \overline{A_p})}{\sqrt{\sum(A_e - \overline{A_e})^2 \sum(A_p - \overline{A_p})^2}} \quad (8)$$

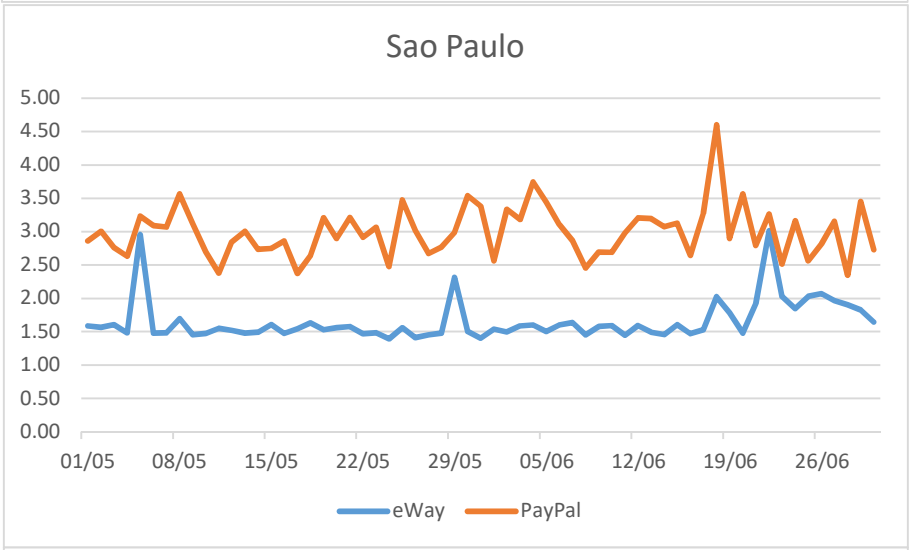
where A_e is average availability of the eWay service, A_p is average availability of the PayPal service computed for one hour, $\overline{A_e}$ is the average availability of eWay service during the monitoring period, and $\overline{A_p}$ is the average availability of PayPal service during the monitoring period.

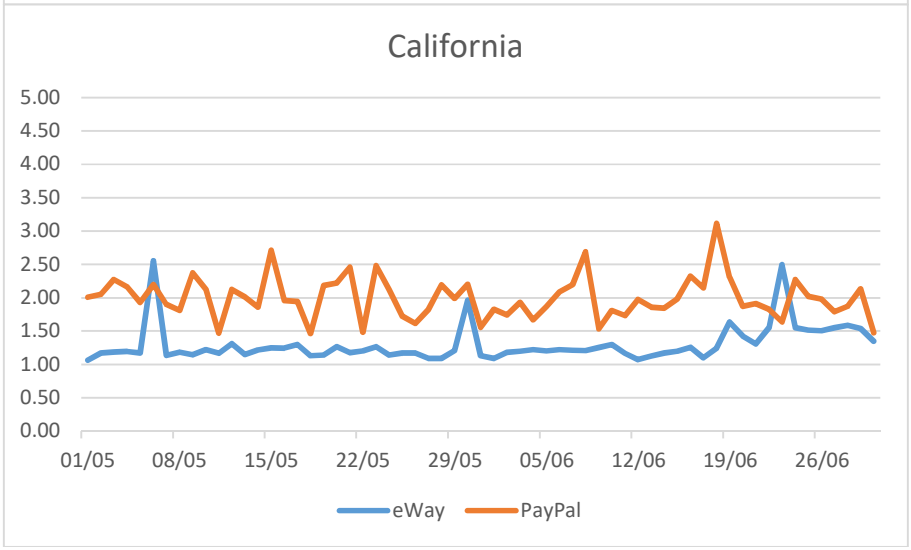
It is evident from the low correlation coefficient values in Table 2 that the underlying factors affecting response time and availability of the two payment services are mutually independent over the monitored period. As the two payment services share the same network connections, this indicates that the source of QoS variability is the service provider system, rather than the network. This implies that improved QoS values may be achievable by deploying RBFT and DFST service substitution fault tolerant strategies. I also note that in environments characterised by reliable low latency network connectivity the QoS values observed at the service consumer site will approximate those published by the service provider.


Mumbai
5.00
4.50
4.00
3.50
3.00
2.50
2.00
1.50
1.00
0.50
0.00
01/05
08/05
15/05
22/05
29/05
05/06
12/06
19/06
26/06
eWay
PayPal
Seoul
5.00
4.50
4.00
3.50
3.00
2.50
2.00
1.50
1.00
0.50
0.00
01/05
08/05
15/05
22/05
29/05
05/06
12/06
19/06
26/06
eWay
PayPal
Singapore
5.00
4.50
4.00
3.50
3.00
2.50
2.00
1.50
1.00
0.50
0.00
01/05
08/05
15/05
22/05
29/05
05/06
12/06
19/06
26/06
eWay
PayPal


Sydney
5.00
4.50
4.00
3.50
3.00
2.50
2.00
1.50
1.00
0.50
0.00
01/05
08/05
15/05
22/05
29/05
05/06
12/06
19/06
26/06
eWay
PayPal
Tokyo
5.00
4.50
4.00
3.50
3.00
2.50
2.00
1.50
1.00
0.50
0.00
01/05
08/05
15/05
22/05
29/05
05/06
12/06
19/06
26/06
eWay
PayPal
Frankfurt
5.00
4.50
4.00
3.50
3.00
2.50
2.00
1.50
1.00
0.50
0.00
01/05
08/05
15/05
22/05
29/05
05/06
12/06
19/06
26/06
eWay
PayPal


Ireland
5.00
4.50
4.00
3.50
3.00
2.50
2.00
1.50
1.00
0.50
0.00
01/05
08/05
15/05
22/05
29/05
05/06
12/06
19/06
26/06
eWay
PayPal


Sao Paulo
5.00
4.50
4.00
3.50
3.00
2.50
2.00
1.50
1.00
0.50
0.00
01/05
08/05
15/05
22/05
29/05
05/06
12/06
19/06
26/06
eWay
PayPal


California
5.00
4.50
4.00
3.50
3.00
2.50
2.00
1.50
1.00
0.50
0.00
01/05
08/05
15/05
22/05
29/05
05/06
12/06
19/06
26/06
eWay
PayPal

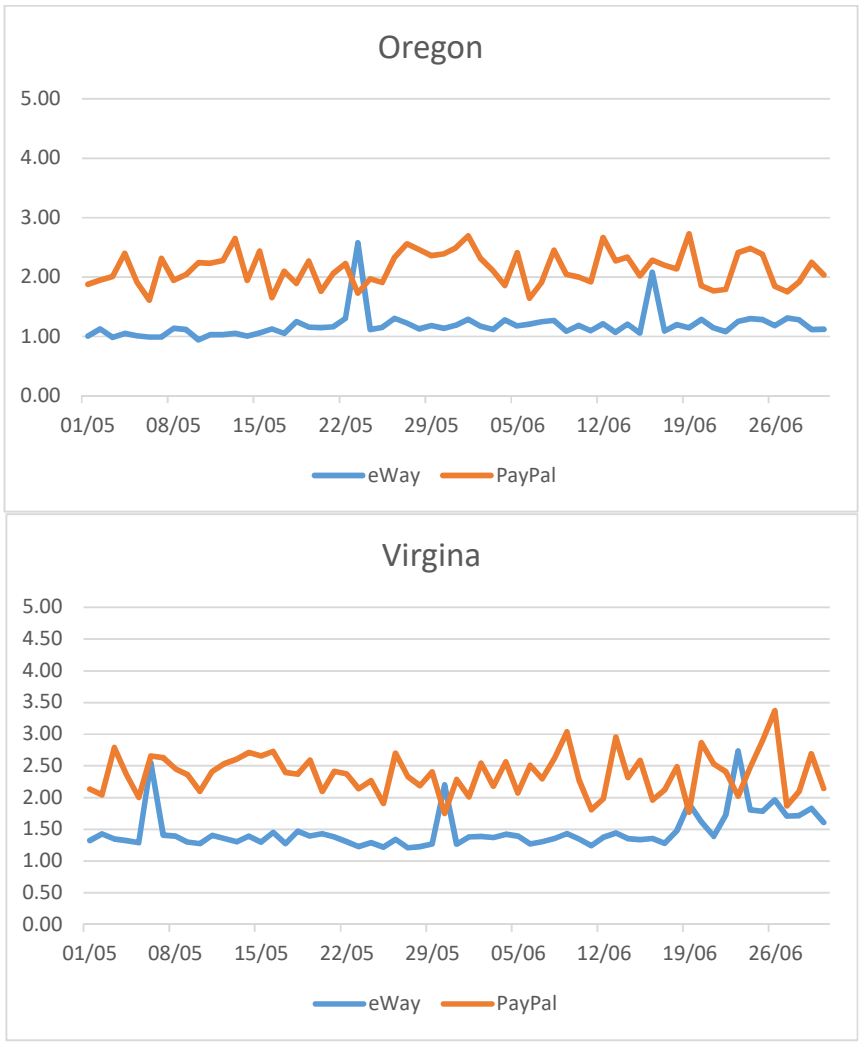
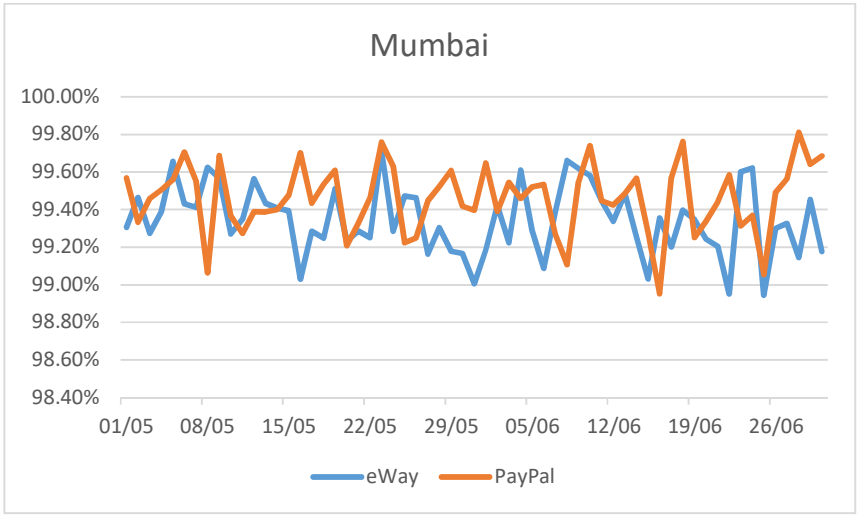
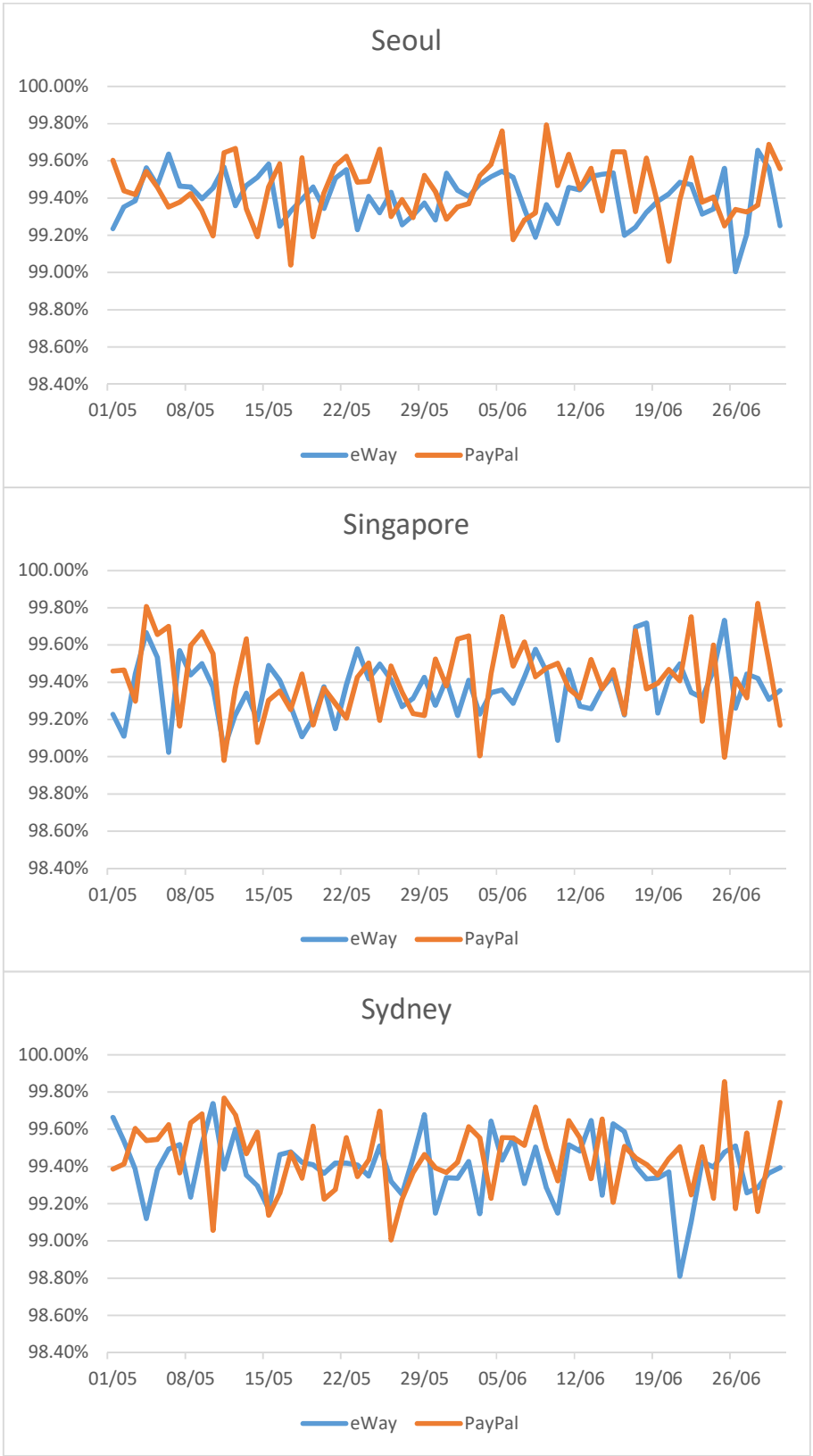
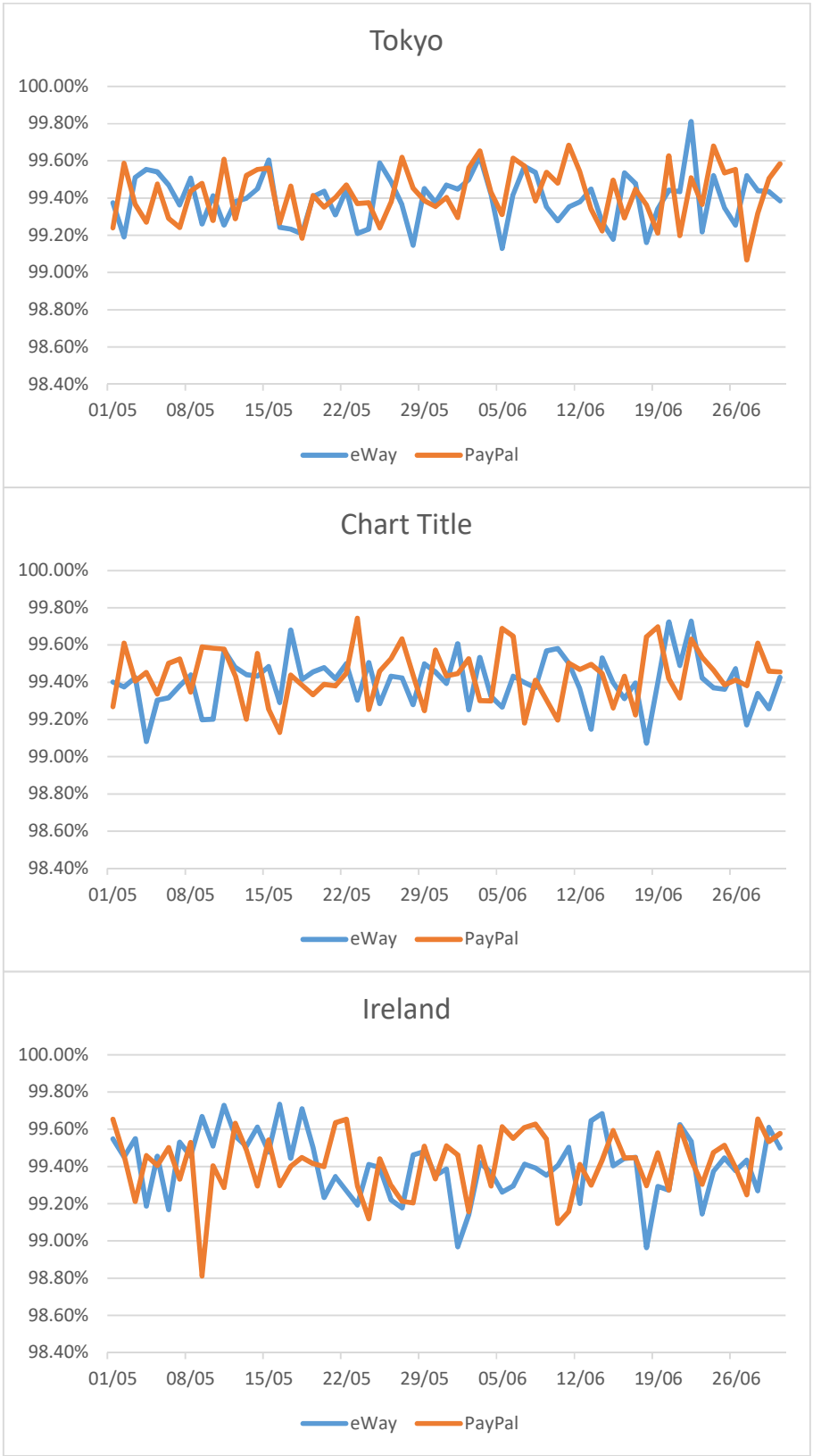
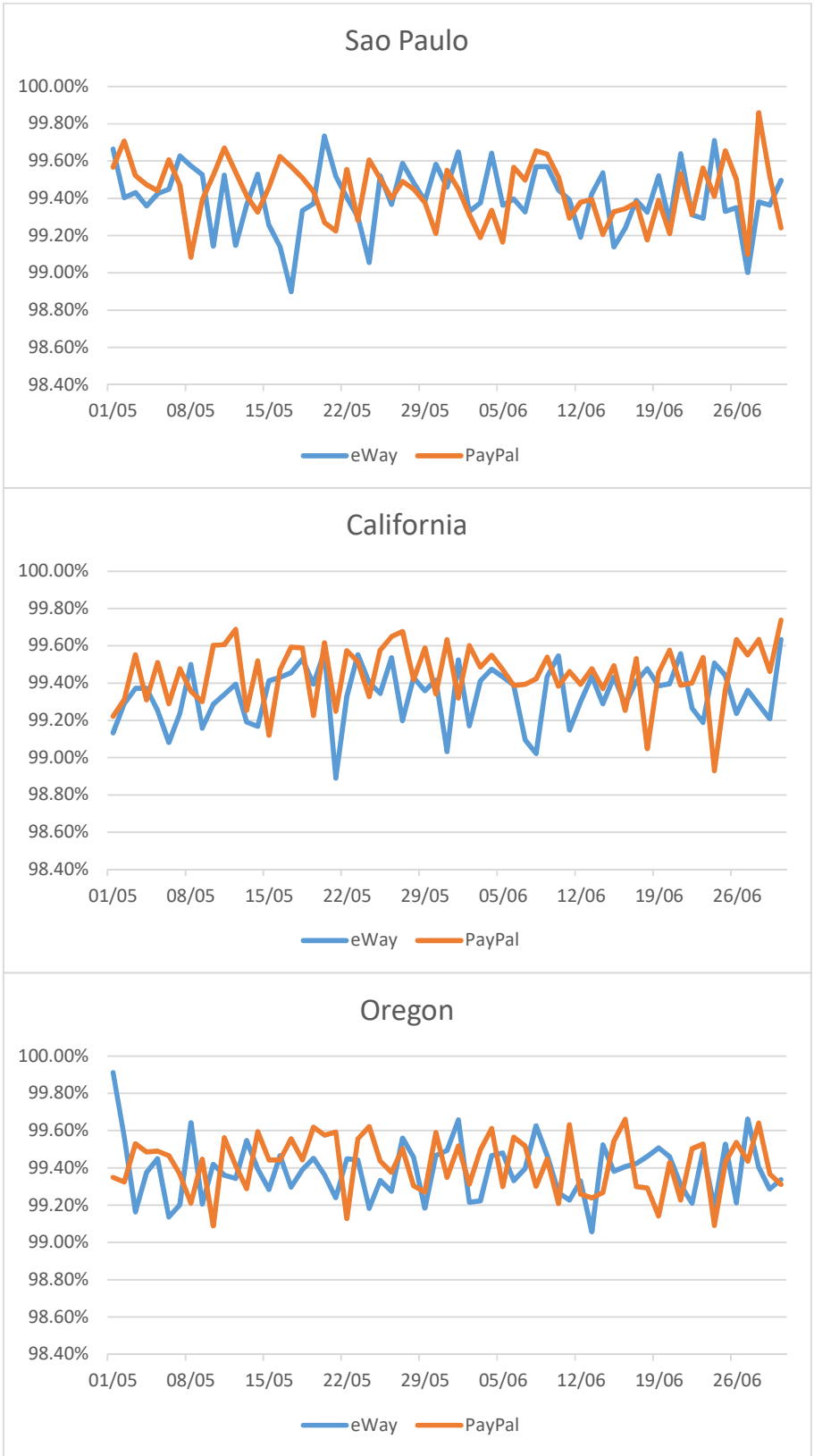


Figure 7.3. Daily average response times of eWay and PayPal services




Seoul
100.00%
99.80%
99.60%
99.40%
99.20%
99.00%
98.80%
98.60%
98.40%
01/05
08/05
15/05
22/05
29/05
05/06
12/06
19/06
26/06
eWay
PayPal
Singapore
100.00%
99.80%
99.60%
99.40%
99.20%
99.00%
98.80%
98.60%
98.40%
01/05
08/05
15/05
22/05
29/05
05/06
12/06
19/06
26/06
eWay
PayPal
Sydney
100.00%
99.80%
99.60%
99.40%
99.20%
99.00%
98.80%
98.60%
98.40%
01/05
08/05
15/05
22/05
29/05
05/06
12/06
19/06
26/06
eWay
PayPal


Tokyo
100.00%
99.80%
99.60%
99.40%
99.20%
99.00%
98.80%
98.60%
98.40%
01/05
08/05
15/05
22/05
29/05
05/06
12/06
19/06
26/06
eWay
PayPal
Chart Title
100.00%
99.80%
99.60%
99.40%
99.20%
99.00%
98.80%
98.60%
98.40%
01/05
08/05
15/05
22/05
29/05
05/06
12/06
19/06
26/06
eWay
PayPal
Ireland
100.00%
99.80%
99.60%
99.40%
99.20%
99.00%
98.80%
98.60%
98.40%
01/05
08/05
15/05
22/05
29/05
05/06
12/06
19/06
26/06
eWay
PayPal


Sao Paulo
100.00%
99.80%
99.60%
99.40%
99.20%
99.00%
98.80%
98.60%
98.40%
01/05
08/05
15/05
22/05
29/05
05/06
12/06
19/06
26/06
eWay
PayPal
California
100.00%
99.80%
99.60%
99.40%
99.20%
99.00%
98.80%
98.60%
98.40%
01/05
08/05
15/05
22/05
29/05
05/06
12/06
19/06
26/06
eWay
PayPal
Oregon
100.00%
99.80%
99.60%
99.40%
99.20%
99.00%
98.80%
98.60%
98.40%
01/05
08/05
15/05
22/05
29/05
05/06
12/06
19/06
26/06
eWay
PayPal

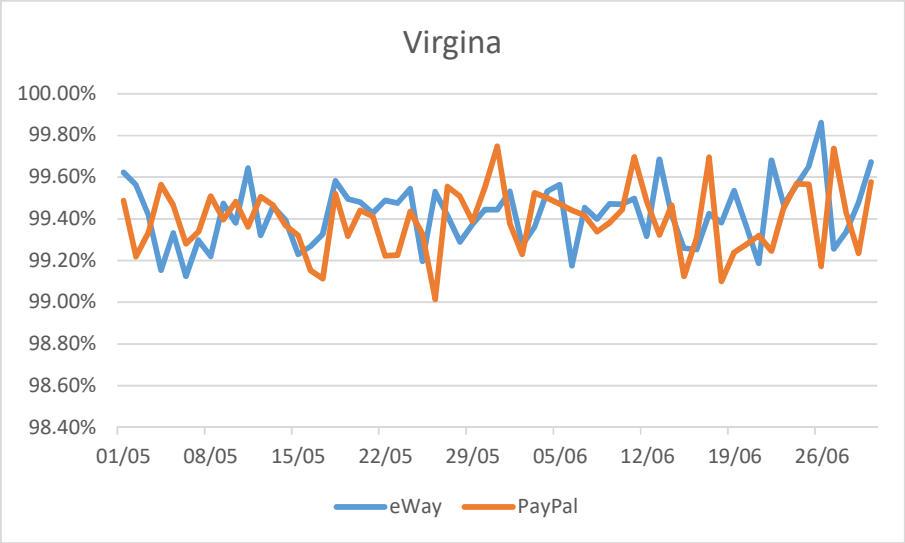


Figure 7.4. Daily average availability of eWay and PayPal services

Figure 7.3 and Figure 7.4 show the daily average response time and availability values for eWay and PayPal services during the monitored period between 1st May and 30th June 2017 for eleven geographic locations across the globe. Figure 7.3 shows that the response time of eWay services is generally better than for PayPal and that the response time of PayPal deployed in the US and Europe is better than the response times deployed in the Asia Pacific. Figure 7.4, shows that the availability of both services varies from 98.8% to 99.8% with PayPal availability slightly better than that of eWay.

7.4. Conclusion

In this chapter, I have argued that consumer-side monitoring of QoS characteristics of cloud services is essential to enable service consumers to make informed decisions about service selection at design-time, and to maintain the good run-time performance of cloud-based enterprise applications. Service consumers need to supplement QoS information published by cloud providers with data obtained independently using consumer side monitoring taking into account location-based information, as the QoS values measured at the consumer deployment site (i.e., at the site where the enterprise application is running) may vary from those published by cloud service providers.

The results obtained using AWS (Amazon Web Services) platforms deployed in eleven sites across four geographic regions to monitor eWay and PayPal payment services

indicate that both services achieved availability values above 99.9% during most of the measurement period 1st May to June 30th August 2017. It is evident from the low correlation coefficient values that the underlying factors affecting response time and availability of the two payment services are mutually independent. As the two payment services share the same network connections, this indicates that the source of QoS variability is the service provider system, rather than the network. This implies that improved QoS values may be achievable by deploying RBFT and DFST service substitution fault tolerant strategies. Using a combination of QoS information published by cloud service providers and QoS data measured at different geographic locations by service consumers improves the understanding of performance and reliability trade-offs and can facilitate the selection of more effective optimisation strategies.

In the future work, I plan to collect QoS data over an extended period of time to give more reliable estimates of service availability and response time. I also plan to make the monitoring database publicly available to cloud service consumers to enable sharing of QoS information and to promote a collaborative effort with the aim to improve the accessibility of cloud QoS information.

CHAPTER 8.

CONCLUSION AND FUTURE WORK

8.1. Conclusion

As discussed in the introduction, this thesis is concerned with a lifecycle methodology for the development of cloud service-oriented enterprise application. To examine this research topic, research approach drawing on a literature survey, action research and design science is adopted. In previous chapters, there are four contributions of the thesis that have been presented.

1. The cloud Service Consumer System Development Lifecycle (SC-SDLC) for developing cloud service-oriented enterprise application

Examination of the literature (Chapter 2) revealed that relatively little research on the methodologies to develop cloud-based enterprise applications had been done so far. Therefore, in Chapter 4, Cloud Service Consumer SDLC (SC-SDLC) for developing a cloud enterprise application is proposed. The SC-SDLC is the key contribution of this thesis. It contains five phases: requirements specification, service identification, service integration, service monitoring, and optimisation. The SC-SDLC activities are both to the development methodologies and to the management of cloud services during runtime. It focuses on addressing a number of challenges of using cloud services in enterprise applications such as service selection, service incident management, and service monitoring.

To evaluate the SC-SDLC, a real-world project of developing a hospital management application is used. The lifecycle methodology was demonstrated and enhanced during various stages of this project from requirements analysis to deployment and monitoring. The development team has given a positive feedback for SC-SDLC. Comparing to the traditional methods that were used by the development team, SC-SDLC provides the clear guidelines to develop the cloud service-oriented enterprise application. The detailed activities of lifecycle also assist the team in managing cloud services and addressing the challenges of using cloud services in an enterprise application. The team also give few suggestions to improve the lifecycle, for example, the SC-SDLC should provide the extension of principles and best practices to management activities throughout the lifecycle.

2. The cloud Service Consumer Framework (SCF) that is a comprehensive

supporting tool for SC-SDLC

In existing literature, there are some commercial products and research efforts focusing the tools for management of cloud services such as server repository, cloud service integration framework, service monitoring framework. These works just address a specific challenge of cloud services. A comprehensive framework that performs all these functionalities required for cloud service management from the consumer perspective has not yet appeared in the literature. The existing literature also has lacked a cloud framework that adequately addresses all challenges of cloud services such as service selection, service incident management, service integration and service monitoring. In Chapter 5, I describe the implementation of Service Consumer Framework (SCF) that is used to manage cloud services and to support the entire SC-SDLC. The SCF supports the development of the cloud service-oriented enterprise application by providing many features: requirements management, service portfolio management, and service integration. The framework also provides the capability of failover and cloud service runtime monitoring to address the problem of cloud service incidents and to optimise the use of cloud services in the enterprise application.

3. The cloud service failover strategies to improve the enterprise application reliability

Although cloud service disruptions and service changes create many problems for management of cloud enterprise applications, comparatively little research has been conducted on these problems from service consumer perspective. The existing techniques of cloud service failover to handle service disruptions mostly focus on cloud infrastructure services, the failover for cloud software services has not yet been adequately investigated. The works on the management of service changes from service consumer perspective are almost missing in the literature. In Chapter 6, I proposed the failover strategies that are applied to various cloud models (IaaS, PaaS, SaaS). The cross-provider failover strategy is not only to handle service disruptions but also to address the problems of cloud service change. These different fault tolerance features of the SCF that are used to configure the cloud service failover are implemented. I have estimated the theoretical improvements in service availability that can be achieved using these strategies and compared these values to experimentally obtained results. The

experimental results obtained using the SCF framework are consistent with theoretical predictions and indicate significant improvements in service availability when compared to using cloud services directly.

4. The multi-site monitoring model for cloud service selection and application optimization.

Monitoring of cloud service performance from the consumer side presents an important and challenging research problem because the performance of cloud service is affected by application contexts such as location or network connection. Although some research work on monitoring of QoS characteristics of cloud services is available in the literature, there is currently lack of detailed information about the assessment of the run-time behaviour of cloud services that includes location-based QoS information. Therefore, in Chapter 7, I focus on improving the estimates of availability and response time of cloud services by introducing location-based QoS information. I set up a simulation environment to monitor QoS characteristics of eWay and PayPal services across eleven locations in four geographical regions to obtain a more accurate estimate of response time and availability for specific deployment locations of consumer enterprise applications. I collect the QoS information independently of the information published by cloud service providers by recording payment transaction log data in a monitoring database. This multi-site monitoring is a model in which many cloud service consumers can cooperatively contribute the service runtime logs to analysis the cloud service performance in various locations. Besides monitoring of currently using cloud services, service consumers also know the performance of the other services available in their locations to consider service migration if another service offers better performance, more reliable or lower price.

Despite the above implications and contributions of the research, this thesis has few limitations. (1) Although the SC-SDLC is developed and evaluated through a real-world case study, the lifecycle needs to be applied to different case studies, for example, cloud migration project, application enhancement project, or out-sourcing project. It also needs to apply for the project from various domains with different service consumer expectations to indicates the advantages and advantages of the lifecycle. (2) Some features of SCF (for example, the unified interfaces of cloud services) are designed for some types of cloud services. The generalisations, therefore, have not claimed for all

types of cloud services because the standardisation of cloud APIs is not mainly investigated in this thesis.

8.2. Future Work

Based on the suggestions from the case study, the next step of this research will consider developing a set of best practices and principles for the SC-SDLC. I am also applying the SC-SDLC for another real-world project, named Silent Bull[19], that is a cloud application for assisting a mobile servicing business manage scheduling of staff as well as customer service delivery. This project is a different type of project from the FMP case study because the end-users are low IT skills and expect an easy-to-use and user-friendly enterprise application. Applying SC-SDLC in different projects to gain the feedback and suggestions that help to improve the lifecycle and its activities.

For management of cloud services, an approach for forecasting future performance on the basis of analyzing the historical data of cloud services need to be investigated. The current monitoring approaches are able to generate detailed QoS history which contains several QoS attributes recorded at regular intervals. The future research direction is to analyse this data to find repeating patterns and trends and to forecast future cloud service performance issues and/or security risks. This also provides an approach to automatically warn the cloud service consumers of significant changes and trends in the cloud service performance, so they can take pre-emptive measures in time to avoid service disruption and degradation. The development of visual workflow engine with the graphical user interface for design workflow is also a future work.

An investigation of cloud service interconnections and dependencies is also a research direction. In practice, a cloud service can be implemented or deployed using a number of cloud services. For example, Amazon Web Service (AWS) provides the infrastructure service for a large range of cloud services such as Dropbox and Salesforce. The availability of Dropbox and Salesforce are depended on the availability of AWS. The

[19] Company website: http://www.silentbull.com.au/
Application test url: http://13.55.161.47/
Username: thai Password: 123456

correlation coefficient analysis to observe the relationship among cloud services will help service consumers develop better monitoring strategies for improving enterprise application reliability.

Bibliography

ACETO, G., BOTTA, A., DE DONATO, W. & PESCAPÈ, A. 2013. Cloud monitoring: A survey. *Computer Networks,* 57**,** 2093-2115.

ADAMS, L. A. & COURTNEY, J. F. Achieving relevance in IS research via the DAGS framework. System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on, 2004. IEEE, 10 pp.

ALBOGHDADY, S., WINTER, S., TAHA, A., ZHANG, H. & SURI, N. C'MON: Monitoring the Compliance of Cloud Services to Contracted Properties. Proceedings of the 12th International Conference on Availability, Reliability and Security, 2017. ACM, 36.

ALJAWARNEH, S. A., ALAWNEH, A. & JARADAT, R. 2017. Cloud security engineering: Early stages of SDLC. *Future Generation Computer Systems,* 74**,** 385-392.

ALMALKI, J. & SHEN, H. A Lightweight Solution to Version Incompatibility in Service-Oriented Revision Control Systems. Proceedings of the ASWEC 2015 24th Australasian Software Engineering Conference, 2015. ACM, 59-63.

AMMANN, P. & OFFUTT, J. 2008. *Introduction to software testing*, Cambridge University Press.

ANDRIKOPOULOS, V., BENBERNOU, S. & PAPAZOGLOU, M. P. 2012. On the Evolution of Services. *IEEE Transactions on Software Engineering,* 38**,** 609-628.

ANWAR, A., SAILER, A., KOCHUT, A. & BUTT, A. R. Anatomy of cloud monitoring and metering: A case study and open problems. Proceedings of the 6th Asia-Pacific Workshop on Systems, 2015. ACM, 6.

ARCHER, L. B. 1964. *Systematic method for designers*, Council of Industrial Design.

ARMBRUST, M., FOX, A., GRIFTH, R., JOSEPH, A., KATZ, R., KONWINSKI, A., LEE, G., PATTERSON, D., RABKIN, A. & STOICA, I. 2009. Above the clouds: A berkeley view of cloud computing. *Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, CA*.

ARUN, S., CHANDRASEKARAN, A. & PRAKASH, P. 2017. CSIS: Cloud Service Identification System. *International Journal of Electrical and Computer Engineering (IJECE),* 7**,** 513-520.

BABAR, M., UR RAHMAN, A. & ARIF, F. 2017. Cloud Computing Development Life Cycle Model (CCDLC). *In:* FERREIRA, J. & ALAM, M. (eds.) *Future Intelligent Vehicular Technologies: First International Conference, Future 5V 2016, Porto, Portugal, September 15, 2016, Revised Selected Papers.* Cham: Springer International Publishing.

BAKSHI, K. & BESER, L. 2016. Cloud Reference Frameworks. *Encyclopedia of Cloud Computing***,** 71-88.

BARYANNIS, G., GAREFALAKIS, P., KRITIKOS, K., MAGOUTIS, K., PAPAIOANNOU, A., PLEXOUSAKIS, D. & ZEGINIS, C. Lifecycle management of service-based applications on multi-clouds: a research roadmap. Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds, 2013. ACM, 13-20.

BAUDE, F., FILALI, I., HUET, F., LEGRAND, V., MATHIAS, E., MERLE, P., RUZ, C., KRUMMENACHER, R., SIMPERL, E. & HAMMERLING, C. ESB federation for large-scale SOA. Proceedings of the 2010 ACM Symposium on Applied Computing, 2010. ACM, 2459-2466.

BAUER, T., BUCHWALD, S., TIEDEKEN, J. & REICHERT, M. 2015. A SOA Repository with Advanced Analysis Capabilities-Improving the Maintenance and Flexibility of Service-Oriented Applications.

BERNSTEIN, P. A. & HAAS, L. M. 2008. Information integration in the enterprise. *Communications of the ACM,* 51**,** 72-79.

BREITER, G. & BEHRENDT, M. 2009. Life cycle and characteristics of services in the world of cloud computing. *IBM Journal of Research and Development,* 53**,** 3:1-3:8.

BRENNER, M. Classifying ITIL Processes; A Taxonomy under Tool Support Aspects. Business-Driven IT Management, 2006. BDIM '06. The First IEEE/IFIP International Workshop on, 07-07 April 2006 2006. 19-28.

BUTTERFIELD, R., MAKSUTI, S., TAUBER, M., WAGNER, C. & BICAKU, A. Towards Modelling a Cloud Application's Life Cycle. 6th International Conference on Cloud Computing and Services, 2016.

BYGSTAD, B. & AANBY, H.-P. 2010. ICT infrastructure for innovation: A case study of the enterprise service bus approach. *Information systems frontiers,* 12**,** 257-265.

CALERO, J. M. A. & AGUADO, J. G. 2015. Comparative analysis of architectures for monitoring cloud computing infrastructures. *Future Generation Computer Systems,* 47**,** 16-30.

CARDOSO, A. & SIMÕES, P. CloudComputing and Security. European Conference on Information Warfare and Security, 2012. Academic Conferences International Limited, 70.

CHAKRABORTY, A., BAOWALY, M. K., AREFIN, A. & BAHAR, A. N. 2012. The role of requirement engineering in software development life cycle. *Journal of emerging trends in computing and information sciences,* 3**,** 723-729.

CHAPPELL, D. 2004. *Enterprise service bus*, " O'Reilly Media, Inc.".

CHAUHAN, N. S. & SAXENA, A. 2013. A green software development life cycle for cloud computing. *IT Professional,* 15**,** 28-34.

CHEMUTURI, M. 2013. Establishment of Requirements. *Requirements Engineering and Management for Software Development Projects.* New York, NY: Springer New York.

CHEN, L. 2012. Integrating Cloud Computing Services Using Enterprise Service Bus (ESB). *Business and Management Research,* 1**,** p26.

CHHABI RANI, P., RAJIB, M. & BIBUDHENDU, P. 2017. Software Development Methodology for Cloud Computing and Its Impact. *Resource Management and Efficiency in Cloud Computing Environments.* Hershey, PA, USA: IGI Global.

CIUFFOLETTI, A. 2016. Application level interface for a cloud monitoring service. *Computer Standards & Interfaces,* 46**,** 15-22.

COLE, R., PURAO, S., ROSSI, M. & SEIN, M. 2005. Being proactive: where action research meets design research.

CONSORTIUM, I. G. L. 2009. IMS Service Oriented Architecture (SOA). Adoption of Service Oriented Architecture for Enterprise Systems in Education: Recommended Practices. Technical Report,< http://www. imsglobal. org/soa/soawpv1p0/imsSOAWhite Paper_v1p0. html>, 2009.[Last accessed on June 2010.].

DENG, X. & XING, C. A QoS-oriented optimization model for web service group. Computer and Information Science, 2009. ICIS 2009. Eighth IEEE/ACIS International Conference on, 2009. IEEE, 903-909.

DICK, J., HULL, E. & JACKSON, K. 2017. Writing and Reviewing Requirements. *Requirements Engineering.* Cham: Springer International Publishing.

EEKELS, J. & ROOZENBURG, N. F. 1991. A methodological comparison of the structures of

scientific research and engineering design: their similarities and differences. *Design Studies,* 12**,** 197-203.

FALLON, L. & O'SULLIVAN, D. 2014. The aesop approach for semantic-based end-user service optimization. *IEEE Transactions on Network and Service Management,* 11**,** 220-234.

FARRELL, K. 2011. *Cloud Lifecycle Management: Managing Cloud Services from Request to Retirement* [Online]. BMC Software. Available: http://www.bmc.com/blogs/hybrid-cloud-delivery-managing-cloud-services-from- request-to-retirement [Accessed 03/02/ 2015].

FEUERLICHT, G. Understanding service reusability. International Conference Systems Integration, 2007. Department of Information Technologies and Czech Society for Systems Integration.

FEUERLICHT, G. & THAI TRAN, H. Adapting service development life-cycle for cloud. Proceedings of the 17th International Conference on Enterprise Information Systems-Volume 3, 2015. SCITEPRESS-Science and Technology Publications, Lda, 366-371.

FEUERLICHT, G. & TRAN, H. T. Enterprise Application Management in Cloud Computing Context. Proceedings of the 16th International Conference on Information Integration and Web-based Applications & Services, 2014a. ACM, 517-523.

FEUERLICHT, G. & TRAN, H. T. 2014b. *Service consumer framework: Managing service evolution from a consumer perspective*.

FIELD, L., MEMON, S., MÁRTON, I. & SZIGETI, G. 2014. The EMI Registry: Discovering Services in a Federated World. *Journal of Grid Computing,* 12**,** 29-40.

GAMA, N., SOUSA, P. & DA SILVA, M. 2013. Integrating Enterprise Architecture and IT Service Management. *In:* LINGER, H., FISHER, J., BARNDEN, A., BARRY, C., LANG, M. & SCHNEIDER, C. (eds.) *Building Sustainable Information Systems.* Springer US.

GARG, S. K., VERSTEEG, S. & BUYYA, R. 2013. A framework for ranking of cloud computing services. *Future Generation Computer Systems,* 29**,** 1012-1023.

GARTNER 2017. Gartner Says Worldwide Public Cloud Services Market to Grow 18 Percent in 2017.

GHAMRY, A. M., ALKALBANI, A. M., TRAN, V., TSAI, Y.-C., HOANG, M. L. & HUSSAIN, F. K. Towards a Public Cloud Services Registry. International Conference on Web Information Systems Engineering, 2017. Springer, 290-295.

GREGOR, S. & HEVNER, A. R. 2013. Positioning and presenting design science research for maximum impact. *MIS quarterly,* 37**,** 337-355.

GUHA, R. 2013. Impact of Semantic Web and Cloud Computing Platform on Software Engineering. *In:* MAHMOOD, Z. & SAEED, S. (eds.) *Software Engineering Frameworks for the Cloud Computing Paradigm.* London: Springer London.

HAJLAOUI, J. E., OMRI, M. N., BENSLIMANE, D. & BARHAMGI, M. QoS Based Framework for Configurable IaaS Cloud Services Discovery. Web Services (ICWS), 2017 IEEE International Conference on, 2017. IEEE, 460-467.

HEVNER, A. & CHATTERJEE, S. 2010. *Design research in information systems: theory and practice*, Springer Science & Business Media.

HEVNER, A. R. 2007. A three cycle view of design science research. *Scandinavian journal of information systems,* 19**,** 4.

HOCHSTEIN, A., ZARNEKOW, R. & BRENNER, W. ITIL as common practice reference model for IT service management: formal assessment and implications for practice. e-Technology, e-Commerce and e-Service, 2005. EEE '05. Proceedings. The 2005 IEEE International Conference on, 29 March-1 April 2005 2005. 704-710.

IEEE 1990. IEEE Standard Glossary of Software Engineering Terminology. *IEEE Std 610.12-1990***,** 1-84.

ITIL. 2014. *What is ITIL* [Online]. ITIL. Available: http://www.itil.org/en/vomkennen/itil/ueberblick/index.php [Accessed 2014].

IZZAT, A. 2013. Software Development Methodologies for Cloud Computing. *Software Development Techniques for Constructive Information Systems Design.* Hershey, PA, USA: IGI Global.

JAGLI, D. & YEDDU, S. 2017. CloudSDLC: Cloud Software Development Life Cycle. *International Journal of Computer Applications,* 168.

JANSEN, W. & GRANCE, T. 2011. Sp 800-144. guidelines on security and privacy in public cloud computing.

JOHNSTON, A. 2009. *Interfaces for musical expression based on simulated physical models.* Citeseer.

JOSHI, K., FININ, T. & YESHA, Y. Integrated Lifecycle of IT Services in A Cloud Environment. The 3rd International Conference on the Virtual Computing Initiative (ICVCI), 2009 USA.

JOSHI, K., FININ, T. & YESHA, Y. 2016. Automating Cloud Services Lifecycle Through Semantic Technologies. US Patent 20,160,149,769.

JOSHI, K. P., YESHA, Y. & FININ, T. 2014. Automating cloud services life cycle through semantic technologies. *IEEE Transactions on Services Computing,* 7**,** 109-122.

JOSHI, N. 2015. Issues and Approaches towards Effective Service Versioning. *Quality Management Practices for Global Excellence,* 1**,** 316.

KARKOŠKOVÁ, S. & FEUERLICHT, G. 2014. ITIL AS A FRAMEWORK FOR MANAGEMENT OF CLOUD SERVICES. *International Journal of Research in Engineering and Technology,* 3**,** 17.

KASHFI, H. 2017. Software Engineering Challenges in Cloud Environment: Software Development Lifecycle Perspective.

KOHLBORN, T., KORTHAUS, A. & ROSEMANN, M. Business and Software Service Lifecycle Management. Enterprise Distributed Object Computing Conference, 2009. EDOC '09. IEEE International, 1-4 Sept. 2009 2009. 87-96.

KOTHARI, C. R. 2004. *Research methodology: Methods and techniques*, New Age International.

KRISHNA, R. & JAYAKRISHNAN, R. 2013. Impact of Cloud Services on Software Development Life Cycle. *In:* MAHMOOD, Z. & SAEED, S. (eds.) *Software Engineering Frameworks for the Cloud Computing Paradigm.* London: Springer London.

KUECHLER, B. & VAISHNAVI, V. 2008. On theory development in design science research: anatomy of a research project. *European Journal of Information Systems,* 17**,** 489-504.

KUMAR, N., ZADGAONKAR, A. & SHUKLA, A. 2013. Evolving a new software development life cycle model SDLC-2013 with client satisfaction. *International Journal of Soft Computing and Engineering (IJSCE),* 3**,** 2231-2307.

LAKSHMI, H. & MOHANTY, H. RDBMS for Service Repository and Composition. The 4th International Conference on Advanced Computing (ICoAC), 2012. 13-15.

LEAU, Y. B., LOO, W. K., THAM, W. Y. & TAN, S. F. Software development life cycle AGILE vs traditional approaches. International Conference on Information and Network Technology, 2012. 162-167.

LEE, C., LEHOEZKY, J., RAJKUMAR, R. & SIEWIOREK, D. On quality of service optimization with discrete QoS options. Real-Time Technology and Applications Symposium, 1999. Proceedings of the Fifth IEEE, 1999. IEEE, 276-286.

LI, L., YE, F. & HUANG, Q. 2017. The Research of QoS Monitoring-Based Cloud Service Selection. *In:* BAROLLI, L., ZHANG, M. & WANG, X. A. (eds.) *Advances in Internetworking, Data & Web Technologies: The 5th International Conference on*

Emerging Internetworking, Data & Web Technologies (EIDWT-2017). Cham: Springer International Publishing.

LIN, M., YAO, Z. & HUANG, T. 2016. A hybrid push protocol for resource monitoring in cloud computing platforms. *Optik - International Journal for Light and Electron Optics,* 127**,** 2007-2011.

LIU, F., TONG, J., MAO, J., BOHN, R., MESSINA, J., BADGER, L. & LEAF, D. 2011. NIST cloud computing reference architecture. *NIST special publication,* 500**,** 292.

LU, W., HU, X., WANG, S. & LI, X. 2014. A multi-criteria QoS-aware trust service composition algorithm in cloud computing environments. *International Journal of Grid and Distributed Computing,* 7**,** 77-88.

MARCH, S. T. & SMITH, G. F. 1995. Design and natural science research on information technology. *Decision support systems,* 15**,** 251-266.

MCNAUGHTON, B., RAY, P. & LEWIS, L. 2010. Designing an evaluation framework for IT service management. *Information & Management,* 47**,** 219-225.

MEIER, J., HILL, D., HOMER, A., JASON, T., BANSODE, P., WALL, L., BOUCHER JR, R. & BOGAWAT, A. 2009. Microsoft application architecture guide. *Patterns and Practices, 2nd edition, Microsoft Corporation*.

MELER, R. C., NEWELL, W. T. & PAXER, H. L. 1969. Simulation in business and economics. *Academy of Management Journal (pre-1986),* 12**,** 392.

MELL, P. & GRANCE, T. 2011. The NIST definition of cloud computing.

MENGE, F. Enterprise service bus. Free and open source software conference, 2007. 1-6.

MICROSOFT. 2016. *CORREL function* [Online]. Microsoft. Available: https://support.office.com/en-us/article/CORREL-function-995dcef7-0c0a-4bed-a3fb-239d7b68ca92 [Accessed 22 August 2016].

MICROSOFT. 2017. *What are public, private and hybrid clouds?* [Online]. Available: https://azure.microsoft.com/en-au/overview/what-are-private-public-hybrid-clouds/ [Accessed 10/09/2017].

MONTES, J., SÁNCHEZ, A., MEMISHI, B., PÉREZ, M. S. & ANTONIU, G. 2013. GMonE: A complete approach to cloud monitoring. *Future Generation Computer Systems,* 29**,** 2026-2040.

MUPPALLA, A. K., PRAMOD, N. & SRINIVASA, K. G. 2013. Efficient Practices and Frameworks for Cloud-Based Application Development. *In:* MAHMOOD, Z. & SAEED, S. (eds.) *Software Engineering Frameworks for the Cloud Computing Paradigm.* London: Springer London.

MWANSA, G. & MNKANDLA, E. Migrating agile development into the cloud computing environment. Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on, 2014. IEEE, 818-825.

NAVINKUMAR, R. & RAGHUL, M. 2016. QOS RANKING PREDICTION FOR CLOUD SERVICE.

NIEVES, M. 2014. Best practice in the cloud: an introduction. *Retrieved,* 10**,** 2014.

NOOR, T. H., SHENG, Q. Z., NGU, A. H. & DUSTDAR, S. 2014. Analysis of web-scale cloud services. *IEEE Internet Computing,* 18**,** 55-61.

NUNAMAKER JR, J. F., CHEN, M. & PURDIN, T. D. 1990. Systems development in information systems research. *Journal of management information systems,* 7**,** 89-106.

PAETSCH, F., EBERLEIN, A. & MAURER, F. Requirements engineering and agile software development. Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on, 2003. IEEE, 308-313.

PANIGRAHI, C. R., MALL, R. & PATI, B. 2016. Software Development Methodology for Cloud Computing and Its Impact. *Resource Management and Efficiency in Cloud Computing Environments*, 286.

PAPAZOGLOU, M. 2008. Web Services Development Lifecycle. *Web services: principles and technology.* Pearson Education.

PAPAZOGLOU, M. P. Service-oriented computing: Concepts, characteristics and directions. Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on, 2003. IEEE, 3-12.

PAPAZOGLOU, M. P., ANDRIKOPOULOS, V. & BENBERNOU, S. 2011. Managing Evolving Services. *IEEE Software,* 28**,** 49-55.

PAPAZOGLOU, M. P. & HEUVEL, W.-J. V. D. 2007. Business process development life cycle methodology. *Commun. ACM,* 50**,** 79-85.

PEFFERS, K., TUUNANEN, T., ROTHENBERGER, M. A. & CHATTERJEE, S. 2007. A design science research methodology for information systems research. *Journal of management information systems,* 24**,** 45-77.

POT'VIN, K., AKELA, A., ATIL, G., CURTIS, B., GORBACHEV, A., LITCHFIELD, N., NELSON, L. & SHARMAN, P. 2013. Cloud Lifecycle Management. *Expert Oracle Enterprise Manager 12c.* Apress.

POVEDANO-MOLINA, J., LOPEZ-VEGA, J. M., LOPEZ-SOLER, J. M., CORRADI, A. & FOSCHINI, L. 2013. DARGOS: A highly adaptable and scalable monitoring architecture for multi-tenant Clouds. *Future Generation Computer Systems,* 29**,** 2041-2056.

PROGRAMMABLEWEB 2017.

QU, L., WANG, Y. & ORGUN, M. A. Cloud service selection based on the aggregation of user feedback and quantitative performance assessment. Services computing (scc), 2013 ieee international conference on, 2013. IEEE, 152-159.

QU, L., WANG, Y., ORGUN, M. A., LIU, L. & BOUGUETTAYA, A. Context-aware cloud service selection based on comparison and aggregation of user subjective assessment and objective performance assessment. Web Services (ICWS), 2014 IEEE International Conference on, 2014. IEEE, 81-88.

RACKSPACE. 2017. *What is a Virtual Private Cloud?* [Online]. Available: https://www.rackspace.com/en-au/library/what-is-virtual-private-cloud [Accessed].

REDDY, C. M. & NALINI, N. FT2R2Cloud: Fault tolerance using time-out and retransmission of requests for cloud applications. Advances in Electronics, Computers and Communications (ICAECC), 2014 International Conference on, 10-11 Oct. 2014 2014. 1-4.

REHMAN, Z.-U., HUSSAIN, O. K. & HUSSAIN, F. K. 2015. User-side cloud service management: State-of-the-art and future directions. *Journal of Network and Computer Applications,* 55**,** 108-122.

REPSCHLAEGER, J., ZARNEKOW, R., WIND, S. & TUROWSKI, K. Cloud Requirement Framework: Requirements and Evaluation Criteria to Adopt Cloud solutions. ECIS, 2012. 42.

RIMAL, B. P., JUKAN, A., KATSAROS, D. & GOELEVEN, Y. 2011. Architectural requirements for cloud computing systems: an enterprise cloud approach. *Journal of Grid Computing,* 9**,** 3-26.

ROCHA, L. A. 2013. Development of Cloud Applications in Hybrid Clouds with Support for Multi-scheduling. *In:* MAHMOOD, Z. & SAEED, S. (eds.) *Software Engineering Frameworks for the Cloud Computing Paradigm.* London: Springer London.

ROSEMANN, M., FIELT, E., KOHLBORN, T. & KORTHAUS, A. 2009. Business Service

Management.

ROSSI, M. & SEIN, M. K. 2003. Design research workshop: a proactive research approach. *Presentation delivered at IRIS,* 26**,** 9-12.

ROTEM, R., ZELOVICH, A. & FRIEDRICH, G. 2016. Cloud services discovery and monitoring. Google Patents.

RUZ, C., BAUDE, F., SAUVAN, B., MOS, A. & BOULZE, A. Flexible SOA Lifecycle on the Cloud Using SCA. Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2011 15th IEEE International, Aug. 29 2011-Sept. 2 2011 2011. 275-282.

SAFY, F. Z., EL-RAMLY, M. & SALAH, A. Runtime Monitoring of SOA Applications: Importance, Implementations and Challenges. Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on, 25-28 March 2013 2013. 315-319.

SCHMIDT, M. T., HUTCHISON, B., LAMBROS, P. & PHIPPEN, R. 2005. The Enterprise Service Bus: Making service-oriented architecture real. *IBM Systems Journal,* 44**,** 781-797.

SCHMIDT, R. Conceptualisation and lifecycle of cloud based information systems. Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2012 IEEE 16th International, 2012. IEEE, 104-113.

SCROGGINS, R. 2014. SDLC and development methodologies. *Global Journal of Computer Science and Technology*.

SEIN, M. K., HENFRIDSSON, O., PURAO, S., ROSSI, M. & LINDGREN, R. 2011. Action design research. *MIS quarterly***,** 37-56.

SHETTY, J. & D'MELLO, D. A. Repository Design Strategies and Discovery Techniques for Cloud Computing. 2013 International Conference on Green Computing, Communication and Conservation of Energy (ICGCE), 2013. IEEE, 761-766.

SIMON, H. A. 1996. *The sciences of the artificial*, MIT Press.

SØRENSEN, C. 2002. *This is not an article: Just some thoughts on how to write one*, LSE, Department of Information Systems.

SUNDARESWARAN, S., SQUICCIARINI, A. & LIN, D. A Brokerage-Based Approach for Cloud Service Selection. Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on, 24-29 June 2012 2012. 558-565.

SUNEJA, S., ISCI, C., KOLLER, R. & DE LARA, E. 2016. Touchless and always-on cloud analytics as a service. *IBM Journal of Research and Development,* 60**,** 11: 1-11: 10.

SUSMAN, G. I. 1983. Action research: a sociotechnical systems perspective. *Beyond method: Strategies for social research***,** 95-113.

TAKEDA, H., VEERKAMP, P. & YOSHIKAWA, H. 1990. Modeling design process. *AI magazine,* 11**,** 37.

TANG, L., DONG, J., ZHAO, Y. & ZHANG, L.-J. Enterprise cloud service architecture. Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on, 2010. IEEE, 27-34.

TANG, M., DAI, X., LIU, J. & CHEN, J. 2017. Towards a trust evaluation middleware for cloud service selection. *Future Generation Computer Systems,* 74**,** 302-312.

TRAN, H. T. & FEUERLICHT, G. Service repository for cloud service consumer life cycle management. European Conference on Service-Oriented and Cloud Computing, 2015. Springer, 171-180.

TRAN, H. T. & FEUERLICHT, G. Improving reliability of cloud-based applications. European Conference on Service-Oriented and Cloud Computing, 2016a. Springer, 235-247.

TRAN, H. T. & FEUERLICHT, G. Optimization of Cloud-Based Applications Using Multi-site QoS Information. International Conference on Research and Practical Issues of Enterprise Information Systems, 2016b. Springer, 325-338.

TRAN, H. T. & FEUERLICHT, G. 2016c. Service Development Life Cycle for Hybrid Cloud Environments. *JSW,* 11**,** 704-711.

TRIPATHI, A., PATHAK, I. & VIDYARTHI, D. P. 2017. Integration of analytic network process with service measurement index framework for cloud service provider selection. *Concurrency and Computation: Practice and Experience,* 29.

TSAI, W. T., ZHOU, X., CHEN, Y. & BAI, X. 2008. On Testing and Evaluating Service-Oriented Software. *Computer,* 41**,** 40-46.

TURK, D., FRANCE, R. & RUMPE, B. 2014. Limitations of agile software processes. *arXiv preprint arXiv:1409.6600*.

TUTEJA, M. & DUBEY, G. 2012. A research study on importance of testing and quality assurance in software development life cycle (SDLC) models. *International Journal of Soft Computing and Engineering (IJSCE),* 2**,** 251-257.

UR REHMAN, Z., HUSSAIN, O. K. & HUSSAIN, F. K. Multi-criteria IaaS service selection based on QoS history. Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on, 2013. IEEE, 1129-1135.

VITHARANA, P. & JAIN, H. 2012. A Knowledge Based Component/Service Repository to Enhance Analysts' Domain Knowledge for Requirements Analysis. *Information & Management,* 49**,** 24-35.

VON ALAN, R. H., MARCH, S. T., PARK, J. & RAM, S. 2004. Design science in information systems research. *MIS quarterly,* 28**,** 75-105.

VOS, J. 2017. A requirements engineering framework for rural hospitals in developing countries.

VUKOJEVIC-HAUPT, K., HAUPT, F., KARASTOYANOVA, D. & LEYMANN, F. Service Selection for On-demand Provisioned Services. The 18th International Enterprise Distributed Object Computing Conference (EDOC), 2014 Germany. IEEE, 120-127.

WAGNER, C., HUDIC, A., MAKSUTI, S., TAUBER, M. & PALLAS, F. Impact of critical infrastructure requirements on service migration guidelines to the cloud. Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on, 2015. IEEE, 1-8.

WALLS, J. G., WIDMEYER, G. R. & EL SAWY, O. A. 1992. Building an information system design theory for vigilant EIS. *Information systems research,* 3**,** 36-59.

WALTON, T. 2016. Cloud Foundry – PaaS for cloud applications.

WEBER, I., NEPAL, S. & ZHU, L. 2016. Developing dependable and secure cloud applications. *IEEE Internet Computing,* 20**,** 74-79.

WENMIN, L., WANCHUN, D., XIANGFENG, L. & CHEN, J. A history record-based service optimization method for QoS-aware service composition. Web Services (ICWS), 2011 IEEE International Conference on, 2011. IEEE, 666-673.

YANG, K., OU, S., AZMOODEH, M. & GEORGALAS, N. 2006. Model-based service discovery—prototyping experience of an OSS scenario. *BT technology journal,* 24**,** 145-150.

YAZDANOV, L. & FETZER, C. Dolen: User-side multi-cloud application monitoring. Future Internet of Things and Cloud (FiCloud), 2014 International Conference on, 2014. IEEE, 76-81.

YU, J., SHENG, Q. Z., HAN, J., WU, Y. & LIU, C. 2012. A Semantically Enhanced Service Repository for User-centric Service Discovery and Management. *Data & Knowledge Engineering,* 72**,** 202-218.

ZALAZAR, A. S., RODRIGUEZ, S. & BALLEJOS, L. C. Handling Dynamic Requirements in Cloud Computing. Simposio Argentino de Ingeniería de Software (ASSE 2015)-JAIIO 44 (Rosario, 2015), 2015.

ZHANG, L.-J. & ZHOU, Q. CCOA: Cloud computing open architecture. Web Services, 2009.

ICWS 2009. IEEE International Conference on, 2009. Ieee, 607-616.

ZHAO, L., SAKR, S. & LIU, A. 2015. A Framework for Consumer-Centric SLA Management of Cloud-Hosted Databases. *IEEE Transactions on Services Computing,* 8**,** 534-549.

ZHAO, X., YIN, J., ZHI, C. & CHEN, Z. 2017. SimMon: a toolkit for simulation of monitoring mechanisms in cloud computing environment. *Concurrency and Computation: Practice and Experience,* 29.

ZHENG, Z., LYU, M. & WANG, H. 2015. Service fault tolerance for highly reliable service-oriented systems: an overview. *Science China Information Sciences,* 58**,** 1-12.

ZHENG, Z. & LYU, M. R. 2015. Selecting an Optimal Fault Tolerance Strategy for Reliable Service-Oriented Systems with Local and Global Constraints. *IEEE Transactions on Computers,* 64**,** 219-232.

ZHENG, Z., WU, X., ZHANG, Y., LYU, M. R. & WANG, J. 2013. QoS ranking prediction for cloud services. *IEEE transactions on parallel and distributed systems,* 24**,** 1213-1222.

ZIBIN, Z. & LYU, M. R. A Distributed Replication Strategy Evaluation and Selection Framework for Fault Tolerant Web Services. Web Services, 2008. ICWS '08. IEEE International Conference on, 23-26 Sept. 2008 2008. 145-152.

ZIBIN, Z., ZHOU, T. C., LYU, M. R. & KING, I. 2012. Component Ranking for Fault-Tolerant Cloud Applications. *Services Computing, IEEE Transactions on,* 5**,** 540-550.

ZISMAN, A., SPANOUDAKIS, G., DOOLEY, J. & SIVERONI, I. 2013. Proactive and reactive runtime service discovery: A framework and its evaluation. *IEEE Transactions on Software Engineering,* 39**,** 954-974.

ZOBEL, J. 2004. *Writing for computer science*, Springer.

ZUO, W., BENHARKAT, A. N. & AMGHAR, Y. Holistic and Change-centric Model for Web Service Evolution. 2014 IEEE World Congress on Services, 2014. IEEE, 250-253.

APPENDIX A.

LIST OF QUALITY OF SERVICE ATTRIBUTES

This section gives a reference of common QoS attributes that are considered during system development lifecycle. To give further explanation of Quality of Service terms defined in Chapter 1, Meier et al. (2009) categorized these QoS attributes are into four groups (Table B.1): design, runtime, system, and user qualities.

Table B.1. List of QoS attributes (Meier et al., 2009).

Category	Quality attribute	Description
Design Qualities	Conceptual Integrity	Conceptual integrity defines the consistency and coherence of the overall design. This includes the way that components or modules are designed, as well as factors such as coding style and variable naming.
	Maintainability	Maintainability is the ability of the system to undergo changes with a degree of ease. These changes could impact components, services, features, and interfaces when adding or changing the functionality, fixing errors, and meeting new business requirements.
	Reusability	Reusability defines the capability for components and subsystems to be suitable for use in other applications and in other scenarios. Reusability minimizes the duplication of components and the implementation time.
Run-time Qualities	Availability	Availability defines the proportion of time that the system is functional and working. It can be measured as a percentage of the total system downtime over a predefined period. Availability will be affected by system errors, infrastructure problems, malicious attacks, and system load.
	Interoperability	Interoperability is the ability of a system or different systems to operate successfully by communicating and exchanging information with other external systems written and run by external parties. An interoperable

		system makes it easier to exchange and reuse information internally as well as externally.
	Manageability	Manageability defines how easy it is for system administrators to manage the application, usually through sufficient and useful instrumentation exposed for use in monitoring systems and for debugging and performance tuning.
	Performance	Performance is an indication of the responsiveness of a system to execute any action within a given time interval. It can be measured in terms of latency or throughput. Latency is the time taken to respond to any event. Throughput is the number of events that take place within a given amount of time.
	Reliability	Reliability is the ability of a system to remain operational over time. Reliability is measured as the probability that a system will not fail to perform its intended functions over a specified time interval.
	Scalability	Scalability is the ability of a system to either handle increases in load without impact on the performance of the system, or the ability to be readily enlarged.
	Security	Security is the capability of a system to prevent malicious or accidental actions outside of the designed usage, and to prevent disclosure or loss of information. A secure system aims to protect assets and prevent unauthorized modification of information.
System Qualities	Supportability	Supportability is the ability of the system to provide information helpful for identifying and resolving issues when it fails to work correctly.
	Testability	Testability is a measure of how easy it is to create test criteria for the system and its components, and to execute these tests in order to determine if the criteria are met. Good testability makes it more likely that faults in a system can be isolated in a timely and effective manner.
User Qualities	Usability	Usability defines how well the application meets the requirements of the user and consumer by being intuitive, easy to localize and globalize, providing good access for disabled users, and resulting in a good overall user experience.

APPENDIX B.

THE FMP BUSINESS REQUIREMENTS

1. Revision History

Date	Version	Description	Author
08.06.2016	1.0	Draft the first version	Thanh Nguyen
15.06.2016	1.1	Update Business Requirements	Son Nguyen
15.06.2016	1.2	Update specifications	Thai Tran
20.06.2016	1.3	Update workflows	Son Nguyen
26.06.2016	1.4	Update Mockups and Validations	Son Nguyen
01.07.2016	1.5	Update workflow	Thai Tran
08.07.2016	1.6	Update Client, Package, Company Mock-up	Thai Tran
10.07.2016	1.7	Add system log, statistic reports	Thanh Nguyen
12.07.2016	1.8	Add Front Desk mockup	Thai Tran
01.08.2016	1.9	Update some mockups per Amir comments	Thai Tran

2. Purpose

This document is intended to describe Requirement Specifications of FMP Application including functional requirements and non-functional requirements.

3. Company Overview

Family Medical Practice (FMP) is Vietnam's leading international primary health care provider with a high standard of Client care and friendly atmosphere.

FMP's clinics are strategically located across Vietnam: Hanoi in the North, Da Nang in the Center and Ho Chi Minh City (Saigon) in the South. With nationwide operational advantage, FMP provides international standard medical services in a safe, professional, and welcoming environment for all our customers.

4. Project Scope

FMP is designed to deploy as separated instances in one Ho Chi Minh clinic and one Ha Noi clinic.

5. Business Requirements

The purpose of building the application is to replace the current manual process with a better process managed by computer software.

5.1. User Roles

This system is used by different users having the following roles

1. IT Admin / System administration: user management, assign roles and privileges, reset password etc. The IT admin is the highest hierarchy and can manage all roles including assigning user admin.

2. User Admin: has all of the permissions to manage all of the data, configure lab's parameters, enable/disable modules for each lab (One lab at Hanoi and one lab at HCMC), create and manage tests, articles, 3rd integration parameters.

3. The nurse has the following abilities:

- Lookup clients, add/edit required fields about the client medical information,

- View details of clients

- Enter test result values for those tests taken at nurse station
- Add/remove lab tests
- Add vaccination including a vaccination plan
- Print barcode stickers for lab tests
- Refer client to another test station queue
- The nurse will also own to integrate with the Front desk about additional test/Vaccinations and including the cost of each
- Mark the samples are collected or not collected (collect later or are refused to be collected)
- Mark those tests as done which are taken place at nurse
- Can see the additional tests which are not included in the initial order packages
- Send messages to front desks or doctors
- Review a Doctor request for retest
- Attach files/images for tests

4. Technician:

- Look up clients
- View details of clients
- Enter test result values
- Attach files/images.

5. Doctor:

- Look up clients

- View details of client
- View EMR
- Do consulting (input comment/suggestion/conclusion).
- Integrate with the nurses like for request for additional test
- Request for additional external tests
- Document a request for a client to come for additional visitation and associate it with the Front Desk
- Choose articles which

6. Back Office:

- View the Care 1 schedule (availability)
- Schedule medical check-up, add/upload, edit, remove a list of Clients for companies
- Create/edit companies
- Create/edit contracts
- Create/edit packages (add/remove tests from package)

7. Front Desk:

- View the Care 1 schedule (availability)
- Manage all schedule for all types of clients coming for the Care a and including Managing the all doctor's availability (plan the backend / doctor schedule- Roster),
- Schedule private and company medical check-up
- Lookup clients
- View details of a Client

- Create new client profile (non-company check-up)
- Update Medical History forms (4 languages),
- Print client membership cards
- Lookup a scheduled check-up
- Lab manager has following the permissions:
- Enter lab test results
- Add or remove tests from an order
- Create/edit tests and test groups
- Mark a lab test sample as received or refused
- Attach files/images to the lab tests

8. Clinic manager should have all of the permissions like the “User Admin” as above

5.2. Assumptions

Manage action includes search, list, view detail, create, update, and remove information.

Remove: There was no actual deletion within the system when a user removed an entity, this entity will be marked as deleted, and later on, when pulling data from the database, these deleted entities are not returned.

Generically: The application and by IT Admin will own a UI to restore removed records.

Create/update action: Users can create/update a new/existing entity with a form or import data from an excel file with a predefined format.

5.3. Requirements

The system should have the following functions:

5.3.1. Manage application settings and constraints.

System administration can manage the general information or settings such as:

1. Security: doctors, nurses, managers, receptionists, administrators can log into the application and use their assigned functionalities of the application. They are able to manage their accounts information such as changing the password or reset their password. Their activities against the application will be recorded in the log. Each action, regardless if by the user, System / API, the application will be documented in the log file. The application will provide a way to archive both log file and old Medical data collected over the years

2. Manage a list of FMP clinics: Administrators can manage the list of clinics (locations). The information of a clinic contains the name, phone and address. The application should support adding clinics. For this project, there are only 2 clinics for now.

3. Manage a list of packages of FMP (That is a privilege of also a user admin). A test package is a template which contains a group of tests that will be applied to clients of a company and/or a Private package. Some tests for a package will be assigned to the Clients with the age and gender conditions. While signing a contract, companies can require changing (add or remove) list of the test in the packages.

4. Manage a list of FMP users, roles and privileges in different locations.

- User Admin: has all of the permissions to manage all of the data, configure lab's parameters (That is a privilege of also a user Admin / Lab Manager), enable/disable modules for each lab, create and manage tests, articles, 3rd integration parameters

- Nurse: Look up Clients, view details of Clients, enter test result values, add / remove tests, create orders to the lab for using the Roche, print test barcode, create vaccination request including vaccination plan, inform Front Desk for additional tests / costs, route the client by the queue management

- Technician: look up clients, view details of clients, enter test result values, attach files/ images

- Doctor (with speciality): lookup clients, view details of clients, view/edit EMR (including all client medical history), do consulting (input comment/suggestion), route the client by the queue management, request for additional tests, refer 3rd party in and out the FMP, attach document/ images

- Back Office: scheduled medical check-up, add/upload list of clients per contract, create packages, remind package

- Front Desk: manage the all doctor's availability (plan the backend / doctor schedule), schedule private and company client/employee medical check-up, lookup clients, view details of clients, create new files for new clients (Private check-up), insert/update client medical history form (4 languages), print membership card, look up client check-up

5. Manage contents/articles to be included in the report.

- Admin and Back Office can see a list of articles that can be included in the final medical report. The system will also support the option of configuring the page sequence of the report be sent to the client. Because of the size of the report, a storage service with security settings is deployed to allow clients to download their report.

- Admin and Back Office can create new, edit with the rich text editor or delete existing contents / articles.

- Article content is saved in HTML format.

- Admin and Back Office can import articles from pdf format file.

6. Manage 3rd party integration settings: Administrators are able to configure the third-party integration settings to DIACOM devices/server and COBAS IT connection parameters.

5.3.2. Manage contracts

When a company signs a contract with FMP, the contract contains:

- Company details

- Package or list packages and the tests included in each package
- Number of employees (clients) in each package and the total number

The Care1 application must have the following abilities to manage the contracts:

Manage the list companies

BO has the permission to sees list of companies in a table/grid with the following column: company name, the active package name(s) with FMP (if the company has an active package/contract with FMP), address, main contact person name, creation date.

The list is sorted by creation date by default. The application needs to support: Search a company and/ or by start typing a company string the all contain it will appear.

The grid columns head will own a filter option to sort/filter by name, by package name, address or advanced filter where Back Office can filter all company's properties. The list also has to page with 20 items per page and action column where Back Office can edit or delete company

The following fields are required to create a new company: company name, company address, company telephone no, company tax no. A company can have one or many contact persons; each contact person entry requires the following fields: name, phone number, email, title.

Creating a sub-company is the same as a company except that the parent company need to be specified, its addresses can be the same as the parent company.

Back Office can view the company details includes the following sections: basic information as above. A paging list that shows company active/inactive contract(s) and sort by created time as default.

Back Office can also create package here and sort/filter contracts by creation date / time, name, status. Click on each package will show contract detail page including the list of packages. The main package properties are Tests per package and the parameters to run the test like age, gender, etc.

Back Office can import client per contract from by uploading an excel. Client (employees) list: A paging list that shows company's Clients (employees) and sort by

alphabet as default. Click on Client will show Client detail page

Manage company check-up packages.

Back Office clicks on package management menu, sees a list of packages in a table/grid with the following columns: package name, company name, check-up dates, status (new, in progress, reported, done), 'Valid From', Expiration Date. The list is sorted by check-up date of company's packages by default. For packages, which do not have check-up date, we will sort by alphabet all columns heads and cross-application supposed to own the sorting. capability. The application needs to support: Search a company and/ or by start typing a company string the all contain it will appear.

The main package properties are Package Name, package cost, Tests per package and the parameters to run the test like age, gender, 'Valid From', Expiration Date...

The red colour will be used to alert the company that not finishing check-up or late report sending in the list. The grid columns head will own a sort/filter option to filter by package name, company name, check-up date, status or advanced filter where Back Office can sort/filter all package's properties.

Back Office can duplicate a package to copy the package content to save time in creating similar packages. The package type can be a company, private, visa and work permit. A package can be replicated an old package or to re-activate. Replicate package creates a package with the same information with original package except for the name. Creating new package, Back Office needs to fill in the information: package type (e.g. company, work permit, visa check-up, private), package name, package price, check-up dates, package notes, package valid dates ("Valid From" and "Expiration Date"), isActive: leave it as unchecked if it not ready to used, status of the package.

In the form to create a package, there is an option to select a predefined package in the system to copy the values from them.

If selecting predefined package type, for example, Canada Visa Check-up, all tests will be populated accordingly. While replicating/re-active a package, the list of tests like the package properties can be editable. For each package, we can add Tests and Clients into.

Clicking on each package, it will show package details page with name, price, note, and

status. Back Office can edit package properties. Back office/ Front desk/ Clinic Manager will be able to Re-activate a package as well as deactivate a package (for de-active, there will be a validation of a non-existing /associated active clients (in process): name, note, list of test which can be sorted/filtered by test name, list of client which can be Sorted/ filtered by all column head: name, ID, Passport number, package name, title in company, phone number, barcode, gender, check-up date, view check-up status and report for each Client from the list. The check-up date is scheduled later on by the Back office/ Front Desk

A client can be associated with one-to-many packages, the packages can be bought by the company which he/she work for, or they can be bought by the client himself or herself when doing a health check-up as Private. For each Client Medical Report, there will be reminder function to remind doctor / nurse to update or finalize it. The report cannot be edited by changing status to "done".

Manage list of Clients (employees)

Clients can be created by Back Office or Front Desk. Company's clients will be created mostly by Back Office, mostly by upload a client list from a pre-defined file holding a specific template.

Individual client (adding/editing client details and associate him with a pre-defined package) will be done by Front Desk by the appropriate role privilege. The application will support a client request for adding a non-included test from the test list The application also supposed to support a removing of a test/s or replacing it from the test list. The application will also support to change the cost of the tests for those (the cost change is an assigned privilege).

A Client can own a Private and / or a company package at a specific time. The application will need to support the tests by associating tests to a non-company "packages": Visa check-up per country, work permit Check-up, or Vaccination.

There is a section to manage the Clients. Clicking on the Client menu, we display a list of Clients in a table/grid with name, address, ID, Passport, phone number, barcode, date of birth, company or any other, active package, active check-up. The list is sorted by active check-up date by default. For Clients who do not have any active check-up, we

will sort by alphabet.

The company list column heads will support filter/sorting a Client by name, address, ID, Passport, gender, barcode, phone number, barcode, date of birth, age, company, active package, active check-up or advanced filter where we can sort/filter all Client's properties. The list also has to page with 20 items per page and action column where we can edit or delete Client

Back Office works with the company to sign a contract, creates company or searches company in the Care1 system to create Clients for the company. The application will support Back Office to upload/import the list of employees for a contact from an excel file, and it is supposed to alert while adding/ uploading a list of employees about duplications. Also, Back Office can handle the task manually as follows:

Back Office checks if clients are already in the company or not (by searching)

If Clients are already in the system, Back Office can assign them to the company. If Clients are already in the system but belong to other companies, Back Office can assign them one more company. Client can't be associated with 2 companies at the same time (02 active packages one from each of the companies) actually the Data Base and for the Client properties has to support multi instances of the company name by 'Date From', 'Date To' (covers the periods he works for each of the companies)

If Clients are not in the system, Back Office / Front Desk clicks on "Create New" button on top of Client list to create a new Client

Individual Clients will own a pre-defined package of Visa Packages or Work permit package. The private check-up will own a 'cost' property as well as for each test, and the application will support the user to issue a bill for the Client (The application will not support the billing by itself). The application will own the package price and the price of each test associated with it. While assigning a client to a package, the application will support adding a non-included test as well as test "removal" from it. The application has to support the User by displaying the all tests been added as well as been removed (including the costs of each) with an appropriate UI for a later on billing.

Creating a new Client, Back Office needs to fill in the basic information: full name, ID, passport, barcode (auto-generated), gender, date of birth, age (auto-generated), email,

phone number, company (current company is filled automatically), title (position in company), insurance company

The application can process the excel file which column headers must have the exact names, for instance, “full name” column header will be only recognized it is titled correctly, if it is titled as “full name”, there will be an error message saying that “full name” column is missing. The following are some notes about the values for some of the required columns:

- Date of birth must be in the format DD-MM-YYYY. Column header must be titled as “DOB.”

- Gender: valid values are male, female and other. Column header must be titled as “Gender.”

- Full name: alphabet characters only, space char is in the middle of words. Column header must be titled as “Full name.”

- Package name: must use be one names of the valid packages for the current company. Column header must be titled as “Package.”

If there is an error(s) when importing Client, all detail log, e.g. what field has an error, will be reported via log messages as the mock-up.

Back Office can view Client details screen with two tabs:

Basic information as described above.

Check-ups: divided into two tables, one for a current check-up and one for past/completed check-up. Each paging table is sorted by check-up date as default and includes the columns as company name, package name, check-up date, status

At Care1, Front Desk meets a client and ask he/she for a membership card or full name, company or DOB and then looking up the check-up of the client in the check-up management screen for that day.

- If the check-up does exist on the management screen and the time slot is not too sooner or later, Front Desk will mark the check-up entry as checked-in.

- In case a client comes sooner or later than the schedule, Front Desk can check the availability and rearrange if it is possible and mark it as checked-in.

- If there is no check-up scheduled, the Client will be asked for the service that he/she desire to order. If the ordered service is visa or work permit, then Front Desk simply creates the check-up at the slot with that type of check-up for that client.

- If this is the first-time client visits Care1, the client profile will be created with the information provided

- With the private check-up, Client can order any test that is applicable to his/her age, gender and marital status.

- The client can ask for additional applicable tests besides the test set included in the initially ordered package; Front Desk can see the total price and additional fee for the additional ordered tests.

Manage billing and invoice

The application auto-generates bill for companies using the information of check-up packages (price, number of clients in packages). It also manages extra payment for clients per private check-up request.

5.3.3. EMR (Electronic Medical Record)

Users can manage Medical record of a Client with privileges. The application must provide several forms with validations so that nurse/technician/doctor can enter data easily. Those forms should support attachments. The attachments can be images, pdf document, etc. The EMR module must support the all client medical history from the previous visitation.

It is important to have the efficient UI that doctors can compare test results easily and in the visualized way. Basically, it should have the control to specify the date range to view the records and display information in the grid mode. The headers the grid will be the dates which the test result is collected.

For the lab test and vital signs, there is an option to display the series of values over the

time in the line chart mode. The x-axis will be the dates of the previous visitations; y-axis is values of the test results collected at those times.

For those test results, which are collected with attachment, the user can view the images in the full mode or download the attachment easily.

6. Technical Solution

6.1. Approach

The proposed technical solution is based on the following factors:

- Proven frameworks and technologies: an architecture based on well-defined and established frameworks, libraries and technologies.

- Iterative methodology: applying proven iterative methodology will help to manage the project better. It forces a more consistent and manageable level pressure on the project which leads to more manageable level of overtime pay across the project and avoids finding last-minute major breakages

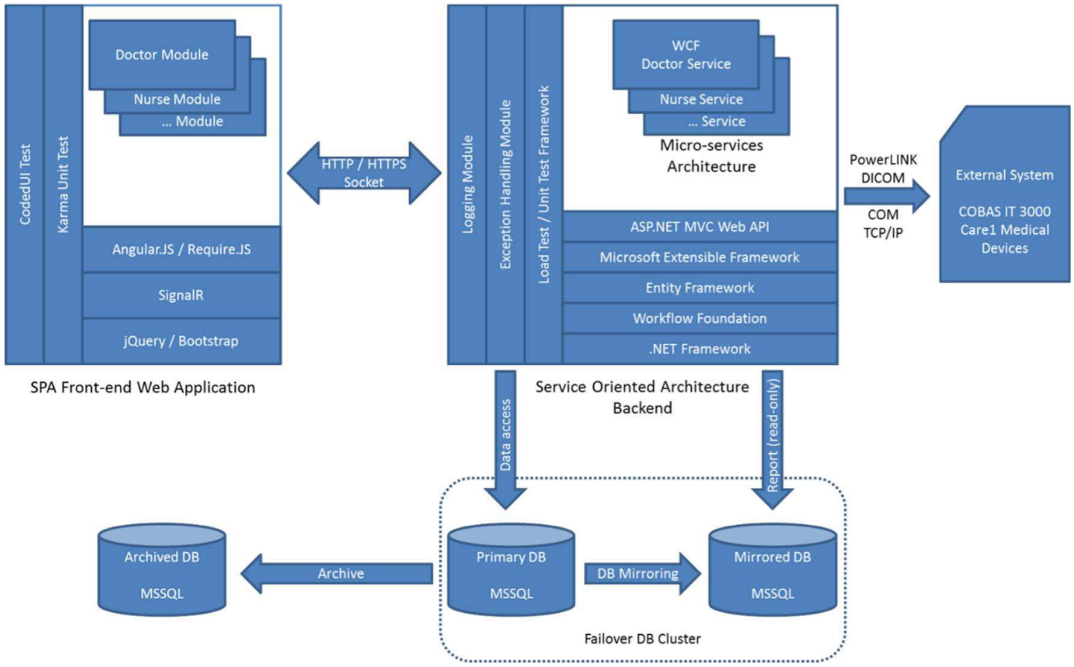
- Well, satisfy the non-functional requirements: all non-functional requirements described above needs to be met.

6.2. High-Level Architecture

Based on the functional and non-functional requirements described above, we would like to propose the high-level architecture of the EMR application as following:

6.2.1. Front-end Web Application

This is a web application written as an HTML5 Single Page Application (SPA) using the frameworks like AngularJS, RequireJS, Signal IR, jQuery and Bootstrap. Users will access this front-end application through the favourited browsers. Up requested, the front-end will retrieve data from various backend services through HTTP/HTTPS restful JSON web service APIs. Besides updating the UI through PULL requests, the front-end will also refresh its data using PUSH mechanism provided by the SignalR library whenever there are changes to data on the backend system so that users don't need to manual click



on the refresh button to update the new data.

Architecture wise, by using the AngularJS framework, it makes the front-end application a pure SPA Model-View-Controller (MVC) application which totally separates the layer of UI and Business Logic on the backend and improves the maintainability of the code. AngularJS also allows isolation between view and data structures on the web client. Through using AngularJS view template, we can have a dynamic UI view for different user preferences. It also allows us to architect the client in different view modules like Doctor Module, Nurse Module which makes the flexibility of upgrade and maintenance much easier. Together with the RequireJS which provides the mechanism for on-demand module loading and caching, it creates a good performance and good user-experience web application. Moreover, Karma Unit Test and CodedUI Automation Test are also implemented in the front-end application to enforce the quality requirement in Continuous Integration approach.

6.2.2. Backend

The backend is a set of restful JSON web services. It is called by the front end to retrieve data either stored in the database or provided by the external medical devices. The backend provides data to front end either through PULL requests using HTTP/HTTPS method or PUSH requests through web socket provided with the SignalR library. The

backend will interact with them to retrieve data from external medical devices using interfaces like: Powerlink, DICOM, COM or TCP/IP.

The backend will be implemented using various frameworks and libraries to ensure the performance, scalability, flexibility and maintainability factors:

- NET Framework: This is the base framework for all components in the backend.

- Workflow Foundation: This is used to build the Workflow Engine for the EMR application. It allows the EMR application to execute the workflows based on the predefined activities and their routines. Its re-host UI designer also allows the users to add new activities, redefine routings, workflow parameters using the drag-and-drop style.

- Entity Framework: ORM framework to talk to MSSQL Server. Will reduce the effort of mapping tables/views into objects and increase code readability.

- Microsoft Extensible Framework (MEF): A library to allow creating an extensible application, this will provide a solution for plug-and-play modules by making hard dependency and soft dependency.

- ASP.NET MVC Web API: Having a separated business service with the UI view will allow a dynamic UI view and layer responsibility.

- Logging Module: This is a module used for application logging throughout the system. It allows users to configure logging at various levels to reveal the system operations behind the scene. By doing so, the administrator can troubleshoot the system much easier by looking at the log.

- Exception Handling Module: A module for application exception handling. It helps to collect all details when there are unexpected crashes or when the application misbehaves. Together with the logging module, it helps the administrator troubleshoot the problem and bring the system back up to running faster.

- Load Test / Unit Test Framework: These frameworks help to create and enforce unit tests for the system. It also helps to test and make sure our system can handle the loads

that we expect.

6.2.3. Database

The EMR application will be implemented using MSSQL Standard Edition. There will be three database instances as follows:

Principal DB: This is the primary and read/write DB instance. The EMR application will mainly interact with this instance. This principal DB instance will be transactional replicated to the mirrored DB instance using database mirroring mechanism so that at all time the mirrored instance contains the same data as the principal DB instance. Together with the mirrored DB instance, a failover DB cluster is created. If there is an unexpected problem happens to the principal instance causing it offline, the role switch-over will be executed making the mirrored instance to take over the principal role and keep the system continue functioning.

Mirrored DB: This is the read-only DB instance. Its data will be kept updated and the same as principal DB instance all the time using the database mirroring mechanism. It is designed to take over the principal role whenever there is any problem happens to the principal instance and keep the system continue functioning. It is also designed to offload the principal DB instance by handling the report (heavy read-only) operations of the system.

Archived DB: This DB instance is used to archive the old data from the principal instance with the threshold specified by the administrator.

6.3. Non-functional Requirements

6.3.1. User Friendly

The front-end of EMR application will be implemented using Bootstrap CSS framework which was created by Twitter and is adopted by the modern web applications. By using the Bootstrap CSS framework, it helps to create a sleek, user-friendly and intuitive user interface. Together coupled with AngularJS view template engine and the SignalR PUSH technology, it helps to deliver a responsive and good user-experience web application. Besides, with the use of HTML5/CSS/Javascript Single Page Application, all EMR

application features are taken place inside the browser without the need to install 3rd party components on user devices which will help to streamline the setup on the user PC (just install the latest version of Firefox or Chrome) as well as avoiding many other unexpected issues with the 3rd party components.

6.3.2. Performance & Scalability

This system is carefully designed to achieve the best performance and scalability for all three major components.

Front-end web application

By applying the RequireJS and AngularJS library, only the views and scripts are loaded when needed, and they will be cached on the user's browser once they are downloaded. Because of the caching capability, the front-end web application will work pretty fast after the first use. Besides, SignalR is used to push the changes from server to the client only when there are changes which reduce the unnecessary pull requests from the browser which in turn increase the performance for both the front- end and the backend application. Also, because of the architecture of HTML5 Single Page Application, all UI rendering process logic is pushed to handle the client's browser which greatly reduces the load of the web server. To further scale the front-end web application for more concurrent users, the front-end web application can be deployed to multiple IIS web server and have them load balance using Windows Network Load Balancing Feature (Please refer to the Logical Deployment Diagram section).

Backend

The backend is designed with Service Oriented Architecture using the Micro-services pattern. Basically, they are a set of services which can be deployed to one physical/logical server or multiple physical/logical servers. For a high load service, it can also be deployed to a Network Load Balancing cluster with multiple servers to distribute its load. Hence, the system can scale nicely.

Database

The database will be optimized its performance by improving the database index through analyzing the query plan of the heavy and frequent queries which can be identified using the SQL Query Profiler tool. Besides, the monthly report will be generated and stored in a summary table so that it doesn't take time to run monthly report views. Moreover, the mirrored instance is set up to keep in-sync with the principal instance which will help to offload the report/read-only query from the principal instance. Also, the outdated data will be archived to the archived instance to keep the principal instance running at its best performance.

6.3.3. Flexibility

The flexibility of the system is achieved through the use of AngularJS view engine, microservices pattern, Microsoft Extensible Framework as well as the Workflow Foundation. With AngularJS view engine, each module view on the frontend application is isolated in its own structure and can be updated/replaced separately without impacting the others. This similar flexibility on the backend is achieved by using the micro-services pattern and the Microsoft Extensible Framework. New elements can be deployed to the system in a plug-and-play manner. Together with the Workflow Foundation, the administrator can make a change to the operation workflow by changing its route, parameters as well as new activities in a drag-and-drop style through the Workflow Re-hosting designer.

6.3.4. Availability

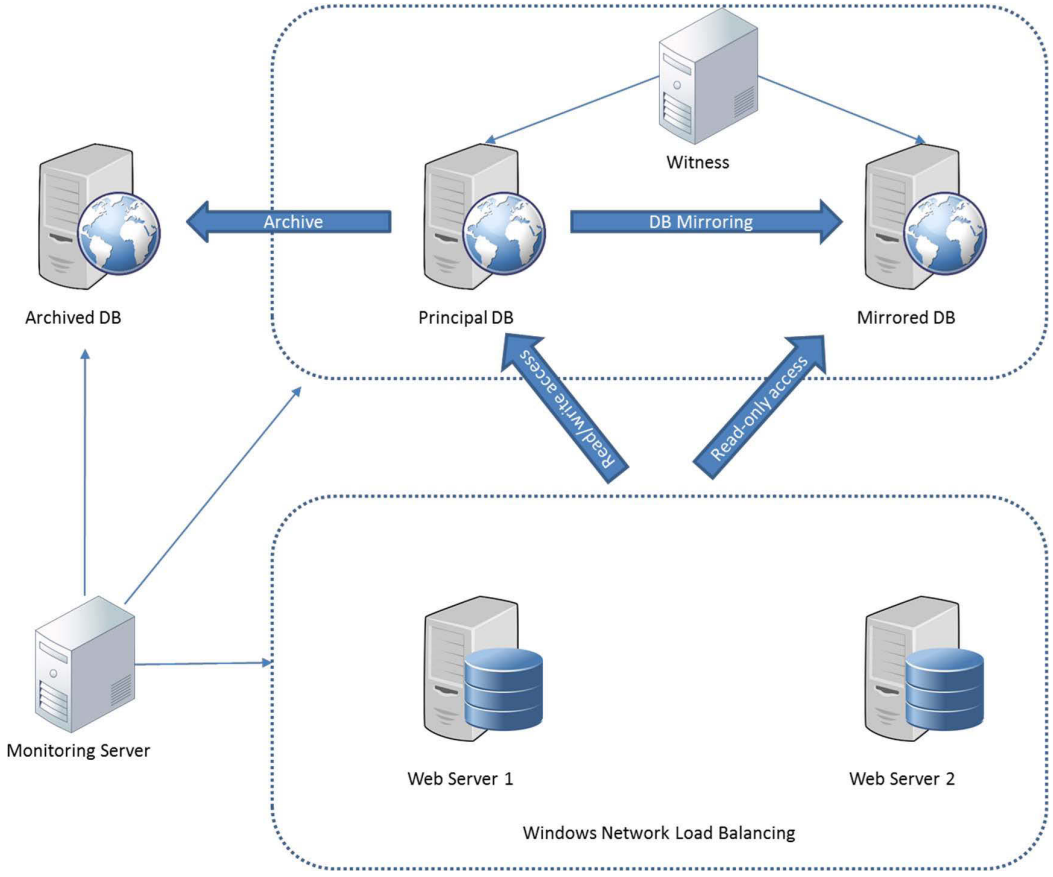
Based on the scalability design of the system, all components of the system can be deployed to run on multiple instance servers. This will remove the single breakdown point issue. Anyone of the running instances is down for any reason; the system will move the load to another running instance and keep the system functions properly. Besides, unit testing, logging and exception handling mechanism is in place which will help the administrator to troubleshoot and trace down the root cause of the problem easily if there is an unexpected incident happens during the operation as well as to bring the down instance back online as short time as possible. The availability of the application should be 99.9%

6.3.5. Maintainability

With “Single Responsibility Principle” and “Modularization” in mind, the system is designed to decouple its components into the different independent module, and layers with each one handle its own responsibility. The separation of code is clear and will be easy to maintain.

6.4. Database Migration

The existing UBQ data and schema needs to be fully understood by examining the training data and testing around with the UBQ application to create the relationship between the tables. SQL Server Migration Assistant for MySQL will be used to migrate the data from UBQ to the new EMR system. An incremental data migration approach will also need to be designed so that most of the data is migrated to the new system before the live switch from UBQ to the new EMR system. During the live switch, only a small remaining part of the data needs to be migrated to the new system; this is to reduce the



live switch time as well as to minimize the unexpected risks happens during the switch.

6.5. Logical Deployment Diagram

Web server 1 and the web server 2 will be hosting the Front-end application and backend application and are put inside the Windows Network Load Balancing cluster to balance the load as well as working as the failover mechanism to pick up the load if either one dies.

Principal database instance will be working as the read-write database and will be synchronously transactional replication to the mirrored database instance so that the mirrored instance can help to offload the heavy read-only queries. Archived instance will receive the outdated data from the principal database through the MSSQL job scheduler. Witness server will monitor the health of principal and mirrored instance and will automatically handle the role switch if there is an issue with the principal instance.

Monitoring server will periodically monitor the health (connectivity, service, CPU, memory, storage...) of all servers in the system and will send notification emails or SMS to the administrator when certain events (e.g. server is down, network is down...) occur or certain pre-defined thresholds (CPU usage reaches 80%, HDD usage reaches 90% ...) are met.

APPENDIX C.

CLOUD SERVICE ADAPTOR INTERFACES

1. Payment services

1.1. Payment generic interface

```
public interface IPaymentService : IServiceAdaptor
    {
        [OperationContract]
        PaymentResponse Pay(PaymentRequest request);

        [OperationContract]
        RefundResponse Refund(RefundRequest request);

        [OperationContract]
        AccountResponse Check(AccountRequest request);

    }
```

1.2. eWay service

```
public class eWayAdaptor : IPaymentService
    {
        private string _apiKey = "";
        private string _rapidEndpoint = "https://api.sandbox.ewaypayments.com/";
        private string _password = "";

        public int Attemps = 3;
        public int Delay = 5000;

        private Transaction ProcessPayRequest(PaymentRequest request)
        {
            var transaction = new Transaction();
            transaction.TransactionType = TransactionTypes.Purchase;

            transaction.Customer = new Customer();
            transaction.Customer.CardDetails = new CardDetails();
            transaction.Customer.CardDetails.Name = request.Card.HolderName;
            transaction.Customer.CardDetails.Number = request.Card.CardNumber;
            transaction.Customer.CardDetails.ExpiryMonth =
request.Card.ExpiryMonth.ToString("00");
            transaction.Customer.CardDetails.ExpiryYear =
request.Card.ExpiryYear.ToString("00");
            transaction.Customer.CardDetails.CVN =
request.Card.CCV.ToString("000");

            transaction.PaymentDetails = new PaymentDetails();
            transaction.PaymentDetails.TotalAmount = (int)(request.Amount * 100);
```

```
            transaction.PaymentDetails.CurrencyCode = "AUD";
            transaction.PaymentDetails.InvoiceNumber = request.InvoiceId;
            transaction.PaymentDetails.InvoiceReference =
request.TransactionId.ToString();
            transaction.PaymentDetails.InvoiceDescription = request.Comment;
            return transaction;
        }

        private PaymentResponse ProcessPayResponse(CreateTransactionResponse
parResponse)
        {
            var response = new PaymentResponse();
            if (parResponse.TransactionStatus.Status.Value)
                response.Result = ServiceResult.Success;
            else
            {
                var errorCodes = new string[] { "S5000", "S5085", "S5086",
"S5087", "S5099", "F9023", "F7000", "D4403", "D4406", "D4459", "D4496", "S9996",
"S9902", "S9992" };
                response.Result = ServiceResult.Error;
                var codes =
parResponse.TransactionStatus.ProcessingDetails.ResponseMessage.Split(new[] { ",
" }, StringSplitOptions.None);
                foreach (var code in codes)
                {
                    if (errorCodes.Contains(code))
                        response.Result = ServiceResult.Error;

response.Errors.Add(RapidClientFactory.UserDisplayMessage(code, "EN"));
                }
            }
            return response;
        }

        public PaymentResponse Pay(PaymentRequest request)
        {
            using (var context = new Context())
            {
                var response = new PaymentResponse();
                var adaptorLog = new ServiceLog(request);
                adaptorLog.ServiceCode = "eWay";
                adaptorLog.Name = "eWay";
                adaptorLog.Type = ServiceLogType.Adaptor;

                adaptorLog.Delay = Delay;
                adaptorLog.Location = request.Location;

                var ewayClient = RapidClientFactory.NewRapidClient(_apiKey,
_password, _rapidEndpoint);
                var transaction = ProcessPayRequest(request);

                var n = 1;

                adaptorLog.ResponseTime.Start = DateTime.Now;
                while (n <= Attemps)
                {
                    var log = new ServiceLog(request);
                    log.ServiceCode = "eWay";
                    log.Name = "eWay";
```

```
                    log.Type = ServiceLogType.CloudService;
                    log.Attemps = n;
                    adaptorLog.Attemps = n++;
                    log.ResponseTime.Start = DateTime.Now;
                    log.Location = request.Location;
                    try
                    {
                        log.ResponseTime.Start = DateTime.Now;
                        var result = ewayClient.Create(PaymentMethod.Direct,
transaction);
                        log.ResponseTime.End = DateTime.Now;
                        adaptorLog.ResponseTime.End = DateTime.Now;
                        response = ProcessPayResponse(result);
                    }
                    catch (Exception ex)
                    {
                        log.ResponseTime.End = DateTime.Now;
                        adaptorLog.ResponseTime.End = DateTime.Now;
                        response.Result = ServiceResult.Error;
                        response.Errors.Add(ex.Message);
                    }

                    log.Message = response.Message;
                    log.Result = response.Result.ToString();
                    context.Set<ServiceLog>().Add(log);

                    if (response.Result != ServiceResult.Error)
                        break;

                    if (n > Attemps)
                        break;

                    Thread.Sleep(Delay);
                }
                adaptorLog.Result = response.Result.ToString();
                adaptorLog.Message = response.Message;
                if (Attemps>1)
                    context.Set<Log>().Add(adaptorLog);
                context.SaveChanges();

                return response;
            }
        }
    }
```

1.3. PayPal service

```
public class PayPalAdaptor : IPaymentService
{
        public string User = "";
        public string Password = "";
        public string Partner =" ";
        public string Vendor = "";
        public string Url = "pilot-payflowpro.paypal.com";
        public string Application = "Service Consumer Framework";
        public string Version = "1.0";
        public int Attemps = 3;
        public int Delay = 5000;
```

```
        public bool ChangeAmount1 { get; set; }
        public bool ChangeAmount2 { get; set; }

        public PayPalAdaptor()
        {
        }

        private SaleTransaction ProcessPayRequest(PaymentRequest request)
        {
            var invoice = new Invoice();
            invoice.Amt = new Currency(new decimal(request.Amount),
request.Currency);
            invoice.CustRef = request.UserId.ToString();

            var cardDetails = new CreditCard(request.Card.CardNumber,
request.Card.ExpiryDate);
            cardDetails.Cvv2 = request.Card.CCV.ToString("000");
            cardDetails.Name = request.Card.HolderName;

            var card = new CardTender(cardDetails); // credit card
            var userInfo = new UserInfo(User, Vendor, Partner, Password);
            var connection = new PayflowConnectionData(Url, 443, 45, "", 0, "",
"");


            var transaction = new SaleTransaction(userInfo, connection, invoice,
card, PayflowUtility.RequestId);
            transaction.ClientInfo = new ClientInfo {IntegrationProduct =
Application, IntegrationVersion = Version};

            transaction.Verbosity = "LOW";


            return transaction;
        }
        private PaymentResponse ProcessPayResponse(Response parResponse)
        {
            var error = new StringBuilder();
            var response = new PaymentResponse();
            response.Result = ServiceResult.Invalid;
            response.ServiceCode = "PayPal";
            var result = parResponse.TransactionResponse;
            if (result == null)
            {
                response.Result = ServiceResult.Error;
                error.AppendLine("Unknown Error: Response is null");
                return response;
            }
            if (result.Duplicate == "1")
            {
                error.AppendLine("Duplicate Response: Duplicate Transaction");
                response.Result = ServiceResult.Invalid;
            }

            // Evaluate Result Code
            if (result.Result < 0)
            {
                // Transaction failed.
```

```
                error.AppendLine("System Error: There was an error processing
your transaction. Please contact Customer Service.");
                error.AppendLine("Error: " + result.Result);
                response.Result = ServiceResult.Error;
            }
            else if (result.Result == 0)
            {
                response.Errors.Clear();
                response.Result = ServiceResult.Success;
            }
            else if (result.Result == 13)
            {
                response.Result = ServiceResult.Invalid;
                error.AppendLine("Your Transaction is pending. Contact Customer
Service to complete your order.");
            }
            else if ((result.Result == 23 || result.Result == 24))
            {
                response.Result = ServiceResult.Invalid;
                error.AppendLine("Invalid credit card information. Please re-
enter.");
            }
            else if (result.Result == 126)
            {

                if (result.AVSAddr != "Y" || result.AVSZip != "Y")
                {
                    response.Result = ServiceResult.Invalid;
                    error.AppendLine("Your billing information does not match.
Please re-enter.");
                }
                else
                {
                    response.Result = ServiceResult.Invalid;
                    error.AppendLine("Your Transaction is Under Review. We will
notify you via e-mail if accepted.");
                }
            }
            else if (result.Result == 127)
            {
                response.Result = ServiceResult.Invalid;
                error.AppendLine("Your Transaction is Under Review. We will
notify you via e-mail if accepted.");
            }
            else
            {
                // Error occurred, display normalized message returned.
                response.Result = ServiceResult.Error;
                error.AppendLine("Error Code: " + result.Result + " - Error
Message: " +result.RespMsg);
            }

            // Get the Transaction Context and check for any contained SDK
specific errors (optional code).
            // This is not normally used in production.
            var transCtx = parResponse.TransactionContext;
            if (transCtx != null && transCtx.getErrorCount() > 0)
            {
                response.Result = ServiceResult.Error;
```

```
                error.AppendLine("Transaction Context Errors: " + transCtx);
            }

            if (response.Result != ServiceResult.Success)
                response.Errors.Add(error.ToString());

            return response;
        }

        public PaymentResponse Pay(PaymentRequest request)
        {
            using (var context = new Context())
            {
                var response = new PaymentResponse();
                var adaptorLog = new ServiceLog(request);
                adaptorLog.Name = "PayPal";
                adaptorLog.ServiceCode = "PayPal";

                adaptorLog.Delay = Delay;
                adaptorLog.Type = ServiceLogType.Adaptor;
                adaptorLog.Location = request.Location;
                adaptorLog.ApplicationCode = request.ApplicationCode;

                var saleTransaction = ProcessPayRequest(request);

                var n = 1;
                adaptorLog.ResponseTime.Start = DateTime.Now;
                while (n <= Attemps)
                {
                    var log = new ServiceLog(request);
                    log.ServiceCode = "PayPal";
                    log.Name = "PayPal";
                    log.Type = ServiceLogType.CloudService;
                    log.Attemps = n;
                    adaptorLog.Attemps = n++;
                    log.ApplicationCode = request.ApplicationCode;
                    log.Location = request.Location;
                    try
                    {
                        log.ResponseTime.Start = DateTime.Now;
                        var result = saleTransaction.SubmitTransaction();
                        log.ResponseTime.End = DateTime.Now;
                        adaptorLog.ResponseTime.End = DateTime.Now;
                        if (result != null)
                            response = ProcessPayResponse(result);
                        else
                        {
                            response.Result = ServiceResult.Error;
                            response.Errors.Add("Connection Error: Cannot connect
the service.");
                        }
                    }
                    catch (Exception ex)
                    {
                        log.ResponseTime.End = DateTime.Now;
                        adaptorLog.ResponseTime.End = DateTime.Now;
                        response.Result = ServiceResult.Error;
                        response.Errors.Add(ex.Message);
                    }
```

```
                    log.Message = response.Message;
                    log.Result = response.Result.ToString();
                    context.Set<ServiceLog>().Add(log);
                    if (response.Result != ServiceResult.Error)
                        break;
                    if (n > Attemps)
                        break;
                    // let's wait few seconds to see if this is a temporary
network issue.
                    Thread.Sleep(Delay);
                }
                adaptorLog.Result = response.Result.ToString();
                adaptorLog.Message = response.Message;
                if (Attemps > 1)
                    context.Set<Log>().Add(adaptorLog);
                context.SaveChanges();

                return response;
            }
        }


    }
```

1.4.Stripe service

```
public class StripeAdaptor : IPaymentService
    {
        public int Attemps = 3;
        public int Delay = 5000;

        private StripeChargeCreateOptions ProcessPayRequest(PaymentRequest
request)
        {
            var tokenOptions = new StripeTokenCreateOptions()
            {
                Card = new StripeCreditCardOptions()
                {
                    Number = request.Card.CardNumber,
                    ExpirationYear = request.Card.ExpiryYear,
                    ExpirationMonth = request.Card.ExpiryMonth,
                    Cvc = "123"
                }
            };

            var tokenService = new StripeTokenService();
            var stripeToken = tokenService.Create(tokenOptions);
            var transaction = new StripeChargeCreateOptions()
            {
                Amount = (int) (request.Amount * 100),
                Currency = "aud",
                Description = request.InvoiceId,
                SourceTokenOrExistingSourceId = stripeToken.Id,
            };
```

```
            return transaction;
        }

        private PaymentResponse ProcessPayResponse(StripeCharge parResponse)
        {
            var response = new PaymentResponse();
            response.Result = ServiceResult.Success;
            response.Errors.Add(parResponse.FailureMessage ?? "");
            return response;
        }

        public PaymentResponse Pay(PaymentRequest request)
        {
            using (var context = new Context())
            {
                var response = new PaymentResponse();
                var adaptorLog = new ServiceLog(request);
                adaptorLog.ServiceCode = "Stripe";
                adaptorLog.Name = "Stripe";
                adaptorLog.Type = ServiceLogType.Adaptor;
                adaptorLog.Delay = Delay;
                adaptorLog.Location = request.Location;
                var chargeService = new StripeChargeService();
                var n = 1;

                adaptorLog.ResponseTime.Start = DateTime.Now;
                while (n <= Attemps)
                {
                    var log = new ServiceLog(request);
                    log.ServiceCode = "Stripe";
                    log.Name = "Stripe";
                    log.Type = ServiceLogType.CloudService;
                    log.Attemps = n;
                    adaptorLog.Attemps = n++;
                    log.ResponseTime.Start = DateTime.Now;
                    log.Location = request.Location;
                    try
                    {
                        log.ResponseTime.Start = DateTime.Now;
                        var myCharge = ProcessPayRequest(request);
                        var result = chargeService.Create(myCharge);
                        log.ResponseTime.End = DateTime.Now;
                        adaptorLog.ResponseTime.End = DateTime.Now;
                        response = ProcessPayResponse(result);
                    }
                    catch (StripeException e)
                    {
                        log.ResponseTime.End = DateTime.Now;
                        adaptorLog.ResponseTime.End = DateTime.Now;
                        response.Result = ServiceResult.Success;
                        response.Errors.Add(e.StripeError.Code + ": " +
e.StripeError.Message);
                        switch (e.StripeError.ErrorType)
                        {
                            case "card_error":
                                response.Result = ServiceResult.Error;
                                break;
                            case "api_connection_error":
                                response.Result = ServiceResult.Error;
```

```
                            response.Errors.Add("api_error");
                            break;
                        case "api_error":
                            response.Result = ServiceResult.Error;
                            response.Errors.Add("api_error");
                            break;
                        case "authentication_error":
                            response.Result = ServiceResult.Error;
                            break;
                        case "invalid_request_error":
                            break;
                        case "rate_limit_error":
                            break;
                        case "validation_error":
                            break;
                        default:
                            response.Result = ServiceResult.Error;
                            break;
                    }
                }
                catch (Exception e)
                {
                    log.ResponseTime.End = DateTime.Now;
                    adaptorLog.ResponseTime.End = DateTime.Now;
                    response.Result = ServiceResult.Error;
                    response.Errors.Add(e.Message);
                }

                log.Message = response.Message;
                log.Result = response.Result.ToString();
                context.Set<ServiceLog>().Add(log);

                if (response.Result != ServiceResult.Error)
                    break;

                if (n > Attemps)
                    break;

                Thread.Sleep(Delay);
            }
            adaptorLog.Result = response.Result.ToString();
            adaptorLog.Message = response.Message;

            if (Attemps>1)
                context.Set<Log>().Add(adaptorLog);
            context.SaveChanges();

            return response;
        }
    }

}
```

2. Storage services

2.1. Storage generic interface

```
public interface IStorageService : IServiceAdaptor
{
        [OperationContract]
        Task<byte[]> Download(string path);

        [OperationContract]
        Task Upload(string path, byte[] content);

        [OperationContract]
        Task<ListFolderResponse> List(string path);
}
```

2.2. Google Drive service

```
public class GoogleDriveAdaptor : IStorageService
    {
        static string[] Scopes = { DriveService.Scope.Drive };
        static string ApplicationName = "Peyton Adaptor";
        private UserCredential Authenticate()
        {
            UserCredential credential;

            using (var stream =
                new FileStream("secret.json", FileMode.Open, FileAccess.Read))
            {
                string credPath = System.Environment.GetFolderPath(
                    System.Environment.SpecialFolder.Personal);
                credPath = Path.Combine(credPath, ".credentials/drive-dotnet-
quickstart.json");

                credential = GoogleWebAuthorizationBroker.AuthorizeAsync(
                    GoogleClientSecrets.Load(stream).Secrets,
                    Scopes,
                    "user",
                    CancellationToken.None,
                    new FileDataStore(credPath, true)).Result;
                Console.WriteLine("Credential file saved to: " + credPath);
            }
            return credential;

            // Create Drive API service.

        }

        public async Task<byte[]> Download(string path)
        {
            var service = new DriveService(new BaseClientService.Initializer()
            {
                HttpClientInitializer = Authenticate(),
                ApplicationName = ApplicationName,
            });
            var fileId = "";
            // Define parameters of request.
            FilesResource.ListRequest listRequest = service.Files.List();
            listRequest.PageSize = 100;
```

```
            listRequest.Fields = "nextPageToken, files(id, name)";
            listRequest.Q = "name = '" + path + "'";
            // List files.
            IList<Google.Apis.Drive.v3.Data.File> files =
listRequest.Execute().Files;

            if (files != null && files.Count > 0)
            {
                foreach (var file in files)
                {
                    fileId = file.Id;
                    break;
                }
            }
            using (var stream = new MemoryStream())
            {
                var downloadFile = await
service.Files.Get(fileId).DownloadAsync(stream);
                return stream.ToArray();
            }
        }

        public Task Upload(string path, byte[] content)
        {
            throw new NotImplementedException();
        }

        public Task<ListFolderResponse> List(string path)
        {
            throw new NotImplementedException();
        }
    }
```

2.3. Dropbox service

```
public class DropboxAdaptor : IStorageService
    {
        private string _token = "";
        public async Task<byte[]> Download(string path)
        {
            using (var dbx = new DropboxClient(_token))
            {
                using (var response = await dbx.Files.DownloadAsync("/" + path))
                {
                    return await response.GetContentAsByteArrayAsync();
                }
            }
        }

        public async Task Upload(string path, byte[] content)
        {
            using (var dbx = new DropboxClient(_token))
            {
                using (var mem = new MemoryStream(content))
                {
                    await dbx.Files.UploadAsync( "/" + path,
WriteMode.Overwrite.Instance, body: mem);
```

```
                }
            }
        }

        public async Task<ListFolderResponse> List(string path)
        {
            using (var dbx = new DropboxClient(_token))
            {
                var list = await dbx.Files.ListFolderAsync("/" + path);
                var response = new ListFolderResponse();
                response.Folders =
list.Entries.Where(i=>i.IsFolder).Select(i=>i.Name).ToList();
                response.Files = list.Entries.Where(i => i.IsFile)
                    .Select(i => new FileInformation() {Name = i.Name, Size =
i.AsFile.Size}).ToList();
                return response;
            }
        }
```

APPENDIX D.

QUESTIONNAIRE

Questionnaire of Service Consumer System Development Life-cycle and Service Consumer Framework.

This questionnaire will be used to evaluate the Service Consumer System Development Life-cycle (SC-SDLC) and Service Consumer Framework (SCF) that is using in UBQ replacement project. As discussed in the project proposals and in development methodology documents of the project, please provide your overall feedback as well as the feedback on each phase of the SC-SDLC. No personal information is collected. If you have any question, please email me, Hong Thai Tran (tranhongthai@gmail.com). Thanks for your time.

Part 1: Service Consumer System Development Lifecycle

Question 1: Which SDLC is used to apply in the previous projects in your company?

Question 2: In overall, does the SC-SDLC provide the methodology that covers the activities of the projects?

Question 3: What is the improvement of the development process when you apply the SC-SDLC in this project?

Question 4: In your opinion, what are the improvements needed for the SC-SDLC?

Question 5: What are your feedback and suggestions of **Requirement Specification** phase?

Question 6: What are your feedback and suggestions of **Service Identification** phase?

Question 7: What are your feedback and suggestions of **Service Integration**

phase?

Question 8: What are your feedback and suggestions of **Monitoring** phase?

Question 9: What are your feedback and suggestions of **Optimization** phase?

Part 2: Service Consumer Framework

Question 1: In overall, does the SCF provide the supporting tools for implementing the projects?

Question 2: What is the advantages and limitations of using the SCF to support the SC-SDLC?

Question 3: In your opinion, what are the improvements needed for the SCF?

Question 4: What are your feedback and suggestions of **Service Repository** phase?

Question 5: What are your feedback and suggestions of **Service Adaptor** phase?

Question 6: What are your feedback and suggestions of **Monitoring Centre** phase?

---------- Thank you ----------

APPENDIX E.

SOFTWARE

The Service Consumer Framework software, developed using .Net technologies, is uploaded on GitHub at https://github.com/tranhongthai/SCF.