A Reactive Architecture for Cloud-Based System Engineering

David Ebo Adjepon-Yamoah

School of Computing

Newcastle University

Newcastle-upon-Tyne, UK

A thesis submitted for the degree of

Doctor of Philosophy

November 2018

I would like to dedicate this thesis to my loving family.

Acknowledgements

Many people have supported me during my Ph.D. studies. First, I owe a debt of gratitude to my Supervisor, Prof. Alexander (Sascha) Romanovsky, who has given me the opportunity to undertake a PhD. Also, he has provided me with invaluable support, guidance and crucial insights throughout these four years.

Family is everything to me, and they have endured a lot because of my studies. Thank you to my mother, uncles and aunties.

A special thanks to my examiners: Emeritus Professor Malcolm Munro, Durham University and Dr. Nigel Thomas, Newcastle University.

I would also like to thank Dr. Phil Lord, Dr. Paul Ezhilchelvan, and Dr. David Kirk, who formed my progression panel. They provided invaluable comments and suggestions, helping to keep the work on the right track. I am equally grateful to Dr. Leo Freitas and Dr. Paolo Missier, the members of my thesis committee panel.

Being a PhD student sometimes has its ups and downs. I was lucky to have supportive colleagues ensuring that the good times outweighed the rough moments. In particular, I want to thank Sami, Maryam, Ehsan, and Razgar.

I especially want to thank all the staff members of the Centre for Software Reliability, Newcastle University. I am also grateful to my co-authors: Dr. Linas Laibinis and Dr. Elena Troubitsyna from Abo Akademi University, Finland; Dr. Zhenyu Wen from University of Edinburgh; and Dr. Alexei Iliasov and Paulius Stankaitis from Newcastle University.

Thank you all!

Abstract

Software system engineering is increasingly practised over globally distributed locations. Such a practise is termed as Global Software Development (GSD). GSD has become a business necessity mainly because of the scarcity of resources, cost, and the need to locate development closer to the customers. GSD is highly dependent on requirements management, but system requirements continuously change. Poorly managed change in requirements affects the overall cost, schedule and quality of GSD projects. It is particularly challenging to manage and trace such changes, and hence we require a rigorous requirement change management (RCM) process. RCM is not trivial in collocated software development; and with the presence of geographical, cultural, social and temporal factors, it makes RCM profoundly difficult for GSD. Existing RCM methods do not take into consideration these issues faced in GSD. Considering the state-of-the-art in RCM, design and analysis of architecture, and cloud accountability, this work contributes:

1. an alternative and novel mechanism for effective information and knowledge-sharing towards RCM and traceability.
2. a novel methodology for the design and analysis of small-to-medium size cloud-based systems, with a particular focus on the trade-off of quality attributes.
3. a dependable framework that facilitates the RCM and traceability method for cloud-based system engineering.
4. a novel methodology for assuring cloud accountability in terms of dependability.
5. a cloud-based framework to facilitate the cloud accountability methodology.

The results show a traceable RCM linkage between system engineering processes and stakeholder requirements for cloud-based GSD projects, which is better than existing approaches. Also, the results show an improved dependability assurance of systems interfacing with the unpredictable cloud environment. We reach the conclusion that RCM with a clear focus on traceability, which is then facilitated by a dependable framework, improves the chance of developing a cloud-based GSD project successfully.

Contents

Page

Contents **v**

List of Figures **x**

List of Tables **xiv**

1 Introduction **1**

1.1 Motivation 1

1.2 Problem Statement and Approach 4

1.2.1 Problem Statement 4

1.2.2 Research Aim and Objectives 4

1.3 Research Contributions 6

1.4 List of Publications 7

1.5 Thesis Structure 9

2 Background and Related Work **12**

2.1 Global Software Development 12

2.1.1 Overview 12

2.1.2 Requirement Management 13

2.1.3 Artefact Repositories 21

2.1.4 Global Software Development Tool Support 24

2.1.5 Software Process Improvement Methods 27

2.1.6 Quality Management of Software Processes 29

2.2 Software Architecture 31

2.2.1 Overview . . . 31
2.2.2 Software System Dependability . . . 32
2.2.3 Quality Attributes Characterisation . . . 36
2.2.4 Designing Software Systems . . . 37
2.2.5 Dependability Evaluation Methods . . . 37
2.2.6 Classifying the Size of Architectures for Target Analysis . . . 47
2.3 Cloud Deployment Environment . . . 51
2.3.1 Cloud Computing . . . 51
2.3.2 Issues with Cloud Computing . . . 52
2.3.3 Cloud Dependability Assurance . . . 54
2.3.4 Cloud Accountability Analysis . . . 57
2.3.5 Trust Assurance Methods . . . 62
2.4 Summary . . . 69

3 Reactive Architecture **72**
3.1 Introduction . . . 73
3.2 Challenges of GSD . . . 73
3.2.1 Effective Information and Knowledge Sharing . . . 73
3.2.2 Automation . . . 74
3.2.3 Diversity of Tools . . . 74
3.2.4 System Dependability . . . 75
3.2.5 Accountable Cloud . . . 75
3.3 Reactive Architecture . . . 76
3.3.1 Mechanism for Requirements Change Management and Traceability . . . 76
3.3.2 Design and Analysis Methodology for Cloud-Based Architecture 81
3.3.3 Assuring Dependability in the Cloud . . . 87
3.4 Summary . . . 89

4 Managing Requirement Change and Traceability **91**
4.1 Introduction . . . 92
4.2 Reactive Middleware . . . 92
4.2.1 Overview . . . 92

4.2.2 Change Management and Traceability as a Service 93
4.2.3 Shared Artefacts Repository System 106
4.2.4 System Engineering Tool Support System 106
4.3 GSD Requirements Discussion 108
4.4 Summary 109

5 Designing Architectures for Global Software Development 110
5.1 Designing the Reactive Architecture 111
5.1.1 Architecture Interface 114
5.1.2 Components 114
5.1.3 Connectors 115
5.1.4 Constraints 116
5.1.5 Dependability 117
5.2 Classifying the Size of the Reactive Architecture 117
5.3 Analysing the Reactive Architecture 120
5.3.1 Present cloud-ATAM 120
5.3.2 Present the Project Drivers 122
5.3.3 Present the Architecture 127
5.3.4 Identify Architectural Approaches 131
5.3.5 Generate the Quality Attribute Utility Tree and Scenarios . . 136
5.3.6 Analyse the Architectural Approaches 138
5.3.7 Present Results 139
5.4 Summary 139

6 Cloud Accountability 140
6.1 Introduction 141
6.2 Cloud Accountability System 142
6.3 Cloud Accountability Methodology 144
6.4 Summary 148

7 Evaluation 149
7.1 Reactive Middleware 150
7.1.1 Analysis of Expert Review 152

7.1.2 Airlock Control System Case Study . . . 157
7.2 Cloud Accountability System . . . 171
7.2.1 Collection . . . 173
7.2.2 Examination . . . 181
7.2.3 Analysis . . . 182
7.2.4 Reporting . . . 185
7.2.5 Summary . . . 187
7.3 Reactive Architecture . . . 188
7.3.1 Comparative Study . . . 189
7.3.2 Utility Tree Analysis Mechanism . . . 192
7.3.3 Stakeholders' Brainstorming Analysis Mechanism . . . 202
7.3.4 cloud-ATAM Analysis Report . . . 210
7.4 Conclusions . . . 211

8 Conclusion and Future Work 213
8.1 Conclusion . . . 213
8.2 Limitations . . . 215
8.2.1 Designing a Cloud-Based Architecture . . . 215
8.2.2 Ensuring Traceability with the Reactive Middleware . . . 216
8.2.3 Cloud Accountability Analysis . . . 216
8.2.4 Constitution of the Expert Panel . . . 217
8.3 Future Work . . . 217
8.3.1 Shared Artefacts Repository Mining and Machine Learning . . . 218
8.3.2 Tool Support for the cloud-ATAM . . . 218
8.3.3 Cloud Accountability of Other Dependability Attributes . . . 219
8.3.4 Future Case Study: Artificial Bee Colony Model-Inspired Traffic Light Control System . . . 220

References 221

Appendix A: Expert Review of Change Management and Traceability Process Model 250
A.1 Overview of CM-T Process Model . . . 250

A.2 Adherence to CMMI Characteristics 251
A.3 Limited Scope 252
A.4 Consistency 253
A.5 Understandability 254
A.6 Ease of Use 255
A.7 Verifiability 255
A.8 Constitution of the Expert Panel 256

Appendix B: Reactive Middleware Implementation Details 257
B.9 Package Diagram of the Airlock Control System Case Study 257
B.10 Class Diagram of the Airlock Control System Case Study 258
B.11 Interaction Diagram of the Artefacts Monitoring System 259
B.12 Applying the System Engineering Toolbox to Formal Verification . . 260

Appendix C: Requirements and Sample Use Cases of the Reactive Architecture 263
C.13 Requirements 263
C.14 Use Cases 264
C.14.1 Client Management 268
C.14.2 Adding Tools to the Supporting Toolbox 270
C.14.3 Saving Artefacts 270
C.14.4 Downloading Artefacts 271
C.14.5 Sharing Artefacts 271
C.14.6 Change Management of Artefacts 271
C.14.7 Traceability of Artefacts 271
C.14.8 Primary Back-Up Repository 272
C.15 Questionnaire for cloud-ATAM Stakeholders' Brainstorm 273

Appendix D: Cloud Accountability System Implementation Details 277
D.16 Java Code Snippet for Metrics Data Collection on AWS 277

List of Figures

2.1 Olsen's Change Management Model [161] 15
2.2 Ince's Change Process Model [100] 15
2.3 Spiral Like Change Management Process [145] 16
2.4 Requirement Change Management Model [153] 17
2.5 ISO/IEC 12207 Software Processes 23
2.6 CMMI's Maturity Levels 26
2.7 Comparison of CMMI & SPICE Models 28
2.8 Quality Attributes Characterisation (from [43]) 35
2.9 Comparison of Architecture Evaluation Methods 41
2.10 Comparison of Current Architecture Evaluation Methods 44
2.11 The "Nines" of Availability [224] 56
2.12 Architecture of Virtual Machine Introspection (VMI) 58

3.1 Overview of Proposed Mechanism for Requirements Change Management and Traceability 77
3.2 Requirements Change Management and Traceability Processes 80
3.3 cloud-ATAM: Adapted ATAM with Two-Staged Analysis Approach . 82

4.1 Reactive Middleware Interactions 93
4.2 Overview of the Components of the Reactive Middleware 95
4.3 PMBOK® Process Group for System Engineering Life-Cycle 98
4.4 Candidate Processes Reflecting a CMMI Level 2 (Baseline) Capability 102
4.5 Change Management and Traceability Process Model 103

5.1 Layered View of Reactive Architecture 111

5.2 Overview of Reactive Architecture Use Cases 118
5.3 *cloud-ATAM* Concept Interactions 121
5.4 Component and Connector View of the Reactive Architecture 128
5.5 Utility Tree with Prioritised Scenarios 135

6.1 Conceptual Model of the Cloud Accountability System 141
6.2 Evidence Collector 142
6.3 Forensic Process 144

7.1 The Airlock Control System 156
7.2 Snippet of the Terminal Output for the Reactive Middleware User Management Service for the Airlock Control System Case Study . . . 162
7.3 Snippet of the Terminal Output for the Reactive Middleware Requirements Management Service for the Airlock Control System Case Study 163
7.4 Snippet of the Terminal Output for the Reactive Middleware Requirements Management Service (Priority) for the Airlock Control System Case Study 164
7.5 Snippet of the Terminal Output for the Reactive Middleware Change Management Service for the Airlock Control System Case Study . . . 166
7.6 Snippet of the Terminal Output for the Reactive Middleware Traceability Service for the Airlock Control System Case Study 167
7.7 Snippet of the Terminal Output for the Reactive Middleware Notification Service for the Airlock Control System Case Study 169
7.8 AWS/EC2 CAS Test-Bed Instances 171
7.9 Test-Bed's VMs Introspection in Two AWS Availability Zones 172
7.10 Some AWS/EC2 Instance and System Metrics 174
7.11 vmiGuestFDS: AWS CloudWatch Line Graphs for Some Metrics . . . 175
7.12 vmiGuestRM: AWS CloudWatch Line Graphs for Some Metrics . . . 175
7.13 vmiGuestSAR: AWS CloudWatch Line Graphs for Some Metrics . . . 176
7.14 vmiGuestSET: AWS CloudWatch Line Graphs for Some Metrics . . . 176
7.15 LibVMI API Functions used in the vmiMONITOR Instance 177
7.16 Processing Times (ms) from the Evidence Sources 180
7.17 Evidence-based Trust Analysis of Reactive Middleware's VM 183

7.18 CAS XML Log for Cloud Agents 186
7.19 cloud-ATAM Two-Staged Evaluation Approach 189
7.20 Component and Connector View of Reactive Architecture for Scenario (P1) Analysis 198
7.21 The Role Played by the Quality Attributes in Architecture Evaluation 204
7.22 Adequacy of Scenario-Based Methods for Architecture Evaluation . . 205
7.23 Maturity of ATAM for Architecture Evaluation 205
7.24 Representativeness of the Reactive Architecture's Requirements . . . 206
7.25 Relevance of the Reactive Architecture's Constraints and Quality Attributes 206
7.26 Usefulness of Reactive Architecture's Components, Relationships and Scenarios 207
7.27 Clarity of the Presentation of *cloud-ATAM* 207
7.28 Well Presentation of Quality Attribute Characterisation 208
7.29 Coverage of Attribute-Specific Questions 208
7.30 Adequately Analysed Scenarios of Reactive Architecture 209
7.31 Sound Reasoning of *cloud-ATAM* Analysis 209

1 (A.1) Overview of CM-T Process Model 250
2 (A.2) Adherence To CMMI Characteristics 251
3 (A.3) Limited Scope 252
4 (A.4) Consistency 253
5 (A.5) Understandability 254
6 (A.6) Ease of Use 255
7 (A.7) Verifiability - How clear is this presentation of the model? . . . 255

8 (B.9) Package Diagram of the Airlock Control System Case Study . . 257
9 (B.10) Class Diagram of the Airlock Control System Case Study . . . 258
10 (B.11) Interaction Diagram of the Airlock Control System Case Study 259
11 (B.12)Theorem Provers ToolBox Interactions in the AWS ECS Cloud 261

12 (C.14) UML Sequence Diagram: Client Management 265
13 (C.15) UML Sequence Diagram: Adding Tools to the Supporting Toolbox 266

14 (C.16) UML Sequence Diagram: Saving Artefacts 267
15 (C.17) UML Sequence Diagram: Download Artefact 267
16 (C.18) UML Sequence Diagram: Sharing Artefacts 268
17 (C.19) UML Sequence Diagram: Change Management of Artefacts . . 269
18 (C.20) UML Sequence Diagram: Traceability of Artefacts 269
19 (C.21) UML Sequence Diagram: Primary Repository Back-Up 270

List of Tables

3.1 Utility Trees vs. Scenario Brainstorming 84

4.1 Mapping the Key Steps for Effective Requirements Management and Traceability Processes with Proposed Change Management and Traceability Services 94
4.2 Description of Artefacts in the Shared Artefacts Repository 96
4.3 Global Software Development Management Guidelines 97
4.4 CM-T Process Model Validation 101
4.5 Meeting the High Level Requirements of an Effective GSD Framework 107

5.1 COSMIC FSM Software Project Size Classification Benchmark 118
5.2 Classification of Functional Processes based on the Reactive Architecture Use Cases 119
5.3 Mapping Requirements to Quality Attributes of Reactive Architecture 123
5.4 Mapping Architectural Styles to Requirements of Reactive Architecture 134
5.5 Classified Quality Attribute Scenarios according to Types 137

7.1 Evaluation Criteria and Related CMMI Level 2 Capability Questions 151
7.2 Classified Requirements of the Airlock Control System 158
7.3 Classified GSD Guidelines Steps 159
7.4 Classified Steps of Cloud Accountability Method with The Forensic Process Phases (NIST SP800-86 Guide) 173
7.5 Classification of Availability and Reliability for the Test-Bed VMs . . 181
7.6 Comparative Study of ATAM and cloud-ATAM 191
7.7 Prioritised Quality Attribute Scenarios 194

7.8 Prioritised (**X**) Quality Attribute Scenarios 195
7.9 Classified Quality Attribute Scenarios according to Types 196
7.10 Prioritised Quality Attribute Scenarios (Ordered) 197
7.11 Analysis of Sensitivities, Trade-offs, Risks & Non-Risks for the Utility Tree 199
7.12 Analysis of Performance Scenario - **P1** 200

8.1 Mapping High-Level Requirements to Components of Reactive Architecture 215

2 (A.8) The Expert Panel 256

3 (B.13) Performance Benchmark 261

Chapter 1

Introduction

This chapter initially describes the motivations behind the thesis and the main topics related to this work in Section 1.1. The research problem statement and our approaches are presented in Section 1.2, and the research contributions in Section 1.3. Section 1.4 lists publications related to the thesis. Finally, the thesis structure is presented in Section 1.5.

1.1 Motivation

Global software development (GSD) is characterised by globally distributed teams which are made up of stakeholders from different geographic locations, and different national and organisational cultures. Many software development companies nowadays strive for the utilisation of benefits offered by GSD such as: access to large skilled labour pool, improving time to market, reduced software development costs by delegating work to countries with low labour cost, to produce better quality product [175], [157], [115], [109], [8]. The collaboration among globally distributed teams is based on the team members' communication. However, the coordination and control of communication forms the main challenges of GSD [193]. Such challenges are generally influenced by distance: geographical, socio-cultural and temporal. The physical distance between remote team members is identified as the geographical distance. Also, the extent to which members of a team vary with regards to language, social status, religion, economic conditions, politics, and basic assumptions constitutes

socio-cultural distance. Furthermore, cultural issues, such as attitudes toward hierarchy, communication styles, time, and need for structure, are often different. Some researchers [180], [17] point out that the dispersion of work force constitutes a drop in productivity in GSD. Others like [121], indicate that the productivity of globally distributed team members decreases by up to 50% compared to that of co-located team members. However, an even more critical issue is that GSD is highly dependent on requirements management. The specific challenge here is that system requirements continuously change. Primarily, poorly managed change in requirements affects the overall cost, schedule and quality of GSD projects [190], [124]. Here, it is particularly challenging to manage and trace such changes. It is identified from literature [6], [86], [116] that, issues related to coordination and control of communication affecting requirement change management (RCM) arise when there is no effective information and knowledge-sharing mechanisms towards change management and traceability. This however requires a rigorous RCM process. RCM is not trivial in co-located software development; and with the presence of geographical, cultural, social and temporal factors in GSD, it makes RCM profoundly difficult for GSD. Existing RCM methods do not take into consideration these issues faced in GSD.

In recent years, cloud computing has been identified as a well suited deployment environment or delivery model for web-based services and especially for complex systems. Cloud computing [103] is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g. servers, storage, etc.) that can be rapidly provisioned and released with minimal management effort or service provider interaction. System engineering stands to benefit from the scalability, parallelism, cost-effectiveness, global multi-user access and flexibility features of the cloud platform. With such a global-focused service delivery model, cloud computing can suitably facilitate the deployment and operations of GSD.

However, the dependability challenge of cloud computing as a result of the rapid evolution of the cloud topology as well as uncontrolled or malicious impact of co-located systems, pose a major issue in its adoption. In the State of Cloud Security 2016 report [45] of the Cloud Security Alliance, it is noted that the challenge to the adoption of cloud computing is mainly security, compliance, lack of expertise and/or resources, and performance. That said, there have been some frequent occurrences

of cloud service failure events. Two of such events reported by [222] are first, the Azure cloud services of Microsoft that had two service interruptions attributed to system failure in March 2015. This affected most parts of their central and eastern United States service areas. Secondly, with regards to the Apple Cloud Services, about 11 iCloud-related features experienced serious technical issues on May 20, 2015. This includes iCloud Account and iCloud mail, by which 2 million iCloud users were affected. Here, the availability and reliability of cloud based systems are affected. In such a situation, the efficient operation of GSD can be adversely affected.

In order for GSD to be dependable in the cloud environment, two approaches need to be considered. Since the disciplined design of a system is as much relevant as the influence on the behaviour of that system by its environment [65], the design as well as the deployment environment are of immense importance. Firstly, the design of GSD systems must master the costs and the quality of the development of such software systems, relative to the rapid evolution of the topology of the cloud environment. Here, it is imperative to consider the overall effect of design decisions, the inherent trade-offs between quality attributes (such as availability, security, reliability, performance), and the trade-offs required to address user, system, and business requirements [20]. It is however essential to have a software system design approach that yields itself readily to an implicit analysis or evaluation method for cloud-based systems. Since the cloud environment is characterised by rapid interactions between quality attributes, the state-of-the-art of evaluating cloud-based systems in this context is lacking. Most notable software system evaluation methods mainly focus on independent quality attributes: modifiability [111], [26], [129], maintainability [148], flexibility [130], etc. Other methods [28], [24] that consider multiple quality attributes do not factor the "trade-off analysis" of system quality attributes.

Secondly, the concept of cloud accountability [77], [11], especially for dependability has not been adequately addressed. This means that the cloud behaviours affecting system dependability should be transparent to relevant parties, hence the call for accountability [11], [77], [187], [137]. It is important to mention that cloud accountability has been widely applied towards assuring the security of cloud-based systems in several research such as [82], [179], [23], [191], [201]. In our literature review, we found no work that employs the *forensic auditing techniques* of cloud accountability

towards the assurance of dependability especially for "availability" and "reliability". It is relevant that cloud users, such as system engineers, are assured of the availability and reliability of the cloud platform they use for GSD.

With these observations in mind, the problem statement for this thesis is formulated below, as well as the thesis approach to the problem.

1.2 Problem Statement and Approach

1.2.1 Problem Statement

To design and evaluate an effective mechanism for requirements change management and traceability, facilitated by a dependable framework to improve the chances of successfully undertaking cloud-based GSD to meet stakeholders' needs.

1.2.2 Research Aim and Objectives

In this thesis, we address the problems mentioned in the motivations by proposing a cloud-based Reactive Architecture (RA), which supports cloud-based system engineering. This alternative approach as against the current state-of-the-art, prioritises the definition of an effective information and knowledge-sharing mechanism towards change management and traceability, as well as providing a dependable framework for the proposed mechanism. Our main aim is to design and evaluate this Reactive Architecture. To achieve this aim, we propose an approach that relates two key components with their associated objectives:

1. A set of GSD management guidelines for requirements change management and traceability. Here, we identify the following objectives:

    - *OB1*: Define a change management and traceability (CM-T) process model, which applies a software process improvement method to ensure the maturity of the RCM and traceability processes,
    - *OB2*: Identify a standard quality management framework to facilitate a significant level of quality for the proposed CM-T process model,

- *OB3*: Validate the CM-T process model using an expert panel review process, and
- *OB4*: Demonstrate the defined management guidelines by applying it to an Airlock Control System case study.

2. A cloud-based Reactive Architecture to facilitate the defined GSD management guidelines. This approach is undertaken in two ways:

   (a) Provide a bespoke methodology that facilitates the design and analysis of small-to-medium size GSD architectures like the Reactive Architecture, interfacing with the unpredictable cloud environment. Here, the identified objectives are below:
   - *OB5*: Define the methodology for small-to-medium size GSD architectures.
   - *OB6*: Validate the methodology using a comparative study with current approaches,
   - *OB7*: Demonstrate the methodology by applying it to the design of the Reactive Architecture, and then
   - *OB8*: Analyse the quality attribute trade-off of the Reactive Architecture.

   (b) Present a method that is guided by a forensic model to perform virtual machine introspection for the purpose of assuring the dependability of the Reactive Architecture deployed to the cloud environment. Here, some objectives are identified:
   - *OB9*: Define the cloud accountability methodology.
   - *OB10*: Develop a cloud accountability system which facilitates the presented method,
   - *0B11*: Demonstrate the method by applying it to a cloud-based testbed of the Reactive Architecture, and
   - *0B12*: Conduct an evidence-based trust analysis on the derived evidence for the purpose of dependability assurance of the cloud-based Reactive Architecture.

1.3 Research Contributions

The research presented in this thesis makes several key contributions:

1. The design, development and evaluation of a novel Reactive Middleware, that supports a set of management guidelines for high quality GSD change management and traceability. The middleware facilitates a novel change management and traceability process model, within the context of quality management for GSD projects. An expert review panel process is conducted to assess the maturity of the process model. Also, an Airlock Control System case study is used to demonstrate the GSD management guidelines.
2. The proposal of a novel methodology for the design and analysis of small-to-medium size cloud-based systems. This method considers the unpredictable character and rapidly evolving topology of the cloud deployment environment, and its impact on dependability in the bespoke design of systems to be deployed to the cloud. The methodology targets systems that are classified within the range of small to medium size. A comparative study of the current state-of-the-art methods is initially undertaken to identify methods that present a high potential to remedy this challenge.
3. The design, development and evaluation of the Reactive Architecture, which is used for cloud-based system engineering. The Reactive Architecture presents critical components for system engineering such as the introduced Reactive Middleware, with a Shared Artefacts Repository, and a System Engineering Toolbox.
4. The proposal of a novel methodology for assuring cloud accountability in terms of dependability. A forensic analysis process is taken to guide the data collection, examination, evidence analysis, and reporting of information.
5. The design and development of the Cloud Accountability System to facilitate the cloud accountability methodology. The Cloud Accountability

System is used to conduct virtual machine introspection of some key components of the Reactive Architecture, where data is collected also from the Cloud Service Providers based on availability and reliability related metrics. Also, an evidence-based trust analysis of the reported information from the forensic process is conducted, to assure cloud users of the dependability of the cloud environment.

1.4 List of Publications

Our list of eleven publications are classified based on the technical chapters they support, or as other publications below:

- **Chapter 3**: Reactive Architecture
  (a) **D. E. Adjepon-Yamoah**, A. Romanovsky, and A. Iliasov. A Reactive Architecture for Cloud-Based System Engineering. In Proceedings of the International Conference on Software and Systems Process, ICSSP 2015 (Tallinn, Estonia), pages 77-81, August 2015.
- **Chapter 4**: Managing Change and Traceability
  (b) **D. E. Adjepon-Yamoah**. Towards Dependable Change Management and Traceability for Global Software Development. In Fast Abstract Proceedings of the 12th International European Dependable Computing Conference, EDCC 2016 (Gothenburg, Sweden), September 5-9, 2016.
  (c) A. Iliasov, L. Laibinis, E. Troubitsyna, **D. E. Adjepon-Yamoah**, and A. Romanovsky. Refinement-based Approach to Co-engineering Requirements and Formal Models. In (CS-TR-1456) Technical Report, Newcastle University, 13 pages, March 2015.

  The following three works are considered in Chapter 4 as cloud-based tool support in the Reactive Architecture:

(d) A. Iliasov, P. Stankaitis, **D. Adjepon-Yamoah**, and A. Romanovsky. Rodin Platform Why3 Plug-In. In Proceedings of the 5th International ABZ Conference 2016 ASM, Alloy, B, TLA, VDM, Z, ABZ 2016 (Linz, Austria), 6 pages, May 2016.

(e) A. Iliasov, P. Stankaitis, and **D. E. Adjepon-Yamoah**. Event-B and Cloud Provers. In Proceedings of the Automated Reasoning Workshop 2015 Bridging the Gap between Theory and Practice, ARW 2015 (Birmingham, UK), pages 11-12, April 2015.

(f) A. Iliasov, **D. E. Adjepon-Yamoah**, P. Stankaitis and A. Romanovsky. Putting Provers on the Cloud. In "Work In Progress" of the 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2015 (Turku, Finland), March 4-6, 2015.

- **Chapter 5**: Designing Architectures for Global Software Development

(g) **D. E. Adjepon-Yamoah**. cloud-ATAM: Method for Analysing Resilient Attributes of Cloud-Based Architectures. In book: Software Engineering for Resilient Systems, Chapter: Engineering Resilient Systems, pp.105-114.
This publication introduces our method for designing dependable small-to-medium size cloud-based architectures. Here, it is demonstrated by designing the Reactive Architecture.

- **Chapter 6**: Cloud Accountability

(h) **D. E. Adjepon-Yamoah**, and Z. Wen. Assuring Dependable Cloud-Based System Engineering: A Cloud Accountability Method. In Proceedings of the 12th International European Dependable Computing Conference, EDCC 2016 (Gothenburg, Sweden), September 5-9, 2016.

- **Chapter 7**: Evaluation

(i) **D. E. Adjepon-Yamoah**. cloud-ATAM: Method for Analysing Resilient Attributes of Cloud-Based Architectures. In book: Software Engineering for Resilient Systems, Chapter: Engineering Resilient Systems, pp.105-114.

In this chapter, the method introduced in Chapter 5 is used to conduct the trade-off analysis of relevant quality attributes of the developed Reactive Architecture.

- **Other Publications**:

  (j) A. Iliasov, P. Stankaitis, and **D. Adjepon-Yamoah**. Static Verification of Railway Schema and Interlocking Design Data. In book: Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification, pp.123-133.

  (k) A. Iliasov, P. Stankaitis, **D. Adjepon-Yamoah**, and A. Romanovsky. A Rodin Plug-in for Constructing Reusable Schematic Lemmas. In Proceedings of the 6th Rodin User and Developer Workshop, pp.5-6, 23rd May 2016.

1.5 Thesis Structure

Chapter 2 - Background and Related Works

Presents an overview of the research areas relevant to this work, and terms that will be used in later chapters.

Chapter 3 - Reactive Architecture

The solutions to the problems introduced in the previous chapters are described as high-level functions of a cloud-based framework proposed in this thesis as the Reactive Architecture. Here, all the solutions are justified, and the definition of our original contributions are clearly presented.

Chapter 4 - Managing Requirements Change and Traceability

Presents the Reactive Middleware component of the Reactive Architecture, that facilitates a set of GSD management guidelines for change management and traceability. We propose a novel change

management and traceability process model, within the context of quality management of GSD.

Chapter 5 - Designing Architectures for Global Software Development

This chapter presents a novel methodology for designing and analysing dependable small-to-medium sized cloud-based systems. Also in this chapter, the cloud-focused Architecture Trade-off Analysis Methodology (i.e. cloud-ATAM) is demonstrated by designing the Reactive Architecture to meet stakeholders' requirements.

Chapter 6 - Cloud Accountability

Presents a novel cloud accountability methodology for assuring the dependability of cloud-based systems. This chapter defines the methodology, and introduces it as a forensic process model in terms of data collection, examination, analysis and reporting. Also, an evidence-based trust analysis approach is introduced as a means of providing trusted information for the assurance of the dependability of the cloud environment.

Chapter 7 - Evaluation

Presents the evaluation and analysis of the Reactive Architecture and its components. First, the change management and traceability process model facilitated by the Reactive Middleware introduced in Chapter 4 is reviewed by an expert panel review process. Also, the GSD management guidelines facilitated by the Reactive Middleware is demonstrated using an Airlock Control System. Secondly, the Cloud Accountability System facilitating the Cloud Accountability Method introduced in Chapter 6, is demonstrated by applying it to a cloud-based test-bed of some components of the Reactive Architecture. Then, an evidence-based trust analysis is conducted on the information generated. Finally, the cloud-ATAM introduced in Chapter 5 is validated through a comparative analysis with state-of-the-

art methods for analysis software architectures. Also, the two-staged qualitative analysis approach (i.e. Utility Tree mechanism and Stakeholders' Brainstorming mechanism) provided by cloud-ATAM is used to analyse the Reactive Architecture.

Chapter 8 - Conclusions

Presents a summary of the findings presented throughout this document and speculates on potential future research made possible by our findings.

Chapter 2

Background and Related Work

This chapter provides an overview of the relevant background material motivating and underpinning the work conducted in this thesis. We begin by introducing GSD and approaches with the potential of improving the development experience in Section 2.1. An overview of software architecture as a suitable GSD framework is presented in Section 2.2. Section 2.3 presents a discussion about cloud computing as a deployment environment for GSD, and its accountability in the context of the provided service level agreement. A discussion on trust assurance methods is provided in Section 2.3.5. We draw our conclusions in Section 2.4.

2.1 Global Software Development

2.1.1 Overview

Many software development projects are globally distributed in nature [180], resulting in the evolution of the term Global Software Development (GSD) [83], [85]. Many factors motivate the need to implement models such as GSD, including: the need to capitalise on globally dispersed resources, wherever they are located [181]; the business advantages of proximity to the market, including

knowledge of customers and local conditions, as well as the good will engendered by local investment [218]; the quick formation of virtual corporations and virtual teams to exploit market opportunities; and, pressure to improve time-to-market by using time zone differences in "follow-the-sun" development [83]; the need for flexibility to capitalize on merger and acquisition opportunities wherever they present themselves [21], [84]. This model typically involves a team in a so-called "home" site, generating requirements based on customer interactions, and farming out parts of those requirements to several geographically diverse "global" sites for implementation [150].

GSD seems to have become a business necessity for various reasons, including cost, scarcity of resources, and the need to locate development closer to the customers. In fact, it is fast becoming a pervasive business phenomenon [50]. Some companies like IBM, British Airways, British Telecom and General Electric have moved parts of their internal software development operations to countries like India and Ireland [118]. Fundamentally, GSD involves communication for information exchange, coordination of teams, activities and artefacts so they contribute to the overall objective, and finally the control of teams [46].

2.1.2 Requirement Management

Communication, coordination and control issues arise largely when there is no effective information and knowledge-sharing mechanisms [6], [86]. In GSD, due to lack of common understanding between geographically dispersed teams, requirements management is particularly difficult. Problems in the requirements phase have a wide impact on the success of software development projects, but have an even greater impact on the success of GSD projects [138]. Here, changes to requirements have to be adequately managed, effected, traceable, and all relevant stakeholders have to be informed. Changes that are inadequately managed affect product quality [190], [124]. Hence, the requirement change management plays a vital part of software requirements engineering process in GSD. However, the communication issue and requirement change management in GSD are given very little consideration as compared to localised software development

[115], [30], [119], [117]. In order to make any meaningful headway for GSD, the closely related concepts of requirements change management, and traceability of these requirements through the development life-cycle in the context of the distributed development resources, need to be appropriately considered.

We therefore briefly introduce and discuss current research approaches for change management (Section 2.1.2.1) and traceability (Section 2.1.2.2) as critical areas that have the potential of facilitating effective requirements management for GSD.

2.1.2.1 Change Management

An important aspect of software system engineering is change management. Since change in the system engineering process is inevitable and has a high influence in determining the success of the process, it must be managed with utmost discipline. Change management is a disciplined process for introducing required changes into the information technology environment [214], [188]. It ensures that changes to software (and sometimes, hardware) are managed and conducted in a way that costs are met, risks are reduced, and that the business needs and goals of a company are satisfied with the highest degree of confidence and optimisation. We first discuss some relevant models for change management in collocated software development.

Olsen's Change Management Model

This change management model [161] identifies the software development process as a queue of changes that need to be made. A primary assumption of this model is that all work done by software designers change. Here, the model can be applied to both software development and maintenance as it is not life-cycle dependent (refer to Figure 2.1). The sources of changes are made available by the users who suggest possible requirement changes. These changes are then passed to the "manage change" section where these changes are managed by change managers. The approved changes are passed on to the *implementation* section where necessary changes are made in the software. After completing implementation, "verification" begins by testing code and by inspecting papers.

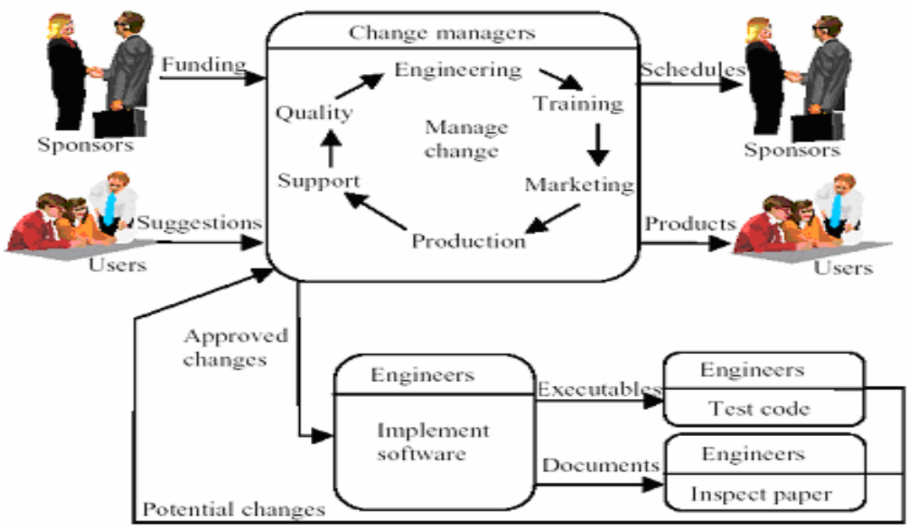


Figure 2.1: Olsen's Change Management Model [161]

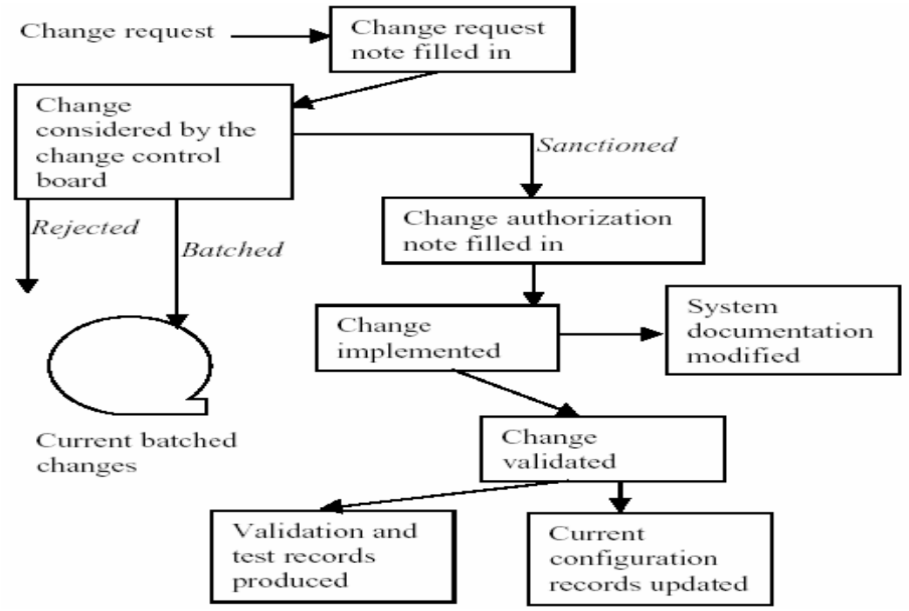


Figure 2.2: Ince's Change Process Model [100]

When a change has been implemented and verified it is then passed back to change managers who will then release the change in a product.

Ince's Change Process Model

Ince's model [100] focuses on how software configuration management relates to software change management. This model (refer to Figure 2.2) has two main sources of change requests, i.e. customer and development team. In order for the change process to be initiated, a change request must be initiated in a software project. All such change requests are recorded in a change request note. The change control board then considers the suggested change. The change control board can reject the change, batch the change (the change will take place but not immediately) or accept the change. If the request for the change is successful, a change authorisation note must be filled. After this the change can be implemented and a system's documentation is modified. After implementation the change is validated. Validation and test records are then produced to document the changes that have taken place. Finally, the configuration records are updated and the staff is informed about the new changes.

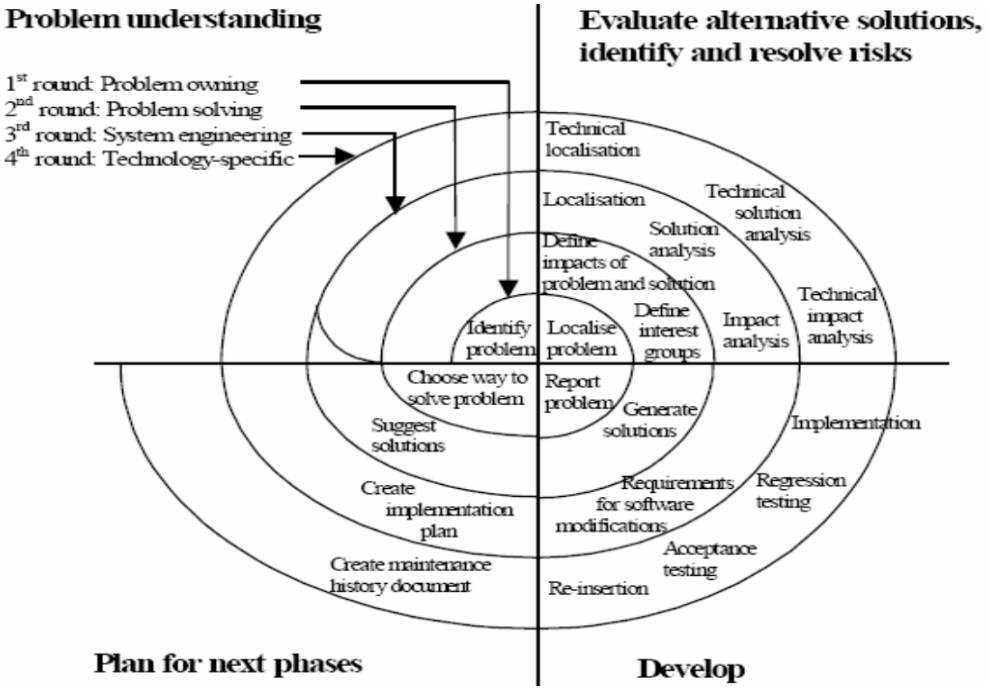


Figure 2.3: Spiral Like Change Management Process [145]

Spiral-Like Change Management Process

This model [145] presents the change management process as a four-cycle or round process (refer to Figure 2.3): the first cycle is "problem owning", the second is "problem solving", the third is "system engineering", and the final cycle is *technology-specific*. The first cycle of this model is the initial cycle; here the owner of a problem begins this cycle. A problem can be a request to add a new feature or services in the system, or to fix a problem in the existing system. At the end of the first cycle the owner decides whether a change needs to be made, and if necessary, how it should be accommodated. The second cycle is required only if the change needs to be investigated from a non-technical viewpoint. This leads to the third cycle, which is the planning stage. This involves the examination of the change from a system viewpoint, and makes implementation plans for the final cycle. The fourth cycle generates, implements and verifies the technical solution. The change is finished and the results of this change are recorded.

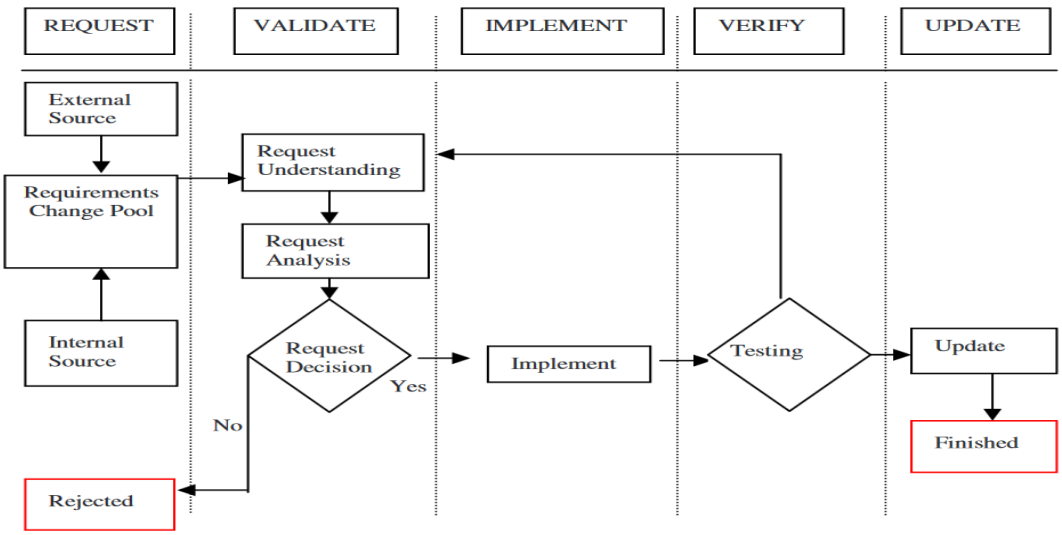


Figure 2.4: Requirement Change Management Model [153]

At this point, our attention is drawn to relevant change management models and approaches that focus on requirements management for GSD. We begin by discussing a prominent work [153] in this area, that further motivates some similar works ([115], [200], [125]) during the decade. We conclude this section by also considering other approaches that suggest agile and hybrid methods as potential solutions.

Niazi's Requirement Change Model

In this work [153], a requirements change management model as well as its framework [138] are presented for GSD. This work implements a CMMI Level 2 specific practice - SP 1.3-1 - manage requirements changes. The model is based on both an empirical study that have been carried out, and then an extensive literature review of software process improvement (SPI) and requirements engineering (RE). The model is based on five core elements identified from literature and interviews: request, validate, implement, verify and update (refer to Figure 2.4). Within each of these elements, there are some specific activities that need to take place during requirements change management process. This work shows that the requirements change management model is clear, easy to use and can effectively manage the requirements change process. However, more case studies are needed to evaluate this model in order to further assess its effectiveness in the domain of RE process.

It is also important to point out that the agile concepts and methods [19], [51], [146] are also receiving attention in this area. Agile and lean practices are used to support a more dynamic handling of software products, especially in terms of RE. The notable approach found from literature is Scrum [168], [107]. Also, others [211] point to the use of hybrid approaches as viable options. That said, [136] identifies that GSD and agile seemingly creates a contradiction, since distributed software engineering requires a number of rules and formalisms to coordinate the different teams spread across the globe, but agile software development is, on the other hand, strongly driven by immediate and direct communication and collaboration of people - quite often in small and co-located teams. Hence, agile approaches need more collaboration which cannot be effectively achieved

in a distributed setting for GSD [88]. Also, the lack of requirements documentation causes problems especially when managing changes to requirements and maintaining traceability [151]. Furthermore, Wagner [219] emphasises that the lack of written, and traceable requirements can make it difficult to maintain developed tool support in the long run.

2.1.2.2 Traceability

Software requirements traceability enables software engineers to ensure consistency among the artefacts created during the development and maintenance of software products [55]. Requirements express needs and constraints of a software product, and traceability allows to describe and follow requirements steps [171],[122]. That said, traceability facilitates an easier verification and validation of requirements. Hence, traceability supports software management, software evolution, and validation. Here, when changes are made in a software product, traceability is fundamental to analyse the impact of such changes. Also, it facilitates the understanding, capturing, tracking, and verification of software artefacts, their relationships, and dependencies with other artefacts during the software life-cycle [71].

An effective traceability approach depends on several factors such as architecture, technical modelling tools, among others. The implementation of traceability in the industry is still a challenge. Literature [42; 55; 72; 73] has shown some reasons for that: cost of implementation, different viewpoints of stakeholders, difficulties of maintaining updated requirements' information, and integrating all generated data or artefacts from software development life-cycle. From these reasons, it is apparent how critical requirements traceability is to the software development RCM processes. Some approaches have been introduced to address various aspects of the traceability challenge, and are discussed briefly below.

Focusing on the communication challenge introduced by the inevitable involvement of a group of stakeholders in GSD, [13] presents a multi-perspective requirements traceability (MUPRET) framework which deploys ontology as a knowledge management mechanism to intervene mutual "understanding" with-

out restricting the freedom in expressing requirements differently. This work provides an approach to handle and resolve issues resulting from such heterogeneity, by tracing and managing changes relating to GSD requirements. Ontology matching as a reasoning mechanism is applied here to automatically generate fine-grained traceability relationships. Such relationships are identified by deriving semantic analogy of ontology concepts which represent requirements elements. Finally, the precision and recall of these traceability relationships generated by the framework are verified by comparing with a set of traceability relationships manually identified by users as a proof-of-concept of the framework. However, this framework is limited in two ways: (1) it emphasises on tracing multi-perspectives in the requirements analysis phase, and (2) it also focuses on requirements that are expressed in terms of natural language.

In this work [170], the concept of Just-In-Time RE which presents the idea of *reactivity* to requirement changes is applied to agile projects to record their requirements (so-called feature requests) in an issue tracker. This work observes that in open source projects, there are large networks of feature requests that are linked to each other. They stress that in both situations when trying to understand the current state of the system, and to understand how a new feature request should be implemented, it is important to know and understand all these tightly related feature requests. However, the authors identify that there is still a lack of tool support to visualise and navigate these networks of feature requests. The first step the authors provide in this direction is to see whether they can identify additional links that are not made explicit in the feature requests, by measuring the text-based similarity with a Vector Space Model (VSM) [135] using Term Frequency - Inverse Document Frequency (TF-IDF) [105] as a "weighting factor". Also, they show that a high text-based similarity score is a good indication for related feature requests. With this in place, they conclude that a TF-IDF VSM can aid the creation of horizontal traceability links. This then provides a new perspective for developers exploring the feature request space. That said, there are three shortcomings of this work: (1) there are no measures on thresholds, recall and precision for the retrieval of those links before hand, (2) there is no tool support for the automatic creation of

feature request networks so that developers can benefit more from the horizontal traceability links, and (3) it does not compare additional information retrieval approaches for a given problem domain.

Finally, the main focus of the authoritative European CESAR project (Cost-Efficient Methods and Processes for Safety Relevant Embedded Systems) [178] is the facilitation of full traceability of a requirement throughout the development chain and even the entire supply chain. To achieve this aim, CESAR adopted interoperability and traceability technologies proposed by the Open Services for Life-cycle Collaboration (OSLC) [27], [31], [160]. OSLC is a cross-industry initiative aiming to define standards for compatibility of software life-cycle tools. It is critical to the CESAR project because OSLC aims to make it easy and practical to integrate software used for development, deployment, and monitoring or tracing applications. The elementary concepts and rules are defined in the OSLC Core Specification which sets out the common features that every OSLC Service is expected to support using the terminology and generally accepted approaches of the World Wide Web Consortium (W3C). Here, the OSLC-CM (Change Management) specification provides details of its approach to traceability by adopting known technologies like resource description framework (RDF) and linked-data [80]. Another significant work [142] that looks at a common industry challenge where a system model which is composed of several sub-models, and which may have been developed using different tools. In this work, the authors present a new approach facilitated by OSLC to support traceability in the OpenModelica software where the traceability information is exchanged with other life-cycle tools through a standardised interface and format. The main limitation of this work is that, it focuses largely on the requirements and specification phases of the development life-cycle.

2.1.3 Artefact Repositories

Central to the achievement of a high standard of RCM and traceability, lies in a common facility to support communication and collaboration of GSD team resources. This facility is a common repository of artefacts. Such a repository

stores all artefacts that are either created or generated during the SDLC phases. We consider some of such repositories below.

The Open Source Component and Artefact Repository System (OSCAR) [33] is provided with in the European Commission backed GENESIS project [34]. This work introduces a software artefact repository that provides its contents with some awareness of their own creation. To achieve such awareness, the concept of "active" artefacts are introduced and are distinguished from their passive counterparts by their enriched *meta-data model*. Such a meta-data model reflects the work-flow process that created them, the actors responsible, the actions taken to change the artefact, and various other pieces of organisational knowledge. This enriched view of an artefact is intended to support re-use of both software and the expertise gained when creating the software. A distinguishing feature from other organisational knowledge systems is that, the meta-data is intrinsically part of the artefact and may be populated automatically from sources including existing data-format specific information, user supplied data and records of communication. The authors emphasise the increased importance of such a feature in the world of "virtual teams" where transmission of vital organisational knowledge, at best difficult, is further constrained by the lack of direct contact between engineers and differing development cultures. Our work draws inspiration from the notion of active artefacts and their awareness for change.

Another repository [54] identified in literature is used for the storage of artefacts for controlled experimentation in software testing. This repository is presented within an infrastructure that supports controlled experimentation with testing and regression testing techniques. This infrastructure primarily stores, artefacts (programs, versions, test cases, faults, and scripts) that enable researchers to perform controlled experimentation and replications. However, the challenges of this infrastructure is that it has no mechanisms for artefacts sharing, as well as to facilitate community development of the infrastructure. Also, it has no mechanisms for the contribution of additions to it in the form of new fault data, new test suites, and variants of programs and versions that function on other operational platforms.

Also, Sysiphus [37] is identified as a distributed environment providing a uniform framework for system models, collaboration artefacts, and organisational models. Fundamentally, Sysiphus encourages participants to make communication and issues explicit in the context of system models and become aware of relevant stakeholders. The authors specifically focus on the problem of externalising issues with their context, stakeholders, and organisational roles in distributed settings such as GSD. This work addresses the challenge of capturing sufficient knowledge as a side effect of development, while structuring it for long-term use.

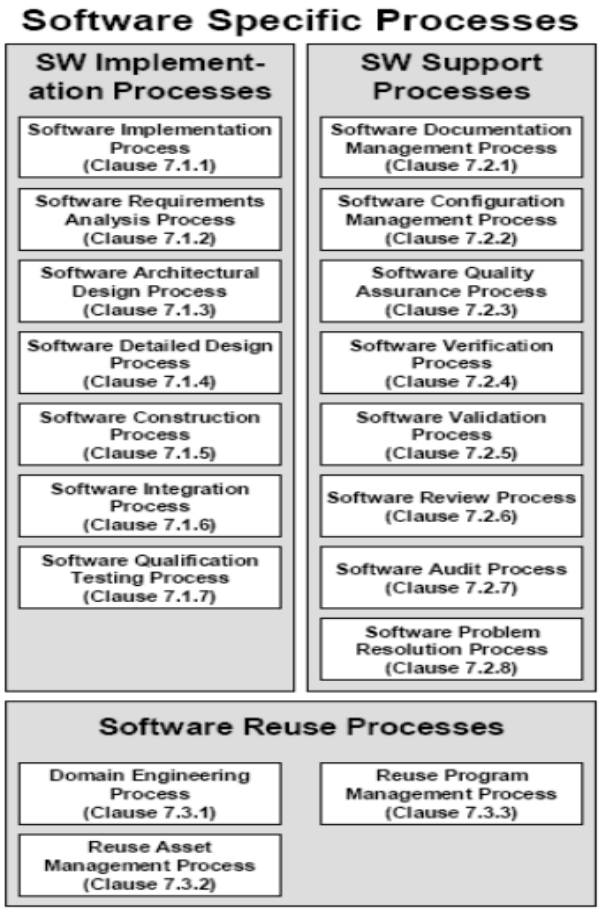


Figure 2.5: ISO/IEC 12207 Software Processes [102]

In practice, software version control repositories such as Git [197], GitHub [70],

SVN [10], and Mercurial [143]. are largely being used in the industry. The main challenges we identified in our experience of this set of repositories are that, their application is often focused narrowly on the development (or implementation) phase of SDLC, and also change management is often enforced based on trust of the change actor (i.e. a Software Developer).

2.1.4 Global Software Development Tool Support

The current challenges in Global Software Development (GSD) necessitate support from software tools with special features. We briefly consider some works from literature that identify tools that facilitate some operations that are crucial in GSD.

The first work [174], presents a set of tools with special features and explains why these features are desirable for the tools in the context of GSD, and how these features are related to the principal challenges in this environment. The authors therefore present a survey of the tools that provide such features. The tools included in the survey were classified through the use of the ISO/IEC 12207 standard processes [102] (see Figure 2.5) to determine which process is supported by each tool. Generally, they provide two main groups. The first group is made up of tools that support Project Processes such as ActiveCollab [4], Assembla [14], Maven [208], Jira [15], Rational Team Concert [92], etc. Here, these tools mainly support activities relating to project management. These tools specifically integrate features to support the Project Planning Process and Project Assessment and Control Process of the ISO/IEC 12207. Also, the second group is composed of tools supporting Implementation Processes such as Rational Requirements Composer [93], IBM Rational DOORS [92], etc. The processes included in this group are Software Requirements Analysis Process, Software Architectural Design Process, Software Detailed Design Process, Software Construction Process and Software Integration Process.

Another work [126], identifies that the distribution of tasks to sites is one central activity in global software development project planning. Due to the large

number of assignment possibilities, tool support seems to be adequate for supporting the evaluation and selection of task assignments. This work presents TAMRI, a planning tool for identifying task assignments based on multiple criteria and weighted project goals. Its implementation combines a distributed systems approach with Bayesian networks. The tool can be adapted to specific organisational environments by exchanging the underlying Bayesian network. The authors present an overview of task distribution approaches, gives three application scenarios for the tool, and shows the implementation of the tool as well as its application in the scenarios.

Also, in this work [49], the authors propose TIPMerge, a novel tool that recommends developers who are best suited to perform merges, by taking into consideration developers' past experience in the project, their changes in the branches, and dependencies among modified files in the branches of distributed projects. They evaluate TIPMerge on 28 projects, which included up to 15,584 merges with at least two developers, and potentially conflicting changes. On average, 85% of the top three recommendations by TIPMerge correctly included the developer who performed the merge. Best results of recommendations were at 98%. Their interviews with developers of two projects reveal that in cases where the TIPMerge recommendation did not match the actual merge developer, the recommended developer had the expertise to perform the merge, or was involved in a collaborative merge session. This work like most past works, considers the fact that the integration of changes across branches is not easy, and often leads to failures. They also emphasise that there has been little work to recommend developers who have the right expertise to perform a branch integration. That said, the identified challenge with this work is that, without a *consistently stringent facility* to effect changes through the merging process, reliance or trust based on past experiences can be subjective and not very effective.

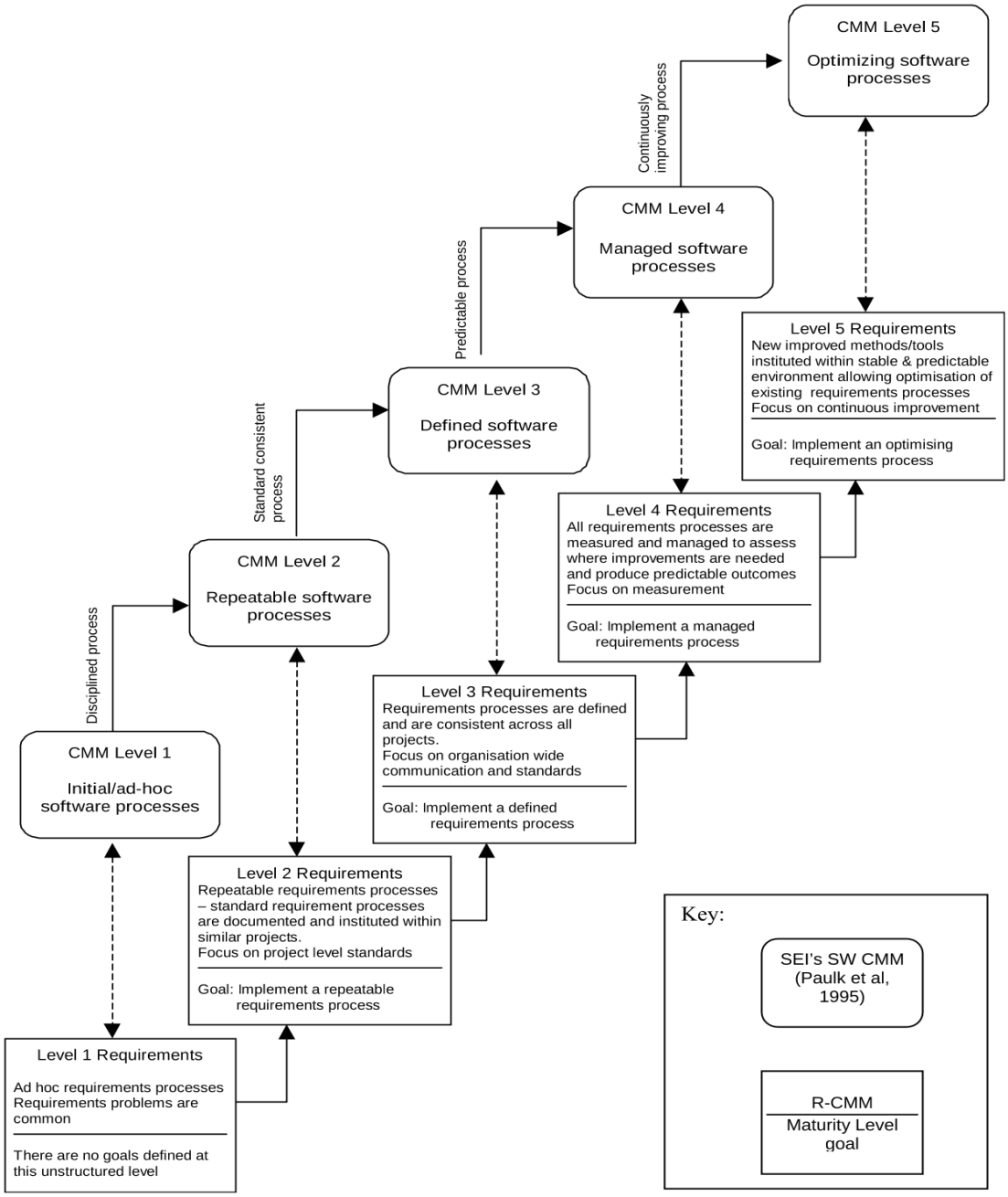


Figure 2.6: CMMI's Maturity Levels [22]

2.1.5 Software Process Improvement Methods

Software Process Improvement (SPI) has been a widely used approach promoted by software engineering researchers, with the intention of helping organisations to develop high quality software more efficiently [153]. SPI frameworks such as the process capability maturity models (e.g. Capability Maturity Model Integration (CMMI) [41] and ISO/IEC 15504 (SPICE) [155], [154], [156]) are provided for defining and measuring processes and practices that can be used by organisations that develop software. We will briefly discuss the CMMI model, which is the most widely used SPI process capability maturity model [7]. Also, ISO/IEC 15504 (SPICE) will be discussed, and then an overview of the current state of SPI frameworks is presented to conclude this section.

2.1.5.1 Capability Maturity Model Integration (CMMI)

The CMMI model is consistent with the international standard ISO/IEC 15504 [226]. The most well-known representation of CMMI is the "staged" representation, which has five "levels" of process maturity for organisations [153] (refer also to Figure 2.6). Here, each of the five levels is composed of several "process areas" [205], and for each process area, there are several *goals* that are defined, and in turn contain different *practices*. To reach a maturity level, the goals of the process areas for that level and all lower levels must be satisfied. These practices help in understanding how to achieve maturity goals, and serve as examples of the activities to be addressed when undertaking a SPI programme. Level 2 maturity is the first level that defines a collection of "process capabilities" that focus on supporting process areas, but also includes some project management and engineering process areas. The two goals in Level 2 are Specific Goal one (SG1): *Manage Requirements*, and Generic Goal two (GG2): "Institutionalise Managed Process". To achieve CMMI maturity level 2, projects must ensure that processes are planned and executed in accordance with a defined policy; projects must employ skilled people who have adequate resources to produce controlled outputs; must involve relevant stakeholders; are monitored, controlled, and reviewed; and are evaluated for adherence to their

process descriptions. Here, the process discipline shown by maturity level 2 helps to ensure that existing practices are retained during times of stress.

CMMI - Continuous	SPICE
Process Categories	
Process Management	Organization
Project Management	Management
Engineering	Engineering
Support	Support
	Customer-Supplier
Capability Levels	
Optimizing	Optimizing
Quantitatively Managed	Predictable
Defined	Established
Managed	Managed
Performed	Performed
Incomplete	Incomplete

Figure 2.7: Comparison of CMMI & SPICE Models

2.1.5.2 ISO/IEC 15504 (SPICE)

The International Standards Organisation's ISO/IEC 15504 standard for Software Process Improvement and Capability Determination (SPICE) [156] is an important model for software process assessment, improvement and capability determination. The ISO/IEC 15504 much like CMMI comprises of maturity levels and each maturity level has attributes assigned to it. An organisation fulfilling all the attributes of a level is said to be on that maturity plane. Also, CMMI and SPICE models are similar in their major classifications: "Process Categories", "Capability Levels", and composing processes (see Figure 2.7). In terms of differences, an obvious one is that SPICE introduces one new process area (i.e. Customer-Supplier). Also, researchers (such as [57]) identify the SPICE model as relatively complex than the CMMI model; the SPICE model contains extra process areas (i.e. Operational Process, Management Process, Process Alignment Process).

2.1.5.3 Overview of SPI Frameworks

The challenge here is that, the failure rate of SPI initiatives is generally very high; estimated as 70% [152]. Also, it is a very complex set of practices or activities, with its accompanying costs. The significant investment and limited success are reasons for many organisations (especially small to medium-sized) being reluctant to embark on a long path of systematic process improvement. Fundamentally, the population of organisations that have adopted process capability maturity model is only a part of the entire population of software developing organisations [134], [153]. It takes significant time to fully implement an SPI initiative [155], [154], [156]. Some experts (such as [22]) in SPI have attempted to provide a streamlined set of activities/practices for CMMI, that are reasonably applicable in the industry. This above mentioned research largely motivates our work. CMMI is a popular choice because it is freely available, trained resources and quality professionals of CMMI are available, and there is also not much awareness in industry about ISO/IEC 15504.

2.1.5.4 Using An Expert Panel In A Model Validation Exercise

From [56], it is noted that small samples of experts can be used to develop and test explanations, and more so in the early stages of model development. Others such as [59], have used small samples to gain expert feedback to evaluate and support model development. The value of expert knowledge is also recognised in an evaluation of software quality that suggests methods to formally capture expert judgement [183]. The reliability of using expert judgement is shown in other work such as [131], [120]. Some notable studies such as [22], [58], [59], and validate improvement models and measurement 'instruments' by inviting a panel of experts to complete a detailed questionnaire.

2.1.6 Quality Management of Software Processes

A widely applied approach for managing the quality of software processes is the Project Management Body of Knowledge (PMBOK) [176]. It is a collection of

processes and *knowledge areas* accepted as best practice for the project management profession. PMBOK is an internationally recognised standard (ANSI/PMI 99-001-2008 and IEEE 1490-2011) that provides the fundamentals of project management. PMBOK introduces *five basic process groups*, as well as *ten knowledge areas*, which is typical of almost all projects. The basic concepts are generic and hence applicable to a wide range of projects, programmes and operations. The five basic process groups are: (1) *Initiating*, (2) *Planning*, (3) *Executing* , (4) *Monitoring and Controlling*, and (5) *Closing*. Processes overlap and interact throughout a project or system engineering phase. In PMBOK, processes are described in terms of: i) *inputs* - such as documents plans, designs, etc.; ii) *tools and techniques* - such as mechanisms applied to inputs; and iii) *outputs* such as documents, products, etc. Also, the ten knowledge areas are: (1) project integration management, (2) project scope management, (3) project time management, (4) project cost management, (5) project quality management, (6) project human resource management, (7) project communications management, (8) project risk management, (9) project procurement management, and (10) project stakeholder management. Each knowledge area contains some or all of the project management processes.

Other approaches such as OSLC [160] and CMMI [41] provide features to achieve or support quality management in software processes. First, OSLC as an industrial standard that targets tools used during a products life cycle and enables their integration and interoperability [63]. To enable interoperability, different specifications, called domains, need to be provided. More precisely, an OSLC Domain is one ALM (Application Lifecycle Management) or one Product Lifecycle Management (PLM) topic area such as Quality Management (QM), Architecture Management (AM), Requirements Management (RM). With regards to quality management, OSLC defines a common set of resources, formats and RESTful services for Quality Management tools to interact with other Application Lifecycle Management (ALM) tools. This includes test execution tools such as functional and performance test tools in addition to source control, defect management, and code development tools. Second, CMMI integrate traditionally separate organisational functions, set process improvement goals and

priorities, provide guidance for *quality processes*, and provide a point of reference for appraising current processes.

2.2 Software Architecture

2.2.1 Overview

With the increasing size of software systems now spanning global scale, the concept of software architecture continues to provide a disciplined process for planning and building frameworks that can sustain such systems. Even though different researchers provide different perspectives to the *disciplined processes* of software architecture, it is strongly argued that they permit designers to describe complex systems using abstractions that make the overall system intelligible [66]. Furthermore, software architecture is identified as a more disciplined basis for architectural design that has the potential to significantly improve the ability of software engineers to construct effective software systems. Some leading literature in system engineering indicate that software architecture can have a positive impact on at least four aspects of software development: *understanding* ([207], [67], [65]), *reuse* ([169], [18]), *evolution* ([87]), and *analysis* ([20], [169]).

Software architecture design focuses on understanding a system, its reusability, evolution and analysis to meet a desirable quality, as well as controlling development cost. The quality of such systems are usually identified as dependability quality attributes such as performance, reliability, availability, security, etc. In essence, software architecture seeks to build a bridge between business requirements and technical requirements. A good design is sufficiently flexible to be able to handle the natural drift that occurs over time in hardware and software technology, as well as in user scenarios and requirements [65]. It is crucial to consider the overall effect of design decisions, the inherent trade-offs between quality attributes, and the trade-offs required to address user, system, and business requirements [20]. In view of this, we first provide an overview of the dependability of software architecture, and briefly introduce some relevant

quality attributes. Then, an assessment of some widely used software architecture evaluation methods are discussed in light of quality attribute sensitivity and trade-off analysis below.

2.2.2 Software System Dependability

Dependability is defined in [16] as the ability to avoid service failures that are more frequent and more severe than is acceptable. Here, the concept of dependability leads to *trust* (i.e. accepted dependence). In this context, the delivered service is the behaviour of the system, as it is perceived by its *user*, which is another system that interacts with the *provider* and receives the service. Another work [164], points out that dependability advocates user trust and customer confidence from a value perspective in doing business, it affects the bottom line of an organisation in product development or service provision demanding attention to ascertain dependability performance value. However, to assess whether a system satisfies the requirements of dependability is not an easy task, especially when complex and globally distributed systems are involved. Moreover, such an assessment is further hampered by the fact that dependability is an encompassing concept which embraces a set of different attributes, whose emphasis and importance depends on the characteristics of the system or application being analysed. We provide further definitions and taxonomy of dependability in the following sections.

2.2.2.1 Definitions and Taxonomy

As mentioned earlier, dependability is an integrating concept which embraces a number of different, but complementary attributes [16], that correspond to different viewpoints of the system. Here, we briefly introduce some system quality attributes:

- **Availability** - ability of a system to be in a state to perform as required;
- **Reliability** - ability of a system to perform as required, and without failure for a given time under a given set of conditions;

- **Safety** - absence of disastrous results for the users and the environment;
- **Performance** - ability of a system to satisfy the service application for a series of operational time interval;
- **Security** - ability of a system to prevent unauthorised access and to protect from intrusions without revealing sensitive information;
- **Recoverability** - ability of a system to recover from a failure, without corrective maintenance.

We further discuss some of these quality attributes, which can quantify the dependability of software systems in different perspectives. We begin by providing an overview of the *dependability metrics* that facilitate the definition of a set of quality attributes.

Metrics are commonly used in engineering as measures of system performance for a given quality attribute [52]. Most often, metrics are computed based on an analytical model that describes the behaviour of a system as a function of parameters associated with these attributes. Generally, this analytical model contains the notion of state, which when combined with the inputs to a system provides a way to uniquely identify the system at any time. Here, the *probability* of the occurrence of a given state as a function of time, and the *average time* before a given system state occurs are considered to compute such models. Such dependability metrics are the:

(a) *Mean Time To Failure (MTTF)*: Average time a system takes to fail. This metric is often referred to as the *average uptime*.

(b) *Mean Time To Recover/Repair (MTTR)*: Average recovery time for a system.

(c) *Mean Time Between Failures (MTBF)*: Average time between two successive failures.

(d) *Operational Time*: The total time an operational system is under observation.

We introduce the three quality attributes and their corresponding dependability metrics that define them below.

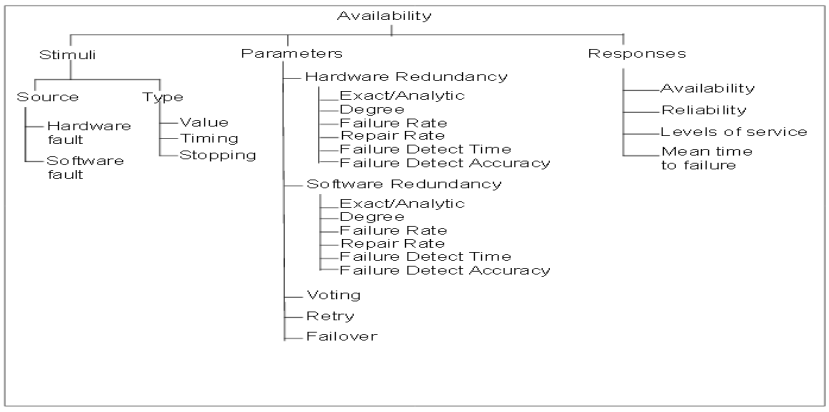
Availability of Software Architecture: As introduced earlier, availability of a system is its ability to be in a state to perform as required. It also describes a system's behaviour in the presence of error treatment mechanisms such as *redundancy* (i.e. system replication, checkpointing). Availability can be classified as either *instantaneous* or *steady-state*. *Instantaneous availability* is the probability that a system is performing correctly at time, t, and its equal to *reliability* for non-repairable systems. On the other hand, *steady-state availability* is the probability that a system will be operational at any random point of time. Usually in practice, the *steady-state availability* is considered when observing an operational system over a period of time. Here, availability is the ratio of uptime and the sum of uptime, scheduled downtime, and unscheduled downtime. Mathematically, steady-state availability (A) is expressed as:

$$A = \frac{MTBF}{MTBF + MTTR} \tag{2.1}$$

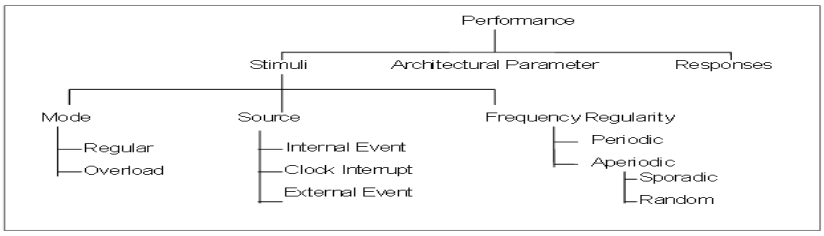
Reliability of Software Architecture: The reliability of a system (i.e. software, hardware) is defined to be the probability that the system performs as required, without failure, for a given time interval, under given conditions. Reliability is a function of time, and it gets smaller as time increases. Here, the assumption is that the system is fully operational at *t=0*, and it indicates failure-free interval of operation. Reliability (R) can be expressed mathematically as:

$$R = e^{-(\frac{Time}{MTBF})} \tag{2.2}$$

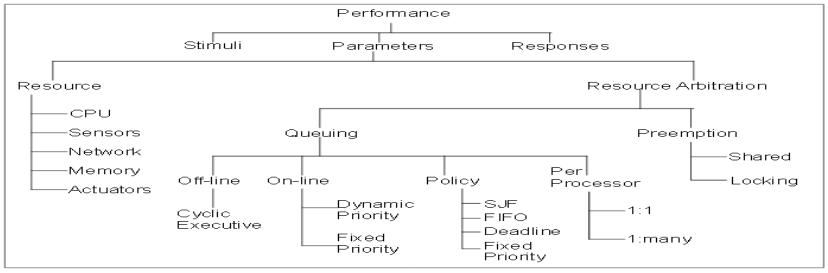
Performance of Software Architecture: Performance involves the allocation and adjustment of resources in order to meet the timing requirements of a system. In this situation, timing behaviour is determined by allocating resources according to the demands for such resources, choosing between conflicting re-



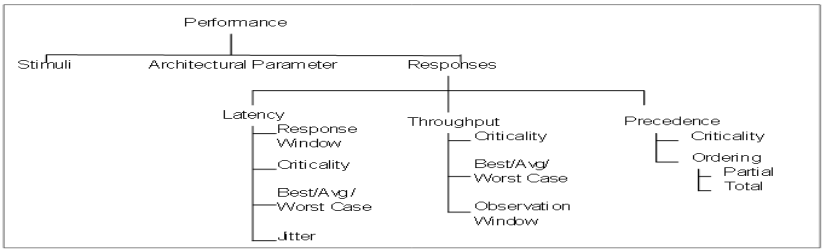
(a) Availability Characterisation



(b) Performance Characterisation - Stimuli



(c) Performance Characterisation - Architectural Parameters



(d) Performance Characterisation - Responses

Figure 2.8: Quality Attributes Characterisation (from [43])

quests for resources, and managing resource usage [140]. To manage performance, three main techniques are worth mentioning:

- *resource allocation* - these are policies for meeting resource demands by allocating various resources,
- *resource arbitration* - these are policies for choosing between various requests of a single resource, and
- *resource usage* - these are strategies affecting the usage of resource.

In assessing the performance of a software system, some of the metrics considered are system availability, response time, latency, completion time, service time, bandwidth, throughput, etc.

2.2.3 Quality Attributes Characterisation

Evaluating an architectural design against quality attribute requirements requires a precise characterisation of the quality attributes of concern. From the knowledge that is already in the various quality attribute communities, we identified the characterisations from [43] as widely used, due to its effectiveness. These characterisations serve as starting points, which can be detailed when preparing to conduct the architecture design or analysis. Here, each quality attribute characterisation is divided into three categories: *external stimuli*, *architectural decisions*, and *responses*. The *external stimuli* are the events that cause the architecture to respond or change. Also, to analyse an architecture for adherence to quality requirements, these requirements need to be expressed in concrete and measurable or observable quantities. Such quantities are described as the *responses*. Then, *architectural decisions* are those aspects of an architecture that have a direct impact on achieving attribute *responses*.

With regards to the *availability* quality attribute, its *stimuli* are from *Source* (i.e. hardware or software faults), and *Type* (i.e. value, timing, and stopping). The *parameters* that apply to the *stimuli* are the *hardware redundancy*, *software redundancy*, *voting*, *retry*, and *failover*. Finally, the *responses* generated

are usually in the form of *availability*, *reliability*, *levels of service*, and *mean time to failure*. The availability characterisation is depicted in Figure 2.8(a) with further details. The *performance* quality attribute presents a relatively elaborate characterisation. Figure 2.8(b) shows the *stimuli* for performance as *mode*, *source*, and *frequency regularity*. The performance *parameters* considered in architectural decisions are mainly *resource* such as *CPUs*, *sensors*, *networks*, *memories*, *actuators*, etc. and *resource arbitration* in the form of *queuing* and *pre-emption* (see Figure 2.8(c)). Finally, the *responses* from the performance characterisation for architectural decision are *latency*, *throughput*, and *precedence* (see Figure 2.8(d)).

2.2.4 Designing Software Systems

In the design of software architecture, quality attributes are a fundamental consideration in determining the level of dependability of that architecture. This determination is achieved by using architectural evaluation methods. In this section, we discuss some architectural evaluation methods. Then we conduct comparative studies to identify a suitable evaluation method. To achieve this, some relevant evaluation criteria are defined.

2.2.5 Dependability Evaluation Methods

Architecture evaluation has attracted many researchers and practitioners during the last 20 years. Software Architecture can be evaluated at different stages of software development life-cycle (SDLC). An architecture is evaluated to compare alternatives for identifying strength and weaknesses, at the early phase of design. Several approaches to evaluate software architecture exist: scenario-based, mathematical modeling, simulation-based, and experience-based [184]. It is observed that, scenario-based approaches are most widely used in practice. This approach is considered as a more matured, reliable and easy to implement in practical situations [166]. Here, we briefly introduce some relevant approaches

(in two parts: (1) old but relevant methods, and (2) relatively current methods) below:

First among the *old but relevant methods*, is the *Software Architecture Analysis Method (SAAM)* [111] which performs evaluation on software architecture to identify *risks* present in architecture, and also able to express the *modifiability quality attribute*. SAAM is an established method and has been applied in several case studies, which includes user interface development environment, internet information systems, keyword in context (KWIC), revision control system, global information system, and embedded audio system [112], [43]. SAAM meets the criteria of involving *all major stakeholders*, and it is the only method that has tool support (i.e. Ex:SAAMTOOL), even though partial.

Secondly, the *Architecture Trade-off Analysis Method (ATAM)* is inspired by the notion of architectural styles, the quality attribute analysis communities, and the SAAM (i.e. the predecessor to the ATAM). The ATAM is intended for sensitivity and trade-off analysis of an architecture with respect to its quality attributes which covers a broader scope (i.e. performance, availability, security, modifiability, etc.) [113]. ATAM is also relatively flexible since it allows for the definition of new quality attributes. In such a situation, the properties of interest for these attributes will have to be explicitly described during its early steps. The ATAM easily accommodates new quality-dependent analysis [43]. It also involves all relevant stakeholders. ATAM consists of nine steps: (1) Present the ATAM, (2) Present the Business Drivers, (3) Present the Architecture, (4) Identify Architectural Approaches, (5) Generate the Quality Attribute Utility Tree, (6) Analyse the Architectural Approaches, (7) Brainstorm and Prioritise Scenarios, (8) Analyse the Architectural Approaches, and (9) Present Results. Note that steps (6) and (8) are similar and can be considered redundant.

Also, the *SAAM for Complex Scenarios (SAAMCS)* extends the SAAM. Its main goal is to assess risk during system modification. SAAMCS handles specific problems, and hence it is designed to implement *modifiability* or *flexibility* quality attributes [130]. This method is applied to the final document of architecture design. Here, stakeholder involvement is same as SAAM. SAAMCS

defines a measurement instrument to identify complex scenarios in the system. To this end, SAAMCS presents a two dimensional framework diagram to locate complex scenarios. Currently, SAAMCS has no tool support for method automation.

Another method is the *Scenario-Based Architecture Re-engineering (SBAR)* [24]. It drives the architecture re-design for reliability and performance quality attributes. The main goal of SBAR is to introduce an iterative process of quality evaluation and architecture transformation. It supports multiple quality attributes like ATAM but differs from ATAM since it uses different evaluation techniques: scenario-based, mathematical modelling, simulation, and experience-based. In scenario-based approach, SBAR defines scenarios for each quality attribute and maps the performance of architecture. SBAR has been applied in fire-alarm system [35], measurement system [112], and dialysis system [24].

The *Extending SAAM by Integration in the Domain (ESAAMI)* is similar to SAAM, however it differs as it considers the existence of a reusable knowledge-base. ESAAMI integrates the existing knowledge in the reuse based development and domain-centric process [148]. It introduces the concept of reusable products (or "protoscenario") that are deployed during the steps of the method. The stakeholder involvement here is the same as in SAAM.

The *Architecture-Level Modifiability Analysis (ALMA)* is also a scenario-based architecture evaluation method, which analyses software architecture for modifiability quality attribute. ALMA assesses modifiability property by employing indicators, such as risk assessment, and maintenance cost prediction. Originally, ALMA is a combination of two architecture evaluation methods, created by [26], and [130]. Here, both methods are based on scenarios and uses similar structures. ALMA provides five steps for evaluation: (1) setting the analysis goal, (2) describing software architectures, (3) eliciting scenarios, (4) evaluating change scenarios, and (5) interpreting the results. It has been applied to analyse various software architecture case studies, such as haemo-dialysis system (a medical treatment device), mobile positioning center assessment (a telecommu-

nications service provider system), ComBAD Framework assessment (a domain specific architecture for administrative systems) [129]. ALMA considers various sets of stakeholders for different activities.

The *Architecture Level Prediction of Software Maintenance (ALPSM)* has been developed to predict software maintainability during software architecture design [25]. Such predictions are used for balancing maintainability of quality attributes, or to compare architectural alternatives. ALPSM considers inputs such as requirement specification, architecture design, software engineer expertise, and historical maintenance data. However, its outputs are estimated maintenance efforts and maintenance profile. Some benefits of ALPSM are that it is practically used in architecture design and it gives not only prediction but also gives improved understanding of requirements. ALPSM has been validated for Haemo-Dialysis System [25].

The final method is the *Software Architecture Comparison Method (SACAM)*, which provides basis for selecting architecture by comparing different candidate architectures [28]. A set of criteria based on business goals of an organisation is used by SACAM to compare architectures. It helps organisations to explore architecture design mainly in product line architectures. That said, SACAM is used for architecture design in many architectures from different vendors or contractors.

2.2.5.1 Comparative Study

We conduct a comparative study of the software architecture evaluation methods, based on a defined set of criteria. Considering software systems that are distributed and deployed using the cloud deployment model, we focus on the fact that this environment is characterised by random evolution which affects the dependability of deployed systems. The main criteria for our comparative study are that the evaluation methods should have:

(a) **A goal of sensitivity and trade-off analysis**:- In a rapidly evolving environment, an efficient evaluation method should be capable of identi-

Figure 2.9: Comparison of Architecture Evaluation Methods

Comparison elements / **Evaluation Method**	***ATAM (Kazman et al, 1999)***	**SAAM (Kazman et al, 1998)**	**ALMA (Lassing et al, 1999 and Bengtsson et al, 2004)**	**SAAMCS (Lassing et al, 1999)**	**SBAR (Bengtsson and Bosch, 1998)**	**ALPSM (Bengtsson and Bosch, 1999)**	**ESAAMI (G. Molter, 1999)**	**SACAM (Bergey et al, 1999)**
Method's Goal	*Sensitivity and Trade-off analysis*	Risk identification, architectural suitability	Maintenance cost prediction, risk assessment, SA selection	Risk assessment, developing complex scenarios for flexibility quality attribute	SA re-engineering to achieve quality attributes	Prediction of software maintainability	Applies SAAM in a reuse-based development and domain centric process	Comparison of several software architectures from different domains
Evaluation Approaches	*Combines questioning and measuring techniques*	Scenario based functionality and change analysis	Depends on analysis objectives	Scenario-based(for complex scenarios)	*Multiple approaches*	Scenario-based	Preparation of analysis template to collect protoscenario	Collect comparison criteria
Quality Attributes	*Multiple*	Modifiability	Maintainability	Flexibility	*Multiple*	Maintainability	Modifiability	*Multiple*
Applicable Project Stage	*After SA /detailed or iterative improvement process*	Once functions assigned to modules	During architectural design	Final version of the software architecture	System extension or re-engineering stage	During architectural design	Final version of the software architecture	During architectural design
Method's Activities	*Six activities in four phases*	Six activities with some activities in parallel	Five activities executed sequentially	Three activities, two executed in parallel	Three activities carried out iteratively	Six activities	Same as SAAM but use reusable knowledge base	Six activities, one executed repeatedly
Stakeholders' Involvement	*All*	*Major stakeholders*	Various, for different activities	*Major stakeholders*	Software Architect	Designer	*All*	Designer
Method's Application	*Battlefield control system, remote temperature sensor, etc.*	GIS,WRCS, KWIC, and embedded audio system	Telecommu-nication systems, information systems, embedded systems	Business information systems	Fire-alarm systems, measurement systems, and dialysis systems	Haemodialysis machine	Measurement system	Envisioned software proposed by various contractors
Tool Support	*No*	*Partial, Ex: SAAMTOOL*	No	No	No	No	No	No

fying the quality attributes that change relative to others. Also, it must be able to inform architectural decisions regarding an acceptable trade-off between quality attributes.

(b) **A focus on multiple quality attributes**:- In the described environment above, typically there will be multiple competing quality attributes. It is, however, relevant that an appropriate evaluation method can consider varying quality attributes.

(c) **The involvement of multiple or all architecture stakeholders**:- Stakeholders are the custodians of architecture requirements, and their involvement in the evaluation and development processes are critical. That said, an appropriate evaluation method should actively involve many relevant stakeholders (for requirements coverage).

We are also interested in identifying (d) the SDLC stage(s) that the methods apply to, (e) application of methods in projects, and (f) whether there are tools that support the methods.

A: Summary of Comparative Study

A summary of our comparative study is shown in Figure 2.9. For **Criteria (a)**, we identified that ATAM is the only method that has a goal to analyse system sensitivity and trade-off. **Criteria (b)** is met by ATAM, SBAR, and SACAM for focusing on multiple quality attributes. Finally, ATAM, SAAM, ALMA, SAAMCS, and ESAAMI involve multiple or all stakeholders to meet **Criteria (c)**. An overall assessment shows that even though ESAAMI met two criteria (i.e. *Criteria 2* and *3*), ATAM on the other hand met all three criteria. Furthermore, ATAM can be applied iteratively to SDLC improvement processes, but it has no tool support. Some reported successful applications of ATAM are the role-playing game to teach ATAM [149], architectural evaluation of a data center system [177], assessment of a battlefield control system [110], war game simulation [106], product line architecture [199], control of a transportation system [36], credit card transactions system [133] and a dynamic map system [203].

To ensure that ATAM is effective as a method compared to more recent methods, we compare some relatively recent architecture evaluation methods and approaches to ATAM based on the set criteria above. We begin by providing an overview of the considered methods, and then tabulate their features in the context of a set of relevant criteria.

The first method [206] proposes the *aspectual software architecture analysis method* (ASAAM) which explicitly identify and specify the architectural aspects and make them transparent early in the software development life cycle. In software architecture analysis, it is implicitly assumed that an appropriate refactoring of the architecture design can help in coping with critical scenarios and mending the architecture. The authors also show that, similar to the notion of aspect at the programming level, there are concerns at the architecture design level which inherently crosscut multiple architectural components. Such concerns are referred to as architectural aspects. Here, they indicate that such concerns cannot be localised in one architectural component and which, as such, can not be easily managed by using conventional abstraction mechanisms. In this context, ASAAM introduces a set of heuristic rules that help to derive architectural aspects and the corresponding tangled architectural components from scenarios. They illustrate their approach for architectural aspect identification with the architecture design of a window management system.

Another work [215], that applies scenario-based architecture analysis, provides a motivation that scenarios are especially helpful for visualising and understanding the incorporation of new systems within systems of systems. They argue that, if used as the basis for decisions about candidate designs, then it is important that such decisions can be rationalised, and quantitative assessment is particularly important. Their work introduces an approach for developing complex scenarios, which incorporates the phases of systems development and deployment, is presented and a quantitative method of comparison is described. Their approach is based on the development of measures of merit and measures of performance. The techniques are illustrated using cases that are relevant to Network Enabled Capability.

Figure 2.10: Comparison of Current Architecture Evaluation Methods

Comparison elements	***ATAM (Kazman et al, 1999)***	**ASAAM (2004)**	**Design and Evaluation for Capability (2010)**	**Evaluating a REST Architecture (2014)**	**Automating Performance Assessment (2015)**
Method's Goal	*Sensitivity and Trade-off analysis*	Heuristics to identify architectural concerns/aspects	Quantitative comparison	Risks, design considerations, trade-offs	Reveal performance degradation of architecture scenarios
Evaluation Approaches	*Combines questioning and measuring techniques*	Scenario based with heuristics	Scenario-based	Scenario-based with guidelines)	*Scenario-based*
Quality Attributes	*Multiple*	Modifiability	Capability, Performance	Multiple	*Performance*
Applicable Project Stage	*After SA /detailed or iterative improvement process*	Design	Design, analysis, development	After development	After design
Method's Activities	*Six activities in four phases*	Heuristics and same as SAAM	Quantitative assessment of scenarios	Same as ATAM	Dynamic analysis and code repository mining
Stakeholders' Involvement	*All*	*Major stakeholders*	Major stakeholders	*All*	Major stakeholders
Method's Application	*Battlefield control system, remote temperature sensor, etc.*	Window Management system	Network Enabled Capability	Web Ecosystem of Physical Devices	A system that analysed: 1. SIGAA, 2. Argo UML and 3. Netty
Tool Support	*No*	*No*	No	No	Automated (algorithm)

The use of Representational State Transfer (REST) as an architectural style for integrating services and applications introduces some risks. Importantly, this work, [48] indicates that such risks are failures to effectively address quality attribute requirements such as security, reliability, and performance. They argue that, an architecture evaluation conducted early in the software life-cycle can identify and help mitigate these risks. This work presents guidelines to assist architecture evaluation activities in REST-based systems. These guidelines can be systematically used in conjunction with scenario-based evaluation methods to reason about design considerations and trade-offs. This work also presents a proof of concept to describe how to use the guidelines in the context of an Architecture Trade-off Analysis Method (ATAM) evaluation.

The final approach, [172] begins by describing an exploratory study for the evaluation of the performance quality attribute for releases of the same system. This work aims mainly to reveal performance degradations of architectural scenarios and their possible causes. The study uses three software systems from different domains: (1) a large-scale web system, (2) a UML modeling tool, and (3) a client-server framework for development of network applications. The data collection of the study is accomplished using a scenario-based approach that uses dynamic analysis and code repository mining to provide an *automated way* to reveal degradations of scenarios on releases of software systems. The results of the study show the feasibility of the approach to determine the causes of the performance degradations of scenarios, including the degraded and changed methods of scenarios, and the issues that have affected them.

B: Summary of Comparative Study of Some Relatively Current Methods and ATAM

A summary of our comparative study of ATAM and some recent architecture evaluation methods is tabulated in Figure 2.10. Considering the goal of the methods in **Criteria (a)**, we identified that the method provided for "evaluating REST architecture" is the only one that considers "risks" and "trade-offs" related to multiple quality attributes in designing an architecture. Even though ASAAM identifies risks in its analysis, it classifies them as "concerns" and do

not consider them in relation to multiple quality attributes. With **Criteria (b)**, it is only the method for *evaluating REST architecture* that meets the expectation of considering "multiple quality attributes" (i.e. more than two attributes). It must be mentioned that the "scenario-based method for designing and evaluating architectures for capability" considers two quality attributes (i.e. capability and performance). Finally, even though all the methods consider major stakeholders in their analysis, it is only the method for "evaluating REST architecture" that considers all stakeholders. Here, the method for "evaluating REST architecture" effectively meets **Criteria (c)**.

2.2.5.2 Observations from Final Comparative Study

We identified two features that we consider to be relevant to our work:

(a) Two of these methods (i.e. ASAAM [206] and method for evaluating REST architecture, [48]) introduce a form of directives to guide system architects and relevant stakeholders on how to analyse architectures with their methods (refer to the "Evaluation Approaches" in Figure 2.10). Here, [206] presents "heuristics" to identify architectural concerns (as defined by SAAM), and [48] also provides "guidelines" for identifying risks and trade-offs during the design of an architecture (as defined by ATAM). With these directives, the mentioned works are able to narrow the context and applicability of their proposed methods.

(b) In the work [48], the evaluation method is applied to a "web ecosystem of physical devices" designed as a REST architecture. As all composing services of the cloud environment are designed and deployed as REST web services, this work has a potential of being scaled up to apply to systems that are designed for the cloud environment. With the cloud environment being characterised by rapid topological evolution, and the random interactions of quality attributes, it will be particularly interesting to apply a custom method to facilitate the design and/or evaluation of systems in such an environment.

2.2.6 Classifying the Size of Architectures for Target Analysis

These discussed architecture analysis methods are effective with regards to the quality attributes of concern, team size, etc. However, we identify that none of them consider the size of architecture being analysed. This is relevant because a very large architecture can introduce complexities to the analysis process, which may render it ineffective. Some widely used architecture classification approaches can be used as a reference point in order to focus the analysis process to a particular size of architecture. We consider the Common Software Measurement International Consortium (COSMIC) Functional Size Measurement [141] as a viable option for classifying architectures. This process is simple to achieve, and yet effective in its application. It takes into account that a functional process (FP) can be derived from at least one identifiable Functional User Requirement (FUR) within the agreed scope. The FP is an elementary component of a set of FUR, comprising a unique, cohesive and independently executable set of data movements. The COSMIC measurement phase distinguishes four types of data movements: *entry*, *exit*, *read*, and *write*. To measure the functional size, COSMIC assigns a single unit of measure, one 1 to each data movement. By convention, it is called 1 CFP (Cosmic Function Point). The total size of the software being measured, corresponds to the addition of all data movements recognised by the COSMIC FSM method.

The COSMIC FSM method is based on the application of a set of models, rules, and procedures to a given piece of software, as it is defined from its Functional User Requirements (FUR). The FUR is a subset of the user requirements describing what the software does in terms of tasks and services. This FSM standard is suitable for measuring various types of software (such as business application software, real-time software or web-based and Internet applications, etc). Furthermore, in conformity with ISO 14143-1 (specifically ISO/IEC 19761) [101], the COSMIC method is independent of the implementation decisions embedded in the operational artefacts of the software to be measured and it excludes both the software quality and technical characteristics.

COSMIC FSM is often applied to the International Software Benchmarking Standards Group (ISBSG) [210] release 8 dataset for categorising software project size.

Related Work on Frameworks that Support System Engineering

In this section, we consider related works spanning frameworks or architectures that support system engineering. We discuss these works along two strands: non-cloud-based frameworks and cloud-based frameworks.

2.2.6.1 Non-Cloud-Based Frameworks

With regards to the Evidential Tool Bus [186], the authors propose a tool combination of theorem provers, model checkers, static analysers, test generators, etc. where many tools and methods are used in an ad-hoc combination within a single analysis. This sort of combination requires an integrating platform - a tool bus - to connect the various tools together; but the capabilities required go beyond those of platforms such as Eclipse. In the tool bus, all tools are co-equals, and are mainly coordinating components of the tool bus. The entities exchanged among clients of the bus - proofs, counterexamples, specifications, theorems, abstractions - have logical content, and the overall purpose of the bus is to gather and integrate evidence for verification or refutation. Even though this work provides a roadmap for a tools combination framework for system engineering, it focuses exclusively on applications to formal methods. This focus in our opinion, is very narrow with regards to system engineering. Also, such a tool bus is accessible to a limited number of stakeholders of a system engineering project, and at a limited geographic location.

Another work is the Open Framework for Software Engineering Tools (OPHELIA) [225], [53] platform which provides a unified software engineering tools integration technology. The concept behind the project involves the definition of standardised set of interfaces abstracting functionalities of different kinds

of software development tools. To maintain implementation language independence, Common Object Request Broker Architecture (CORBA) technology [158] is used to define the interfaces. As part of the Object Management Group (OMG) [159] work, they have developed a comprehensive distributed open systems framework known as CORBA. It is a standard for object middleware used in the heterogeneous environment. The main weakness of OPHELIA is of poor memory management, which is inherited from the CORBA platform. Also, the use of the CORBA platform as a middleware for integration introduces overhead costs through its use of brokers or translators.

2.2.6.2 Cloud-Based Frameworks

The cloud platform is currently being applied in various ways to support different combinations of software and system engineering phases. Such applications are generally presented in a form of software systems or architectures. Among other benefits, this platform provides global access to distributed teams for global software development. Some of these systems or architectures for system development are discussed below:

In the Design Assistant Agent for a Vendor (DAAV) system [108], the authors propose a design assistant agent for defining requirement specifications for a multi-lingual design team, which plays a roll of a bridge engineer. They argue the necessity of a bridge engineer for a multi-lingual development team, who bridges gaps between different languages, cultures and social systems of a client and a vendor to define a requirement specification of an application system. Also, they propose a concept of agent-based support system for defining requirement specifications, consisting of two design assistant agents: a definition support subsystem and a language translation web service. The subsystem consists of Graphical Modeling (GM) functions for clients and developers to draw Unified Modeling Language (UML) diagram or Mindmap for defining requirement specifications. In essence, this work introduces a multi-lingual expert system for distributed global software development. This expert system plays a very important role in terms of catering for the socio-cultural aspect

of distributed global software development. However, other essential development aspects involving project co-ordination and control using information and knowledge sharing mechanisms are not considered.

The scalability of the cloud (i.e. Amazon Web Service [1]) is used to test a complex set of test cases of the Google Chrome software. Testing is a critical phase in the software life-cycle. While small-scale component-wise testing is done routinely as part of development and maintenance of large-scale software, the system level testing is much more problematic due to low level of coverage of potential usage scenarios by test cases and high costs associated with wide-scale testing of large software. Wide scale software testing requiring substantial computation and storage resources and where the testing process can be automated through a workflow needing minimal human intervention is representative of the type of testing that may benefit from the use of the cloud. Such testing is generally not practicable on typical desktop computers, due to limitation of resource scalability and consequently time and cost limitations. The ultimate aim of this work is to verify applicability of network analysis methods to analyse software engineering data, so that these methods may be adopted for software analysis to support software testing [212], [163]. That said, only one type of testing tool is used as multiple instances to process the set of test cases. This however, presents a limited perspective in terms of the results from the test tool. A set of multiple test tools with different versions can be coordinated to facilitate a comprehensive testing process, as well as further demonstrate the scalability of the cloud.

Eclipse has an on-going project named Eclipse Orion which is building an integrated web-based tool-set, and targets first the web client languages such as JavaScript, CSS, and HTML. The project aims to move software development to the web as a web browser experience and not by cloning the desktop IDE experience. This is not particularly the direction of our work. We aim to support system engineering lif-cycle processes on the cloud, and not a complete movement of software engineering. A further look into Orion shows that it consists of loosely coupled components written in JavaScript, and server-side services exposed via REST-oriented HTTP APIs. These components and services can be

combined in many different ways to create various kinds of browser-based applications. However, parts of Orion can also be used in traditional desktop clients as well as server side applications. The data being manipulated by such tools can reside either on a remote server or a local machine [209]. Our assessment of local and remote IDEs is that operating on local data offers better performance and facilitates offline usage, but does not provide the always secure, and backed-up nature of storing data on the server [209].

Similar to the Eclipse Orion project is the Codenvy project [47]. The Codenvy structure is also developed as a RESTful web service, which supports IDEs for Java/Android, PHP, JSP, XML, Python, Perl languages, etc. The browser-based clients deploy projects to target platforms on the Cloud. The Codenvy web service is developed to make Platform-as-a-Service (PaaS) such as Amazon Elastic Beanstalk, CloudBees, etc. flexible and convenient to use. Codenvy is built upon a scalable, extensible plug-in architecture that includes an embedded cloud-local builder and runner to package and debug applications. Codenvy includes the usual IDE tools, integrations, and plug-ins such as syntax highlighting, code completion, refactoring, packages, build manager, continuous integration, git, and PaaS deployment. It also provides the flexibility to developers to use their provided software development kit (SDK) to build their own extensions. It integrates a wide range of technologies which can be classified into Languages, Databases, Platforms/Continuous Integration and Build Systems, Repositories and Agile System Development. The developer can create a project and virtually navigate the entire Codenvy IDE in a similar way as in Eclipse. Codenvy does not integrate into desktop IDEs.

2.3 Cloud Deployment Environment

2.3.1 Cloud Computing

Cloud computing [103] is one of the best service provider in the information technology (IT) industry. The on-demand service is the most important char-

acteristic, and others are resource pooling, measured service, broad network access, and rapid elasticity. Cloud Service Providers (CSPs) offer services in three main categories: Infrastructure as a service (IaaS), Platform as a service (PaaS) and software as a service (SaaS). Also, there are four deployment models: *private cloud* which is owned or operated by a single organisation, *public cloud* is available to the general public, but owned and operated by government organisations, institutions, businesses, etc., *community cloud* involves a common infrastructure for organisations of the same community, and *hybrid cloud* which is a composition of two or more clouds. Cloud computing offers some benefits such as low cost, unlimited smart storage, flexibility, improved performance, increased data reliability [78]. Some studies have shown that businesses that adopt SaaS enjoy a return-on-investment of almost 600% [194].

2.3.2 Issues with Cloud Computing

In spite of the mentioned benefits, the cloud platform is faced with some challenges. We identify some of these challenges from literature as lack of customer trust, vague SLAs, perceived lack of reliability, threats to security and privacy, absence of independent quality assurance body, etc. Some of these challenges are discussed below.

2.3.2.1 Lack of Customer Trust

Cloud computing raises more than economical and technical challenges, but also has implications on trust [128]. The highly distributed and non-transparent nature of cloud computing represents a considerable obstacle to the acceptance and market success of cloud services. Potential users of these services often feel that they lose control over their data and they are not sure whether cloud providers can be trusted [74]. Also, with the growing number of CSPs, the customers are facing a challenge to select the best and most appropriate providers from numerous offers. In a typical scenario [61], it is pointed out that a CSP can offer a reasonably secure service while another may not. If the latter charges

half the price, the majority of organisations will opt for the latter one as there is no real way to explore the difference.

2.3.2.2 Weak Service Level Agreements (SLAs)

Standard SLAs in the present cloud market are also one of the obstacles that the cloud users face while adopting the services offered by the cloud providers. These SLAs are perceived to be weak in terms of assuring the needs of users, and require non-ambiguous descriptions to cater for clear definition of responsibilities to all cloud stakeholders. Cloud users might face problems that occur from insufficient security measures, data unavailability, CSP lock-in, hidden costs, and non-transparent infrastructure. In most cases, SLAs are created to protect the CSPs and not the customers. CSPs do not provide SLAs guaranteeing minimum levels of performance [104]. Most of the above mentioned problems are sidelined in current SLAs offered by the CSPs.

2.3.2.3 Perceived Lack of Reliability

Availability of resources in cloud computing is identified as one of the biggest concerns for the cloud users [12]. Here, reference to availability is not only with respect to the reachability of the cloud service, but also the success rate of the transaction. The cloud platform's quality of service involving availability, reliability and performance are of interest. Most cloud providers do not define the availability in this way. CSPs use the *availability* term to show their cloud users the level of reliability they would get regarding the cloud services. Most CSPs offer 99.99% availability for their servers, but it is not clear whether the availability is for a single server where the virtual instance of particular cloud user resides or for all the servers placed in geographically distributed datacenters. Many reported outage incidents in the datacenters of the CSPs, indicate a negative image to the cloud users about the providers regarding reliability [12].

2.3.2.4 Absence of Independent Quality Assurance Body

Some CSPs are offering monitoring tools for the cloud users to monitor their service's availability and performance in real-time with extra charges [75]. An example is Amazon Web Service's CloudWatch service. However, most of the CSPs are not offering these kind of solutions. Those that provide it do not really monitor the SLA compliance. This calls for independent quality assurance bodies for monitoring the performance or quality of the Cloud services, in the context of SLA compliance.

2.3.3 Cloud Dependability Assurance

To gain some understanding of the assurances provided by CSPs for cloud resources, it is important to be able to have reference points, and widely accepted benchmarks that clarify such reference points. This work identifies the cloud SLA as the main reference point for cloud dependability assurance. Also, some dependability assurance information sources are provided to guide the understanding of the SLA.

2.3.3.1 Cloud Service Level Agreements

In this work, we consider Amazon Web Services (AWS) as the choice for the type of cloud computing platform. In light of this, we assess AWS' service level agreement (SLA) for their Elastic Cloud Compute (EC2) service [1]. We identified that this SLA guarantees two things:

(a) That the EC2's application programming interface (API) will be available to allow for the launching of new instances 99.9% of the time.

(b) That at least one user instance will be able to access the Internet 99.9% of the time (specifically, it's an outage if 100% of user instances cannot reach the Internet 99.9% of the time).

The AWS SLA does not specifically cover the reliability of the AWS EC2 instances. Even with the assurances provided for the availability of the AWS EC2 instances, they are rather vague. That said, the AWS SLA provides some grounds for the definition of the boundaries for the assurance of system availability. The AWS SLA however provides some *definitions*, *service credits* and *service commitments*. These are presented below:

- "Monthly Uptime Percentage" is calculated by subtracting from 100% the percentage of minutes during the month in which Amazon EC2 was in the state of "Region Unavailable".
- "Region Unavailable" and "Region Unavailability" mean that more than one Availability Zone in which an instance is running, within the same Region, is "Unavailable" to users.
  (a) "Unavailable" and "Unavailability" mean:- For Amazon EC2, when all of the user's running instances have no external connectivity.
  (b) A "Service Credit" is a dollar credit, calculated as set forth below, that AWS may credit back to an eligible account.

Service Credits are calculated as a percentage of the total charges paid by users (excluding one-time payments such as upfront payments made for Reserved Instances) for either Amazon EC2 in the Region affected for the monthly billing cycle in which the Region Unavailability occurred. For a *Monthly Uptime Percentage* of less than 99.95% but equal to or greater than 99.0%, and less than 99.0%, *Service Credit Percentage* of 10% and 30% are provided respectively. Finally, the Amazon EC2 SLA *commitment* is 99.95% availability for each Amazon EC2 Region.

2.3.3.2 Cloud IaaS 2016 Benchmark

Standard Performance Evaluation Corporation (SPEC)'s first benchmark suite to measure cloud performance - SPEC Cloud IaaS 2016 [198]. However, the benchmark addresses the performance of infrastructure-as-a-service (IaaS) public or private cloud platforms, which is designed to stress provisioning as well

as runtime aspects of a cloud using input-output (I/O) and central processing unit (CPU) intensive cloud computing workloads. SPEC selects the social media NoSQL database transaction and K-Means clustering using Map Reduce as two significant and representative workload types within cloud computing. Also, the test uses the Red Hat Enterprise Linux Openstack Platform 7 and a KVM hypervisor in the Dell Inc. USA cloud environment. The key benchmarking metrics are *scalability*, *elasticity*, and *mean instance provisioning time*.

Availability	*Downtime per year*	*Downtime per week*
90.0 % (1 nine)	*36.5 days*	*16.8 hours*
99.0 % (2 nines)	*3.65 days*	*1.68 hours*
99.9 % (3 nines)	*8.76 hours*	*10.1 min*
99.99 % (4 nines)	*52.6 min*	*1.01 min*
99.999 % (5 nines)	*5.26 min*	*6.05 s*
99.9999 % (6 nines)	*31.5 s*	*0.605 s*
99.99999 % (7 nines)	*0.3 s*	*6 ms*

Figure 2.11: The "Nines" of Availability [224]

2.3.3.3 The "Nines" of Availability

High-demand systems that are commonly in around-the-clock service, the availability is frequently measured by the number of "nines" [224]. If availability is 99.0%, it is stated to be "2 nines", and so on. Figure 2.11 depicts the amount of downtime a system exhibits within one year (365 days) of continuous desired operation and its associated number of nines, which are calculated using this formula using the availability equation (2.1) introduced in Section 2.2.2.1. Obtaining 5 nines or 99.999% availability is an ambitious goal that often requires critical system components that are redundant. Such redundant components can help reduce the time to repair a system.

2.3.4 Cloud Accountability Analysis

Digital forensics is playing an increasing important role in digital introspection for evidence collection especially for investigating criminal activity. According to the Federal Bureau of Investigations (FBI) [182], 4,263 tera-bytes (TB) of data was processed for digital forensic in 7,629 criminal examinations in 2011, as opposed to 439 TB and 880 criminal cases in 2003. With the increased migration of critical information technology services into the cloud, *digital forensics* is poised to become instrumental in investigating vulnerabilities and possible criminal activities committed in cloud environments as well [68]. Such a process can be classified as *cloud accountability*. Here, we define accountability as a clear disclosure of service obligations; faithfully honouring of disclosed obligations, or otherwise assuming the liability for the unsatisfactory performance of the obligations [227] by cloud agents (i.e. service providers, software engineers). Typically, accountability in service is achieved through the enforcement of a legal and paper-based *contract*. In a cloud service context, using a paper-based contract is no longer effective [228]. The current practice is for service providers to publish a terms and conditions page and a text-based SLA for their offerings on their website does. In its plain-text form, a web-enabled paper-based contract can neither be interpreted by software agents, nor be used as a basis for *monitoring* the execution of a contract.

It is important for cloud computing services to provide *assurance* based on detailed trust-focused auditing to enable forensic conclusions to be drawn for the purpose of accountability. It is relevant to note that, the indiscriminate addition of auditing to a run-time environment introduces performance overheads. With cloud accountability:

- the *cloud users* can investigate whether the provider is meeting their expectations according to the service level agreement,
- if violations are reported, the *cloud users* can provide evidence to verify who is responsible, and
- if there is a dispute, the *cloud users* can present proof to a third party (i.e.

regulator or judge).

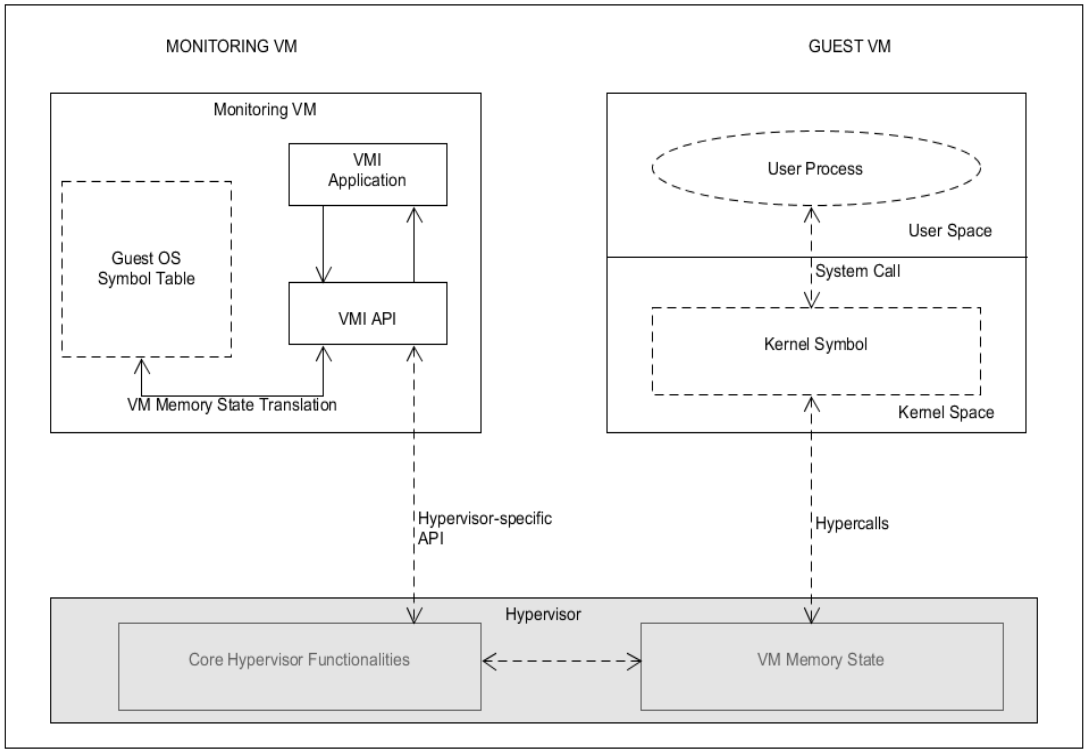


Figure 2.12: Architecture of Virtual Machine Introspection (VMI)

2.3.4.1 Digital Forensic Approach

Digital forensic is about obtaining, preserving, analysing, and documenting digital evidence from digital devices [39]. Digital forensic has been widely accepted and largely used in investigating cloud security issues [147], [192], [195]. This activity is called cloud forensic, which is defined broadly by [185] as "an application of digital forensic science in cloud computing environments". Technically, it consists of a hybrid forensic approach (e.g., remote, virtual, network, live, large-scale, thin-client, thick-client) towards the generation of digital evidence. Generally, cloud forensic is based on the well-established and widely accepted

standard of NIST SP800-86 [114]. The concept of Virtual Machine Introspection (VMI) for digital evidence collection is briefly presented. Also, some sources for digital evidence are introduced.

Virtual Machine Introspection (VMI)

VMI [173] is the technique of locating and accessing the digital forensic evidence on a running virtual machine (VM) (i.e. user-VM) from another isolated running VM (i.e. admin-VM) which is co-located on the same hardware and which has required privileges to access the hypervisor layer. VMI is transparent and does not interrupt the work-flow of the target user-VM nor can it be detected from there. VMI is especially of interest for security-related techniques, e.g. intrusion detection. Fine-grained VMI techniques are used to locate, read and write potential configuration settings of running applications (see Figure 2.12).

The open-source programming library, "LibVMI" is focused on reading and writing memory from virtual machines [127]. Therefore monitoring applications can access the memory state, CPU registers and disk activity of target operating systems in a safe and efficient manner [123]. Memory can directly be read during runtime of virtual machines. Thus, it is possible to create memory dumps for further processing. In order to meet safety and integrity requirements, target VMs can be paused in order to eliminate the chance of acquiring inconsistent snapshots. The library itself is written in C and comes with a Python wrapper to be able to integrate access to VMs to Python scripts [139].

Sources of Evidence

Digital evidence is collected from multiple sources across a cloud system:

- **Hypervisor**: Hypervisors, also known as virtual machine monitors (VMMs), such as Citrix's Xen, VMWare ESXi and Microsoft HyperV, are used to manage virtual machines (VMs) and their various hardware resources (i.e. CPUs, RAM, NICs, hard drive, etc.) [202]. It can provide runtime statistics, but also information can be derived from the hypervisor using advanced techniques like Virtual Machine Introspection (VMI). Evidence collected from the hypervisor can be invaluable, since a VM can be observed

from the outside.

- **The Cloud Management System (CMS)**: CMS is a large source for evidence information. It is the central controlling component of a cloud infrastructure and provides information about user logins, cloud service usage, access rights, configuration, resource provisioning, policies, location, etc. CSPs like Amazon Web Services (AWS) provide monitoring features such as the AWS CloudWatch and application programming interface (API) for such information gathering.

2.3.4.2 Net Present Value

The idea of Net Present Value (NPV) involves the concept of the time value of money and takes into consideration that money spent or obtained in future periods will have a different value than money spent or obtained in the present [132]. NPV has been a standard method for the financial appraisal of projects. NPV calculations can be currently found in every project document (i.e. business case, project plan, etc.), and project managers throughout the world use this methodology to compare the value of different projects against investment targets [223]. In the same light, users of cloud resources should be able to know the value of their cloud investments relative to their investment expectations.

2.3.4.3 Related Work

We identify some related works that provide an approach to ensuring a level of transparency or accountability from CSPs. To this end, we first consider some approaches that encourage or measure transparency towards *security* on the cloud platforms. First, the *A4Cloud FP7 Project* [29] is focused on the accountability of the cloud in terms of data security. In the context of the A4Cloud FP7 Project, accountability concerns data stewardship regimes in which organisations that are entrusted with personal and business confidential data are responsible and liable for processing, sharing, storing and otherwise using the

data according to contractual and legal requirements from the time it is collected until when the data is destroyed (including onward transfer to and from third parties). Similar works are *Cloud Controls Matrix (CCM) Framework* [44] developed by the Cloud Security Alliance (CSA); the *Information Assurance Framework* [38] also developed by the European Union Agency for Network and Information Security (ENISA), *SMICloud Framework* [64] which relies on the service measurement indexes that have been identified by the Cloud Service Measurement Index Consortium (CSMIC) [196], and the Complete, Auditable, and Reportable Approach (C.A.RE) [162]. All of these works are mainly for *intrusion detection*, including [82], [179], [23], [191], [201].

However, we consider the few related systems for assuring dependability in the cloud which are more in line with our work. The three main identified related works are conceptually similar to our work. The first work [69], measures and uses quality of service (QoS) information to predict availability, quantify risk, and consider liability in case of failure. They demonstrate that there is a pressing need for such an understanding and explore a set of benchmarks that offers an interesting characterisation of resource performance variability which can be quite significant. Also, they identify how such information can be used both directly by a user and indirectly via a Cloud Broker in the automatic construction and management of SLAs which reference certain kinds of financial portfolios (i.e. pricing models, risk management). However, this work focuses on cloud performance and considers the automatic construction of SLAs that would incorporate expectations over quality of service by referencing benchmarks.

The second work is the *Phantom* [76] which uses fault injection as a means of assessing the dependability of cloud systems. Generally, it observes probe responses to monitor the cloud environment, and generates an alarm when the quality degrades beyond an acceptable range. It consists of three main components, namely Havoc, Monitor, and Analysis. These components interact independently with the cloud service. More specifically, Havoc injects simulated failure events into the cloud, while Monitor "actively" monitors the cloud's behaviour through probes it issues towards the cloud. Phantom's Analysis component observes probe responses, creates a profile of the cloud's failure detector

and recovery system performance, and generates an alarm when the quality degrades beyond an acceptable range. With this work however, events of failure are simulated instead of actually observing the cloud platform for actual occurrences of failure. That said, Phantom uses its "active meta-monitor" approach to detect degradation of the cloud's own failure detector and recovery systems. On the other hand, the recorded failures are not analysed with respect to the CSPs SLA. Also, evidence gathering is mainly by observing the metric provided by the CSP of an open cloud platform.

Similarly, the *Cloud Broker Architecture* [3] is concerned with fault detection, fault evaluation and taking decision for recovery or migration as a means for assuring dependability. It is composed of five modules that collaborate to assure trust in cloud services with focus on dependability properties. The cloud broker architecture for dependability involves the main actors of the cloud chain namely the cloud consumer and the cloud provider and operates consequently. The three main modules (i.e. service discovery, service composition and service delivery) are aimed to fulfil the commonly inquired roles from the cloud broker. Each module can operate independently. The main shortcoming of the Cloud Broker Architecture is inherited from the CORBA platform [217]. This weakness is that of poor memory management. This weakness leads to ad-hoc solutions with regards to avoiding memory leaks. This has the potential of introducing large overhead cost.

2.3.5 Trust Assurance Methods

Trust is a critical factor in cloud computing; in present practice it depends largely on perception of reputation, and self assessment by providers of cloud services. Here, we briefly discuss some mechanisms generally classified under (a) reputation-based trust, (b) SLA verification based trust, and (c) evidence-based trust, that have been introduced in literature to assure trust for cloud services.

2.3.5.1 Reputation-Based Trust

Trust and reputation are related, but different. Basically, trust is between two entities; but the reputation of an entity is the aggregated opinion of a community towards that entity. Usually, an entity that has high reputation is trusted by many entities in that community; an entity, who needs to make trust judgment on an trustee, may use the reputation to calculate or estimate the trust level of that trustee. Reputation systems are widely used in e-commerce and P2P networks. The reputation of cloud services or CSPs will undoubtedly impact cloud users' choice of cloud services; consequently, cloud providers try to build and maintain higher reputation.

Provisioned cloud resources on-demand are especially vulnerable to cyber attacks (i.e. security breaches, copyright violations, and privacy abuses). The cloud platforms built by Google, IBM, and Amazon all reveal this weaknesses. In this work, [91] the authors propose a new approach to integrating virtual clusters, security-reinforced data-centers, and trusted data accesses guided by reputation systems. A hierarchy of P2P reputation systems is suggested to protect clouds and data-centers at the site level, and to safeguard the data objects at the file-access level. Different security countermeasures are suggested to protect cloud service models: IaaS, PaaS, and SaaS, currently implemented by Amazon, IBM, and Google, respectively.

In this work, [2] they investigate the problem of establishing trust in hybrid cloud computing environments for resource sharing and collaboration. As the scope of federated cloud computing enlarges to ubiquitous and pervasive computing, there will be a need to assess and maintain the trustworthiness of the cloud computing entities. This work presents a fully distributed framework that enables trust-based cloud customer and cloud service provider interactions. The framework aids a service consumer in assigning an appropriate weight to the feedback of different raters regarding a prospective service provider. Based on the framework, the authors developed a mechanism for controlling falsified feedback ratings from iteratively exerting trust level contamination due to falsified feedback ratings. The experimental analysis shows that the proposed framework

successfully dilutes the effects of falsified feedback ratings, thereby facilitating accurate and fair assessment of the service reputation.

Many existing reputation based systems either ignore or give less importance to uncertainty linked with the evidence. The authors in this work, [167] propose an uncertainty model linked with the evidence and define their approach to compute opinion for cloud service providers. Using subjective logic operators along with the computed opinion values, they also propose mechanisms to calculate the reputation of cloud service providers.

Also, this work [216] proposes a combinatorial model for assessing trust dynamism in the cloud services. Cloud services and trust value are assessed based on compliance and reputation. Service logs-based compliance reflects dynamic trust. The reputation has been calculated from cooperative user feedback. Feedback rating is the sight of each user about the appealed services. The exposed services that fulfill the user necessities are ranked according to their trust values and top-k cloud services are suggested to the user. The method is well-organised and noticeably improves service-selection process in cloud applications, in terms of security, reliability, and dynamicity.

In [40], they have proposed a method for trust and reputation evaluation in the cloud through the recommendations of "opinion leaders" and eradicating the effect of troll entities. Trust value was assessed using five factors; accessibility, reliability, data integrity, identity, and ability. Also, they offered a method for opinion leaders and malicious entity identification via three topological metrics, including input degree, output degree, and reputation measures. The method being assessed in various situations where displays the results of accuracy by eliminating the effect of malicious entities and the recommendation of opinion leaders. The results have been obtained with a program, MATLAB. It offers suitable security, integrity, and safety.

2.3.5.2 SLA Verification Based Trust

After establishing the initial trust and employing a cloud service, the cloud user needs to verify and re-evaluate the trust. A service level agreement (SLA) is

a legal contract between a cloud user and a cloud service provider. Therefore, quality of service (QoS) monitoring and SLA verification is an important basis of trust management for cloud computing. A number of models that derive trust from SLA verification have been proposed and some are discussed below.

In this model [79], the authors argue that for business workflow automation in a service-enriched environment such as a grid or a cloud, services scattered across heterogeneous Virtual Organisations (VOs) can be aggregated in a producer-consumer manner, building hierarchical structures of added value. In order to preserve the supply chain, the Service Level Agreements (SLAs) corresponding to the underlying choreography of services should also be incrementally aggregated. This cross-VO hierarchical SLA aggregation requires validation, for which a distributed trust system becomes a prerequisite. Elaborating their previous work on rule-based SLA validation, the authors propose a hybrid distributed trust model. This new model is based on Public Key Infrastructure (PKI) and reputation-based trust systems. It helps to prevent SLA violations by identifying violation-prone services at service selection stage and actively contributes in breach management at the time of penalty enforcement.

Also, the work [62] points out the problem of trust management in a multi-cloud setting based on a set of distributed Trust Service Providers (TSPs). TSPs are divided over the clouds, and they evoke raw trust proof from different sources and in different formats. This proof is information concerning the adherence of the cloud service providers (CSPs) to the Service Level Agreement (SLA) for the offered services and the feedback sent by cloud service users(CSUs). Using this information, they calculated an objective trust and a subjective trust of CSPs. TSPs interconnect between themselves through a trust journal network that permits a TSP to get trust information about a CSP from other TSPs. Examinations showed that their proposed framework is effective and relatively constant in differentiating trustworthy and untrustworthy CSPs in a multi-cloud setting.

In this work [204], the authors have proposed a selection middleware for the cloud service based on trust. They suggest an integrated trust evaluation

method which joins objective trust evaluation and subjective trust evaluation. The objective trust evaluation is based on quality of service (QoS) monitoring while the subjective trust evaluation is based on user feedback scores. Experimentations conducted using a synthesised data set display that their offered technique suggestively outperforms the other trust and reputation approaches. The experiments have been developed using MATLAB.

An issue with this trust mechanism is that many cloud users lack the capability to do fine grained QoS monitoring and SLA verification on their own; a professional and independent third party is needed to provide these services. In a private cloud, there may be a cloud broker or a trust authority, which is trusted in the trust domain of the private cloud; so the trusted broker or trust authority can provide the users in the private cloud the services of QoS monitoring and SLA verification. In a hybrid cloud or inter-clouds, a user within a private cloud might still rely on the private cloud trust authority to conduct QoS monitoring and SLA verification; however, in a public cloud, individual users and some small organisations without technical capability may use a commercial professional cloud entity as trust broker.

2.3.5.3 Evidence-Based Trust

Evidence-based trust is tightly linked to QoS monitoring and SLA verification. Here, the actual performance attributes (i.e. evidence) from the cloud service will have to be compared or analysed relative to the contract (i.e. SLA) provided by the CSP. A trustor's belief in the expected behaviour of trustee is based on the evidence about the trustee's attributes of competency, goodwill, and integrity, with respect to that expectation [89]. In evidence-based trust, the focus is dependent on evidence of the trustee's attributes of competency or performance in the context of the service provided. We briefly introduce some approaches for evidence-based trust below.

We first consider the work [220], which investigates a model dynamic trust-level scheduling (DLS) for the cloud computing. They have been inspired by Bayesian cognitive model and mentioning to the trust relationship models of sociology.

They first proposed a novel Bayesian method based cognitive trust model, then proposed a trust dynamic level scheduling algorithm via mixing the existing DLS algorithm. Theoretic analysis and simulation are demonstrated that the Cloud-DLS algorithm can proficiently meet the requirement of cloud computing workloads in trust and assure the performance of jobs in a secure way. But it provided low dependability, integrity and safety in terms of confidentiality.

In terms of assuring system developers of the trustworthiness of the cloud environment, a joint project between IBM and Microsoft [165] aims to instil greater confidence in computations outsourced to the cloud. System developers are able to verify the correctness of the results returned to them. Pinocchio, a built system for efficiently verifying general computations while relying only on cryptographic assumptions is developed. With Pinocchio, the system developer creates a public evaluation key to describe the computation; this setup is proportional to evaluating the computation once. The cloud-based worker system then evaluates the computation on a particular input and uses the evaluation key to produce a proof of correctness. Since computational power is often asymmetric (particularly for mobile devices), a relatively weak client may wish to outsource computation to one or more powerful workers. Here, the system developer is able to verify the results returned, to guard against malicious or malfunctioning workers. They allow the worker to also shed liability; any undesired outputs are provably the result of data the client supplied. Anyone can use a public verification key to check the proof. The main challenge of Pinocchio is that, it is co-located with the workers in the same cloud environment, related to the same set of cloud hardware infrastructure and management software. To better guard against malicious or malfunctioning workers, Pinocchio should be deployed to a different cloud environment such that security and dependability issues affecting the workers cannot compromise the operations of Pinocchio.

In [5], the authors have explained the role of trust in the cloud computing services based on empirical proof from interviewing managers of financial organisations in Ghana. This is a descriptive paper that is based on literature review and experimental data on exploring reasons for the cloud service acquisitions. A mixture of conferences and attention group discussions was used as approaches

for data collection. Information and technology, and electronic banking managers of five main mercantile banks in Accra, Ghana, between January and July 2013 were interviewed. A sum of ten respondents was interviewed, two in each of the selected banks. A purposive sample technique was used in the choice of informants. This method let the selection of qualified informants to ensure extensiveness and diversity of opinion.

Another work, [90] proposes a fuzzy trust evaluation based on consistency intensity for cloud services. The main objective of this work is to define an assessment model for the cloud services to deal with the fuzzy information and offer a novel fuzzy assessment method based on reliability intensity to examine the quantitative value from the fuzzy information. The offered method can dissolve the problem on the analysis and synthesis of the fuzzy assessment information. An instance of trust assessment of the cloud storage service is presented to confirm that the proposed method can express the opinions of all assessors more adequately. It offered suitable security, reliability, and dynamicity.

Furthermore, another work [221] proposes a cloud trust capacity model for reducing threats of internal troll services. It was used to manage the trust relationship among the guest services, to evaluate the threats to the unknown troll services, and to diminish risk associated with leasing the cloud services and limiting the resource drain caused by troll guest services. Experimental results showed that the suggested model can effectively limit the scale of the troll services and considerably lessen the threats of internal attacks. The proposed mechanism provided better reliability and security.

Another work, [189] have proposed a fuzzy-based trust evaluation scheme for the cloud services. A dynamic trust model based on evidence was also suggested to define the dynamic trustworthiness on services in the cloud environment. It employed fuzzy logic to develop trust in order to handle the uncertainty and uses ordered weight averaging operator to gather the trust values, thus allowing the real-time performance. The proposed scheme uses the QoS parameters as a validation to evaluate the trust for the cloud services. The results in terms of efficiency and effectiveness of the model were established through simulations.

It offered suitable security, reliability, and dynamicity, but it suffered from low integrity, low dependability, low confidentiality and low safety.

Finally, a current work [204] has proposed a new method for recognising the moderating effect of trust on the adoption of cloud-based services. The purpose of this research is the identification of the trust factors in the hypothesis of the cloud services in semiconductor industries. Furthermore, the moderating efficacy of these trust elements related to the technical, organisational, and environmental success factors has been propounded. On the base of a literature survey, an assumptive model has been expanded, and the relations among the hidden variables have been studied by utilising structural equations.

2.4 Summary

We have introduced the concept of global software development (GSD) as a pervasive business phenomenon, and its associated benefits for geographically distributed system engineering. Some leading companies such as IBM, British Airways, British Telecom and General Electric are increasingly adopting this model. However, from the literature, we have identified that the absence of effective information and knowledge-sharing mechanisms, form a crucial part of GSD's state-of-the-art problem affecting collaborative software system development. Due to the lack of understanding between geographically distributed teams in GSD, requirements management is particularly difficult. This problem can be addressed by providing guidelines for the quality management of GSD projects, and apply a software process improvement model for change management and traceability. Here, a *change management and traceability process model* that is capable of managing the increased scale of requirements' changes and their traceability which are characteristic of GSD, has to be introduced by the guidelines. The guidelines must ensure that all major GSD activities such as user management, requirement management, change management, and traceability meet a *quality management standard*. In this regard, a *shared artefacts repository* will play a central role for information and knowledge-sharing. We

believe that such guidelines are capable of facilitating a tight linkage between system requirements, change management and traceability towards quality management of GSD projects. This however, informs our objectives (i.e. *OB1* to *OB4*) in Section 1.2.2. We introduce four objectives: *Objective 1* defines a change management and traceability (CM-T) process model, which applies a software process improvement method to ensure the maturity of the RCM and traceability processes; *Objective 2* identifies a standard quality management framework to facilitate a significant level of quality for the proposed CM-T process model; *Objective 3* validates the CM-T process model using an expert panel review process; and *Objective 4* demonstrates the defined management guidelines by applying it to an Airlock Control System case study.

Also, the challenge of appropriate methods to design and evaluate dependable architectures that support system engineering, directly affects the successful implementation of GSD. GSD frameworks are defined as software architectures. The design and evaluation of the dependability of software architectures are based on quality attributes. In these processes, it is important to consider the overall effect of design decisions, the inherent trade-offs between quality attributes, and the trade-offs required to address user, system, and business requirements. In our opinion, we argue that the existing architecture evaluation methods have limitations when assessing architectures interfacing with unpredictable environments such as the Cloud. This is because the Cloud environment is fundamentally different from the classical environments for which most software evaluation methods were developed. The unpredictability of this environment requires a bespoke and holistic approach that combines aspects of both dynamic (i.e. trade-off) and static evaluation of the quality attributes of software architectures for GSD. Such an approach will be in line with our objectives (i.e. *OB5* to *OB8*) in Section 1.2.2. Here, we present four objectives: *Objective 5* defines a methodology for small-to-medium size GSD architectures; *Objective 6* validates this methodology using a comparative study with current approaches; *Objective 7* demonstrates the methodology by applying it to the design of a small-to-medium size architecture, Reactive Architecture; and then *Objective 8* analyses the quality attribute trade-off of the Reactive Architecture.

Finally, Cloud computing has been introduced as a suitable delivery model for GSD. In fact, GSD seems to be organically integrable with Cloud computing. Some argue that GSD can be improved by the main characteristics of cloud computing such as virtualisation, reduced cost, performance, and multi-tenancy support. That said, this facilitating environment is challenged with lack of customer trust, vague SLAs, perceived lack of reliability, threats to security and privacy, and the absence of an independent quality assurance body. Such concerns lead to the call for cloud accountability. Cloud accountability with respect to security has been explored to a great extent. Here, outlines for the technical requirements have been provided. However, from our literature review we observed that cloud accountability for dependability in areas such as availability and reliability has not been explored. This is a matter of concern since cloud users such as system engineers need to be assured of the dependability of the cloud platform they use for GSD. We believe that a cloud accountability methodology for assuring the dependability of cloud environments is necessary (see our objectives *OB9* to *OB12* in Section 1.2.2). We present these objectives as: *Objective 9* defines a cloud accountability methodology; *Objective 10* develops a cloud accountability system which facilitates the presented method; *Objective 11* demonstrates the method by applying it to a cloud-based test-bed of the Reactive Architecture; and *Objective 12* conducts an evidence-based trust analysis on the derived evidence for the purpose of dependability assurance of the cloud-based Reactive Architecture.

Chapter 3

Reactive Architecture

This chapter mainly contributes a cloud-based framework for system engineering. We refer to this framework as Reactive Architecture. Considering the state-of-the-art in RCM, design and analysis of architecture, and cloud accountability, this work mainly contributes:

1. an alternative and novel mechanism for effective information and knowledge-sharing towards RCM and traceability.
2. a novel methodology for the design and analysis of small-to-medium size cloud-based systems, with a particular focus on the trade-off of quality attributes.
3. a dependable framework that facilitates the RCM and traceability method for cloud-based system engineering.
4. a novel methodology for assuring cloud accountability in terms of dependability.
5. a cloud-based framework to facilitate the cloud accountability methodology.

In this chapter, we provide an introduction of the Reactive Architecture as a cloud-based framework to support system engineering in Section 3.1. Also, the

challenges to the state-of-the-art of GSD is discussed in Section 3.2. In Section 3.3, the Reactive Architecture is discussed as a framework that introduces three approaches that attempt to resolve the discussed challenges. Here, an approach that presents a mechanism for requirements change management and traceability is introduced and justified. This mechanism is facilitated by the central component of the Reactive Architecture called the Reactive Middleware, discussed in Section 3.3.1. We follow up in Section 3.3.2 with a description and justification of our approach for designing and evaluating a cloud-based architecture. Section 3.3.3 describes and justifies our approach to assure the dependability of cloud-based systems. We conclude this chapter by providing a summary in Section 3.4.

3.1 Introduction

The Reactive Architecture aims to support the system engineering process by employing some state-of-the-art methods. This is necessary to meet the complexities of the systems we are building now and in the future. These complexities introduce challenges that are inherent in the nature of system engineering: the changing needs of system stakeholders which is more critical for global software development (GSD), dependable system composition or integration, and automated processes. An overview of these challenges are discussed next.

3.2 Challenges of GSD

3.2.1 Effective Information and Knowledge Sharing

Software development is increasingly carried out in a distributed manner with stakeholders based in different geographical locations. Issues caused by this trend are related to knowledge management, quality control, synchronous collaboration, and risk, project and process management concerns. Here, the absence of effective information and knowledge-sharing mechanisms, form a crucial

part of GSD's current problem affecting collaborative software system development. Due to the lack of understanding between geographically distributed teams, requirements management is particularly difficult. Specifically, GSD, where teams are distributed worldwide, introduces an additional level of complexity to artefact consistency management tasks. In this respect, areas of concern are creating and maintaining links among distributed artefacts, multiple versions of artefacts, and the availability and accessibility of the latest version of any given artefact. Artefact repositories and version control systems are often used to mitigate the effects of distribution however, these facilities are largely limited to the implementation phase of system engineering.

3.2.2 Automation

Tasks associated with artefact consistency management in system engineering, when performed manually, are error-prone, tedious and require substantial effort. Some aspects of artefact consistency management can be more easily automated, such as checking consistency violations, while others may present non-trivial challenges. For example identifying relationships between diverse representations is a complex task due to the heterogeneity of artefacts and the fact that semantics and intentions are not explicitly captured. The extent to which automation is possible is an open problem.

3.2.3 Diversity of Tools

Software life-cycle tasks are undertaken by stakeholders using a variety of software engineering tools. Integrated development environments (IDEs) provide support to produce source code and tests. Higher-level artefacts are created using diagram and analysis tools. An ideal framework, to maximise its applicability in software projects, should not impose any specific application on the user and should be configurable to work with any tool. This therefore, brings forward the issue with the integration of such tools in a seamless manner.

3.2.4 System Dependability

Cloud computing has been introduced as a suitable delivery model for GSD. In fact, GSD can be improved by the main characteristics of cloud computing such as reduced cost, performance, global multi-tenancy support, etc. However, the unpredictability and rapid evolution of the topology of the cloud environment affect the dependability of deployed systems for GSD. In order for GSD to be dependable in the cloud environment, the design of GSD systems must master the costs and the quality of the development of such software systems, relative to the rapid evolution of the topology of the cloud environment. Here, it is imperative to consider the overall effect of design decisions, the inherent trade-offs between quality attributes (such as availability, security, reliability, performance), and the trade-offs required to address user, system, and business requirements. It is however essential to have a software system design approach that yields itself readily to an implicit analysis or evaluation method for cloud-based systems. Since the cloud environment is characterised by rapid interactions between quality attributes, the state-of-the-art of evaluating cloud-based systems in this context is lacking. Most notable software system evaluation methods mainly focus on independent quality attributes, and the few that consider multiple quality attributes do not factor the trade-off analysis of system quality attributes for complex enterprise deployment environments, such as the cloud.

3.2.5 Accountable Cloud

The concept of cloud accountability, especially for dependability has not been adequately addressed. This means that the cloud behaviours affecting system dependability should be transparent to relevant parties, hence the call for accountability. It is important to mention that cloud accountability has been widely applied towards assuring the security of cloud-based systems, however, there is no work that employs the *forensic auditing techniques* of cloud accountability towards the assurance of dependability especially for *availability* and *re-*

liability. It is relevant that *cloud users* such as system engineers are assured of the availability and reliability of the cloud platform they use for GSD.

3.3 Reactive Architecture

The Reactive Architecture provides approaches (related to the contributions of this work) intended to remedy the discussed challenges. These approaches are presented and justified below.

3.3.1 Mechanism for Requirements Change Management and Traceability

Considering the introduced challenges of (1) effective information and knowledge sharing, (2) automation, and (3) diversity of tools, we present the definition of guidelines for managing GSD projects that implement the specific goal - *manage requirements changes* - of CMMI Level 2 (discussed in Section 2.1.5.1). We justify the use and validity of using guidelines or heuristics for identifying risks and trade-offs during the design and analysis of architectures in Section 2.2.5.2. The GSD guidelines present a process model for change management and traceability that supports the implementation of the mentioned specific goal. Also, to support the effective management of the system engineering processes, the GSD guidelines apply a lean derivative of the PMBOK quality management process group (discussed in Section 2.1.6) for project life-cycle practices.

An underlying technology used in the change management and traceability process model is the open services for life-cycle collaboration (OSLC). This technology was briefly introduced in Section 2.1.6. OSLC is an open community, where the main goal is to create specifications for integrating tools, their data and workflows in support of life-cycle processes. Fundamentally, OSLC is based on the concept of linked data. Here, OSLC is organised into work-groups

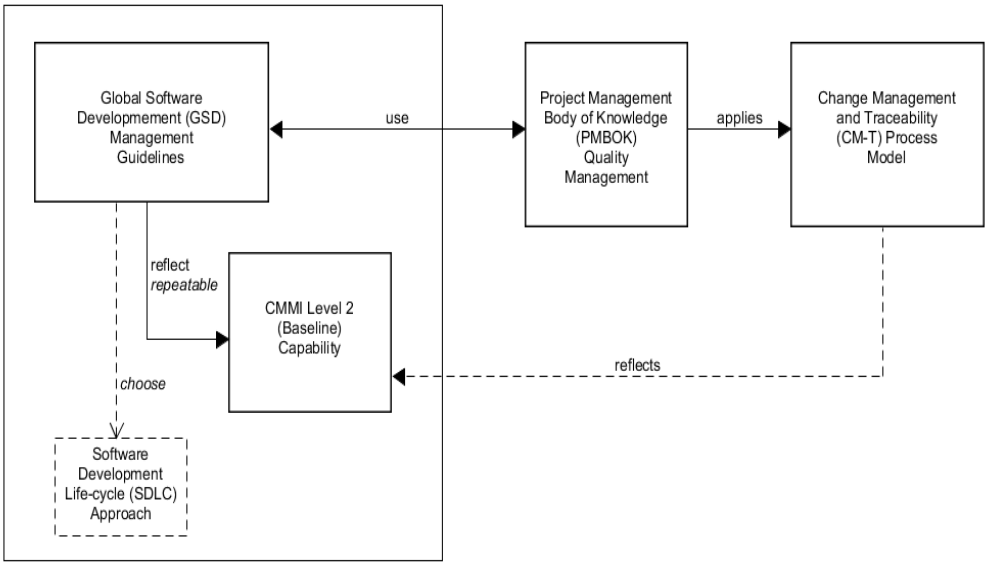


Figure 3.1: Overview of Proposed Mechanism for Requirements Change Management and Traceability

that address integration scenarios for individual topics such as change management, traceability, test management, requirements management and configuration management. Such topics are called OSLC domains. Each work-group explores integration scenarios for a given domain and specifies a common linked data vocabulary for the life-cycle artefacts needed to support the scenarios. In very simple terms, OSLC specifications focus on how the external resources of a particular tool can be accessed, browsed over, and specific change requests can be made. OSLC is not trying to standardise the behaviour or capability of any tool. Instead, OSLC specifies a minimum amount of protocol and a small number of resource types to allow two different tools, data or workflows to work together relatively seamlessly. To ensure coherence and integration across these domains, each work-group builds on the concepts and rules defined in the OSLC Core specification. OSLC Core consists mostly of standard rules and patterns for using HTTP (Hypertext Transfer Protocol) and RDF (Resource Description Framework) that all the domains must adopt in their specifications. It also

defines a small number of resource types that help tools to integrate their activities. In OSLC, each artefact in the life-cycle - a requirement, test case, source file, etc. - is an HTTP resource that is manipulated using the standard methods of the HTTP specification (GET, PUT, POST, DELETE). Each resource has its RDF representation, which allows statements about resources (in particular web resources) in the form of subject/predicate/object (i.e. RDF triple) expressions, such as in linked data. OSLC also supports representations in other formats, like JSON, XML or HTML. The central organising concept of OSLC is ServiceProvider, enabling tools to expose resources and allowing consumers to navigate to all of the resources, and create new ones. Importantly, OSLC allows artefacts (e.g. requirement document) to be exposed as a unit and even specific elements of the artefacts (e.g. individual requirements) can also be exposed as sub-units for monitoring.

Figure 3.1 provides an overview of the requirements change management and traceability mechanism. The GSD guidelines apply a widely used quality management approach to a change management and traceability (CM-T) process model for GSD projects. The CM-T process model complies to or reflects the CMMI Level 2 capability. This mechanism is facilitated by a Reactive Middleware. The Reactive Middleware plays a key role in the Reactive Architecture, and provides a novel Change Management and Traceability-as-a-Service (CM-TaaS) on the cloud platform for system engineering.

3.3.1.1 Reactive Middleware

After considering literature on requirements engineering; both old but significant, and current (discussed in Section 2.1.2) approaches, the identified areas that have not been sufficiently addressed can provide a basis for extracting a high-level set of requirements. Such areas are the independence of artefacts, globally accessible framework for development, large capacity to store different and changing artefacts, automation of change management and traceability processes, and seamless tool integration. Furthermore, we introduce these requirements under the three discussed challenges of effective information and

knowledge sharing, automation, and diversity of tools below:

(a) Effective information and knowledge sharing:

- R1: Artefact independence - The suitable framework should have the capacity to cater for different types or formats of artefacts.
- R2: Supports globally distributed development - Software system development is now practiced globally, a suitable approach should be globally accessible and also provide avenues for solving challenges related to globally distributed software system development.
- R3: Ability to handle different and large numbers of changing artefacts - Since software systems are different in terms of their complexity and size, a suitable approach should consider handling artefacts of varying complexities and large numbers. Also, its capacity for handling changes to such artefacts should scale appropriately (such that it does not affect performance).

(b) Automation:

- R4: Automated as far as possible - In order for the framework to be adopted in software projects and to reduce manual effort, it should provide automated support for consistency management tasks.

(c) Diversity of tools:

- R5: Tool integration - The suitable approach should consider that software artefacts are created and edited in a variety of tools. It should work with both new and existing tools, new and old versions of a tool, and should support seamless integration into different environments.

The Reactive Middleware provides cloud-based services for quality requirements change management and traceability. It is "reactive" because it responds to changes made to artefacts and such changes are consistently propagated to all dependent artefacts and relevant stakeholders. The cloud platform identified and used as a deployment platform for the Reactive Middleware because of the global accessibility of services for collaboration in GSD, scalability to meet

varying complexity of GSD projects, and cost-efficiency. To achieve an effective change management and traceability process, some key steps are adopted. These steps are:

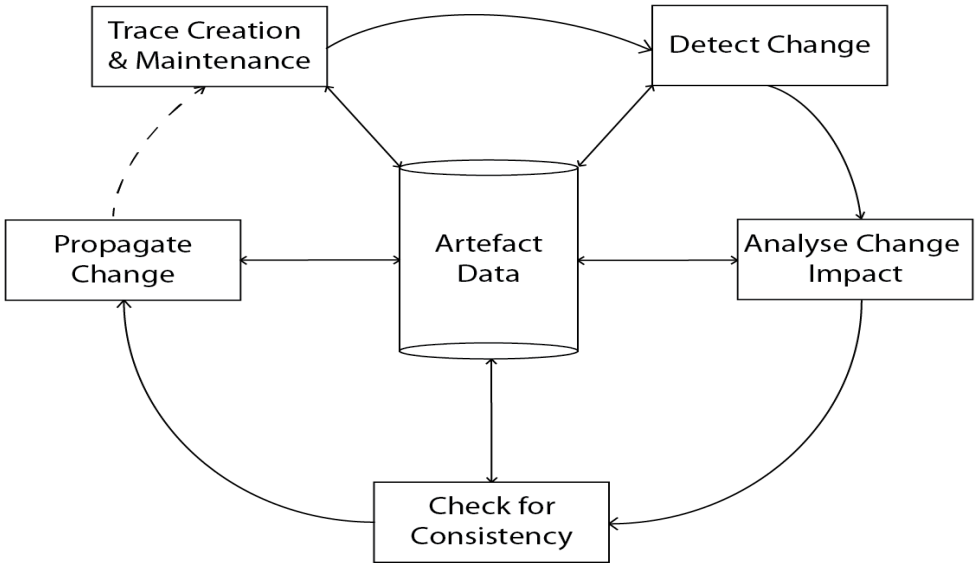


Figure 3.2: Requirements Change Management and Traceability Processes

(a) trace creation and maintenance,

(b) detect change,

(c) analyse change impact,

(d) check for consistency, and

(e) propagate change.

Refer to Figure 3.2 for the interaction of the above mentioned steps. The Artefact Data is a repository of specific details of artefacts, and more contextual data such as the dependencies between artefacts and the trace information of a set of artefacts. Avenues for conflict resolution are also provided. Also, the Reactive Middleware introduces a structured role-based management of system development artefacts. Role in this context is described as stakeholders' responsibility (i.e. privilege) to system artefacts. This approach assigns priority

to system requirements relative to their importance. Also, the approach uses six main privileges (with roles): None:- Have no access to the system artefact(s) for PAWNS, View:- Only sees the system artefact(s) for PAWNS, Modify:- Can see (view) and change the system artefact(s) for MODIFIERS, Review:- Can see and change a modification to system artefact(s), in response to a set of notifications for REVIEWERS, Create:- Can create and modify (view, modify) system artefact(s) for CREATORS, and Own:- Full access (view, modify, review, delete, recall) to the system artefact(s) for TEAM LEADERS. Here, recalled artefacts are reinstated deleted artefacts. In the Reactive Middleware, the original formats of artefacts are maintained but their metadata which identifies the trace links are represented in XML format.

This work aims to answer the research question (RQ): "How can the Reactive Middleware guide system engineering to ensure the continual tight linkage of stakeholders' requirements and system engineering processes?". It hypothesises that changes in system requirements' artefacts are captured and consistently propagated to all the related system engineering processes and stakeholders using a matured process model. This hypothesis constitutes the conceptual foundation of the proposed approach, which is aimed at fulfilling the high-level requirements discussed earlier.

3.3.2 Design and Analysis Methodology for Cloud-Based Architecture

We argue that the existing architecture evaluation methods have limitations when assessing architectures interfacing with unpredictable environments such as the Cloud. The unpredictability of this environment is attributed to the dynamic elasticity, scale, and continuous evolution of the cloud topology. More specifically, this is as a result of the rapid introduction of new services, mashups, unpredictable modes of service use, fluctuations in QoS provision due to unpredictable load or growth, etc. As a result, architectures interfacing such unpredictable environments are expected to encounter many uncertainties. This is also relevant because the cloud environment is fundamentally different from

the classical environments for which most software evaluation methods were developed. It is however, important to focus on, and present holistic approaches combining aspects of both dynamic and static analysis of architecture resilience attributes. From literature, we identify a set of twelve relevant architecture analysis or evaluation methodologies, and conduct comparative studies (refer to at Section 2.2.5.1) based on a set of justified criteria:

(a) A goal of sensitivity and trade-off analysis,

(b) A focus on multiple quality attributes, and

(c) The involvement of multiple or all architecture stakeholders.

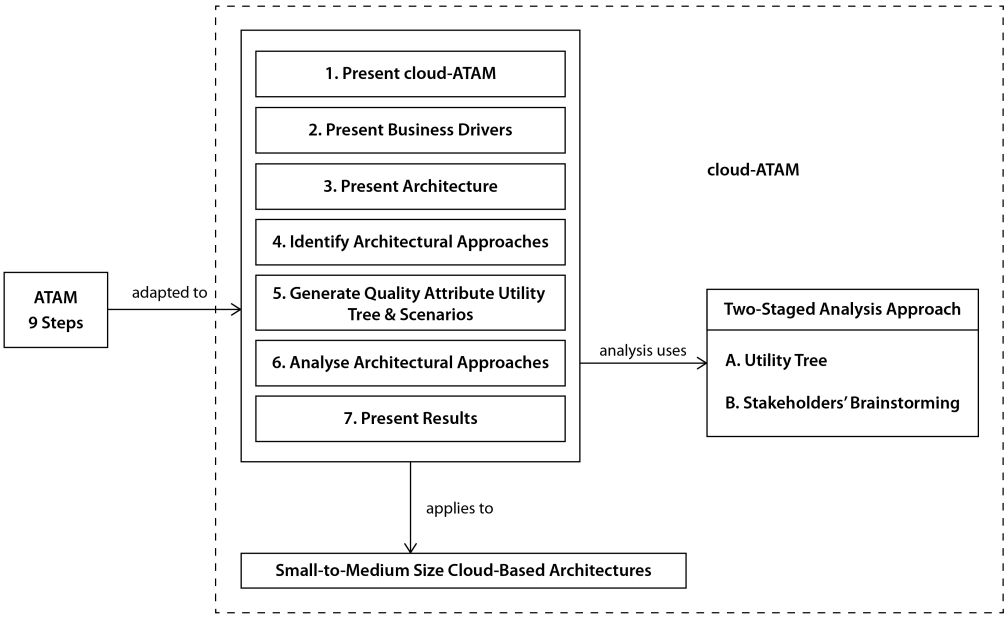


Figure 3.3: cloud-ATAM: Adapted ATAM with Two-Staged Analysis Approach

The comparative studies resulted in the identification of the Architecture Trade-off Analysis Method (ATAM), which satisfied the three criteria above. ATAM has the potential of considering several architectural quality attributes and appropriately represent an architecture, and analysing the sensitivities of and

trade-offs between these multiple quality attributes in a hyper dynamic environment such as the cloud.

That said, ATAM is presented as a generic methodology for analysing the quality attributes of all sizes of architectures. The nine-step approach of ATAM is perceived to be complex especially in terms of analysing small-to-medium sized architectures. There is an obvious need for tailor-made variations of this methodology to fit small-to-medium sized architectures, which are considered to be the largest classification of architectures worldwide.

In this work, we provide an enrichment to ATAM with a derived methodology called cloud-ATAM, which is used to guide the design, analysis and evaluation of cloud-based architectures. The cloud-ATAM also presents a two-stage approach for the qualitative analysis and evaluation of small-to-medium size cloud-based software architectures. Here, the derived methodology is used to design, analyse and evaluate the Reactive Architecture as a proof of concept.

The introduced methodology (i.e. cloud-ATAM) depicted in Figure 3.3, is motivated by the complex and iterative nature of the ATAM even for small-to-medium scale architecture. Typically, small-to-medium scale systems do not need to undertake all the steps of the ATAM. Due to the limited size of such projects, some steps can be combined into a new step and some activities of some steps can be optional. Here, "generating the quality attribute utility tree" process of "Step 5" can be combined with the "prioritising scenarios" process of "Step 7". Also, the duplication of the "analysing the architectural approaches" process of "Step 6" and "Step 8", can be combined into one process. This new step (i.e. "analysing the architectural approaches") can be extended by "noting the impact of scenarios on the architectural approaches" as undertaken in "Step 8". Activities such as critical requirements, technical constraints, and quality attributes are necessary under "Step 2: Present Project Drivers". These changes are particularly important especially in addressing the perceived weakness of ATAM due to its iterative nature which requires a substantial number of human experts on the team at different times. It is often expensive to speculate the availability of such domain experts for small-to-middle size software projects

due to budgetary or time constraints. The resulting steps for cloud-ATAM is provided below:

1. Present the cloud-ATAM
2. Present the Project Drivers
3. Present the Architecture
4. Identify Architectural Approaches
5. Generate the Quality Attribute Utility Tree and Scenarios
6. Analyse the Architectural Approaches
7. Present Results

Table 3.1: Utility Trees vs. Scenario Brainstorming

	Utility Trees	**Facilitated Brainstorming**
Stakeholders	Architects, Project Leader	All stakeholders
Typical Group Size	2 Evaluators; 2-3 Project Personnel	4-5 Evaluators; 5-10 Project-Related Personnel
Primary Goals	1. Elicit, concretise and prioritise the driving quality attribute requirements. 2. Provide a focus for the remainder of the evaluation.	Foster stakeholder communication to validate quality attribute goals elicited via the utility tree.
Approach	Top-down (general to specific)	Bottom-up (specific to general)

Here, steps (1) to (5) are used to design small-to-medium size architectures. Also, cloud-ATAM presents an enrichment in terms of analysis coverage under steps (6) and (7) to ATAM in the form of a two-stage scenario-based analysis and evaluation approach:

(a) Utility Tree, and

(b) Stakeholders' Brainstorming

Table 3.1 highlights the differences between the utility tree mechanism and the stakeholders' brainstorming mechanism.

Utility Tree Analysis Mechanism

Utility tree analysis mechanism provides a top-down mechanism for directly and efficiently translating the project drivers of a system into concrete quality attribute scenarios. We need to understand the relative importance of these project drivers as against other quality attribute drivers to determine where the architecture evaluation focus should be. Utility trees help to detail and prioritise quality goals. We empirically analyse the Reactive Architecture by qualitatively reasoning (i.e. using the Utility Tree Mechanism) about a set of scenarios of the Reactive Architecture, with the goal of identifying sensitivities and trade-offs with system quality attributes, and providing a healthy balance between the risk points identified among the quality attributes of the cloud-based Reactive Architecture.

Stakeholders' Brainstorming Mechanism

The scenario brainstorming mechanism is stakeholder-centric, which elicits points of view from a more diverse and larger group of stakeholders, and verifies, and then builds on the results of the first phase. It involves the evaluation of the cloud-ATAM, and this is undertaken through an organised group work of stakeholders who analyse the trade-off themes on the project drivers. Here, it uses the expert panel review process to evaluate the architecture. The expert panel review process has been discussed in Section 2.1.5.4. This mechanism is relevant in facilitating communication among stakeholders and also the stakeholders with the team of architects of the architecture.

These analysis approaches (i.e. utility tree and stakeholders' brainstorming mechanisms) are based on quality attributes in identified scenarios of an architecture. This work focuses on the sensitivities and trade-offs between multiple dependability quality attributes such as availability and performance. The jus-

tification of the choice of availability and performance is mainly as a result of the nature of the cloud environment, which in turn affects the deployed system such as the Reactive Architecture. As discussed in Section 2.3.2, the cloud environment is characterised by rapid evolution of the cloud topology, which has a high potential of resulting in hardware faults and software errors. Since hardware and software faults are stimuli for availability (discussed in Section 2.2.3), it is very relevant to focus on availability as a quality attribute. In characterising availability, the strategies adopted or parameters are redundancy, voting, retry and failover support. Also, the responses are service availability, reliability, and the level of the service. Here, the cloud service providers (CSPs) use the service level agreements (SLA) to assure clients of the availability and performance of their platforms. However, these SLAs are perceived to be weak and hence, lack customer trust, and the cloud platforms are perceived to lack reliability (see Section 2.3.2.2).

In Section 2.2.2.1, we provide the definitions and taxonomy of software system dependability. Here, a directly proportional correlation is identified between availability and performance. The presence of faults as a result of internal and external events in the cloud platform can affect latency, throughput and precedence. Since cloud resources are mainly shared (e.g. co-tenancy of virtual instances), uncontrolled and malicious activities can also affect the performance of other deployed systems. Even though security is a major area of concern now and essential, we identify availability and performance as more fundamental. Security attacks such as distributed denial of service (DDoS) fundamentally affect availability of service and performance. More so since current literature (see Section 2.2.5.1) do not effectively address the issues of availability and performance for software systems deployed to the cloud environment.

cloud-ATAM generates a number of outputs such as: a prioritised list of quality attributes, a list of architectural decisions made, a map linking architectural decisions to attributes, lists of risk and non-risks, and lists of sensitivities and trade-offs. In this work, we use the derived cloud-ATAM to design the Reactive Architecture, and to analyse the trade-off between the availability and performance attributes. To support the analysis of the Reactive Architecture,

cloud-ATAM considers a non-trivial set of scenarios, and uses a two-staged analysis approach. We answer the research question: "What is the trade-off between availability and performance quality attributes identified by the cloud-ATAM for the cloud-based Reactive Architecture?"

3.3.3 Assuring Dependability in the Cloud

The cloud computing technology of today and the future promises to bring demonstrable benefits to people's lives. It presents an outsourcing model which is attractive for businesses that wish to minimise their computing and storage infrastructure cost. Here, the cloud service provider (CSP) is responsible per the service level agreement (SLA), for the availability of services and clients are free from maintenance and management problems of the resource machines. However, the responsibility of the CSP is often called to question. Literature review (refer to Section 2.3.4.3) shows that these concerns are largely related to the assurance of security (in terms of intrusion detection) and dependability on the cloud. This work pays a closer attention to the dependability assurance provided by the CSPs, with specific focus on availability and reliability. This choice is justified by the large quantity of literature focusing on security (see Section 2.3.4.1), and a few for dependability. From our observation, most of these literature identify some aspects of dependability as very critical. Such aspects are the level of service and reliability (see availability charaterisation in Section 2.2.3). These two aspects are the responses or character of availability and also reliability. Here, a good plan for availability and reliability in terms of software and hardware redundancy can also help to mitigate the impact of security attacks.

With the movement of software engineering from local computers to the cloud for global software development (GSD), software developers need to be assured of the dependability of the engineering support deployed to the cloud and the cloud environment. The predefined and mutually agreed upon business logic and SLA provided by CSPs attempt to assure developers of the cloud performance, availability, reliability, etc. However, due to the cloud platform's

inherent complexity and large scale, production cloud computing systems are prone to various run-time problems caused by hardware and software faults, cloud run-time management decisions and environmental factors (see Section 2.3.2). Such SLAs in this context are considered weak, and unable to guarantee minimum levels of performance (see Section 2.3.2.2). Cloud agents such as system developers, CSPs, and cloud regulators need to be informed about possible or actual violations of the SLAs (when for instance, there is a request time-out due to the developer specifying a longer time-out than the cloud's SLA provides). A robust mechanism is needed for violation detection, notification, logging and a means towards resolution. Once the cause of the violation is found, each violator is regarded as being accountable for their fault.

This however highlights the need for cloud accountability (discussed in Section 2.3.4). Creating accountability in the cloud is seen as a solution to users' lack of trust. Accountability refers to a situation, where both the CSP and the clients are able to check whether the cloud is running the service as agreed. If a problem appears, they should be able to determine which of them is responsible, and to prove the presence of the problem to a third party, such as an arbitrator or a judge. Such an activity should be based on evidence. An advanced approach for using the cloud infrastructure's hypervisor for providing evidence in digital forensics is Virtual Machine Introspection (VMI) (Section 2.3.4.1). VMI leverages the capabilities of the hypervisor to look "inside" the virtual machine during runtime and using information collected this way for evidence-based auditing. The VMI is largely used for system security investigations (i.e. digital forensic) especially in the area of intrusion detection (e.g., detecting malware).

To support the trustworthiness of the data collected and methodology, we consider some trust assurance methods: reputation-based, SLA-based, and evidence-based (see Section 2.3.5). However, we identify that the reputation-based trust assurance methods are very subjective as this trust assurance method considered the opinions of cloud agents. These agents may be biased and will not accurately assess the cloud platform and associated CSPs. In this work, a combination of SLA-based and evidence-based trust assurance methods is considered. These methods are dependent on evidence which can provide a more

accurate representation of the level of trust assurance.

A contribution of this work is a Cloud Accountability System (CAS) which is a component of the Reactive Architecture. It provides dependability assurance of cloud resources to cloud agents. This assurance is provided relative to the cloud SLA and evidence collected from the cloud platform. The CAS facilitates the Cloud Accountability Methodology which is guided by the NIST SP800-86 forensic model, that motivates the collection, examination and analysis of data from the cloud infrastructure, and the generated evidence including logs and context are reported to appropriate cloud agents. This work also presents a novel approach to collecting digital evidence to support cloud-based system dependability, using the VMI technique. This methodology aims to:

- Assure cloud agents of the dependability of the cloud infrastructure with reference to cloud SLAs.
- Quantify (in monetary terms) the violations to SLAs, using the Net Present Value (see Section 2.3.4.2). This can serve as a reference point for compensating cloud agents, as well as providing punitive charges to violators.
- Serve as an evidence-based benchmark for choosing a relatively dependable cloud provider.

The broad research question (RQ) that we seek to answer is "Can a cloud accountability method be used to meaningfully assure availability and reliability of deployed systems, relative to the cloud platform's service level agreement (SLA)?". To answer this question, we validate the hypothesis (H1): "The cloud accountability method can be used to meaningfully assure the availability and reliability of cloud-based systems".

3.4 Summary

This chapter initially discusses the challenges of GSD as the absence of an effective information and knowledge sharing facility, automation of such facility,

application of diverse set of tools, the dependability of such facility, and an accountable cloud as a GSD operational environment. We introduced a cloud-based framework called the Reactive Architecture which has the potential to address the discussed challenges (this meets contribution 3 in Section 1.3). Here, the Reactive Architecture presents three approaches to the challenges:

(a) mechanism for requirement change management and tracebility which is facilitated by the Reactive Middleware (which meets contribution 1 in Section 1.3),

(b) design and analysis methodology for cloud-based architectures (which meet contribution 2 in Section 1.3), and

(c) a methodology for assuring the dependability of systems in the cloud (which meets contributions 4 and 5 in Section 1.3).

The following chapter looks at managing requirement change and traceability. It also elaborates on the design decisions pertaining to the Reactive Middleware, software engineering artefact and their trace links, as well as data representation.

Chapter 4

Managing Requirement Change and Traceability

This chapter contributes a Reactive Middleware which facilitates a set of guidelines defined to manage change and traceability in Global Software Development (GSD). The Reactive Middleware is a critical and central component of the Reactive Architecture, as it provides cloud-based services for user management, requirement management, change management, and traceability of GSD project requirements and system development phases. We first present a generalised process model for change management and traceability for GSD, and then detail our management approach for system engineering processes as part of the presented GSD guidelines. This contribution satisfies *Objectives 1* and *2* presented in Section 1.2.2.

In this chapter, Section 4.1 introduces our work, and the Reactive Middleware that facilitates the defined system engineering guidelines is presented in Section 4.2. Here, the components of the Reactive Middleware are discussed as well as the management guidelines, which is composed of the change management and traceability process model and the system engineering management process approach. Also, Section 4.3 discusses the approaches and services provided by the Reactive Middleware in the context of an optimal set of high level requirements for a GSD framework operating in the cloud environment, presented in

Section 3.3.1.1. Finally, the conclusions of this chapter is drawn and presented in Section 4.4.

4.1 Introduction

In this chapter, we present details of the Reactive Middleware for coordinating and tracing changes made to system engineering artefacts created and/or used at various system engineering phases. In this vein, we present the core process model for change management and traceability along with discussion of the management approach for system engineering processes and coordinating agents. Our GSD management guidelines involving the application of the presented management approach for the change management and traceability process model, are easily generalised to other system engineering phases.

4.2 Reactive Middleware

4.2.1 Overview

The *Reactive Middleware* (RM), which is the chapter's main contribution introduces a structured *role-based management* of system development artefacts. Role in this context is described as stakeholders' responsibility (i.e. privilege) to system artefacts. Also, this approach assigns priority to system requirements relative to their importance. The role-based management approach uses six main privileges (with roles): *None*: Have no access to the system artefact(s) for **PAWNS**, *View*: Only sees the system artefact(s) for **PAWNS**, *Modify*: Can see (view) and change the system artefact(s) for **MODIFIERS**, *Review*: Can see and change a modification to system artefact(s), in response to a set of notifications for **REVIEWERS**, *Create*: Can create and modify (view, modify) system artefact(s) for **CREATORS**, and *Own*: Full access (view, modify, review, delete, recall) to the system artefact(s) for **TEAM LEADERS**. Here, *recalled* artefacts are reinstated deleted artefacts.

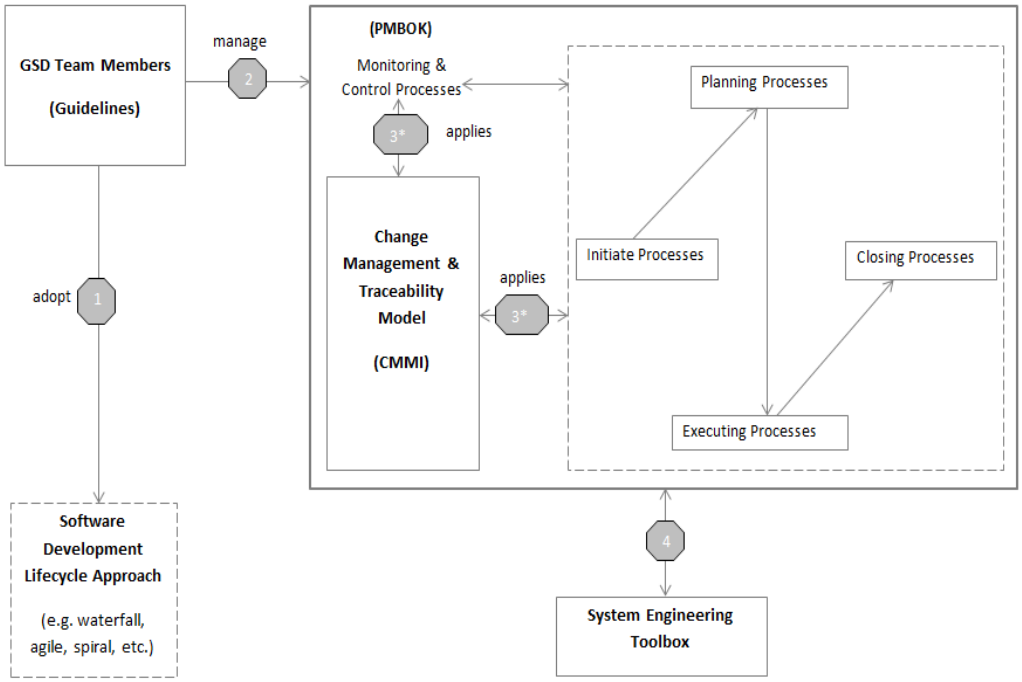


Figure 4.1: Reactive Middleware Interactions

4.2.2 Change Management and Traceability as a Service

The RM is composed of the *Publish/Subscribe system* (PSS) and the *Artefacts Monitoring system* (AMS), which together provides cloud-based services towards user management, requirement management, change management, and traceability. The PSS facilitates the subscription of system stakeholders to relevant artefacts, to which they will initiate change requests or will receive change notifications. Also, the AMS is responsible for monitoring the relevant artefacts for changes, and then triggers the PSS to notify appropriate stakeholders or change agents. The RM interacts with the system stakeholders, System Engineering Tools, and a Shared Artefacts Repository. The RM facilitates management guidelines for GSD projects that applies *quality process management*, to a *change management and traceability process model*. From Figure 4.1, the GSD Team Members have the flexibility to *adopt* any type of software development life-cycle (SDLC) approach (e.g. waterfall, agile, spiral, etc.) that suits

their development style (i.e. *Step 1*). Then following the prescribed *management guidelines* featured by the Reactive Middleware, the GSD Team Members *manage* the development process with the PMBOK process group for system engineering life-cycle (i.e. *Step 2*). When there are *change requests* that are related to the high priority requirements, the GSD change managers *apply* the change management and traceability process (CM-T) model to either *approve*, *note* (i.e. to be applicable in the future) or *disapprove* the request (i.e. *Step 3**). This CM-T model takes into consideration the bidirectional traceability of the change agents (i.e. system stakeholders, artefacts and tools) involved in the change request. System engineering tools form an important change agent in the development process (i.e. *Step 4*).

Table 4.1: Mapping the Key Steps for Effective Requirements Management and Traceability Processes with Proposed Change Management and Traceability Services

5 Key Steps	**Proposed Change Management and Traceability Services**
Trace creation and maintenance	The PSS facilitates the initiation of change requests and receiving change notifications.
Detect change	The AMS monitors the relevant artefacts for changes, and then triggers the PSS to notify appropriate stakeholders or change agents.
Analyse change impact	the GSD Team Members manage the development process and change with its impact using the GSD guidelines.
Check for consistency	When there are change requests that are related to the high priority requirements, the GSD change managers apply the change management and traceability process (CM-T) model to approve, note or disapprove the request.
Propagate change	This CM-T model takes into consideration the bidirectional traceability of the change agents (i.e. system stakeholders, artefacts and tools) involved in the change request.

In Table 4.1, we provide a mapping of the identified five key steps for effective

requirements management and traceability processes (refer to Section 3.3.1.1) with our proposed approach facilitated as services by the Reactive Middleware.

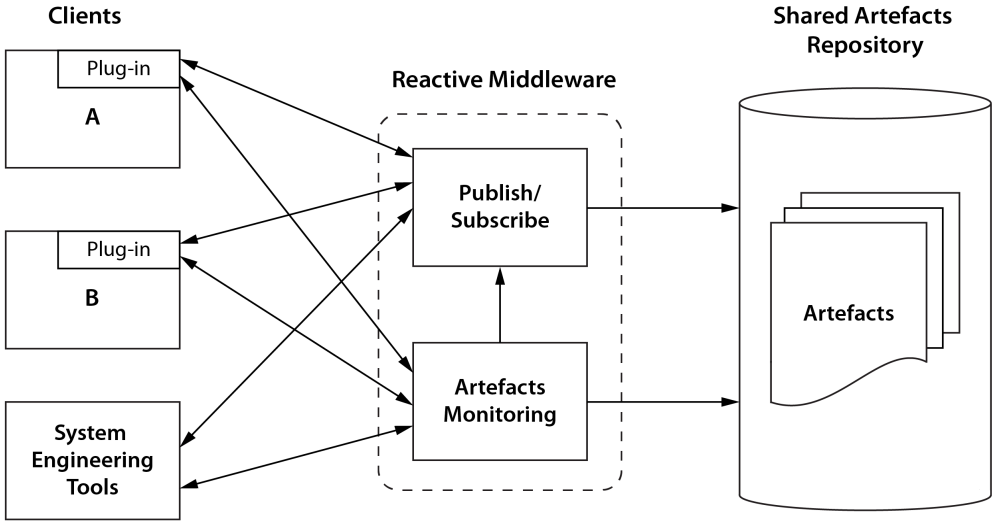


Figure 4.2: Overview of the Components of the Reactive Middleware

4.2.2.1 Publish/Subscribe System

This system implements a Publish/Subscribe mechanism. Here, all actors (i.e. system stakeholders and tools) involved in the development and evolution of an artefact subscribe to that artefact (see Figure 4.2). Artefacts can only be accessed by specific authorised actors. The Artefacts Monitoring System notifies the Publish/Subscribe System when there is a change made to an artefact. The Publish/Subscribe System then identifies and notifies all the actors that have registered their interest in the artefact.

That said, the Publish/Subscribe System specifically provides services for user management as the stakeholders (1) register to use the Reactive Middleware, and (2) roles and privileges are assigned to them. Also, the system provides services for requirements management: (1) subscription to a set of artefacts to which they (2) receive notifications.

Table 4.2: Description of Artefacts in the Shared Artefacts Repository

SDLC Phases	**Artefacts Description**
Requirements	System models are saved as artefacts. These requirements artefacts are obtained from tools such as ProR.
Specification	Developed system model specification are saved as artefacts. The model specification contains elements such as invariants, guards, actions, etc. Such elements are also extracted as dependent artefacts.
Implementation	Implementable source codes of model specifications are saved as artefacts. These source codes are generated with appropriate tools such as EB2ALL. It supports automatic code generation from Event-B to C, C++, Java and C. Another tool example is EventB2Dafny. This tool extends the Boogie and Dafny tools, and allows the use of Dafny static analysis machinery based on design-by-contract principles. This yields artefacts in the form of executable input code for Boogie and Dafny.
Documentation	Documentation artefacts are in the form of: (1) traceability logs, (2) incident reports, and (3) others such as designs, test plans, execution results, etc.

4.2.2.2 Artefacts Monitoring/Interpretation System

The Artefacts Monitoring/Interpretation System (AMS) is a subsystem of the Reactive Middleware. This subsystem interacts with the Shared Artefacts Repository to monitor changes made to artefacts. The OSLC technology (discussed in Sections 2.1.6 and 3.3.1) used for *monitoring* in AMS provides standardised methods to represent, access, and link to resources. With OSLC specifications, tools can freely understand each other's data and artefacts. This makes it easy to better analyse, track, and explore that data to make better decisions. There are feature to support change management, traceability, etc.

In OSLC, resources are identified by OSLC annotations. So projects and their composing elements being developed in IDEs are tagged with these OSLC annotations, to reveal their artefacts to the client plug-in. Information about

Table 4.3: Global Software Development Management Guidelines

Guidelines Steps ID	Guidelines
GS1	System development teams should appoint team leaders.
GS2	These team leaders will constitute the GSD change managers
GS3	System requirements should be classified based on identified dependability quality attributes (i.e. safety, reliability, robustness, etc.), and are then prioritised relative to their importance to the system stakeholders.
GS4	Team leaders must assign roles to all team members with the prioritised requirements in mind, and manage the development process with the adapted PMBOK guide.
GS5	All other change agents especially the system engineering tools should be assigned a default privilege of review.
GS6	All system artefacts should be saved in a shared artefacts repository.
GS7	The privileges (i.e. none, view, modify, review, own) of system stakeholders or change agents will determine the access privileges to system artefacts.
GS8	Change agents must subscribe to relevant artefacts after they are created, in order to receive notifications when they are changed.
GS9	All related artefacts must be linked together to facilitate traceability.
GS10	Changes made to any system artefacts must be logged.
GS11	When changes affect the high priority set of requirements, appropriate local team leader must lead the change request review process (i.e. involving the CM-T model) of the GSD change managers.
GS12	On the other hand, conflicts arising from changes to low priority set of requirements are resolved locally, lead by the local team leader.
GS13	Changes in system artefacts should be traceable to manage its impact on related/linked requirements or artefacts.

these resources/artefacts are gathered and formatted as XML files. These files are sent from the client plug-in periodically to the OSLC sub-system in the Reactive Architecture. Here, all various versions of the XML files are saved. The main functionality of this sub-system is to ensure change management and traceability of artefacts. For traceability, all activities on an artefact such as creation, modification, deletion are recorded and stored. Also for change management, the latest information about artefacts is compared with information from newly arrived XML files. Whenever a change in the files is identified, a notification is sent to the Publish/Subscribe sub-system.

Here, the services provided are requirements management, change management and traceability. Here, changes to artefacts relating to a set of requirements are managed, and then traced to ensure consistency through impact management and logging.

Different types or formats of artefacts are considered here (see Table 4.2). This enhances the capacity of the AMS to support independent artefacts, however metadata are created as XML formats. This format allows for managing traceability.

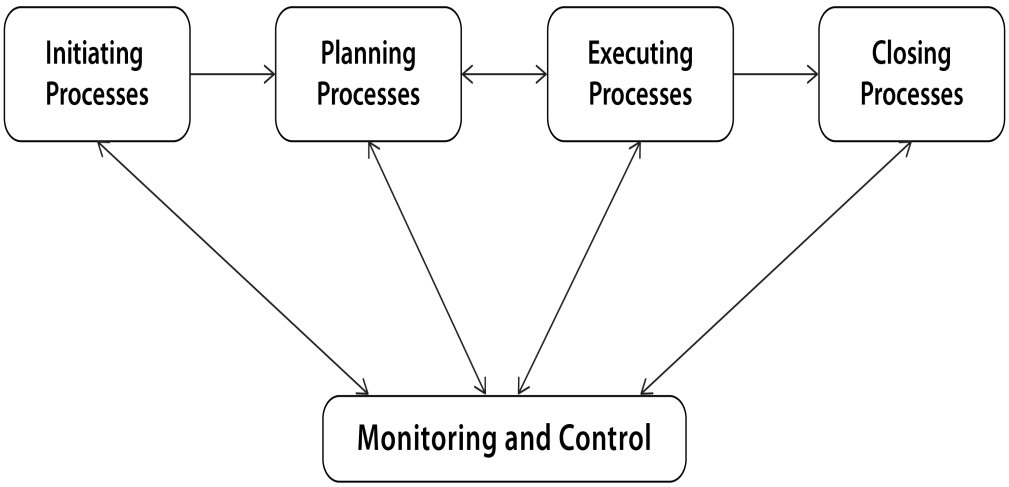


Figure 4.3: PMBOK® Process Group for System Engineering Life-Cycle

4.2.2.3 Management Guidelines for System Engineering

We introduce the defined management guidelines for GSDs (see Table 4.3). The GSD guideline essentially defines a generic development policy for software engineering projects. As part of the GSD guidelines, we present our PMBOK process group for managing quality system engineering processes, and then the change management and traceability (CM-T) process model. The PMBOK process group for managing quality system engineering life-cycle (see Figure 4.3) plays an overarching role in the GSD guidelines. This management approach is applied to the CM-T process model.

4.2.2.4 PMBOK 5-Step Process Group for System Engineering Life-Cycle

The PMBOK is a project management guide that is a well accepted standard which provides a general framework for project management. The PMBOK contains 42 project management (PM) practices organised by two orthogonal categories: *Process Groups* (PG) and *Knowledge Areas* (KA). Here, each of the 42 processes belongs to exactly one process group and to exactly one knowledge area. The PG organisation shows the project's life cycle (see Figure 4.3), involving 5 groups: Initiating, Planning, Executing, Monitoring & Control, and Closing. These five process groups are introduced below:

(a) *Initiating*: An idea for a project will be carefully examined to determine whether or not it benefits the organisation. During this phase, a decision making team will identify if the project can realistically be completed.

(b) *Planning*: A project plan and/or project scope may be put in writing, outlining the work to be performed. During this phase, a team prioritises the project, calculate a budget and schedule, and determine what resources are needed.

(c) *Executing*: Resources' tasks are distributed and teams are informed of responsibilities. This is an appropriate time to bring up important project related information.

(d) *Monitoring and Controlling*: Project managers compare project status and progress to the actual plan, as resources perform the scheduled work. During this phase, project managers may need to adjust schedules or do what is necessary to keep the project on track.

(e) *Closing*: After project tasks are completed and the client has approved the outcome, an evaluation is necessary to highlight project success and/or learn from project history.

The KA organisation groups the processes into 9 knowledge areas, according to their application to a specific aspect of project management, such as cost, schedule, quality, risks, etc. The PMBOK's chapters follow the KA organisation, where the order of processes in each KA is determined by their chronological application in the project, according to the PG organisation.

4.2.2.5 Change Management and Traceability Process Model

The (CM-T) process model is expected to ensure a *matured* change management and traceability processes relative to the specific practices of the CMMI Level 2. To begin the definition of the process model, we indicate the main processes involved in validating the CM-T process model as to:

1. Provide objective(s) for building the model;
2. Show the criteria identified during the initial stages of model development;
3. Design a validation instrument to test the success criteria (to include methods for reporting/analysing responses);
4. Select an expert panel to reflect the population of experts in Software Engineering, Requirements Engineering (RE), and CMMI; and
5. Present results of the validation instrument.

Table 4.4: CM-T Process Model Validation

Criterion	Purpose	Rule	Source
Adherence to CMM Characteristics	The new model should be recognisable as a derivative of established models - both in structure and concept. By tapping into the established models, the CM-T model takes the strengths of a proven improvement structures and becomes more accessible and compatible, avoiding redundant activities.	- CMM maturity level concepts must be implemented - Each level should have a theme consistent with CMM - Requirement engineering (RE) processes must be integrated - The model should be recognisable as a CMM offshoot - The CM-T must be systematic and sequential	Where possible we should adapt existing models rather than create new ones Maturity levels help characterise a process and set out a strategy for its improvement
Limit Scope	CMM goals, RE phases and RE processes define the boundaries of the model. The model does not include all RE processes.	- Key activities relating to technical and organisational RE processes are included - Processes are prioritised. - Processes relate directly to the CM-T process areas - The scope/level of detail should be appropriate (i.e. depth and breadth of processes presented)	It is important to know the scope of the model, i.e. what the model includes and excludes
Consistency	Having an acceptable level of 'construct' validity will help users navigate within levels of maturity as well as between different levels of process maturity. Model development and adaptation depends on an acceptable level of consistency.	- There should be consistent use of terms and CMM features at this level of development - There will be a consistency in structure between model components at the same level of granularity that are modelling different maturity levels.	To understand a model it is important that there is a common language. Each stage of development should describe processes at similar levels of granularity
Understandable	All users of the model should have a shared understanding of the RE process in order to identify where improvement is needed. There should be no ambiguity in interpretation, especially when goals are set for improvement.	- All terms should be clearly defined (i.e. have only one meaning). - All relationships between processes and model architecture should be unambiguous and functional.	The importance of clear definitions. Understanding is a prerequisite for effective process improvement and management
Ease of Use	Over-complex models are unlikely to be adopted as they require extra resources and may be too challenging for the user to interpret without extensive training. The model will have differing levels of decomposition starting with the most high level in order to gradually lead the user through from a descriptive model towards a more prescriptive solution	- The model should be decomposed to a level that is simple to understand - The model should be simple yet retain meaning - The chunks of information should clearly relate as they develop into more complex structures - The model should require little or no training to be used	Usability is a key requirement of any process improvement model
Verifiable	Model strengths and weaknesses need to be tested to help direct future model development. Validation of the model will help to improve the model, add confidence in its representation and help with research in this area.	- The model must be verifiable, i.e. we must be able to test/measure how well model meet its objectives and whether meeting these objectives leads to a high quality CM-T process model.	To assess whether a process is useful, well implemented the model needs to be verifiable

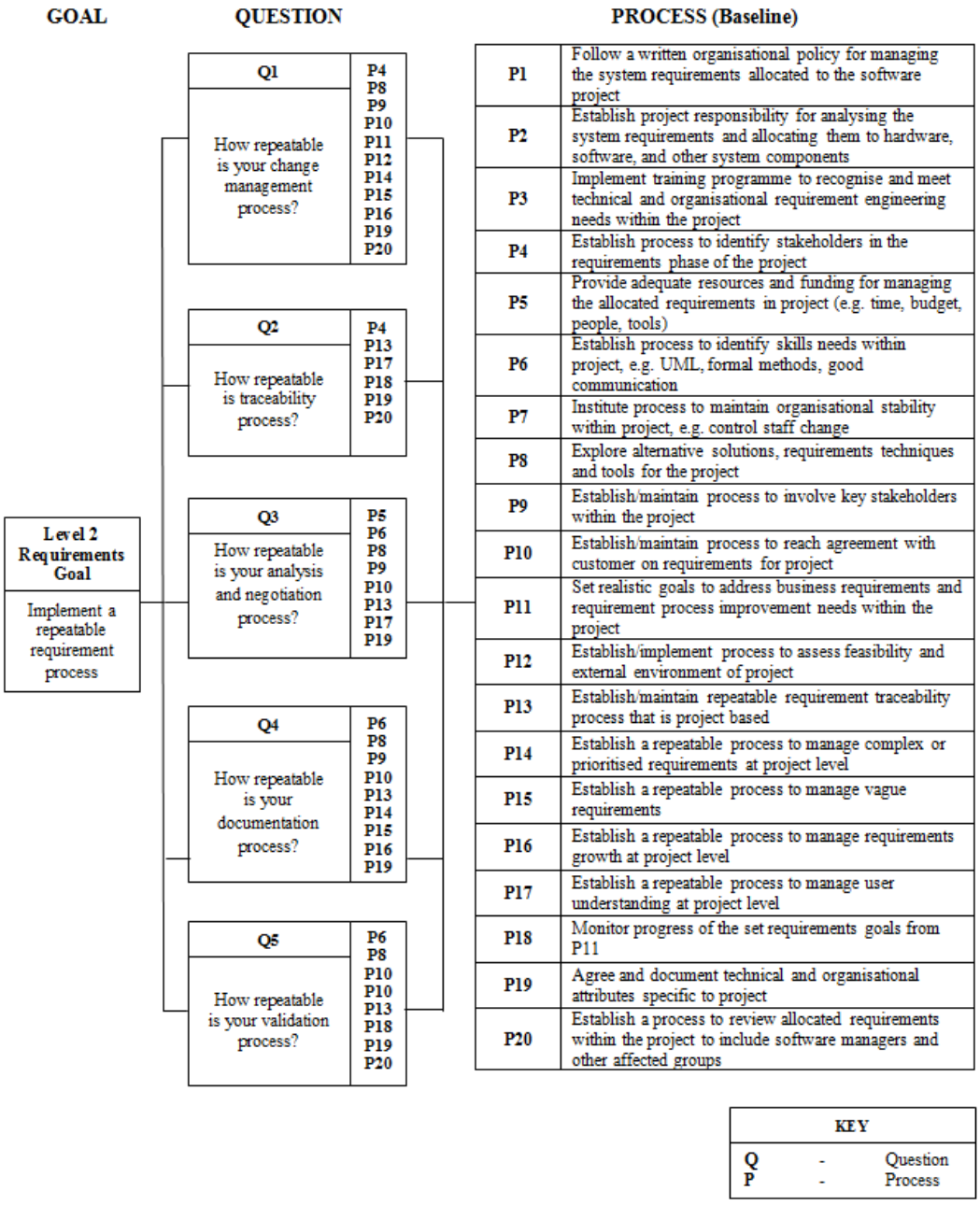


Figure 4.4: Candidate Processes Reflecting a CMMI Level 2 (Baseline) Capability

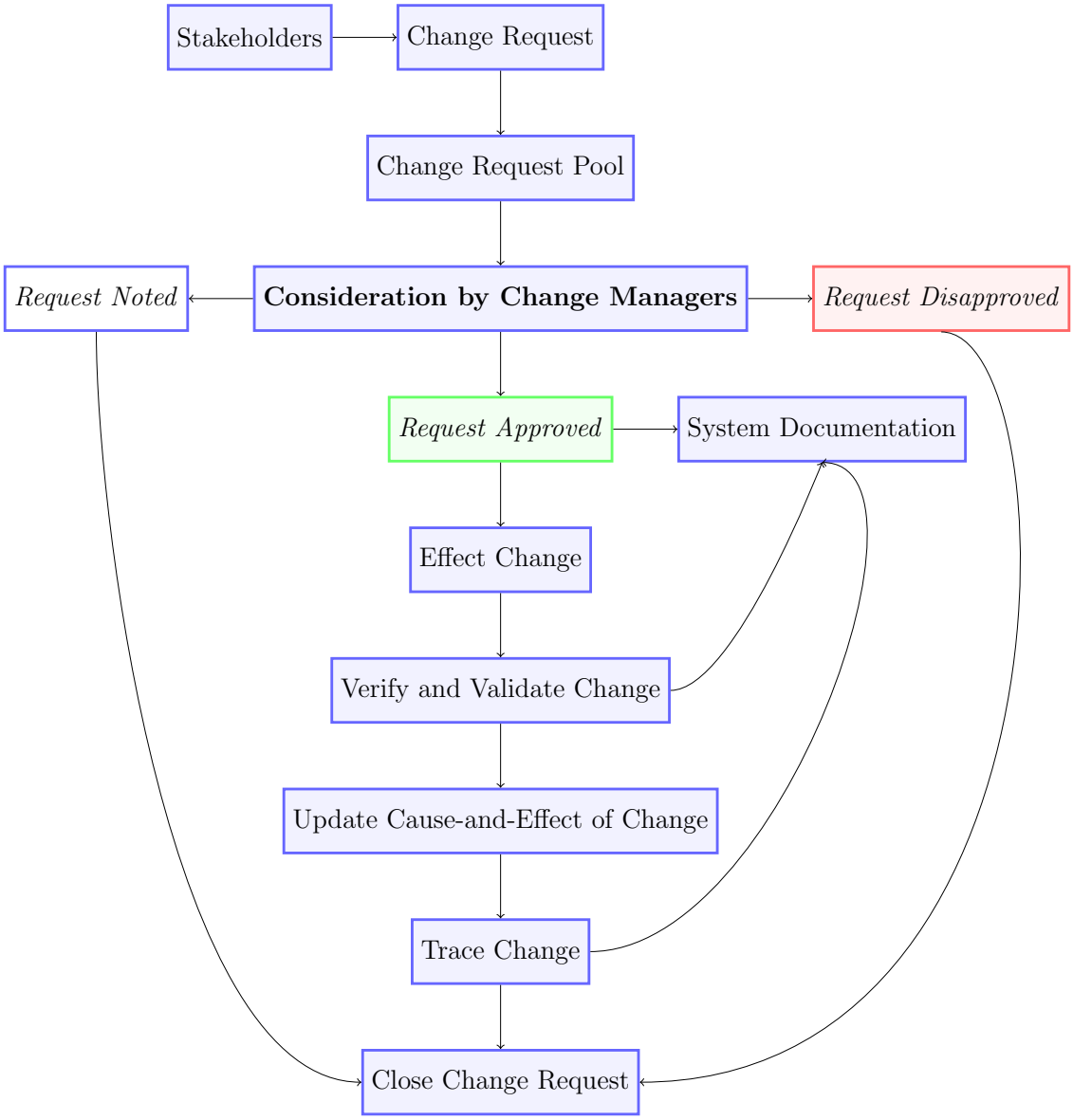


Figure 4.5: Change Management and Traceability Process Model

1. Provide objective(s) for building the model

We aim to develop a model that represents key practices in RE within a maturity framework mentioned earlier. Here, our objectives are:

(a) A primary objective of our CM-T model is to guide software stakeholders to relate key change management and traceability processes to goals in order to prioritise their requirements process improvement activities.

(b) The CM-T process model should strengthen components of the CMMI (involving software requirements) to clearly focus on the change management and traceability processes.

(c) Our model should complement the CMMI (Level 2) so that practitioners are not required to learn another software process improvement methodology.

(d) Finally, we aim to link theory to practice through a model that is easy to use and interpret.

2. Show the criteria identified during the initial stages of model development

We initially identify six relevant success criteria based on CMMI Level 2 baseline capability (see Figure 4.4) to guide the development of the CM-T model. Success criteria were established per evaluation question. These success criteria are presented in Table 4.4. The criteria were identified using a method similar to that used in the SPICE trials (see Section 2.1.5.2) to support the ISO/IEC 15504 emerging standard. The resulting CM-T process model (see Figure 4.5) considers the development teams and system engineering tools as the main *agents of change*. In order for the change process to be initiated, a *change request* must be initiated in a GSD project. All such change requests are drafted as a *change request form*. All submitted *change request forms* go into a centralised *change request pool*. The *change managers* group is made up of all the *GSD team leaders* and other relevant stakeholders. The *change managers* consider and timely decide on all the submitted *change requests*. During meeting sessions for considering submitted *change requests*, the *GSD team leaders*

will lead the consideration of all change requests that originates from their team. Team leaders are expected to have critically assessed, and understand change requests before they are approved for submission. The decisions take a form of an approval, disapproval, or noted. A disapproved change request means a termination of its consideration, and hence not to be effected. Also, a noted change request means an acknowledgement of relevance, but cannot be effected at the point of time in the GSD project. Such a change request, can be reconsidered at a future time determined by the *change managers* or resubmitted when deemed relevant by the *initiator* and approved by the *GSD team leader*. On the other hand, the *initiators* of the *change requests* that are approved are notified to *effect the change*. This change is *verified* and *validated* after the change has been effected. The verification and validation process assesses the immediate impact of the change on the GSD project. A detailed assessment of the *cause-and-effect* of the change is then undertake, and all minor *conflicts* (i.e. affecting less prioritised project requirements) are resolved within the local GSD team. In situations where the initial change affects a *prioritised project requirement*, the *GSD team leader* will advice the most relevant stakeholder to submit a *change request* to the *change managers* for consideration. From the point where an accepted change is being effected till the point where it has been implemented successfully, a process to trace all the change with regards to participating stakeholders, associated software development life-cycle (SDLC) phase, corresponding system engineering tools, impact on other artefacts, etc. is undertaken in parallel. This period also sees the creation or modifications to system documentations. At this point, the *change managers* accept the change and it is marked as successful. Then a generation of notification to all stakeholders of the change, and finally the change request is closed.

3. Design a validation instrument to test the success criteria (to include methods for reporting/analysing responses)

We design a validation instrument to test the success criteria provided in Table 4.4. We choose a questionnaire as a validation instrument since it mainly provide relatively precise responses for evaluations,compared with interviews. This questionnaire is provided in Appendix C.15. We consider a set of experts who

provides their responses to the questionnaire for validation.

The validation process meeting objective (i.e. **4. Select an expert panel to reflect the population of experts in Software Engineering, Requirements Engineering (RE), and CMMI**), and the reporting process meeting objective (i.e. **5. Present results of the validation instrument**) are presented as the validation of the CM-T process model by the expert panel and evaluation of Reactive Middleware with regards to the services it provides respectively in Section 7.1.

4.2.3 Shared Artefacts Repository System

The Shared Artefacts Repository is a component of the Reactive Architecture, and it stores all relevant system engineering artefacts (see Table 4.2). An artefact is one of many kinds of tangible by-products produced during the development of software systems. Some artefacts (e.g. requirements, plans, designs, source code, test plans and results, problem reports, reviews, notes, use cases, class diagrams, and other Unified Modeling Language (UML) models and design documents) help describe the function, architecture, and design of software systems. Other artefacts are concerned with the process of development itself such as project plans, business cases, and risk assessments. For this version of the architecture, focus will be on defining artefacts that describes the functions and design of the components of the architecture.

4.2.4 System Engineering Tool Support System

The Toolbox system is composed of tools that have been classified based on system engineering phases. The Toolbox is a component of the Reactive Architecture. The Reactive Middleware interacts with the Toolbox by notifying system engineering stakeholders of changes that tools in the Toolbox make to artefacts. The tools in the Toolbox can be coordinated in a form of compositions. System engineers can choose to compose the tools into either parallel or

Table 4.5: Meeting the High Level Requirements of an Effective GSD Framework

GSD Requirements	**Description of Approaches**
Effective information and knowledge sharing:	
R1: Artefact independence	Artefacts in the Shared Artefacts Repository are saved in their original formats, some generic XML derivatives are are generated for interoperability , as well as the generation of metadata for each artefact in XML format to facilitate change management and traceability. (**GS6**)
R2: Supports globally distributed development	The Reactive Middleware is deployed to the cloud environment to facilitate global accessibility. Also, it provides GSD services towards user management, requirement management, change management, and traceability. (**GS1**, **GS2**, **GS3**, **GS4**, **GS5**, **GS7**)
R3: Ability to handle different and large numbers of changing artefacts	The Shared Artefacts Repository has a high scalable capacity for varying formats of artefacts. Also, the CM-T process model has been designed to keep up with large volumes of changing to artefacts. An Airlock Control System case study has been provided in Section 7.1.2 to assess this. (**GS9**, **GS10**, and **GS11**)
Automation:	
R4: Automated as far as possible	The Reactive Middleware has been developed as a set of cloud-based REST web services to provide the mentioned set of GSD services. It mainly minimizes the manual effort involved in artefact consistency management involving change management and traceability. (**GS8**, **GS11**, **GS12**, and **GS13**)
Diversity of tools:	
R5: Tool integration	The System Engineering Toolbox provides a set of tools with different versions. These tools are integrated using the OSLC technology such that a workflow can be created. Also, outputs of one tool is reformated as an input for another. Plug-in for the toolbox is provided for variations of the Eclipse development environment. (**GS5**)

sequential scenarios. Such compositions can be applied to a set of tools that are classified across system engineering phases. In this situation, tools that support Requirements Engineering can be composed as a set. Also, tools that support different system engineering phases such as Requirements and Verification, can be composed in a workflow set. To achieve such a workflow, the OSLC technology is used. The OSLC technology facilitates the integration of different tools by providing a standardised liked data formats.

The Toolbox system receives requests from the Architecture Interface, which is then directed to the specified tool(s). The Web Service running the collection of tools implements the OSLC technology. Here, all data and artefacts generated by the tools are annotated as resources. The annotation process allows the definition of the format (plain text, XML, JSON-RDF, etc.) of data or artefacts. After this process, data or artefacts can either be sent to the Shared Artefacts Repository to be saved, or passed on to other tools in a workflow.

Developers have an option of selecting a particular set of tools from the list of tools on the plugin's "properties page". With this selection, artefacts can be passed to these particular tools. This selection can be changed at any time, but it can only take effect in the REQUEST that follows it. We demonstrate the application of the Toolbox to facilitate co-engineering and verification (i.e. model checking and theorem proving) of formal models relative to their requirements in Appendix B.12.

4.3 GSD Requirements Discussion

In Section 3.3.1.1, a set of high level requirements were discussed as optimal for GSD in the cloud environment. Here, we want to reassess our proposed approach of the Reactive Middleware in the context of the mentioned high level requirements. Table 4.5 shows this assessment.

With the successful mapping of the approaches or services provided by the Reactive Middleware and presented by the proposed GSD management guidelines, to the high level requirements of a desirable framework to provide solutions

to the state-of-the-art challenges in practicing GSD in the cloud environment, we are confident that the Reactive Middleware will guide system engineering to ensure the continual tight linkage of stakeholders' requirements and system engineering processes. That said, we evaluate this in Chapter 7.

4.4 Summary

In this chapter, we introduce a cloud-based Reactive Middleware that applies a defined change management and traceability (CM-T) process model, within the context of an adapted PMBOK quality process management approach to GSD. This is in line with Objectives 1 and 2 presented in Section 1.2.2. The Reactive Middleware provides cloud-based services for user managament, requirement management, change management and traceability, and are facilitated by our GSD management guidelines.

To ensure that the defined CM-T process model complies with the CMMI Level 2 (Baseline) Capability, an expert panel review process is used to validate it in Chapter 7. Also in that chapter, we demonstrate how the Reactive Middleware will guide system engineering to ensure the continual tight linkage of stakeholders' requirements and system engineering processes, by applying it to the GSD of an Airlock Control System.

Chapter 5

Designing Architectures for Global Software Development

This chapter presents the description and the design of the Reactive Architecture. The main contributions of this chapter are the introduction of a novel architectural trade-off analysis methodology referred to as cloud-ATAM, and the design of the Reactive Architecture for cloud-based system engineering. These contributions satisfy *Objectives 5* and *7* respectively presented in Section 1.2.2. The introduced methodology (see Section 1.3) assists in analysing the trade-off of quality attributes of software architectures. Here, we focus on software architectures with their sizes spanning the range of "base functional process" (BFP) to "macro functional process" (MFP) as classified with *functional size measurement* under the ISO/IEC 19761:2011 and COSMIC Full Function Point 2.2 standards. This size range is referred to as "small-to-medium size" in this chapter.

In this chapter, we discuss the design and analysis of the the Reactive Architecture using the cloud-ATAM. We begin by giving a preliminary introduction to the Reactive Architecture, and describe its components (i.e. Reactive Middleware, Shared Artefacts Repository System, System Engineering Toolbox, and Cloud Accountability System) and their relationships in Section 5.1. In Section 5.2, we classify the size of the Reactive Architecture using the "COSMIC func-

tional size measurement" approach. Section 5.3 presents a brief overview of the Architecture Trade-off Analysis Method (ATAM) as a precursor to our derived methodology. Then, the derived trade-off analysis methodology - cloud-ATAM - for the design and analysis of the small-to-medium size cloud-based Reactive Architecture is discussed. Finally, Section 5.4 concludes this chapter.

5.1 Designing the Reactive Architecture

We introduce a Reactive Architecture for system engineering in the cloud. This architecture supports various phases of system engineering processes. The main aim is to provide a dynamic and dependable framework that addresses the issues of complexities of system engineering in terms of its processes.

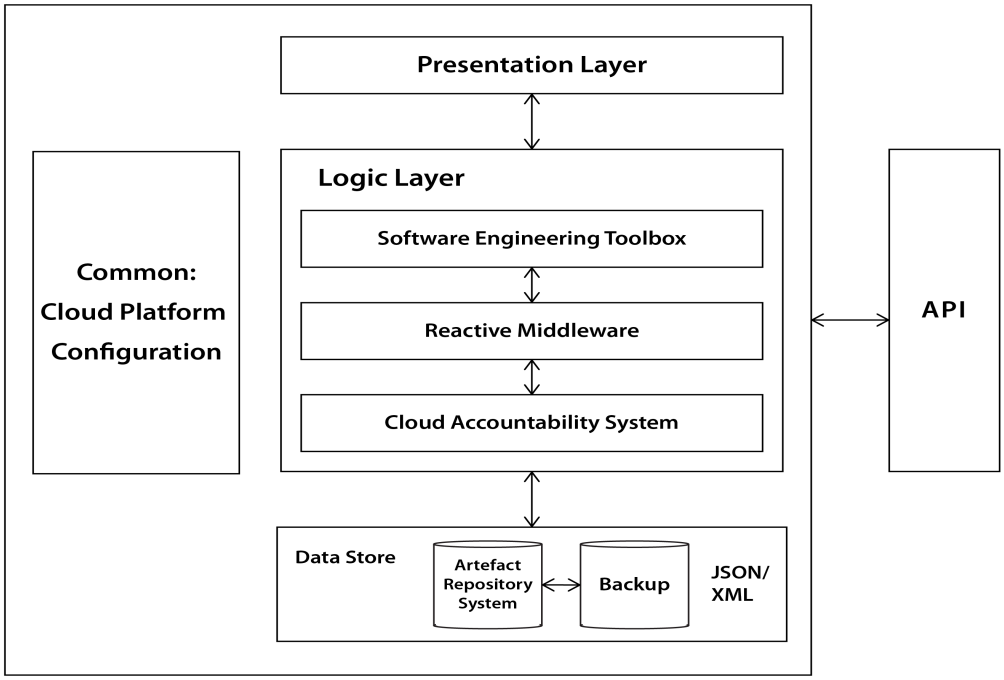


Figure 5.1: Layered View of Reactive Architecture

The main components of this architecture are an Architecture Interface, Reactive Middleware, Shared Artefacts Repository System, System Engineering

ToolBox, and Cloud Accountability System (see Figure 5.1 - Layered View).

For an effective collaboration of system engineers and other stakeholders, there should be a standardised medium or means of reliable information flow. Such information is often a product of the system engineering process. The system engineering process produces different types of artefacts, and changes made to these artefacts will have to be propagated to all relevant stakeholders. There is also a need to record all these changes as well as the system engineering tools and system developers that/who made them. Hence, change management and traceability of artefacts of the development process is crucial. It is therefore necessary to support a kind of reactive system engineering. Here, system developers will be able to react to such changes and ensure traceability and change management of all development artefacts.

To achieve this, we introduce a Reactive Middleware. Our definition of the Reactive Middleware is a system deployed to the cloud platform that provides services for change management, traceability, and stakeholder notification during cloud-based system engineering. The "reactivity" feature of the architecture is supported by this Reactive Middleware, which is largely aided by the Shared Artefacts Repository System. The Reactive Middleware is composed of a publish/subscribe mechanism and a change monitoring system. This middleware reacts to changes made to artefacts saved in the Shared Artefacts Repository. This reaction involves a notification sent to the corresponding system developers and relevant stakeholders to inform them of the changes made. This middleware introduces a form of dynamism and advancement to the system engineering process.

At a high and abstract level, the Reactive Architecture is designed based on the server-client relationship. Service Oriented Architecture (SOA) might be treated as a state of the art approach to the design and implementation of enterprise software, which is driven by business requirements. Within the last decade a number of concepts related to SOA have been developed, including Enterprise Service Bus (ESB), web services, design patterns, service orchestration and choreography and various security standards. Due to the fact that there

are many technologies that cover the area of SOA, the development and evaluation of SOA compliant architectures is especially interesting. The concept of cloud computing represents a combination of most of these concepts and hence, serves as a good SOA example. In view of this, the Reactive Architecture consists of a set of interconnected *web services*. We use web services as our service delivery framework since the cloud platform mainly supports such definition of services. Here, the architecture and its components are mainly servers. The *clients* (i.e. system developers and relevant stakeholders) access the interface of the Reactive Architecture using *plug-ins*, developed into integrated development environments (IDEs). In web services, clients' plug-ins and services are assumed to be loosely-coupled, which means that they are stand-alone systems independent of each other. The web service implementation details and internal structure are hidden from clients.

These web services are classified into architecture interface, Reactive Middleware, System Engineering Toolbox, Shared Artefacts Repository system, and the Cloud Accountability System. The classified web services function differently to achieve their specific goals. All of these web services can be used by clients (i.e. system developers) independently of the others. Here, if a client requires the services of a model checker tool such as ProB, all other tools in the System Engineering Toolbox, as well as all other web services can be disregarded. Also, some web services can interact with each other to create workflows. Communication among the web services is fundamentally based on the web standard Hypertext Transfer Protocol (HTTP), but more specifically on JSON-RPC. JSON-RPC is a remote procedure call protocol encoded in JSON. We use it as it is a very simple protocol, defining only a handful of data types and commands. Also, JSON-RPC allows for (1) "notifications" where data sent to the server does not require a response, which is suitable for workflows, and for (2) "multiple calls" to be sent to the server which may be answered out of order. We also make provisions for XML-RPC as an alternative communication protocol.

The overview of the proposed architecture supporting system engineering on the cloud (see Figure 5.1) is now described briefly in terms of interfaces, components,

connectors, constraints, and dependability.

5.1.1 Architecture Interface

Clients send requests to the Reactive Architecture. The requests are first received by the architecture interface. Here, requests are translated and directed appropriately to the intended component or constituent web service. This function of the architecture interface assists in regulating all requests made to the web services in the Reactive Architecture. In this situation, all requests that do not conform to the expected requests format are classified as malicious; they are dropped and noted.

5.1.2 Components

The main sub-systems of this architecture are briefly introduced as System and Component Interfaces, System Engineering Toolbox, Shared Artefacts Repository, Reactive Middleware, and Cloud Accountability System. System and Component Interfaces provide globally unique names of web services' handlers, based on internet domain names. This is generally specified as a unique uniform resource identifier (URI). Also, the interface determines the capacity and ordering of resource requests. A standard of the relationships between components are described in the component interface(s). This defines the type, means and scope of resource request transactions between components. The System Engineering Toolbox manages central updates of tools, defines the relationships between tools (i.e. establishing workflow of tools), manages system artefacts manipulation processes, and facilitates data presentation in different predefined (standard) formats. Also, the Shared Artefacts Repository manages search and retrieval of artefacts. This repository management supports the Reactive Middleware. These tools and repository management together ensure real-time monitoring of artefacts, change management, and traceability. The Toolbox brings together various tools that support the system engineering phases. It also provides access to various versions of tools, and facilitates communication

between tools. The Shared Artefacts Repository stores system engineering artefacts (source code, test cases, models, patterns, documentations, requirements, etc.), it employs an efficient indexing process (to facilitate prompt access to artefacts), and it asynchronously backs its contents up in a cloud-based repository. The components of the Reactive Architecture are further discussed in Section 5.3.3 and in the following chapters: Reactive Middleware and Shared Artefacts Repository in Chapter 4, and Cloud Accountability System in Chapter 6.

5.1.3 Connectors

A connector defines the type of relationships between the components of a system. In the Reactive Architecture, we consider a set of connectors that are appropriate to use at various facets of its design. Such connectors are:

(a) **Asynchronous Event Notification** (AEN) is a process or procedure that may be used by system component *targets* to notify a system component *initiator* of "events" that occur in the target. More specifically, we identify some components that will be suited to the *asynchronous event notification* connector:

- the Architecture Interface can use the *asynchronous event notification* to notify architecture components when a request is received from a client. Here, client requests are noted as events and the appropriate component(s) are notified accordingly.
- with the Reactive Middleware, the *asynchronous event notification* is able to identify *changes* made to artefacts as events in the Shared Artefacts Repository, and all appropriate component(s) such as the tools in the System Engineering Toolbox is/are notified.
- the coordination of tools in the System Engineering Toolbox for the purpose of creating *workflows* can benefit from *asynchronous event notification* connectors. Here, an event of an instruction defined by the coordination of tools by system developers initiate tools and in turn notify other tools in the workflow.

- the Cloud Accountability System can use the *asynchronous event notification* connectors to collect dependability metrics data from target virtual machines of the Reactive Architecture. In this situation, *timed* dependability metrics collection events from the target virtual machines notify the Cloud Accountability System after metrics are collected.

(b) **Synchronous Procedure Call** (SPC) is a protocol that allows the construction of client-server applications, using a demand/response protocol with management of transactions. The client is blocked until a response is returned from the server, or a user-defined optional time-out occurs. RPC guarantees at-most-once semantics for the delivery of the request. It also guarantees that the response received by a client is definitely that of the server and corresponds effectively to the request (and not to a former request to which the response might have been lost). SPC also allows a client to be unblocked (with an error result) if the server is unreachable or if the server (or virtual machine) has crashed before emitting a response. Finally, this protocol supports a mechanism called *abort propagation* that is, when a thread that is waiting for an reply is aborted, this event is propagated to the thread that is currently servicing the client request. We consider the *synchronous procedure call* connector between the Shared Artefacts Repository and the Back-Up Repository. Here, the extra guarantee of back-up request of artefacts from the Shared Artefacts Repository and the Back-Up Repository, over a dedicated channel is essential. This type of connector assures a high level of a reliable artefacts back-up process.

5.1.4 Constraints

Some general constraints identified in the Reactive Architecture and components design are classified as resources (time, budget, etc.), technology constraint, local standards (development, coding, etc.), public standards (HTTP, XML, XML Schema, RDFXML, WSDL, WADL, etc.), standard communication protocols, standard message formats, and skill profile of architecture developer.

Also, *project constraints* (e.g., time to market or deploy, customer demands, standards, cost, etc.) and *technical constraints* (e.g., COTS, interoperation with other systems, required hardware or software platform, reuse of legacy code, etc.).

5.1.5 Dependability

The Reactive Architecture is expected to be designed with dependability in focus. More precisely, meeting the dependability attributes' requirements of *availability*, *performance*, and *reliability* are important when designing the Reactive Architecture. The architecture's Cloud Accountability System has respective modules to monitor the dependability of all components in the Reactive Architecture. These modules collect dependability metrics from corresponding components to assess the dependability of the Reactive Architecture. Also, the Cloud Accountability System collects dependability metrics from the *Cloud Service Provider* (CSP), and compares them with those from the modules. CSPs such as the Amazon Web Service, make dependability metrics available to all their users' virtual machines. These metrics are mainly classified as availability and reliability metrics. The activities of the Cloud Accountability System is expected to assure the dependability of the Reactive Architecture running in the cloud, to system developers or other relevant stakeholders.

5.2 Classifying the Size of the Reactive Architecture

We complete this section by classifying the size of the Reactive Architecture. Here, we apply the COSMIC FSM approach (see Section 2.2.6) for classifying software project size. Also, we use the requirements (see Appendix C.13) and more specifically the use cases (see Figure 5.2 and Appendix C.14 for functional requirements) of the Reactive Architecture to identify *functional processes* for classification.

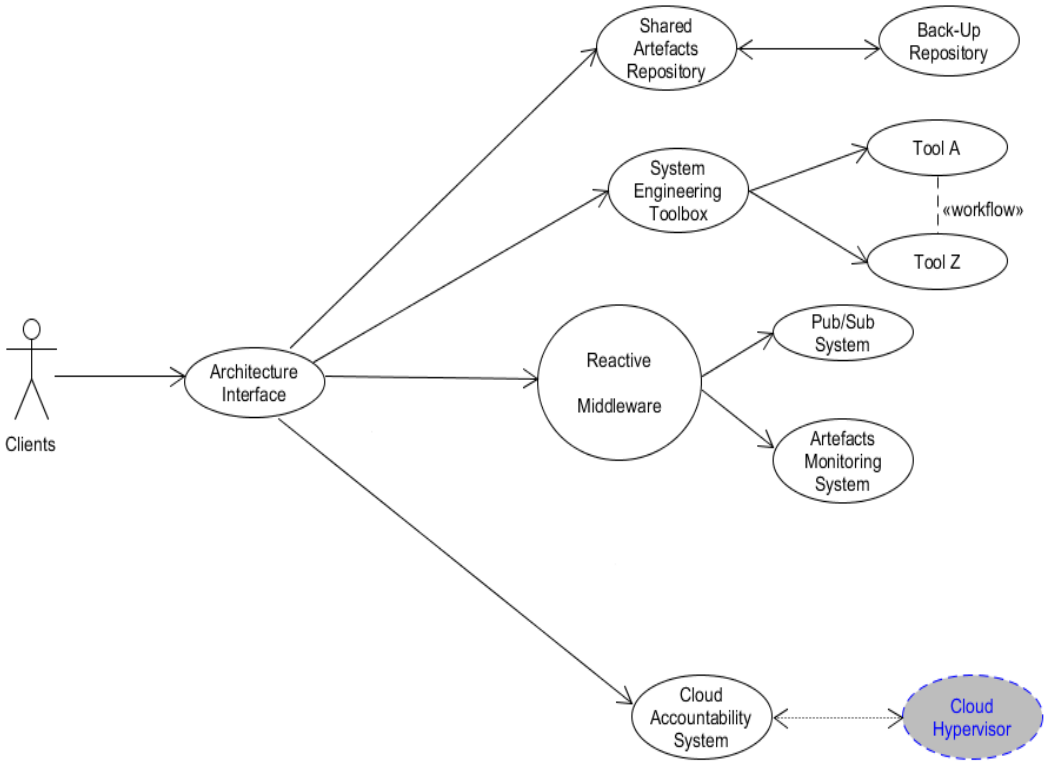


Figure 5.2: Overview of Reactive Architecture Use Cases

Table 5.1: COSMIC FSM Software Project Size Classification Benchmark

Size	**Bin (Full Functional Points)**
XXS	10
XS	30
S	100
M1	300
M2	1000
L	3000
XL	9000
XXL	18000
XXXL	More

Classifying the Functional Processes derived from the Use Cases of the Reactive Architecture			
Use Cases	**Use Case ID**	**Functional Processes**	**TOTAL**
User Management (see Figure 12)	UMUC	Register (fp1), UserID (fp2), SaveUserID (fp3), ackSave (fp4), JoinUserGroup (fp5), ackJoin (fp6), IsGroupAvailable (fp7), ackAvailability (fp8), CreateGroup (fp9), ackCreateGroup (fp10), AddUserToGroup (fp11), ackAdd (fp12), SetUserPrivilege (fp13), ackPrivilege (fp14), UpdateGroup (fp15), ackUpdate (fp16), Unregister (fp17), ackUnregister (fp18), UpdateGroupRemove (fp19), ackUpdateGroupRemove (fp20).	20
Adding Tools (see Figure 13)	ATUC	SelectDefaultTools (fp1), ackChoice (fp2), ParallelComposition (fp3), ackParallelComposition (fp4), SequentialComposition (fp5), ackSequentialComposition (fp6), SelectTool (fp7), ackToolChoice (fp8), ToolSelection (fp9), ackToolSelection (fp10), ChooseToolCoordination (fp11), ackToolCoordination (fp12), CallTools (fp13), ackResults (fp14), ToolAddition (fp15), ackToolAddition (fp16)	16
Saving Artefacts (see Figure 14)	SAUC	SelectArtefact (fp1), ackSave (fp2), ArtefactName (fp3), ackChoice (fp4), ArtefactHash (fp5), ackHash (fp6), ArtefactDate (fp7), ackDate (fp8), ArtefactOwner (fp9), ackOwner (fp10)	10
Download Artefacts (see Figure 15)	DAUC	DownloadArtefact (fp1), ackDownload (fp2), ArtefactName (fp3), ackChoice (fp4), ArtefactHash (fp5), ackHash (fp6), ArtefactDate (fp7), ackDate (fp8), ArtefactOwner (fp9), ackOwner (fp10)	10
Sharing Artefacts (see Figure 16)	ShAUC	ShareWith (fp1), ackAccess (fp2), SetAccessPrivilege (fp3), ackPrivilege (fp4), SetRead (fp5), ackRead (fp6), SetWrite (fp7), ackWrite (fp8), SetOwn (fp9), ackOwn (fp10), SetNoAccess (fp11), ackNoAccess (fp12)	12
Change Management (see Figure 17)	CMUC	ChangeArtefact (fp1), ackNotify (fp2), IsArtChanged (fp3), ackChange (fp4)	4
Traceability (see Figure 18)	TAUC	CreateArtefact (fp1), ackCreate (fp2), InformRepository (fp3), ackTrace (fp4), ChangeArtefact (fp5), ackChange (fp6), InformRepository (fp7), ackTrace (fp8), UseTools (fp9), ackToolsTrace (fp10), InformMiddleware (fp11), ackTrace (fp12), ackToolsChange (fp13)	13
Backup Repository (see Figure 19)	BRUC	AssynchronousBackUp (fp1), ackAssynBackUp (fp2), SynchronousBackUp (fp3), ackSynBackUp (fp4), Restore (fp5), ackRestore (fp6)	6
Evidence Collection	ECUC	ChooseSystemAttribute (fp1), ackAttributeChoice (fp2), TargetVM (fp3), ackTargetSet (fp4), HyperCalls (fp5), ackMetricsCalls (fp6), FormatMetrics (fp7), ackMetricsFormat (fp8), CallCloudAPI (fp9), ackAPICall (fp10), LogActivities (fp11), ackLog (fp12)	12
Arbitrate Metrics	ArMUC	SortMetrics (fp1), ackSort (fp2), ExamineMetrics (fp3), ackExam (fp4), LogActivities (fp5), ackLog (fp6)	6
Audit Metrics	AuMUC	ClassifyMetrics (fp1), ackClassify (fp2), ComputeMeans (fp3), ackCompute (fp4), CompareMeans (fp5), ackCompare (fp6), TriggerRM (fp7), ackTrigger (fp8), ComputeNPV (fp9), ackNPVCompute (fp10), DeriveMonetaryValue (fp11), ackMonetaryValue (fp12), LogActivities (fp13), ackLog (fp14)	14
Manage Notifications	MNUC	SendNotification (fp1), ackNotification (fp2), ClaimCompensation (fp3), ackClaim (fp4), SendTimedReport (fp5), ackTimedReport (fp6), LogActivities (fp7), ackLog (fp8)	8
		TOTAL	**131**

Table 5.2: Classification of Functional Processes based on the Reactive Architecture Use Cases

In classifying the Reactive Architecture, we apply the COSMIC FSM approach for categorising software project size. We reference the categorisation benchmark of software project size with COSMIC FSM in Table 5.1, for classifying the functional processes derived from the use cases of the Reactive Architecture (shown in Table 5.2). Here, *Bin* is the COSMIC full function points, and it is considered to be a "generic set of size category". The total number of functional processes identified from the Reactive Architecture through its use cases are *131*. At this point, we can reliably identify the Reactive Architecture as a "level 1 medium size software project/architecture" as it falls in the **M1** range of more than 100 functional processes but less than or equal to 300 functional processes.

5.3 Analysing the Reactive Architecture

5.3.1 Present cloud-ATAM

One of our main objectives of this chapter is to present a methodology, cloud-ATAM for analysing the trade-off between multiple quality attributes of the small-to-medium scale Reactive Architecture deployed to the cloud environment.

The reasoning in cloud-ATAM is not always highly formal and mathematical, but it is predictive and repeatable. The reasoning might manifest itself as a discussion that follows from the exploration of the architectural approaches that address a scenario; it may be a qualitative model of attribute-specific behaviour of the architecture; or it may be a quantitative model that represents how to calculate a value of a particular quality attribute. Attribute-Based Architectural Styles (ABASs) and quality attribute characterisations provide the technical foundations for creating these reasoning models.

As introduced earlier, the ATAM is considered a mature and validated scenario-based Software Architecture (SA) evaluation method. Similarly, the inputs of cloud-ATAM are scenario elicited by stakeholders and documented descriptions

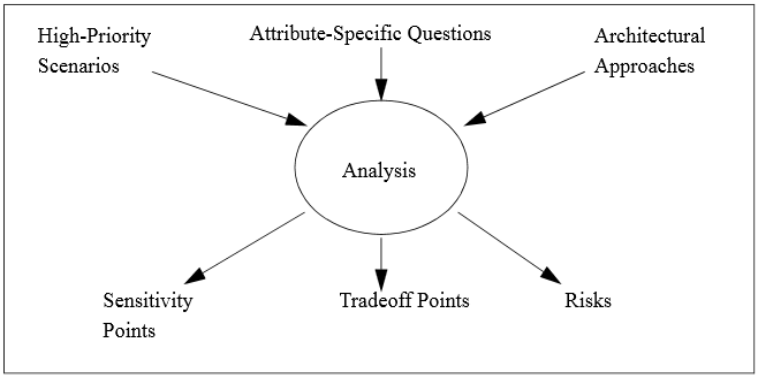


Figure 5.3: *cloud-ATAM* Concept Interactions

of the architecture. The goal of the cloud-ATAM is to analyse architectural approaches with respect to scenarios generated from project drivers for the purpose of identifying "risk points" in the architecture. This is achieved by a disciplined reasoning about SA relating to multiple quality attributes. There are two important classifications of risk points in cloud-ATAM namely "sensitivity points" and "trade-off points". A "sensitivity point" refers to a parameter of the architecture that affects the achievement of one quality attribute. On the other hand, a "trade-off point" refers to a parameter of the architecture that affects the achievement of more than one quality attribute. In this situation, one quality attribute improves and the other degrades. These risk points, together with extensive documentations of the architecture, scenarios, and quality-attributes analyses are the products of cloud-ATAM. The cloud-ATAM also explicitly relates architectural risks and trade-offs to project drivers (i.e. architectural requirements) as shown in Figure 5.3.

1. Present the cloud-ATAM
    - Method
    - Process
    - Agenda
2. Present the Project Drivers
    - Critical requirements

- Technical constraints
- Quality attributes

3. Present the Architecture
   - Architecture drivers
   - High-level architecture views
   - Architecture styles
   - Important scenarios
4. Identify Architectural Approaches
   - Architecture decisions
5. Generate the Quality Attribute Utility Tree and Scenarios
   - Quality tree
   - Scenarios
6. *Analyse the Architectural Approaches*
   - Decisions
   - Risks
   - Trade-offs (Project drivers, Trade-off themes, and Impact of trade-off themes on project drivers)
   - Sensitivity points
7. *Present Results*

The seven steps of cloud-ATAM are provided above. This chapter discusses the design of the the Reactive Architecture which is based on Steps 1 to 5. On the other hand, Steps 6 and 7 are used to analyse or evaluate a designed small-to-medium size cloud-based architecture.

5.3.2 Present the Project Drivers

The predominant drivers of this project are the target quality attributes: *availability* and *performance*. The expectation is that the Reactive Architecture

Reactive Architecture Quality Attributes of Interest		
Quality Attribute Goals	**ID**	**Attribute-Specific Requirements**
Operability	O1	The Reactive Architecture must store all artefacts created in all of its components.
	O2*	It must monitor and trace all changes to these artefacts to inform system developers (i.e. clients). (also P1)
	O3	The System Engineering Toolbox must facilitate sequential and parallel execution of tools in a workflow manner.
	O4	The Cloud Accountability System must gather dependability metrics from several virtual machines, and perform a synchronous analysis of these metrics.
Performance	P1*	It must monitor and trace all changes to these artefacts to inform system developers (i.e. clients). (also O2)
	P2*	The Reactive Middleware must enable heterogeneous access and analysis operations on artefacts in the Shared Artefacts Repository. (also A2)
	P3	Security mechanisms must not degrade defined performance threshold. Specifically, response time for create, delete, update, and display artefact operations should not exceed 5 seconds at peak cloud period and less than 1 second during off-peak period. (also S1)
	P4	The Reactive Architecture must provide high performance and availability to allow it to keep up with the sturdy stream of data and operations on artefacts from the system engineering processes. (also A4)
Scalability	Sc1	The Reactive Architecture must support at least 20 users concurrently.
	Sc2*	The Reactive Architecture must provide capacity to scale quickly to accommodate changing demands of system developers, and failures. (also A1)
Availability	A1*	The Reactive Architecture must provide capacity to scale quickly to accommodate changing demands of system developers, and failures. (also Sc2)
	A2*	The Reactive Middleware must enable heterogeneous access and analysis operations on artefacts in the Shared Artefacts Repository. (also P2)
	A3	Critical systems such as the Reactive Middleware must not constitute a single point of failure which will affect the uptime of the system and the Reactive Architecture. (also R1)
	A4	The Reactive Architecture must provide high performance and availability to allow it to keep up with the sturdy stream of data and operations on artefacts from the system engineering processes. (also P4)
Maintainability	M1	The Shared Artefacts Repository must be backed up asynchronously to facilitate roll-back of repository artefacts.
Reliability	R1	Critical systems such as the Reactive Middleware must not constitute a single point of failure which will affect the uptime of the system and the architecture. (also A3)
Security	S1	Security mechanisms must not degrade defined performance threshold. Specifically, response time for create, delete, update, and display artefact operations should not exceed 5 seconds at peak cloud period and less than 1 second during off-peak period. (also P3)

Table 5.3: Mapping Requirements to Quality Attributes of Reactive Architecture

will maintain a high level and healthy balance between availability and performance. The relationship between the quality attributes are directly dependent on the requirements provided by the stakeholders of the Reactive Architecture. Also, the constraints of the architecture are contributory drivers of the project. Finally, the major stakeholders of the project are presented.

5.3.2.1 Requirements

We present the requirements of the Reactive Architecture in Appendix C.13. However, these requirements are formulated in terms of the architectural components in Table 5.3. Also, we map the requirements of the Reactive Architecture to identified quality attributes; operability, performance, scalability, availability, maintainability, reliability, and security.

5.3.2.2 Constraints

The constraints affecting the Reactive Architecture are identified along two perspectives:

(a) Reactive Architecture:
- Cost of development.
- Time to market, which is dictated by customer demands.
- Skill set of architect.
- Technical standards.

(b) Environment of Architecture (i.e. Cloud Environment):
- Co-location of potentially risky systems on a cloud server.
- Architecture is highly dependent on the dependability of the cloud infrastructure.
- Security of the architecture and data is largely out of the control of the architect.
- Reliance on the performance of Commercial Off-The-Shelf (COTS) products on the cloud platform.

5.3.2.3 Project Stakeholders

The main stakeholders of the Reactive Architecture are briefly introduced below as users of the architecture, and with regards to the *roles* they play in GSD projects:

- **Systems Architect**: They define the architecture of a computerised system (i.e. a system composed of software and hardware) in order to fulfill certain *requirements*. Such definitions include a breakdown of the system into components, the component interactions and interfaces (with the environment, and the user), and the technologies and resources to be used in the design. The Systems Architect's work avoids implementation issues and readily permit unanticipated extensions or modifications in future stages. Due to the extensive experience required for this, the Systems Architect is typically a very senior technician with substantial, but general, knowledge of hardware, software, and similar systems. But above all, the systems architect must be reasonably familiar with the users' domain of experience.
- **Project Managers**: A project manager is the person who has the overall responsibility for the successful initiation, planning, design, execution, monitoring, controlling and closure of a project. The role of the project manager encompasses many activities including: planning and defining scope, activity planning and sequencing, resource planning, developing schedules, time estimating, cost estimating, developing a budget, documentation, creating charts and schedules, risk analysis, managing risks and issues, monitoring and reporting progress, team leadership, strategic influencing, business partnering, working with vendors, scalability, interoperability and portability analysis, controlling quality, and benefits realisation.
- **Cloud Service Provider**: This is the entity that provides the cloud service. The Cloud Service Provider (CSP) owns and controls the cloud computing platform. The services include SaaS (Software as a Service),

PaaS (Platform as a Service), and IaaS (Infrastructure as a Service). Based on the services provided, the CSPs can be broadly categorised into 3 types: Application Provider, Resource Provider, and Infrastructure Provider. The Application Provider directly provides access to an application without the need to know about the resources or layers underneath. Also, the Resource Provider provides virtualisation systems on top of their servers and lets clients buy resources such as RAM, computing cycles and disc space. Then the Infrastructure Provider leases servers and associated infrastructure from their datacenters. The infrastructure includes servers, storage, bandwidth and the datacenter (with power, space and personnel to man them).

- **System Developers**: The System Developer interprets business requirements and translates them into a deployable solution that meets functional and non-functional needs. Their responsibilities are to (1) work with Testers to iteratively develop: the deployable solution, models required for the properly controlled development of the solution, and models and documentation required for the purpose of supporting the solution in live use; and (2) recording/interpreting the detail of any: changes to the detailed requirements, changes to the interpretation of requirements which result in re-work within the solution, information likely to impact on the ongoing evolution of the solution, adhering to technical constraints laid out in the System Architecture Definition, adhering to standards and best practice laid out in the Technical Implementation Standards, participating in any quality assurance work required to ensure the delivered products are truly fit for purpose, and testing the output of their own work prior to independent testing.
- **System Testers**: The System Tester is fully integrated with the System Development Team and performs testing in accordance with the Technical Testing Strategy throughout the project. Their responsibilities are to work with other roles to define test scenarios and test cases for the evolving system. Also, in accordance with the Technical Testing Strategy, the System Tester: carries out all types of technical testing of the solution as a whole,

creates testing products, e.g. test cases, plans and logs, reports the results of testing activities to the Technical Co-ordinator for Quality Assurance purposes, and keeps the Team Leader informed of the results of testing activities.

- **Database Administrators**: A Database Administrator (DBA) is responsible for the performance, integrity and security of a database. They are also involved in the planning and development of the database, as well as troubleshooting any issues on behalf of the users. The DBA role naturally divides into three major activities: ongoing maintenance of production databases (operations DBA); planning, design, and development of new database applications, or major changes to existing applications (development DBA, or architect); and management of an organisation's data and metadata (data administrator). One person may perform all three roles, but each is profoundly different.
- **Tool Developers**: A Tool Developer builds software tools that are used to create, debug, maintain, or otherwise support other programs and applications. Such tools can be combined together to accomplish a software development task in a project.

5.3.3 Present the Architecture

The Reactive Architecture provides support for system engineering to system developers. The architecture is deployed to the cloud to make use of its benefits. Cloud computing provides relatively cheap resources, multi-user access, global access, scalability, etc.

We present two main views of the Reactive Architecture: layered view depicted in Figure 5.1, and component-connector view also depicted in Figure 5.4. A detailed description of the Reactive Architecture has been provided earlier in Section 5.1. We then briefly introduce the components of the Reactive Architecture below.

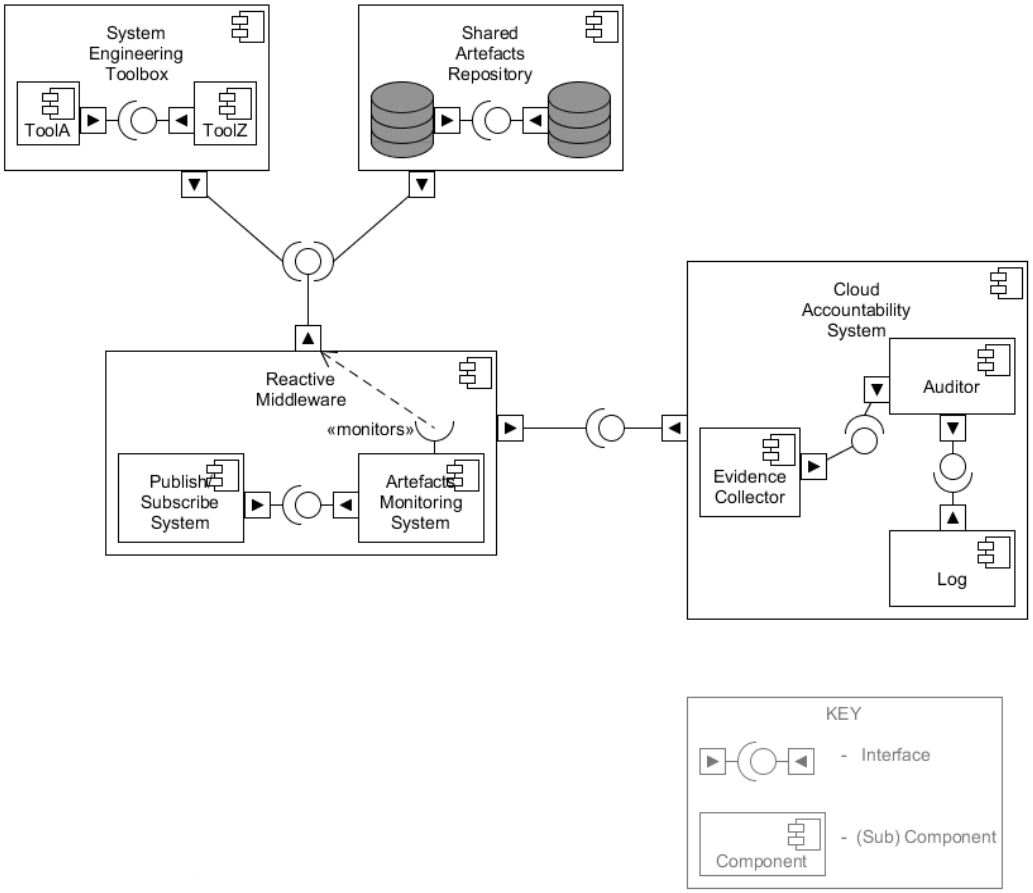


Figure 5.4: Component and Connector View of the Reactive Architecture

5.3.3.1 Reactive Middleware

The *reactivity* feature of the architecture is based on this Reactive Middleware which is largely supported by the Shared Artefacts Repository System. This middleware reacts to changes made to artefacts saved in the Shared Artefacts Repository. This reaction involves a notification sent to the corresponding stakeholders (i.e. system developers, managers) to inform them of the changes made. This Reactive Middleware introduces a form of dynamism and advancement to the system engineering process.

The Reactive Middleware is made up of two sub-systems: Artefacts Monitoring

System (AMS) and Publish/Subscribe System (PSS). The PSS registers *actors* (i.e. tools, and developers) to development artefacts that they are interested in. This registration assigns permissions to only relevant stakeholders or actors. Based on this registration, notifications are generated as a result of changes to the artefacts and are sent to corresponding stakeholders. Also, the AMS provides an industry standard framework for system engineering tool integration that in particular supports requirements, change management and traceability. This AMS traces all changes made to artefacts or elements of artefacts from integrated development environments, and links them to the stored artefacts in the repository. The AMS then triggers a call to the PSS to notify all stakeholders. The Reactive Middleware is further discussed in Chapter 4.

5.3.3.2 Shared Artefacts Repository System

This system is made up of the Shared Artefacts repository and the Back-up repository. Here, all requests from the Architecture Interface is received by the Shared Artefacts Repository. The Shared Artefacts Repository facilitates a recovery process using the Back-up Repository. The Reactive Middleware is used to collect and interlink system engineering artefacts to support traceability and change management. By recording the history of all system development and storing all development artefacts in the dedicated Shared Artefacts Repository, the Reactive Architecture is able to support *reactive* system engineering processes. In this situation, a change in any development artefact is propagated to all relevant stakeholders (i.e. developers), tools, and also to dependent artefacts. The Shared Artefacts Repository is a critical element of this architecture. It is important to mention that the Shared Artefacts Repository also links the artefacts with the tools and system engineering phases in which they are either produced or used. Here, artefacts that specify the origin of artefacts (i.e. tool source), format, etc. are also stored. Also, a timed back-up of all the artefacts in the Shared Artefacts Repository into the Back-up Repository is undertaken. The Shared Artefacts Repository is further discussed in Chapter 4.

5.3.3.3 System Engineering Toolbox

The System Engineering Toolbox is made up of tools for software projects. All client tool-related requests to the Architecture Interface are directed to the Toolbox. The tools in the Toolbox are classified based on the system engineering phase (i.e. Specification, Requirements, Design, etc.) they support. These tools are presented to be interoperable, hence can be coordinated to support system engineering. The main artefacts generated in the toolbox system are those from the tools: processing results, processing time, tool version, etc. These tools are composed into either parallel coordination or sequential coordination scenarios. Such compositions can be applied to a set of tools that supports a particular system engineering phase. The Toolbox also supports tools management; tool patches and updates are installed. Also, new and older versions of tools are made available to developers. The System Engineering Toolbox is further discussed in Chapter 4.

5.3.3.4 Cloud Accountability System

This system consists of components that collect dependability-related data from the components of the Reactive Architecture, to assure system developers of the dependability of the cloud-based architecture. Here, availability and reliability metric data are collected from the Toolbox system, Reactive Middleware and the Shared Artefacts Repository system. The data collected is passed on for examination, analyses and then compared to the CSP's contract (i.e. Service Level Agreement - SLA). Notifications are sent to the system developer(s), the CSP and other stakeholders, when there is a breach to the SLA. Relevant information is provided to the system developer towards the assurance of the dependability of the Reactive Architecture. We discuss the Cloud Accountability System further in Chapter 6.

5.3.4 Identify Architectural Approaches

Our research focuses on the *availability* and *performance* quality attributes of the Reactive Architecture, so we will then pay a closer attention to, and present the architectural requirements related to them. We then characterise these quality attributes, and ask *attribute-specific questions* to have a clearer appreciation of their interaction in the context of the Reactive Architecture. Finally, we identify *architectural approaches* to facilitate the achievement of the architectural requirements.

- **Availability:**
  (a) The Reactive Architecture must provide capacity to scale quickly to accommodate changing demands of system developers, and failures.
  (b) The Reactive Middleware must enable heterogeneous access and analysis operations on artefacts in the Shared Artefacts Repository.
  (c) Critical systems such as the Reactive Middleware must not constitute a single point of failure which will affect the uptime of the system and the Reactive Architecture.
  (d) The Reactive Architecture must provide high performance and availability to allow it to keep up with the sturdy stream of data and operations on artefacts from the system engineering processes.
- **Performance:**
  (a) It must monitor and trace all changes to these artefacts to inform system developers.
  (b) The Reactive Middleware must enable heterogeneous access and analysis operations on artefacts in the Shared Artefacts Repository.
  (c) Security mechanisms must not degrade defined performance threshold. Specifically, response time for create, delete, update, and display artefact operations should not exceed 5 seconds at peak cloud period and less than 1 second during off-peak period.

(d) The Reactive Architecture must provide high performance and availability to allow it to keep up with the sturdy stream of data and operations on artefacts from the system engineering processes.

5.3.4.1 Quality Attribute Characterisations

Evaluating an architectural design against quality attribute requirements requires a precise characterisation of the quality attributes of concern. From the knowledge that is already in the various quality attribute communities, we apply an already created characterisations for the quality attributes of performance and availability. These characterisations have been discussed in Section 2.2.3.

5.3.4.2 Attribute-Specific Questions

Some attribute-specific questions are asked to help narrow the design of the Reactive Architecture, and clarify the expectations of the architecture. Through these questions and the understanding of the attribute characterisations, we aim to improve the architectural documentation. The questions asked are:

- What facilities exist in the software architecture (if any) for self-testing and monitoring of software components? (Availability)
- What facilities exist in the software architecture (if any) for redundancy, liveness monitoring, and fail-over? (Availability)
- How is data consistency maintained so that one component can take over from another and be sure that it is in a consistent state with the failed component? (Availability)
- What is the process and/or task view of the system, including mapping of these processes/tasks to hardware and communication mechanisms between them? (Performance)
- What functional dependencies exist among the software components? (Performance)

- What data is kept in the database? How big is it, how much does it change, who reads/writes it? (Performance)
- How are resources allocated to service requests? (Performance)
- What are the anticipated frequency and volume of data transmitted among the system components? (Performance)

5.3.4.3 Architectural Approaches

We identify some architectural approaches to meet the requirements of the Reactive Architecture, and are enumerated below:

(a) We use the *component-and-connector architectural style* to represent the various components and connections/interfaces of the Reactive Architecture. This is particularly relevant because it expresses the runtime behaviour of the architecture under review.

(b) We avoid the *distributed data repository* approach in designing the Shared Artefacts Repository. This prevents situations such as issues with database consistency and possible modifiability concerns.

(c) The *client-server* approach is a best fit for the data-centric Shared Artefacts Repository system.

(d) The Reactive Middleware will be adequately represented using the *client-server* approach.

(e) Since the Reactive Middleware and the Shared Artefacts Repository constitute a single point of failure, we present the following approaches:

   i. *Backup* of artefacts in the Shared Artefacts Repository. (towards the architecture's *availability* requirements)

   ii. *Distributed services* (or modular set of services) for the components of the Reactive Middleware. (towards the architecture's availability requirements)

Identified Architectural Styles in Reactive Architecture		
Architectural Styles	**ID**	**Requirements**
Data-Centered	DC1	The Reactive Architecture must store all artefacts created in all the components of the main composing systems.
	DC2	The Reactive Middleware must enable heterogeneous access and analysis operations on artefacts in the Shared Artefacts Repository.
Client-Server	CS1	It must monitor and trace all changes to these artefacts to inform system developers (i.e. clients).
	CS2	The System Engineering Toolbox must facilitate sequential and parallel execution of tools in a workflow manner.
	CS3	The Cloud Accountability System must gather dependability metrics from several virtual machines, and perform a synchronous analysis of these metrics.
Event-Driven	ED1	The Shared Artefacts Repository must be backed up asynchronously to facilitate rollback of repository artefacts.
Peer-To-Peer	P2P1	The Cloud Accountability System must gather dependability metrics from several virtual machines, and perform a synchronous analysis of these metrics.

Table 5.4: Mapping Architectural Styles to Requirements of Reactive Architecture

(f) Schema-free NoSQL data management system (DMS) is necessary for the Shared Artefacts Repository to minimise or remove bottlenecks. (towards the architecture's performance requirements)

(g) An independent communication components approach for communication between the Reactive Middleware together with the Shared Artefacts Repository, and the Cloud Accountability System. Such communication approach is particularly relevant for the distributed components of the Cloud

Accountability System.

Each of these approaches was probed for risks, sensitivities, and trade-offs via our attribute-specific questions. Also, we have presented some architectural styles that correspond to some of the Reactive Architecture requirements being considered (see Table 5.4).

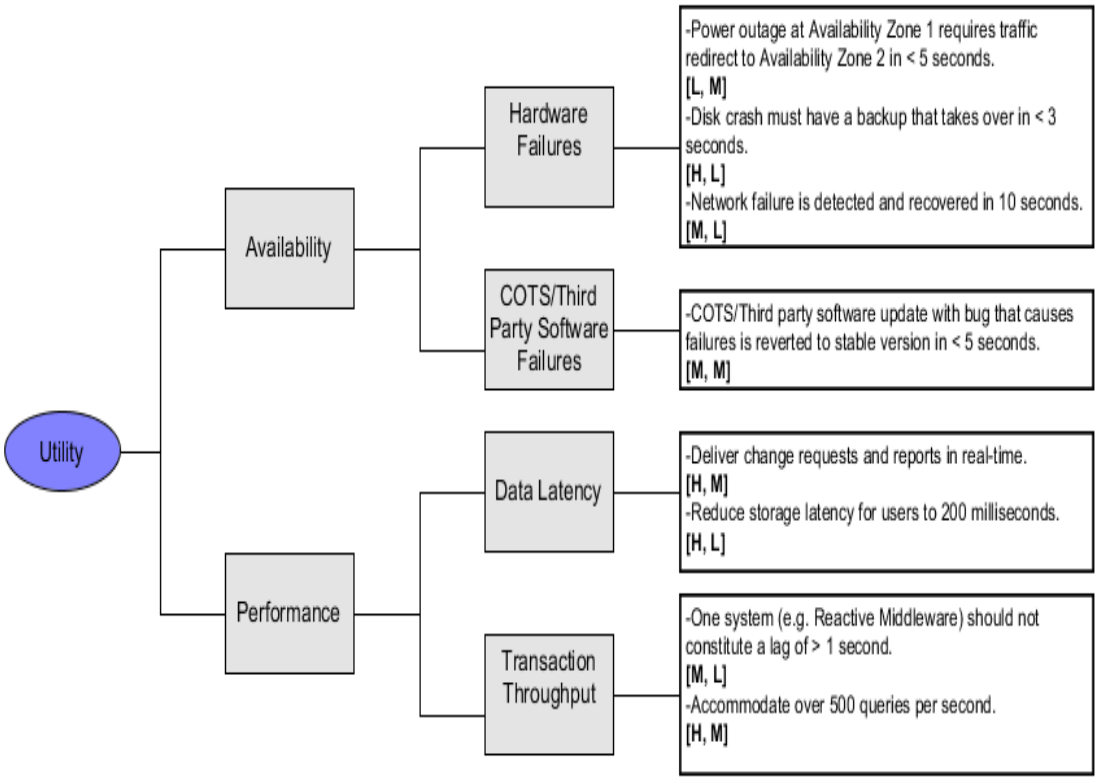


Figure 5.5: Utility Tree with Prioritised Scenarios

5.3.5 Generate the Quality Attribute Utility Tree and Scenarios

5.3.5.1 Eliciting and Prioritising Scenarios

Scenarios are elicited and prioritised in cloud-ATAM using different mechanisms at different times with different stakeholder participation. The two mechanisms used here are utility trees and structured brainstorming. We apply the utility tree mechanism to elicit and prioritise scenarios at this point. We organise a scenario brainstorming with the stakeholder community to reconcile the utility tree. This scenario brainstorming mechanism used, and the reconciliation process are discussed in Chapter 7. The description of our utility tree scenario elicitation approach is provided below.

5.3.5.2 Utility Tree

The utility tree generated in this exercise is shown in Figure 5.5. This presented utility tree guides the remaining analysis process. It is important at this point to prioritise, and refine the Reactive Architecture's most important quality attribute goals. The utility tree starts with "Utility" as the root node. This indicates the general "goodness" of the Reactive Architecture. The **second level** is constituted with the quality attributes of interest: performance and availability. In the **third level**, there are specific quality attribute refinements. From the performance quality attribute, we identify "data latency" and "transaction throughput" as relevant refinements. Such refinements are major determinants of performance. Also, availability is refined to "hardware failures" and "software failures". From this point, we are able to identify attribute goals as "quality attribute scenarios" that are concrete enough for prioritisation and analysis. These quality attribute scenarios form the "leafs" of the utility tree. Here, the "hardware failures" (i.e. see **third level**) refined from availability is further refined into "power outage at Availability Zone 1 requires traffic redirect to Availability Zone 2 in less than 5 seconds", "disk crash must have a backup that takes over in less than 3 seconds", and "network failure is detected and

recovered in 10 seconds". These specific scenarios are prioritised relative to each other and then analysed in Chapter 7.

Table 5.5: Classified Quality Attribute Scenarios according to Types

Scenario Type	**Scenario Type ID**	**Quality Attribute Scenarios**	**Scenario ID**
Use Case	*USC1*	Deliver change requests and reports in real-time.	P1
	USC2	Reduce storage latency for users to 200 milliseconds.	P2
Growth	*GS1*	Disk crash must have a backup that takes over in less than 3 seconds.	A2
	GS2	Network failure is detected and recovered in 10 seconds.	A3
	GS3	COTS/Third party software update with bug that causes failures is reverted to stable version in less than 5 seconds.	A4
	GS4	One system (e.g. Reactive Middleware) should not constitute a lag greater than 1 second.	P3
	GS5	Accommodate over 500 queries per second.	P4
Exploratory	*ES1*	Power outage at Availability Zone 1 requires traffic redirect to Availability Zone 2 in less than 5 seconds.	A1

In cloud-ATAM, we use three types of scenarios: (1) use case scenarios; (2) growth scenarios; and (3) exploratory scenarios. Use case scenarios involve typical uses of the existing system, and are used for information elicitation. Also, growth scenarios cover anticipated changes to the system. Then, exploratory scenarios cover extreme changes that are expected to "stress" the system. These different types of scenarios are used to probe a system from different angles, to further optimise the chances of identifying architectural decisions that are at

risk. In this work, we consider some scenarios bordering use case, growth and exploration. We have provided some use cases derived from our scenarios in Appendix C.14. The scenarios are classified and shown in Table 5.5.

5.3.6 Analyse the Architectural Approaches

As mentioned earlier in Section 5.3, the Steps 6 and 7 of the cloud-ATAM involves the analysis/evaluation and results presentation of a small-to-medium size cloud-based architecture. For the analysis process, we adopt a two-staged approach to qualitatively and quantitatively analysing the design of the Reactive Architecture: Utility Tree analysis, Scenario Brainstorming (stakeholder group work) constitute qualitative analysis approach, and the Attribute-Based Architectural Style (i.e. reliability tri-modular redundancy) facilitate the quantitative reasoning of the quality attributes of the Reactive Architecture.

(a) Utility Tree Analysis:

This mechanism involves the use of the discussed utility tree (see Figure 5.5) for the analysis of quality attribute scenarios to identify sensitivities and trade-offs.

(b) Scenario Brainstorming (Stakeholders' Group Work):

The scenario brainstorming mechanism is *stakeholder-centric*, which elicits points of view from a more diverse and larger group of stakeholders, and verifies, and then builds on the results of the first phase. It involves the evaluation of the *cloud-ATAM*, and this is undertaken through an organised group work of stakeholders who analyse the trade-off themes on the project drivers.

The two qualitative reasoning approaches are conducted and discussed in detail in Chapter 7.

5.3.7 Present Results

The presented set of Reactive Architecture requirements related to the availability and performance quality attributes (see Section 5.3.2) have all been met by first considering relevant "quality attribute-specific questions" in Section 5.3.4, which then led to the identification of appropriate architectural styles or approaches in Section 5.3.4.3.

Based on the information collected during the design of the Reactive Architecture with the cloud-ATAM, the cloud-ATAM team of architects presents the findings to the stakeholders and writes a report detailing this information along with any proposed mitigation strategies. As mentioned earlier, this section is presented in detail in Chapter 7.

5.4 Summary

In this chapter, we contribute a Reactive Architecture for cloud-based system engineering (see Section 1.3). The cloud-ATAM is introduced for the design, analysis and evaluation of small-to-medium size cloud-based architectures (see *Objective 5* of Section 1.2.2.

The cloud-ATAM presents a derived seven-step methodology from an established architecture evaluation methodology known as Architecture Trade-off Analysis Method (ATAM). The Reactive Architecture is designed using the cloud-ATAM (see *Objective 7* of Section 1.2.2, and also analysed using a focal pair of quality attributes (i.e. availability and performance) and the relevant identified quality attribute scenarios of the Reactive Architecture.

Further analysis of the architecture is discussed in Chapter 7. Our study can help system engineers to design and analyse small-to-medium size cloud-based software architectures with quality attributes (such as availability, performance) trade-off being the main focal point. This approach is very practical and dynamic as the deployment environment (i.e. cloud platform) is unpredictable and changes often, hence affecting the quality attributes of systems.

Chapter 6

Cloud Accountability

This chapter contributes a novel methodology for assuring cloud dependability, and the Cloud Accountability System (CAS) which is a component of the Reactive Architecture. The Reactive Architecture is deployed to the cloud platform. The CAS facilitates the Cloud Accountability Method (CAM) that enables the provision of assurance for dependability (i.e. availability, reliability) of the cloud-based Reactive Architecture for system engineering (refer to *Objectives 9* and *10* in Section 1.2). Our work discusses this methodology which is guided by a well-established digital forensic model to assure and inform system engineers of cloud dependability during the use of cloud resources. This digital forensic model shows how digital forensics can support incident handling.

In this chapter, Section 6.1 introduces our research, and the Cloud Accountability System is introduced in Section 6.2. The Cloud Accountability Methodology that facilitates the assurance of dependable cloud environments is presented in Section 6.3. Finally, the conclusions of this research are drawn and presented in Section 6.4.

6.1 Introduction

Cloud computing promises to be an effective business model for staging varying sizes of enterprise systems to provide services to intended users. It is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g. servers, storage, etc.) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

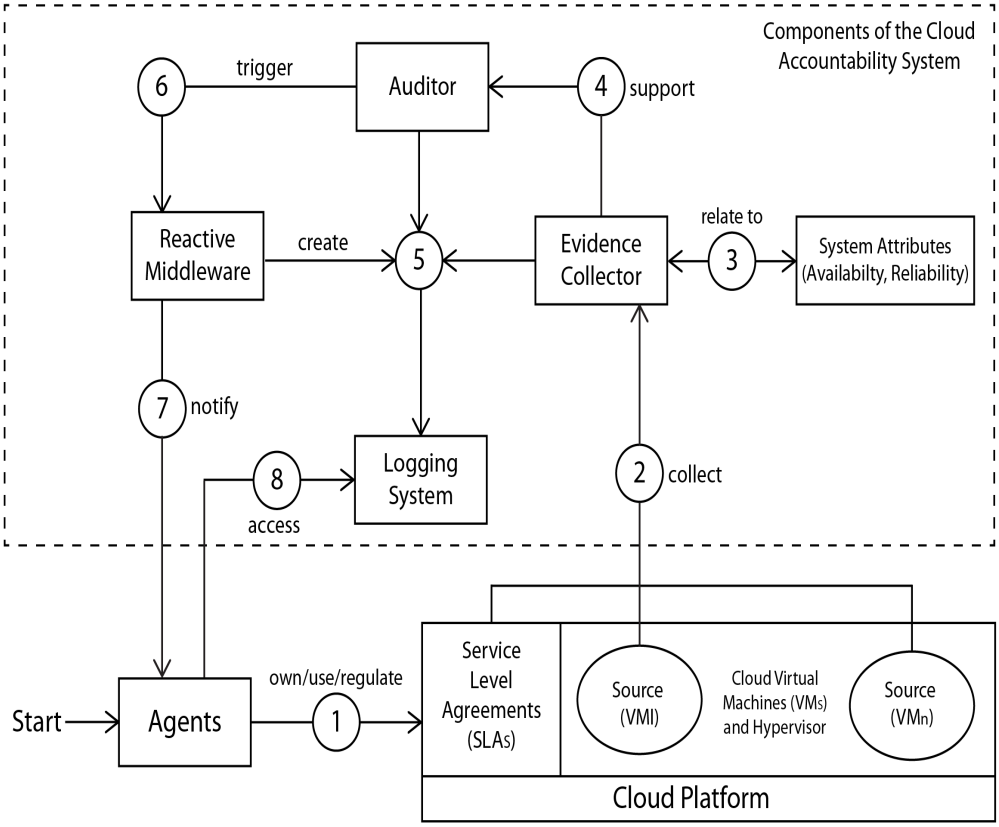


Figure 6.1: Conceptual Model of the Cloud Accountability System

From our literature review (see Section 2.3.4.3), we identify that there is no work that employs forensic auditing techniques to check the CSPs' SLA towards the assurance of dependability especially for availability and reliability. It is relevant that cloud users such as system engineers are assured of the avail-

ability and reliability of the cloud platform they use for GSD. In this work, we collect data from the CSPs and also from the cloud server's hypervisor. The collection of this pair of data until it is processed into evidence, is guided by a digital forensic model. This model is standardised by NIST SP800-86 to support incident handling.

6.2 Cloud Accountability System

The Cloud Accountability System is constituted by four main components (Figure 6.1). These components are the Evidence Collector, Auditor, Reactive Middleware, and a Logging System. The functions of these components are briefly introduced.

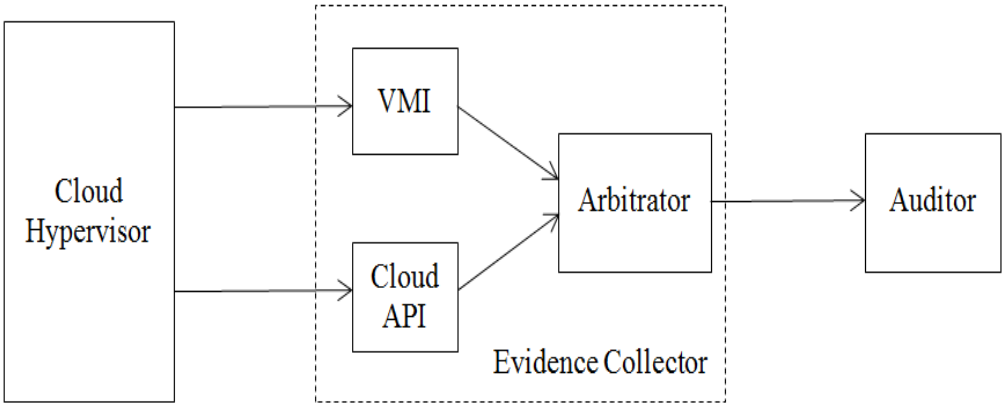


Figure 6.2: Evidence Collector

(a) **Evidence Collector (EC)**: This system collects data for the availability and reliability metrics from the required set of the RA's VM hypervisor using two main approaches; Virtual Machine Introspection (VMI) technique and the Cloud Management System (CMS). Refer to steps *2* and *3* of Figure 6.1. The VMI uses a VMI library in C language, and Python scripts to collect these metrics data. Also, the CMS uses Amazon Web Services (AWS) API known as the AwsSdkMetrics. Figure 6.2 provides

an overview of the Evidence Collector system. The Arbitrator sub-system reconciles the data from the two sources to support data integrity. All relevant activities in steps *2* and *3* are saved as logs (see step *5* in Figure 6.1).

(b) **Auditor**: The Auditor conducts an evidence-based trust analysis. Here, a simplistic yet relatable and effective weighted trust value approach is used. All systems are initialised to zero, the absence of violations gets one, however the presence of it gets negative one. Generally, the Auditor works by collating the set of metrics data relating to availability and reliability from the Evidence Collector (see step *4* in Figure 6.1). Then the availability and reliability are computed for each system being monitored. At this point, the availability and reliability values from each monitored system are obtained from the CSPs API, and then compared with those from the VMI. Violations from the two data sources are checked in the context of the stipulated SLA. A call is triggered to the Reactive Middleware if there are violations (see step *6* in Figure 6.1). Also, the Auditor computes the Net Present Value (NPV) of the investments that cloud agents (i.e. system developers) have made subject to the recorded dependability violations. In this process, the system derives the monetary value of the dependability violations over time to provide a clearer understanding to the investor or cloud agents typically, the system developers. All activities are saved as logs (see step *5* in Figure 6.1). These logs are periodically updated since cloud service performance varies due to the dynamic quality attributes which affects the metrics over time.

(c) **Reactive Middleware (RM)**: The RM helps to manage notifications. The RM notifies system agents of dependability violations, and afford them the option of managing these violations (see step *7* in Figure 6.1). All violations, activities in response to the violations, NPV computations and responsible agents are documented as *logs*. Also, the cloud agents can access the logs of their systems (see step *8* in Figure 6.1).

(d) **Logging System (LS)**: In this system, all the activities of the CAS com-

ponents are documented and saved as logs. This is where all the generated *evidence* and associated cloud *events* are stored. These logs can be used as evidence to claim compensation (i.e. "Service Credits"), and also serve as a source of data for predicting dependability violations using a form of machine learning.

The CAS is implemented to collect data from the RA's VM in run-time as per the metrics that are related to the architecture reliability and availability. Such metrics are Mean Time to Failure (MTTF) or Mean Time Between Failures (MTBF), Mean Time to Repair (MTTR), system operation time, etc. The data gathered are compared with those from the cloud platform and the SLA, and then notifications are triggered to the cloud agents: the system developers, the cloud service provider, and the cloud regulators if there are dependability violations. Figure 6.1 provides a conceptual overview of the CAS.

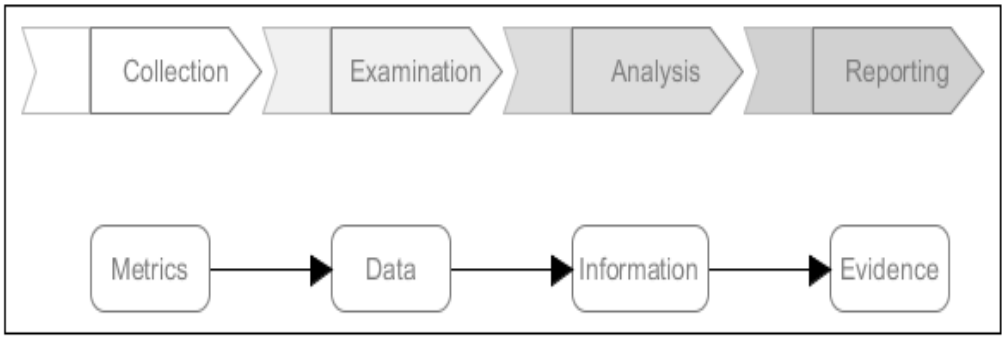


Figure 6.3: Forensic Process

6.3 Cloud Accountability Methodology

This research introduces a Cloud Accountability Method (CAM), which is facilitated by the CAS. This methodology is guided by a digital forensics model. The model is based on the well-established and widely accepted standard of NIST SP800-86. The NIST SP800-86 guide shows how digital forensics can support incident handling.

The forensic process shown in Figure 6.3 comprises the following basic phases:

- **Collection**: The first phase in the process is to identify, label, record, and acquire data from the possible sources of relevant data, while following guidelines and procedures that preserve the integrity of the data. Collection is typically performed in a timely manner because of the likelihood of losing dynamic data such as current network connections.
- **Examination**: Examinations involve forensically processing large amounts of collected data using a combination of automated and manual methods to assess and extract data of particular interest, while preserving the integrity of the data.
- **Analysis**: The next phase of the process is to analyse the results of the examination, using legally justifiable methods and techniques, to derive useful information that addresses the questions that were the reason for performing the collection and examination.
- **Reporting**: The final phase is reporting the results of the analysis, which may include describing the actions used, explaining how tools and procedures were selected, determining what other actions need to be performed (e.g. forensic examination of additional data sources, securing identified vulnerabilities, improving existing security controls), and providing recommendations for improvement to policies, guidelines, procedures, tools, and other aspects of the forensic process. The formality of the reporting step varies greatly depending on the situation.

The steps of our proposed cloud accountability method as well as its algorithm (see Algorithm 1) are provided below:

1. The Evidence Collector (EC) is assigned to the composing systems of the Reactive Architecture (RA), deployed on virtual machines (VMs) in the cloud.
2. The EC collects metrics data for availability, [A] and reliability, [R], including processing times in a synchronised manner from their respective VMs at a defined constant time duration, $[t_i, t_{i+1}]$ (i is initialised to zero).

3. The gathered metrics data are sorted and examined for data integrity by the Arbitrator.
4. The set $[A, R]_N$ is sent to the Auditor using synchronous procedure calls (SPC).
5. The Auditor initialises the trust values of the VMs to *0*.
6. The Auditor classifies $[A_N]$ and $[R_N]$ for the VMs.
7. The Auditor compares the sets of metrics data from the cloud evidence sources (i.e. VMI, CMS) to understand their relationship based on the SLA (i.e. acceptable metric range, $[AMR]_c^m$) of the cloud platform.
8. If any metric data of the two evidence sources violates the SLA's $[AMR]_c^m$, the Auditor assigns a value of *-1* to the trust value of the related VM, and then triggers a call to the Reactive Middleware (RM).
9. All activities including VM users, duration of VMs observation, date, details of failed VMs, analysis of failure, etc. are *logged*.
10. The RM then sends a notification to the cloud agents when all VMs are "unavailable".
11. The RM provides options such as requesting for "Service Credits" from the CSP using the *log* from (9).
12. The Auditor also computes the Net Present Value (NPV) of the set of cloud resources with respect to the processing times from (7), and/or violations from (8).
13. If there was no violation from (8), the Auditor assigns *1* to the trust value of the related VM, an analysis of the comparison in (7), and the NPV from (12) are saved as a *log*, $[l_N]$.
14. This log, $[l_N]$ is sent as a notification to the cloud agents as periodic reports (e.g. monthly, yearly).
15. The method continues in a loop at (2) for time duration,$[t_i, t_{i+1}]$ if none of the assigned VMs in (1) failed.
16. If there is any recorded failure, the method will continue in a loop at (1) for the given time duration, $[t_i, t_{i+1}]$.

Algorithm 1 Analyse Data from Cloud for Evidence-Based Dependability Assurance

Require: $VM_{RM}(A,R), VM_{SET}(A,R), VM_{SAR}(A,R), VM_{FDS}(A,R)$
Ensure: notify (l_N)

$EC_{RM} \leftarrow VM_{RM}(A,R)$
$EC_{SET} \leftarrow VM_{SET}(A,R)$
$EC_{SAR} \leftarrow VM_{SAR}(A,R)$
$EC_{FDS} \leftarrow VM_{FDS}(A,R)$
$duration \leftarrow 0$
$i \leftarrow 0$
$trust_{VM_N} \leftarrow 0$

top:
while $t_i \leq duration \leq t_{i+1}$ **do**
  $arbitrate_{time}(EC_{RM}, EC_{SET}, EC_{SAR}, EC_{FDS})$
  $audit_{SPC}(EC_{RM}, EC_{SET}, EC_{SAR}, EC_{FDS})$
  $audit_{classify}(EC_{RM}, EC_{SET}, EC_{SAR}, EC_{FDS})_R^A$
  $computeA_N, R_N$
  $audit_{metricsdata}(EC_{RM}, EC_{SET}, EC_{SAR})_{CMS}^{VMI}$

  if $(EC_{RM}, EC_{SET}, EC_{SAR}, EC_{FDS})_{CMS}^{VMI}$ violates $SLA's[AMR]_c^m$ **then**
    $trust_{VM_N} \leftarrow -1$
    $l_N(violation_{VM}, trust)$
    notify (l_N)
    $request_{ServiceCredit}()$

    if VM is unavailable **then**
      notify (l_N)
      $audit_{NPV}(audit_{metricsdata}(), violation_{VM}())$
    else $\{l_N(audit_{metricsdata}(), audit_{NPV}()\}$
    end if

  else $\{l_N(audit_{NPV}(), audit_{metricsdata}(), violation_{VM}())\}$
    $trust_{VM_N} \leftarrow 1$
    $l_N(violation_{VM}, trust)$
  end if

end while
goto *top*

6.4 Summary

In this chapter, we make two contributions: a novel methodology for assuring cloud dependability, and the Cloud Accountability System that facilitates the aforementioned methodology. This meets *Objectives 9* and *10* in Section 1.2. Our work aims to provide a meaningful level of assurance of cloud dependability in terms of availability and reliability of cloud based architectures. This assurance is enabled by the proposed methodology. This approach is guided by a well established digital forensic model to assure cloud agents of cloud dependability. This model is standardised by NIST SP800-86 to support incident handling. Here, cloud agents are provided with logs that give concise information about cloud resource activities. The information from our work aims to provide more detailed assessment of the cloud agents' cloud-based resources than what is provided from the CSPs. Here, we check the data based on the SLA from the CSPs, with those collected from the digital forensic process. An evidence-based trust analysis is then conducted.

Chapter 7

Evaluation

The aim of this chapter is to report on the evaluation of the Reactive Architecture proposed in Chapter 3. We conduct three main evaluations which are discussed below. Section 7.1 presents the evaluation of the Reactive Middleware which is undertaken using expert panel review process for validating the maturity of the CM-T process model (see Section 7.1.1). In Section 7.1.2, an Airlock Control System case study is used as a running example for the presented GSD guidelines facilitated by the Reactive Middleware. These are aimed to meet Objectives *3* and *4* in Section 1.2.2.

Also, Section 7.2 presents and discusses the evaluation of the Cloud Accountability Method. Here, the method is applied to the Reactive Architecture to evaluate how meaningful the cloud accountability method can be used to assure availability and reliability of cloud-based systems, relative to the cloud platform's service level agreement. This is in line with meeting Objective *11* in Section 1.2.2.

Then in Section 7.3, we evaluate the Reactive Architecture using a two-staged analysis approach defined by cloud-ATAM. This work mainly contributes the qualitative trade-off reasoning of quality attributes of a cloud-based Reactive Architecture using the two-staged analysis approach of cloud-ATAM (see *Objective 8* in Section 1.2.2). Finally, we draw conclusion in Section 7.4.

7.1 Reactive Middleware

The research evaluates the Reactive Middleware discussed Chapter 4. First, we validate the "change management and traceability (CM-T) process model" by conducting an "expert panel review process" in Section 7.1.1. This is used to assess the maturity of the CM-T process model in light of the CMMI Level 2 specific practices - requirements change management. The next section (i.e. Section 7.1.2) demonstrates the continuous tight linkage between requirements and system engineering processes provided by the introduced GSD guidelines, by an Airlock Control System case study.

At this point, we present our research question as "How can the Reactive Middleware guide system engineering to ensure the continual tight linkage of stakeholders' requirements and system engineering processes?", and a validating hypothesis below:

H1: Changes in system requirements' artefacts are captured and consistently propagated to all the related system engineering processes and stakeholders using a matured process model.

7.1.0.1 Panel Review Process

At this stage, we evaluate whether the motivation for building the Reactive Middlware's CM-T process model is justified, and if the process model is matured in relation to the CMMI Level 2 practice for managing change and traceability. We have motivated the use of expert panels to review or validate a software process model in Section 2.1.5.4. Also, the constitution of the expert panel is presented as a table (i.e. Table 2) in Appendix A.8. Then, we present our analysis of the feedback from the experts on the maturity of the CM-T model processes in light of the CMMI Level 2 (baseline) capability (refer to Table 7.1).

Table 7.1: Evaluation Criteria and Related CMMI Level 2 Capability Questions

Validation Criteria (Table 4.5)	**Rule (Table 4.5)**	**CMMI Level 2 Capability Questions (Figure 4.4)**
Adherence to CMM Charateristics	- CMM maturity level concepts must be implemented - Each level should have a theme consistent with CMM - Requirement engineering (RE) processes must be integrated - The model should be recognisable as a CMM offshoot - The CM-T must be systematic and sequential.	How repeatable are the following processes: (1) change management, (2) Traceability, (3) Analysis & Negotiations, (4) Documentation, and (5) Validation
Limited Scope	- Key activities relating to technical and organisational RE processes are included - Processes are prioritised. - Processes relate directly to the CM-T process areas - The scope/level of detail should be appropriate (i.e. depth and breadth of processes presented)	How repeatable are the following processes: (3) Analysis & Negotiations, and (4) Documentation
Consistency	- There should be consistent use of terms and CMM features at this level of development - There will be a consistency in structure between model components at the same level of granularity that are modelling different maturity levels	How repeatable are the following processes: (1) change management, (2) Traceability, (3) Analysis & Negotiations, (4) Documentation, and (5) Validation
Understandable	- All terms should be clearly defined (i.e. have only one meaning). - All relationships between processes and model architecture should be unambiguous and functional.	How repeatable are the following processes: (3) Analysis & Negotiations, and (4) Documentation
Ease of Use	- The model should be decomposed to a level that is simple to understand - The model should be simple yet retain meaning - The chunks of information should clearly relate as they develop into more complex structures - The model should require little or no training to be used	How repeatable are the following processes: (3) Analysis & Negotiations, and (4) Documentation
Verifiable	The model must be verifiable, i.e. we must be able to test or measure how well model meet its objectives and whether meeting these objectives leads to a high quality CM-T process model.	How repeatable are the following processes: (4) Documentation, and (5) Validation

7.1.1 Analysis of Expert Review

In this expert review, a panel of sixteen experts are constituted to validate the change management and traceability (CM-T) process model. Here, seven experts are from the industry, while nine are from academia. Experts from the industry are selected based on their experience in requirements engineering (RE) and software process improvement (SPI). Also, the research focus (i.e. RE and SPI) and publications of academics informed the selection of these academics. To facilitate the collection of data from the panel, a designed questionnaire is used as our data collection method. A questionnaire is appropriate for the type and nature of data we aim to collect and analyse. In terms of the type and nature of data for our analysis, we classify the questionnaire in relevant sections covering the "assessment of expertise" and the six relevant success criteria for validating the CM-T model. The related analysed results are provided as bar charts in Appendix A. For the type and nature of questions, the responses from the experts are expected to be classified as "Strongly agree", "Agree", "Neutral", "Disagree", "Strongly disagree". The "Neutral" option caters for both "uncertainty" and "no opinion" or "unwillingness to answer". In our analysis, we disregard the reasons for all the selected "Neutral" option. The classification of the questionnaire and the analysed results are provided below:

(A) **Self-Assessment of Expertise**

In the assessment of the expertise of the panel, we identified that 100% of the experts indicated their expertise in both "Software" or "System Engineering", "RE" and in "SPI". However out of the sixteen experts, one indicated no expertise in "CMMI".

(B) **Validating the CM-T Model**

- Validation Overview
  (a) "A matured model for change management and traceability is very relevant for requirement engineering?"

Towards the validation of the CM-T model, 100% of the experts' responses are supportive to *question 1* above. Here, seven (7) experts "strongly agree" and nine (9) of them "agree" to this question.

- Success Criteria One "Adherence to CMMI Characteristics"
  (a) "The classification of questions is representative of the CMMI Level 2 goal?"
  In assessing the adherence of the CM-T process model to CMMI characteristics, 62.5% of the experts "agree". However, six (6) experts indicated "neutral". Here, one (1) of the experts commented that CMMI is not his particular area of expertise, but the five (5) remaining experts chose not to provide further comments on this question.
  (b) "The CMMI processes have been adequately mapped to the identified questions?"
  Unanimously, all the sixteen (16) experts agree that the CMMI processes have been adequately mapped to the identified questions. Here, four (4) experts "strongly agree", and twelve (12) "agree".
- Success Criteria Two "Limited Scope"
  (a) "How complete is the CM-T model relative to the CMMI Level 2 processes?"
  In the assessment of the completeness of the CM-T model relative to the baseline processes of the CMMI Level 2, twelve (12) experts "agree" and four (4) are "neutral".
  (b) "How appropriate is it to include change management processes and traceability processes in one model?"
  Most of the experts (i.e. 87.5%) "agree" that it is appropriate to combine change management processes with traceability process in one model. More precisely, four (4) "strongly agree", and ten (10) "agree". However, one (2) remained "neutral" in response to this question.
  (c) "How well do the questions and assigned processes cover the key activities in change management and traceability of requirements?"
  The experts assessed how well the questions and assigned processes

cover the key activities in change management and traceability of requirements, and they were generally supportive: Four (4) "strongly agree", and eight (8) "agree". However, four (4) chose to be "neutral".

- Success Criteria Three "Consistency"
  (a) "How consistent is the level of detail given within the CM-T model?"
  The assessment of the consistency of the level of detail given within the CM-T model is very supportive (i.e. 100%). Here, five (5) experts "strongly agree", and eleven (11) "agree".
  (b) "All key processes are represented (at a baseline level)?"
  The experts provided a mostly supportive (i.e. 75.0%) response to the assessment that all the key processes associated with the defined questions (see Figure 4.4) are represented at a CMMI Level 2 baseline. Here, two (2) expert "strongly agree", as ten (10) "agree". On the other hand, four (4) provided a "neutral" response.
  (c) "All processes listed are at a similar level of abstraction?"
  In assessing the similarity of the level of abstraction for all the processes, the experts are mostly in agreement (i.e. 75.00%) in their response. This assessment shows five (5) experts "strongly agree", and seven (7) "agree". It was noticed that four (4) indicated a "neutral" position on this "consistency" question.
- Success Criteria Four "Understandability"
  (a) "How easy is it to understand the path from initial goal, to question, to final process?"
  In assessing the "understandability" of the CM-T model, the experts' response to "question 1" above is largely supportive (i.e. 87.50%): one (1) expert "strongly agree", and thirteen (13) others "agree". However, two (2) expert indicated a "neutral" option for this question.
  (b) "Each individual process is easy to understand (i.e. they are clearly defined and unambiguous)?"
  The experts are generally supportive (i.e. 68.75%) in response to "ques-

tion 2". Seven (7) experts "strongly agree" and four (4) experts "agree" that each individual process is easy to understand. However, five (5) experts chose the "neutral" option to this question.

(c) "How clear is this presentation of the model?"
Based on the relevant information about the CM-T model presented to the experts, they unanimously agree that the presentation was clear to understand. More specifically, three (3) experts "strongly agree", as thirteen (13) "agree" to "question 3" above.

- Success Criteria Five "Ease of Use"

(a) "Do you think that a considerable amount of prior knowledge of CMMI is needed to be able to interpret the CM-T model?"
To assess the "ease of use" of the CM-T model, all the experts "disagree" that a considerable amount of prior knowledge of CMMI is needed to be able to interpret the CM-T model. Here, five (5) experts "strongly disagree" and eleven (11) "disagree" to "question 1".

(b) "Dividing the RE process into smaller activities in this way will help practitioners to implement the process?"
Most of the experts (i.e. 87.50%) support the claim that dividing the requirements engineering process into smaller activities in the CM-T model will help practitioners to implement the process. Specifically, five (5) experts "strongly agree", and nine (9) "agree" to "question 2". However, two (2) experts chose to be "neutral".

- Success Criteria Six "Verifiability"

(a) "Has the level of detail provided with the questionnaire allowed you to give a fair assessment of the strengths and weaknesses of the CM-T model?"
In assessing the verifiability of the CM-T model, a majority of the experts (i.e. 81.25%) are supportive that the level of detail provided allowed them to give a fair assessment of the strengths and weaknesses of the CM-T model. To provide more details, five (5) experts "strongly agree", and eight (8) others "agree" to this question. That said, three

(3) experts remained "neutral" in their response.

In terms of validating the CM-T model, a total average of *84.38%* of the experts at least agree (i.e. indicating "Strongly agree" and "Agree") to the six (6) success criteria of the CM-T model. This high percentage of acceptance indicates the high level of conformance of the CM-T model to the CMMI Level 2 baseline processes. The composition of this high percentage is that, an average of *60.94%* of the experts "strongly agree", and also an average of *23.44%* of the experts "agree" to the questions in the questionnaire relating to the success criteria of the CM-T model. That said, *100%* of the experts "disagree" that "a considerable amount of prior knowledge of CMMI is needed to be able to interpret the CM-T model" under the "ease of use" success criteria. With this supportive choice, a new total average of *85.58%* of the experts accept the maturity of the CM-T model. It must also be mentioned that an average of *14.42%* chose to remain "neutral" on the questions.

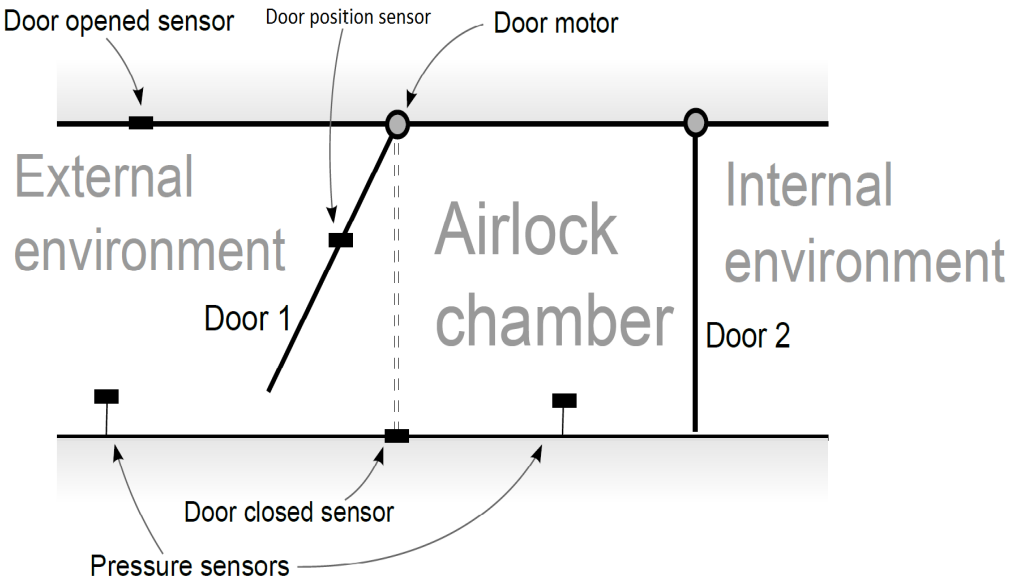


Figure 7.1: The Airlock Control System

For those who remained "neutral", we gathered three general comments from them, that we will consider in future work:

(a) Ambiguous process definitions for the "Consistency" criteria.

(b) The CM-T process model is incomplete for the "Adherence to CMMI Characteristics" criteria.

(c) The assessment component is not self-explanatory for the "Verification" criteria.

After validating the CM-T process model, we then apply the defined GSD guidelines (composing of the PMBOK project quality management approach and the CM-T model) to an airlock control system (ACS) case study.

7.1.2 Airlock Control System Case Study

A submarine airlock control system (ACS) (see Figure 7.1) case study from [94] is used to demonstrate the proposed management guidelines provided by the Reactive Middleware. The main function of the ACS is to separate two areas (i.e. external and internal) with different air pressures and allow users to pass safely between the areas. Let us assume that the pressure outside is lower than inside. In order to allow a user to pass from inside through the airlock into the external area, the system needs to perform the following steps:

- equalise the chamber pressure to that of the internal environment,
- open the second door to allow the user into the chamber,
- close the second door,
- equalise the pressure in the airlock to that of the external environment, and
- open the first door to let the user out.

Table 7.2: Classified Requirements of the Airlock Control System

*Airlock Control System Requirements (Resilient Quality Attribute - **SAFETY**)*		
Requirements Classification	**ID**	**Requirements**
Environment	ENV1	The airlock system separates two different environments. The pressure of the external environment is lower than that of the internal one.
	ENV2	In order to maintain different pressures, the two environments must be physically separated.
	ENV3	The system has two doors and a chamber. Each door when closed separates the chamber from the appropriate environment.
	ENV4	Each door is equipped with three positioning sensors and a two-way motor. The sensors consist of two boolean sensors representing the fully closed (*SNS_CLOSED*) and opened (*SNS_OPENED*) door states, and a range-value position sensor (*SNS_POS*) that returns values in a range between the fully closed and the fully opened states inclusively. The two-way motor (*ACT_MOTOR*) is the actuator that can open and close the door within its physical range of movement.
	ENV5	There is a pressure sensor in each of the areas, three in total (*SNS_PRESSURE_OUT*, *SNS_PRESSURE_CHAMBER*, *SNS_PRESSURE_IN*).
	ENV6	The pressure in the chamber can be changed by the pump actuator (*ACT_PUMP*).
	ENV7	Any of the sensors and actuators may fail to provide a correct function.
Safety	SAF1	The pressure in the chamber must always be between the lower external pressure and the higher internal one.
	SAF2	A door can only be opened if the pressure values in the chamber and the conjoined environment are equal.
	SAF3	Only one door is allowed to be opened at any moment of time.
	SAF4	The pressure in the chamber shall not be changed unless both doors are closed.
Function	FUN1	When in operation, the airlock system must be able to let users pass safely between the two environments via the airlock.

Moreover, the opposite scenario needs to be performed to allow the user pass from outside through the airlock into the external area. The system is equipped with a number of actuators - door motors, a pressure pump, as well as sensors - pressure sensors, door positions sensors and buttons. The goal of the GSD Team members spread over three geographical areas (i.e. Europe, Africa and Australia), is to develop control software that would allow a user to safely pass through the airlock. The GSD teams at Europe, Africa and Asia prioritise the safety and liveness properties (i.e. SAF1, SAF2, SAF3 and SAF4) of the ACS, leaving aside issues of its usability, operation speed, reliability and maintainability. It must be mentioned that safety properties described in this section do not completely cover all safety concerns that would arise for a real system. For example, a user would be required to wear special equipment while in the chamber in order to survive the change of pressure. We implicitly assume that this and other possible safety requirements are satisfied. We only focus on a particular part of system properties described in this section to limit the context of the case study. The high-level requirements of the system are presented in Table 7.2.

Table 7.3: Classified GSD Guidelines Steps

Classification of The GSD Guidelines Steps Into Stages/Services	
Stages/Services	**GSD Guidelines Steps ID**
User Management	GS1 and GS2
Requirement Management	GS3 and GS4
Change Management	GS5, GS6, GS7, GS8, GS10, GS11, and GS12
Traceability	GS9 and GS13

Airlock Control System Development Using the GSD Guidelines

The ACS case study is used as a running example of the Reactive Middleware's GSD guidelines in this section. In this activity, we develop a prototype of the Reactive Middleware and use it to demonstrate its set of services (or functionalities). Here, the development of Company X's (i.e. an assumed owning company name with simulated developers) ACS case study is guided by this set of guidelines. The set of GSD guidelines can be classified with respect to

the Reactive Middleware's services (i.e. User Management, Requirement Management, Change Management, and Traceability - see Table 7.3). However, the ACS case study presents the above classification under the five basic "process groups" (PG) (i.e. Initiating, Planning, Executing, Monitoring and Controlling, and Closing) of the PMBOK specific "knowledge area" (KA) of "project quality management". Company X expects that an effective change management and traceability in the development process of the ACS will improve the quality of the management of the project.

7.1.2.1 Initiating the ACS Project

The ACS is the core product of Company X. Company X sends some of its development activities offshore, but maintains a team of practitioners (i.e. Team Europe) in the European based central office who work mainly from 9 am to 5 pm five days a week. This team focuses on requirements gathering, prioritising requirements (and focus on "safety-related requirements"), developing their core product (i.e. the ACS), managing the offshore or GSD teams, and testing the bespoke software. The GSD teams (i.e. Team Africa and Team Asia) are more focused on product deployment and integration. Such a geographical spread of the set of teams is critical for Company X to be competitive in the airlock control system market, by managing a continuous "follow-the-sun" development approach. This approach ensures a continuous and around the clock development of the ACS, as well as having access to global skilled labour.

As part of Company X's ACS development policy, the Reactive Middleware (RM) is used to aid the effective management of the project, as well as to provide an automated facility to ensure that a matured (i.e. CMMI-compliant) change management and traceability process model is applied.

7.1.2.2 Planning the ACS Project

Team Europe focuses on the development of the airlock chamber (see Figure 7.1) of the ACS. Since the airlock chamber interfaces with both the external

environment and internal environment, the team can manage the changes that affect the "safety properties" of the ACS, and their traceability. Also, Team Africa and Team Asia are responsible for developing the external environment and internal environment respectively. In this project, the GSD teams use the cloud-based Reactive Middleware to manage the changes and traceability affecting the "safety properties" of the ACS during development. It is also identified to be essential that the development processes meet the "CMMI Level 2 practice". Company X decides to apply the GSD guidelines to its development process. Before the execution of the ACS project, Team Europe generates some project diagrams to guide the execution process. Here, a class diagram (see Figure 9), a package diagram (see Figure 8), and an interaction diagram (see Figure 10) are provided.

7.1.2.3 Executing the ACS Project

At this point, team leaders are appointed for the three respective GSD teams (i.e. Team Europe, Team Africa, Team Asia). These activities are guided by the **user management** set of guidelines (i.e. **GS1** and **GS2**) defined under the GSD guidelines.

> **GS1:** System development teams should appoint team leaders.
> **GS2:** These team leaders will constitute the GSD change managers.

Here, team leaders play the role of "Team Leader" with an associated privilege of "Own" where they have permission to perform any activities on development artefacts (such as specification, requirements, configurations, documentation, etc.). In the same light, team leaders assign roles (e.g."Reviewer", "Modifier", "Pawn", etc.) and privileges (i.e."Review", "Modify", "View" respectively) to team members. Then the team leaders form the development supervisory team referred to as the GSD Change Managers. This GSD Change Managers team performs a crucial role of managing changes that affects prioritised ACS "safety requirements", and "trace" the changes' cause-and-effect on requirements and

```
**********************
USER MANAGEMENT
**********************

Creating TEAM AFRICA USERS..
        User Details: (Ato Adjepon-Yamoah, TL02, TeamLeader, Own)
        User Details: (Akosua Adjettey, R002, Reviewer, Review)
        User Details: (Adetolu Lacriox, M003, Modifier, Modify)

Creating TEAM ASIA USERS..
        User Details: (Olu Jensen, TL03, TeamLeader, Own)
        User Details: (Feng Jiabao, R003, Reviewer, Review)
        User Details: (Kate Supha, M004, Modifier, Modify)

Creating TEAM EUROPE USERS..
        User Details: (Joe Sinclair, TL01, TeamLeader, Own)
        User Details: (Arun Gupta, R001, Reviewer, Review)
        User Details: (Jao Xhiabao, M001, Modifier, Modify)
        User Details: (Oliver Annan, M002, Modifier, Modify)
        User Details: (Javier Mendes, P004, Pawn, View)


Creating GSD CHANGE MANAGERS...
UserID: TL01, [Joe Sinclair]
UserID: TL02, [Ato Adjepon-Yamoah]
UserID: TL03, [Olu Jensen]


PROJECT NAME: AirLockControlSystem
PROJECT TEAM SET: teamAfrica, teamAsia, teamEurope
```

Figure 7.2: Snippet of the Terminal Output for the Reactive Middleware User Management Service for the Airlock Control System Case Study

associated artefacts. Figure 7.2 shows a snippet of the Reactive Middleware's user management service.

The next set of activities relating to the ACS requirements are guided by the **requirements management** set of guidelines (i.e.**GS3** and **GS4**) defined under the GSD guidelines.

> **GS3:** System requirements should be classified based on identified resilient quality attributes (i.e. safety, reliability, robustness, etc.), and are then prioritised relative to their importance to the system stakeholders.
> **GS4:** Team leaders must assign roles to all team members with the prioritised requirements in mind, and manage the development process with the adapted PMBOK guide.

```
**********************
REQUIREMENT MANAGEMENT
**********************

*LIST of Airlock Control System Requirements:

[Requirement: 11]:- [Priority: false], [ReqID: SAF4], [Requirement:The
<pressure> in the <chamber> shall not be changed unless both <doors> are
<closed>.]

[Requirement: 1]:- [Priority: false], [ReqID: ENV1], [Requirement:The airlock
system separates two different environments. The pressure of the external
environment is lower than that of the internal one.]

[Requirement: 12]:- [Priority: false], [ReqID: FUN1], [Requirement:When in
operation, the airlock system must be able to let users pass safely between
the two environments via the airlock.]

[Requirement: 2]:- [Priority: false], [ReqID: ENV2], [Requirement:In order to
maintain different pressures, the two environments must be physically
separated.]

[Requirement: 3]:- [Priority: false], [ReqID: ENV3], [Requirement:The system
has two doors and a chamber. Each door when closed separates the chamber from
the appropriate environment.]

[Requirement: 4]:- [Priority: false], [ReqID: ENV4], [Requirement:Each door
is equipped with three positioning sensors and a two-way motor. The sensors
consist of two boolean sensors representing the fully closed (SNS_CLOSED) and
opened (SNS_OPENED) door states, and a range-value position sensor (SNS_POS)
that returns values in a range between the fully closed and the fully opened
states inclusively. The two-way motor (ACT_MOTOR) is the actuator that can
open and close the door within its physical range of movement.]

[Requirement: 5]:- [Priority: false], [ReqID: ENV5], [Requirement:There is a
pressure sensor in each of the areas, three in total (SNS_PRESSURE_OUT},
SNS_PRESSURE_CHAMBER, SNS_PRESSURE_IN).]

[Requirement: 6]:- [Priority: false], [ReqID: ENV6], [Requirement:The
pressure in the chamber can be changed by the pump actuator (ACT_PUMP).]

[Requirement: 7]:- [Priority: false], [ReqID: ENV7], [Requirement:Any of the
sensors and actuators may fail to provide a correct function.]

[Requirement: 8]:- [Priority: false], [ReqID: SAF1], [Requirement:The
<pressure> in the <chamber> must always be between the <lower external
pressure> and the <higher internal pressure>.]

[Requirement: 9]:- [Priority: false], [ReqID: SAF2], [Requirement:A <door>
can only be <opened> if the <pressure values> in the <chamber> and the
conjoined <environment> are equal.]

[Requirement: 10]:- [Priority: false], [ReqID: SAF3], [Requirement:Only one
<door> is allowed to be <opened> at any moment of <time>.]
```

Figure 7.3: Snippet of the Terminal Output for the Reactive Middleware Requirements Management Service for the Airlock Control System Case Study

```
*SET Safety Requirements to PRIORITY:

[Requirement: 11]:- [Priority: true], [ReqID: SAF4], [Requirement:The
<pressure> in the <chamber> shall not be changed unless both <doors> are
<closed>.]

[Requirement: 1]:- [Priority: false], [ReqID: ENV1], [Requirement:The airlock
system separates two different environments. The pressure of the external
environment is lower than that of the internal one.]

[Requirement: 12]:- [Priority: false], [ReqID: FUN1], [Requirement:When in
operation, the airlock system must be able to let users pass safely between
the two environments via the airlock.]

[Requirement: 2]:- [Priority: false], [ReqID: ENV2], [Requirement:In order to
maintain different pressures, the two environments must be physically
separated.]

[Requirement: 3]:- [Priority: false], [ReqID: ENV3], [Requirement:The system
has two doors and a chamber. Each door when closed separates the chamber from
the appropriate environment.]

[Requirement: 4]:- [Priority: false], [ReqID: ENV4], [Requirement:Each door
is equipped with three positioning sensors and a two-way motor. The sensors
consist of two boolean sensors representing the fully closed (SNS_CLOSED) and
opened (SNS_OPENED) door states, and a range-value position sensor (SNS_POS)
that returns values in a range between the fully closed and the fully opened
states inclusively. The two-way motor (ACT_MOTOR) is the actuator that can
open and close the door within its physical range of movement.]

[Requirement: 5]:- [Priority: false], [ReqID: ENV5], [Requirement:There is a
pressure sensor in each of the areas, three in total (SNS_PRESSURE_OUT),
SNS_PRESSURE_CHAMBER, SNS_PRESSURE_IN).]

[Requirement: 6]:- [Priority: false], [ReqID: ENV6], [Requirement:The
pressure in the chamber can be changed by the pump actuator (ACT_PUMP).]

[Requirement: 7]:- [Priority: false], [ReqID: ENV7], [Requirement:Any of the
sensors and actuators may fail to provide a correct function.]

[Requirement: 8]:- [Priority: true], [ReqID: SAF1], [Requirement:The
<pressure> in the <chamber> must always be between the <lower external
pressure> and the <higher internal pressure>.]

[Requirement: 9]:- [Priority: true], [ReqID: SAF2], [Requirement:A <door> can
only be <opened> if the <pressure values> in the <chamber> and the conjoined
<environment> are equal.]

[Requirement: 10]:- [Priority: true], [ReqID: SAF3], [Requirement:Only one
<door> is allowed to be <opened> at any moment of <time>.]
```

Figure 7.4: Snippet of the Terminal Output for the Reactive Middleware Requirements Management Service (Priority) for the Airlock Control System Case Study

Firstly, the ACS requirements are assigned as either a priority or not, and then they are uniquely identified (see Figure 7.3). The GSD Change Managers set the safety requirements as the priority of the project's quality attributes (see Figure 7.4).

After all these activities have been undertaken, the three GSD teams begin the development of the ACS. Here, the development of the three ACS environments are initialised as well as the doors, sensors and actuators.

7.1.2.4 Monitoring and Controlling the ACS Project

The GSD Change Managers together with some project stakeholders at the European based central office, "monitor and control" the development process to meet the project requirements. Particular attention is placed on the set of safety related ACS requirements. In this activity, a set of the GSD guidelines are for **change management** (i.e.**GS5**, **GS6**, **GS7**, **GS8**, **GS10**, **GS11**, and **GS12**) and **traceability** (i.e.**GS9** and **GS13**).

GS5: All other change agents especially the system engineering tools should be assigned a default privilege of review.
GS6: All system artefacts should be saved in a shared artefacts repository.
GS7: The privileges (i.e.none, view, modify, review, own) of system stakeholders or change agents will determine the access privileges to system artefacts.
GS8: Change agents must subscribe to relevant artefacts after they are created, in order to receive notifications when they are changed.
GS10: Changes made to any system artefacts must be logged.
GS11: When changes affect the high priority set of requirements, appropriate local team leader must lead the change request review process (i.e.involving the CM-T model) of the GSD change managers.
GS12: On the other hand, conflicts arising from changes to low priority set of requirements are resolved locally, lead by the local team leader.

```
**********************
CHANGE MANAGEMENT - CM
**********************

Creating CHANGE REQUESTS...

*CHANGE REQUEST POOL:

Change Request: [CR1685232414]:
(ActionPPMachine.class, ROO3, SAF3, 'Prioritising the opening of door
requests from both the internal or external environments')

Change Request: [CR2807444458]:
(artName1, ROO3, ENV3, 'Colour of door is critical to safety')


*Is Priority of Requirement (SAF3) HIGH?: [true]

*[Sign-Off CHANGE REQUEST]: 'approve', 'note' OR 'disapprove':-

Decision: [approve]

Change Request Details:
Change Request: [CR1685232414]:
(ActionPPMachine.class, ROO3, SAF3, 'Prioritising the opening of door
requests from both the internal or external environments')

Change Request decision is [approve]
```

Figure 7.5: Snippet of the Terminal Output for the Reactive Middleware Change Management Service for the Airlock Control System Case Study

During the development process, a set of "change requests" relative to the ACS requirements are raised. This process requires the Change Management and Traceability (CM-T) process model (see Section 4.2.2.5) of the Reactive Middleware. The steps of the CM-T process model are followed to resolve all "change requests".

Change requests are expected to contain information about the artefact involved, the identification of the initiating stakeholder, the relevant requirement identification, and the change request details. Here, two change requests are submitted to the change request pool (see Figure 7.5). The GSD Change Managers consider the change requests based on their priority, and then select which one to make a decision on. At this point, GSD Change Managers decide that the change request with a unique identification *CR1685232414* is of high priority (i.e. as a safety related request - SAF3) and needs to be considered further.

```
*****************************
TRACE ARTEFACT CHANGE - Trace
*****************************

[Trace] Trigger Change in artefact by [R003]:- Feng Jiabao


[CHANGE MANAGEMENT -CM] process(es)...
[CM- SAME FILENAMES] Retrieved Artefact Name: 'e:\ActionPPMachine.class'
[CM- SAME HASH] Hash of Input Artefact [-614419020], and Hash of Ouput
Artefact [-614419020] ---> ChunkSize: 261120

[TRACEABILITY -trace] process(es)...

[Trace Changed Artefact [e:\ActionPPMachine.class] --> '2016/11/19 00:03:49'
begins...]
            [Trace --> UserID]: R003,
            [Trace --> Artefact]: e:\ActionPPMachine.class,
            [Trace --> Notification]: [CM- TRIGGER] Change Request on:
*'e:\ActionPPMachine.class' by User[R003]

[Trace --> Changed Artefact [e:\ActionPPMachine.class] -->2016/11/19 00:03:49
ends]

[CM- TRIGGER] Change Request on: *'e:\ActionPPMachine.class' by User[R003]

[Trace Summary]
***************

[Trace]:---> CHANGED ARTEFACT:
Artefacts Details: [ART2537339]:- 'Only one <door> is allowed to be <opened>
at any moment of <time>.'

[Trace]:---> CHANGED AGENT: (R003) - Feng Jiabao

[Trace]:---> CAUSE-AND-EFFECT on ARTEFACTS:
(1)Artefacts Details: [ART2537337]:- 'The <pressure> in the <chamber> must
always be between the <lower external pressure> and the <higher internal
pressure>.'

(2)Artefacts Details: [ART2537338]:- 'A <door> can only be <opened> if the
<pressure values> in the <chamber> and the conjoined <environment> are
equal.'

(3)Artefacts Details: [ART2537340]:- 'The <pressure> in the <chamber> shall
not be changed unless both <doors> are <closed>.'

Sent message: '[CM- TRIGGER] Change Request on: *'e:\ActionPPMachine.class'
by User[R003] : Thread-1 : 972210874' :--> 1659654801 : Thread-1

Received: '[CM- TRIGGER] Change Request on: *'e:\ActionPPMachine.class' by
User[R003] : Thread-1 : 972210874'

Notification: [Key: 1498854882]--> Value: '[CM- TRIGGER] Change Request on:
*'e:\ActionPPMachine.class' by User[R003] : Thread-1 : 972210874'
```

Figure 7.6: Snippet of the Terminal Output for the Reactive Middleware Traceability Service for the Airlock Control System Case Study

The change request with a unique identification *CR280744458* is related to the ACS' environmental requirements (i.e. ENV3).

That said, the details of the change request being considered is that 'prioritising the opening of door requests from both the internal and external environments'. The team leader for the team (i.e.Team Asia) that presented the change request defends the criticality of the request, and leads the decision-making process. This is a critical change request as it is important to only have one *door* of the ACS open at a given time, and hence door opening requests that occur simultaneously from both the internal and external environments must be prioritised. This ensures a level of safety. As a result of the need for such a change, the change request (i.e. *CR1685232414*) is approved by the GSD Change Managers.

The decision taken for *CR1685232414* is logged as part of the documentation of the change request. The initiator (i.e.*R003*) of the change request is notified to "effect the change". Effecting this change requires a close monitoring by the team leader of Team Asia to make sure that it is undertaken as expected. When the process of effecting the change is completed. The team leader assesses the process, and then "verifies and validates" the change. The log of the verified and validated change is updated accordingly.

The next step undertakes a detailed assessment of the "cause-and-effect" of the change on project artefacts, and all minor conflicts (i.e.involving less prioritised project requirements) are resolved within the local GSD team. In situations where the initial change affects a prioritised project requirement, the team leader advises the most relevant stakeholder to submit a change request to the GSD change managers for consideration. During the assessment of the "cause-and-effect" of the change, it identified to have an impact on three requirement artefacts (see Figure 7.6). The affected requirements are ENV6, SAF2, and SAF4. Here, the minor conflict relating to the ENV6 ACS environment requirement is resolved within Team Asia. However, the conflict involving prioritised safety requirements (i.e.SAF2 and SAF4) are submitted as change requests to the change request pool to be considered by the GSD Change Managers. This requires the spawning of a new change process based on the CM-T

process model.

```
****************************
Notification To Stakeholders
****************************

[INFO] Team (teamAsia) Members to receive Notification:
UserID: TL03, [Olu Jensen]
UserID: R003, [Feng Jiabao]
UserID: M004, [Kate Supha]


 INFO |
[Notify User(s)]:
[UserID: TL03, [Name:Olu Jensen]:
Notification: [Key: 1498854882]--> Value: '[CM- TRIGGER] Change Request on:
*'e:\ActionPPMachine.class' by User[R003] : Thread-1 : 972210874'

 INFO |
[Notify User(s)]:
[UserID: R003, [Name:Feng Jiabao]:
Notification: [Key: 1498854882]--> Value: '[CM- TRIGGER] Change Request on:
*'e:\ActionPPMachine.class' by User[R003] : Thread-1 : 972210874'

 INFO |
[Notify User(s)]:
[UserID: M004, [Name:Kate Supha]:
Notification: [Key: 1498854882]--> Value: '[CM- TRIGGER] Change Request on:
*'e:\ActionPPMachine.class' by User[R003] : Thread-1 : 972210874'
```

Figure 7.7: Snippet of the Terminal Output for the Reactive Middleware Notification Service for the Airlock Control System Case Study

From the point where an "approved" change is effected till the point where it has been implemented successfully, a process to "trace" this change with regards to participating stakeholders, associated software development life-cycle (SDLC) phase, corresponding system engineering tools, impact on other artefacts, etc. is undertaken in parallel. The activities for tracing changes also facilitates "roll-back" in situations where the resulting conflicts from the "cause-and-effect" of a change is highly undesirable relative to the project requirements. This activity is guided by the set of GSD guidelines (i.e. GS9 and GS13). During this process, the log of this change is updated.

GS9: All *related* artefacts must be linked together to facilitate traceability.
GS13: Changes in system artefacts should be traceable to manage its impact on related/linked requirements or artefacts.

Lastly, the GSD Change Managers accept the change and the change process is marked as successful. Then a "notification" is generated and distributed to all the relevant stakeholders of the change in Team Asia (see Figure 7.7). The change request is then "closed".

7.1.2.5 Closing the ACS Project

After the ACS development process involving a series of change processes that are "monitored and controlled" relative to the ACS requirements, the system is demonstrated to the project stakeholders (i.e.Project Approval Board of Company X, and relevant users). The stakeholders undertake an evaluation of the ACS according to the prioritised requirements and expectations. During the evaluation process, highlights of the ACS development process are identified and discussed. Also, lessons are learnt from the process.

7.1.2.6 Summary

In this section, we introduce a cloud-based Reactive Middleware that applies a defined change management and traceability (CM-T) process model, within the context of an adapted PMBOK quality process management approach to GSD. This is in line with *Objectives 1* and *2* presented in Section 1.2.2. The Reactive Middleware provides cloud-based services for user managament, requirement management, change management and traceability, and are facilitated by our GSD management guidelines.

To ensure that the defined CM-T process model complies with the CMMI Level 2 (Baseline) Capability, the CM-T process model is validated using an expert

panel review process (see Section 7.1.1) where a total average *85.58%* of the experts supported the maturity of the CM-T process model. Also, we demonstrate the application of the GSD management guidelines provided by the Reactive Middleware with an Airlock Control System case study (see Section 7.1.2). Here, we highlight the continual tight linkage of stakeholders' requirements and system engineering processes towards change management and traceability, through the application of our prototype of the Reactive Middleware to the case study. This continual tight linkage between the set of requirements and the system engineering processes is enabled using the Reactive Middleware which supports the bidirectional tracking of prioritised system requirements.

7.2 Cloud Accountability System

In this section, we aim to provide support for assuring the dependability (i.e. availability and reliability) of the cloud for system engineering with a cloud accountability method (see *Objective 5* of Section 1.2.2). This assurance is achieved in relation to the CSPs' SLA. We however, conduct an assessment of the SLA for the relevant cloud platform to this work in Section 2.3.3.1. Here, a set of assured values for cloud availability and compensation or "service credit" are also identified. To achieve our objective, we define our research question as "Can a cloud accountability method be used to meaningfully assure availability and reliability of deployed systems?"

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS
vmiGuestFDS	i-0211824ca098bbce2	t2.micro	eu-west-1b	running	2/2 checks ...	None	ec2-52-212-
vmiMONITOR	i-05e50b2d2c3953f31	t2.micro	eu-west-1c	running	2/2 checks ...	None	ec2-52-213-
vmiGuestRM	i-6d16b7ad	t2.micro	eu-west-1b	running	2/2 checks ...	None	ec2-52-50-5
vmiGuestSAR	i-a61bba66	t2.micro	eu-west-1b	running	2/2 checks ...	None	ec2-52-19-1
vmiGuestSET	i-c61abb06	t2.micro	eu-west-1b	running	2/2 checks ...	None	ec2-52-212-

Figure 7.8: AWS/EC2 CAS Test-Bed Instances

In this section, we evaluate our work by answering our research question and draw conclusions on this chapter. We begin by describing how we apply the cloud accountability method (CAM) (see Section 6.3) to the Reactive Architecture (RA). To achieve this, we implement a prototype of the cloud accountability system (CAS) to facilitate the active monitoring of the RA. Our evaluation is performed using a cloud-based test-bed in the Amazon Web Service (AWS) Elastic Cloud Compute (EC2) environment. The AWS/EC2 test-bed is run in the *eu-west-1b* and *eu-west-1c availability zones* (i.e.geographic locations of AWS cloud infrastructure/service) in Ireland. Such a decision is necessary since these availability zones provide the least network latency to Newcastle University where the evaluation is conducted. It also allows for the monitoring or introspection of the group of VMs for the set of RA components from another availability zone. For the test-bed, we create a set of 5 Linux virtual machines (i.e. Ubuntu 16.04 x64 t2.micro with 1 GB of memory, 1 vCPU, SSD Volume Type and variable ECU - see Figure 7.8).

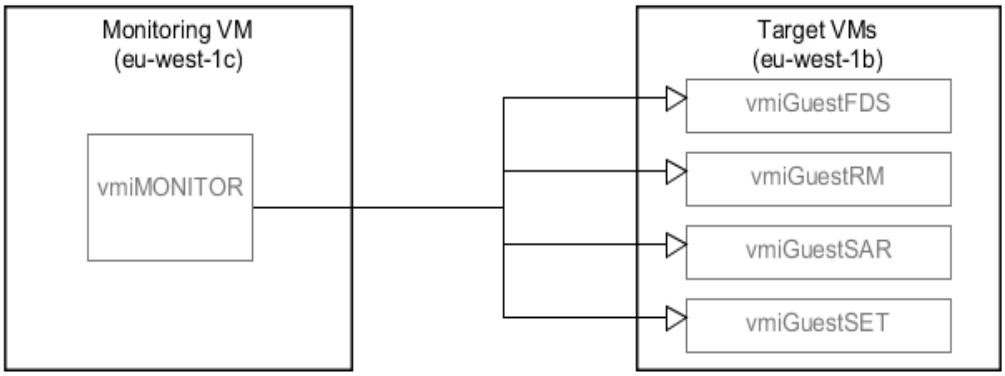


Figure 7.9: Test-Bed's VMs Introspection in Two AWS Availability Zones

The 5 instances are classified into two main VMs; the *target VMs* and the *monitoring VM* (see Figure 7.9). The *target VMs* are constituted with *vmiGuestFDS* (i.e. Formal Decomposition System), *vmiGuestRM* (i.e. Reactive Middleware), *vmiGuestSAR* (i.e. Shared Artefacts Repository), and *vmiGuestSET* (i.e. System Engineering Toolbox) VM instances. The *target VMs* are located at the *eu-west-1b* availability zone. Also, the *monitoring VM* is the *vmiMONITOR*

VM instance located at the *eu-west-1c* availability zone. The separation of the availability zones allows for the effective introspection of the *target VMs* by the *vmiMONITOR* VM, without being subjected to the same dependability situation at the *eu-west-1b* availability zone.

Table 7.4: Classified Steps of Cloud Accountability Method with The Forensic Process Phases (NIST SP800-86 Guide)

The Forensic Process Phases (NIST SP800-86 Guide)	**Cloud Accountability Method Steps**
Collection	1, 2, 4, 5, 15 and 16
Examination	3 and 6
Analysis	7, 8, 11 and 12
Reporting	9, 10, 13, and 14

Accountability Method for Cloud Dependability Assurance
We classify the steps of the presented cloud accountability method according to the phases of the NIST SP800-86 guide (see Table 7.4). The conceptual model (i.e. Figure 6.1) of the cloud accountability system is also used to support this classification.

7.2.1 Collection

To implement the CAS, we look at the source of digital evidence collection. In our work, we identify two sources for accessing data related to dependability metrics from the *hypervisor* of the RA virtual machines (discussed in Section 2.3.4.1). The choice for the two sources of data is because the data from the CSPs are perceived to be untrustworthy, so a second data source using a reliable digital forensic approach is considered to check the former data source. These are:

7.2.1.1 AWS CloudWatch

The API used here is the *AmazonCloudWatchClient* API, which provides data related to metric statistics. The *metrics data* provided by the AWSCloudWatch-

Client API also provide information for CPU utilisation, status checks for VMs and system, network packets, network state, disk write operations, disk read operations, CPU credit usage, and CPU credit balance.

Metric	Description
StatusCheckFailed	Reports whether the instance has passed both the instance status check and the system status check in the last minute. This metric can be either 0 (passed) or 1 (failed). Units: Count
StatusCheckFailed_Instance	Reports whether the instance has passed the instance status check in the last minute. This metric can be either 0 (passed) or 1 (failed). Units: Count
StatusCheckFailed_System	Reports whether the instance has passed the system status check in the last minute. This metric can be either 0 (passed) or 1 (failed). Units: Count

Figure 7.10: Some AWS/EC2 Instance and System Metrics

Status check metrics are available at 1 and 5 minute frequencies. However, we focus exclusively on the *CPU utilisation*, *status checks for VMs*, and *status check for system* metrics data collected at a frequency of 5 minutes. This time frequency is appropriate since it is long enough to accommodate our 1 minute data collection call from the target VMs, and network latency which can be unpredictable. We believe that this set of metrics are representative for providing a reliable picture of the dependability of a set of cloud-based VMs. All the mentioned metrics are also provided to users of AWS as graphs in AWS CloudWatch relative to users' AWS resources (e.g. EC2). In this work, we consider metrics from the AWS CloudWatch API, as well as the generated graphs from AWS CloudWatch. In Figure 7.10, we provide a brief description of some of the focal metrics we consider in our evidence collection. Also, a snippet of Java code showing the implementation of AWS/EC2 CloudWatch API for collecting the data for the *CPU utilisation* metric in *Listing 1 (see Appendix D.16)*. This same approach is used to collect the other three focal metrics.

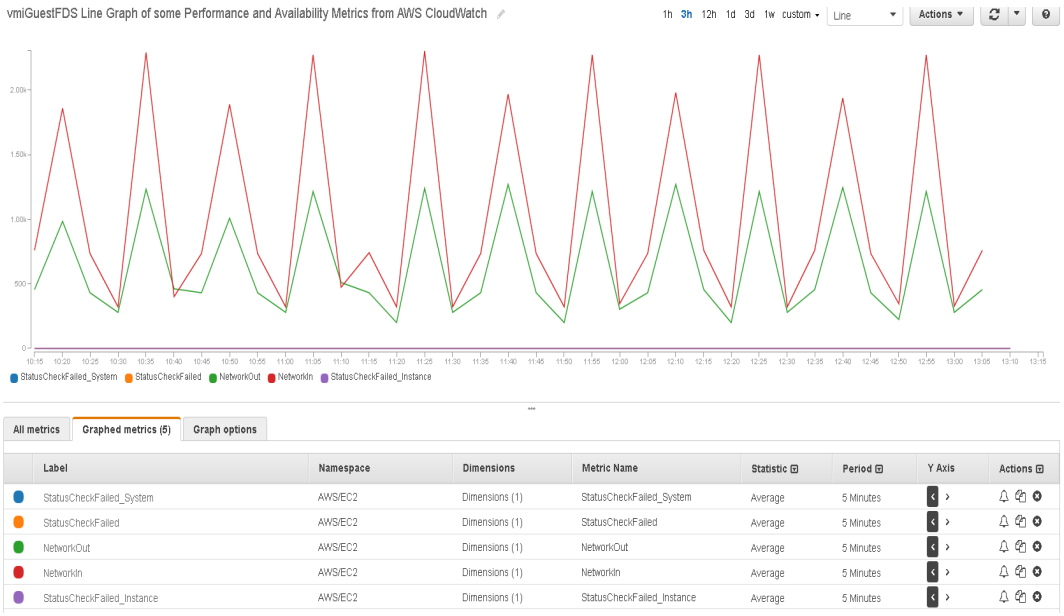


Figure 7.11: vmiGuestFDS: AWS CloudWatch Line Graphs for Some Metrics

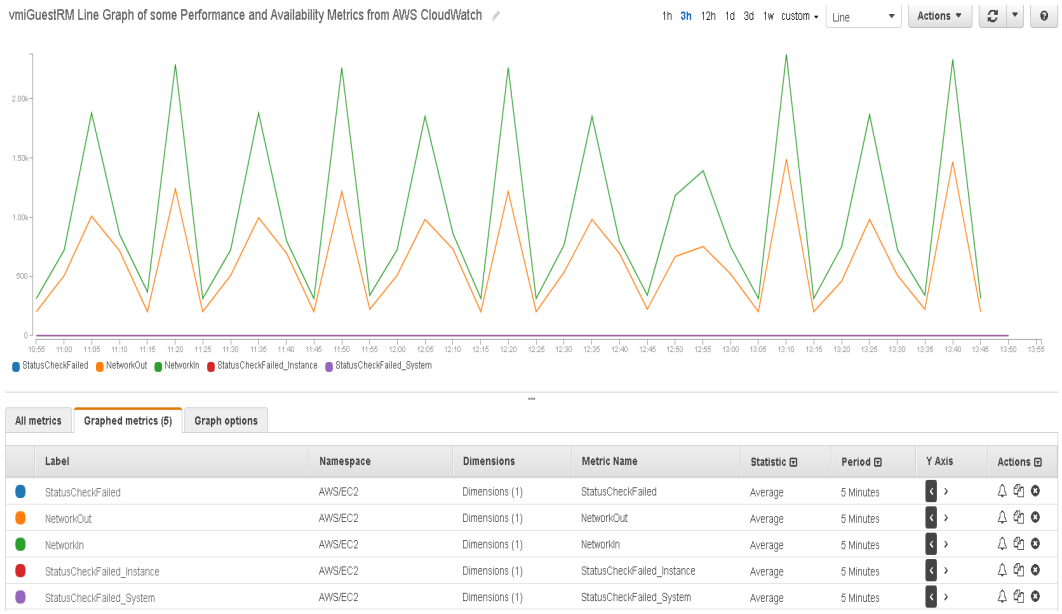


Figure 7.12: vmiGuestRM: AWS CloudWatch Line Graphs for Some Metrics

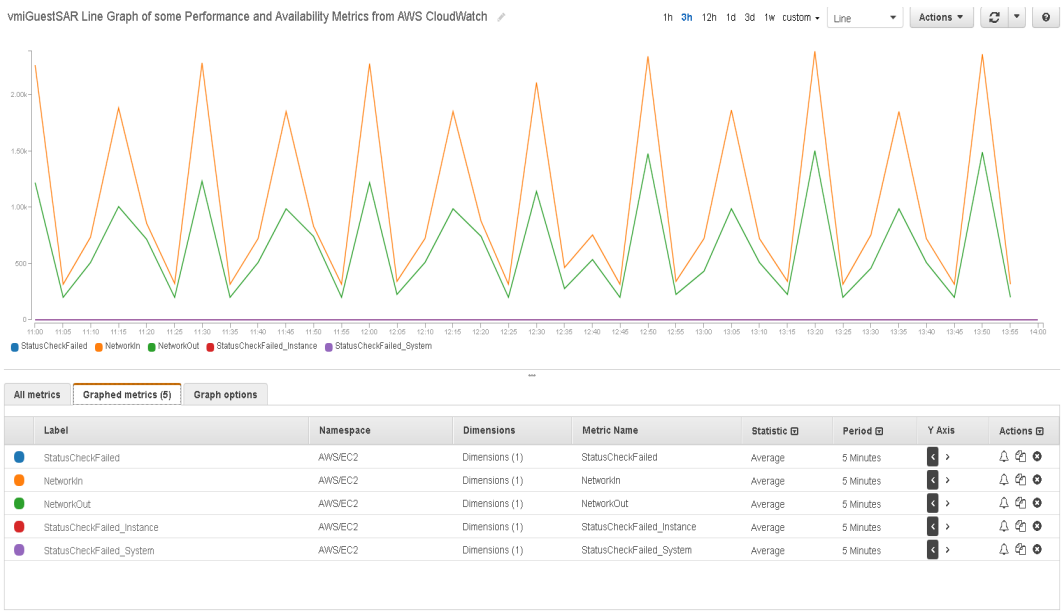


Figure 7.13: vmiGuestSAR: AWS CloudWatch Line Graphs for Some Metrics

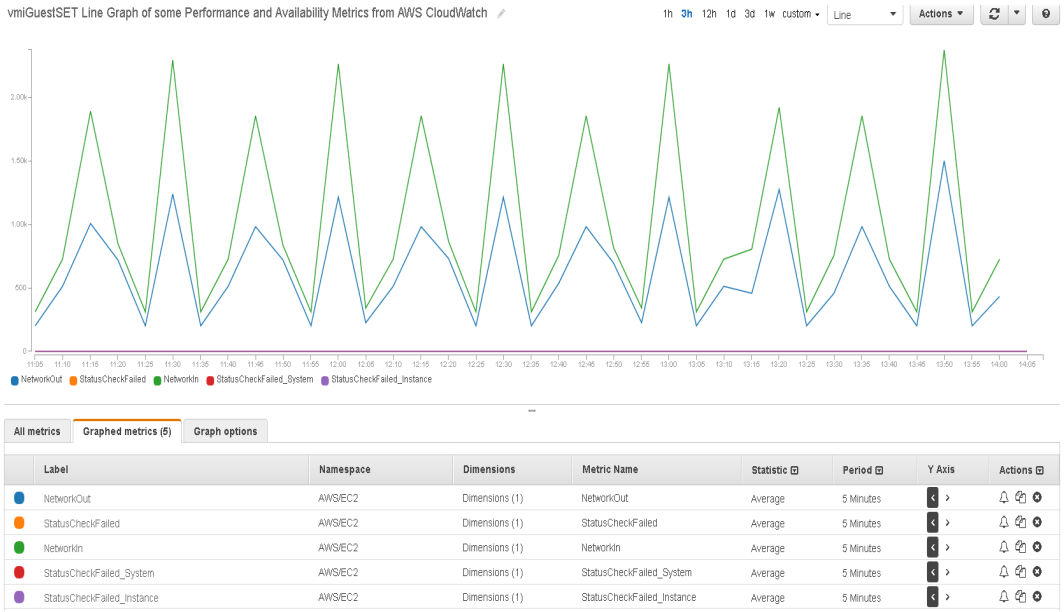


Figure 7.14: vmiGuestSET: AWS CloudWatch Line Graphs for Some Metrics

```
/**
 * Gets the physical address and page size of the VA
 * as well as the addresses of other paging related structures
 * depending on the page mode of the VM.
 *
 * @param[in] vmi LibVMI instance
 * @param[in] dtb address of the relevant page directory base
 * @param[in] vaddr virtual address to translate via dtb
 * @param[in,out] info Pointer to the struct to store the lookup information in
 * @return VMI_SUCCESS or VMI_FAILURE of the VA is invalid
 */
status_t vmi_pagetable_lookup_extended(
    vmi_instance_t vmi,
    addr_t dtb,
    addr_t vaddr,
    page_info_t *info);
```

(a) Page Table Lookup Function

```
/**
 * Returns the path of the Linux system map file for the given vmi instance
 *
 * @param[in] vmi LibVMI instance
 * @return String file path location of the Linux system map
 */
const char * vmi_get_linux_sysmap(vmi_instance_t vmi);
```

(b) System Map Function

Figure 7.15: LibVMI API Functions used in the vmiMONITOR Instance

Initially, we took a look at the focal metrics as presented by the AWS CloudWatch for the test-bed. From Figures 7.11, 7.12, 7.13, and 7.14, we observed a set of line graphs that indicate that the instances are "healthy" per the set of metrics. Here, the data for the *status checks* metrics indicate results of "0", implying that there has not been any recorded failures for the target VMs in the 3-hour observation period. That said, our work takes a step further to collect our own set of evidence relative to the said set of metrics. We consider a well used digital forensic process known as *virtual machine introspection* as our reliable means of accessing such data. This process is discussed below. Here, with the aid of the metrics provided by the APIs, we aim to deduce the state of dependability of the CAS target VM instances by obtaining the mean time to failure ($MTTF$) or mean time between failures ($MTBF$), mean time to recover

(*MTTR*), and *system operation time*.

7.2.1.2 Virtual Machine Introspection (LibVMI)

The LibVMI API for virtual machine introspection is considered for evidence collection. Here, we first install Xen with *dom0* getting 1GB RAM assigned and 1 dedicated vCPU core on the vmiMONITOR VM. The LibVMI API provides functions such as "vmi-pagetable-lookup", specifically "vmi-pagetable-lookup-extended" to check the performance status of VMs. Importantly, it returns VMI_SUCCESS for an "active" and "healthy" VM, or VMI_FAILURE if the VM is "invalid" or "inactive". To further check for VM introspection failure, we consider "vmi-get-linux-sysmap" to show a linux system path to an active VM. Figure 7.15 shows the set of LibVMI API functions that are considered for introspection by the Evidence Collector system on the four target VMs.

1: The Evidence Collector (EC) is assigned to the composing systems of the Reactive Architecture (RA), deployed on virtual machines (VMs) in the cloud.
2: The EC collects metrics data for availability, [A] and reliability, [R], including processing times in a synchronised manner from their respective VMs at a defined constant *time duration*, $[t_i, t_{i+1}]$ (i is initialised to zero).
4: The set $[A, R]_N$ is sent to the Auditor using synchronous procedure calls.
5: The Auditor initialises the trust values of the VMs to *0*.

It must be said that other equally effective LibVMI API functions can be used, but we are also cautions since the performance of both the monitoring VM (i.e. vmiMONITOR) and the target VMs are sensitive to the size of request parameters and the resulting return values. The API functions called by "module-list" (such as "vmi-pagetable-lookup-extended()") have fewer parameters and small return values (i.e., primitive variable type) compared to "process-list" such as "vmi_read_pa( )", which returns specified size of physical memory in binary will introduce more significant overhead. This identified API function call is used to

help in observing the times between VMs fail (i.e. obtaining MTBF or MTTF), times for VMs to recover (i.e. obtaining MTTR), and system operation time. That said, we proceed to apply our defined CAM to the CAS test-bed.

With the provided overview for our evidence collection approach supported by **Steps 1**, **2**, **4** and **5**, we synchronise the collection of *operation times* for the respective VMs from both the AWS CloudWatch and LibVMI APIs evidence sources. This activity is undertaken between 11:00 GMT and 14:00 GMT which falls within the range of time considered to be AWS/EC2 peak time in the Ireland availability zone. Such a time period sees a higher level of user requests, processing and scaling of AWS/EC2 resources. We presume that it will be relatively easier to identify dependability issues at this time period. Also, the "trust values" for the VMs are initialised to 0.

15: The method continues in a loop at (2) for time duration,$[t_i, t_{i+1}]$ if none of the assigned VMs in (1) failed.
16: If there is any recorded failure, the method will continue in a loop at (1) for the given time duration, $[t_i, t_{i+1}]$.

During the 3-hour period, the evidence collection process for both evidence sources is repeated every 5 minutes in line with **Steps 15** and **16** of the cloud accountability methodology.

Results After 3 (hr)				
VMI Time (HH:mm)	Introspection Time for Reactive Architecture Instances (ms)			
	vmiGuestFDS	vmiGuestRM	vmiGuestSAR	vmiGuestSET
11:00	358.9100474	370.4532001	362.3901615	368.9284697
11:05	402.4200017	421.3621982	409.8791003	413.6350998
11:10	301.4189565	313.8665493	306.0723782	308.1937303
11:15	297.8903279	317.1003756	308.9685978	312.7643812
11:20	313.9864337	325.7511986	322.1276639	324.7610075
11:25	349.3487084	409.1083746	403.4632698	405.6768527
11:30	419.6762789	479.7667729	452.9875453	467.2397629
11:35	591.4008266	607.9875236	602.6296398	605.7553753
11:40	550.2486899	581.9764272	575.8276272	580.1098338
11:45	512.8007165	529.8762282	522.4200909	527.1009313
11:50	420.9743678	444.1098747	430.8891071	433.8600201
11:55	357.1238735	483.6572761	360.6826887	362.9761101
12:00	373.6189474	469.2702891	371.3915638	375.6569127
12:05	307.2290375	319.3899188	311.9551051	313.9010798
12:10	311.1829513	320.5449531	313.3771778	317.5432301
12:15	499.1203273	93.2335726	518.1200923	511.6776723
12:20	313.6174435	321.5113181	320.1036738	321.1010207
12:25	379.8417021	399.2103796	389.4632142	393.4234735
12:30	357.7100278	388.1646221	359.5232673	371.8884201
12:35	589.1456006	601.9875236	591.3109129	600.3001502
12:40	470.2486899	487.7212702	476.1011893	483.9638371
12:45	412.2973143	430.2900421	425.7507005	497.1998203
12:50	367.4004711	381.9011253	330.1754662	330.6617236
12:55	371.8015578	380.5567106	374.6131891	380.1391556
13:00	334.0648725	351.2902506	347.0417628	348.2182318
13:05	352.2037276	368.6743919	366.9710383	367.6350193
13:10	371.1080564	373.8165412	372.0723787	373.1007302
13:15	300.8907252	311.1003756	303.1600748	307.4321847
13:20	303.8304332	306.2300107	304.2766301	305.6947523
13:25	351.4235882	372.2010567	364.4637121	369.9768528
13:30	357.7100278	368.1646227	359.5238673	361.8904203
13:35	391.1296064	394.9875236	392.7132122	393.7098532
13:40	372.2426843	387.7702764	376.9986124	383.9638323
13:45	382.7983186	364.1764042	362.5103251	367.1020317
13:50	427.7913875	371.1127638	370.7219792	371.6203974
13:55	419.1689932	381.1698247	376.8909574	384.1750054

(a) LibVMI API

Results After 3 (hr)				
AWS Time (HH:mm)	AWS/EC2 API Metrics Collection Time for Reactive Architecture Instances (ms)			
	awsec2FDS	awsec2RM	awsec2SAR	awsec2SET
11:00	308.8097518	321.7528	311.89006	318.828368
11:05	352.3197061	372.6617981	359.3789988	363.5349981
11:10	251.3186609	265.1661492	255.5722767	258.0936286
11:15	247.7900323	268.3999755	258.4684963	262.6642795
11:20	263.8861381	277.0507985	271.6275624	274.6609058
11:25	299.2484128	360.4079745	352.9631683	355.576751
11:30	379.5759833	431.0663728	402.4874438	417.1396612
11:35	541.300531	559.2871235	552.1295383	555.6552736
11:40	504.1483943	533.2760271	525.3275257	530.0097321
11:45	462.7004209	481.1758281	471.9199894	477.0008296
11:50	370.8740722	395.4094746	380.3890056	383.7599184
11:55	337.0235779	434.956876	310.1825872	312.8760084
12:00	351.5186518	420.569889	320.8914623	295.556811
12:05	257.1287419	270.6895187	261.4550036	263.8009781
12:10	261.0826557	271.844553	262.8770763	267.4431284
12:15	466.0200317	84.5331725	467.6199908	461.5775706
12:20	263.5171479	272.810918	269.6035723	271.000919
12:25	329.7414065	350.5099795	338.9631127	343.3233718
12:30	307.6097322	339.464222	309.0231658	321.7883184
12:35	539.045305	483.2871235	540.8108114	550.2000485
12:40	420.1483943	439.0208701	425.6010878	433.8637354
12:45	362.1970187	381.589642	375.250599	447.0997186
12:50	317.3001755	333.2007252	279.6753647	280.5616219
12:55	321.7012622	331.8563105	324.1130876	330.0390539
13:00	283.9645769	302.5898505	295.5416613	298.1181301
13:05	302.895432	319.9739918	316.4709368	317.5349176
13:10	361.1077608	325.1161411	321.5722772	323.0006285
13:15	250.7904296	262.3999755	252.6599733	257.332083
13:20	253.7301376	257.5296106	253.7765286	255.5946506
13:25	301.3232926	323.5006566	313.9636106	319.8767511
13:30	307.6097322	319.4642226	309.0237658	311.7903186
13:35	341.0293108	346.2871235	342.2131107	343.6097515
13:40	322.1423887	339.0698763	328.4985109	333.8637306
13:45	362.698023	355.4760041	312.0102236	347.00193
13:50	418.7910954	322.4123637	320.2218777	321.5202957
13:55	389.0686976	332.4694246	326.3908559	334.6749037

(b) AWS CloudWatch API

Figure 7.16: Processing Times (ms) from the Evidence Sources

7.2.2 Examination

3: The gathered metrics data are sorted and examined for data integrity by the Arbitrator.
6: The Auditor classifies [A_N] and [R_N] for the VMs.

The two sets of collected processing times are first assessed by the Arbitrator for "outliers" in relation to **Step 3**. Here, we consider processing times that span a period of *100 ms* to *650 ms* as the acceptable processing time, and hence any other processing time is classified as an "outlier". However, we identify processing times less than 100 ms as a "failure" of the VM. This failure classification is informed by an earlier assessment of the the minimum latency to each VM, and it was observed to be an average of 119 ms. Also, this decision is complemented by at least one evidence source API check of the VM in consideration.

Following from this, an "outlier" was observed for the vmiGuestRM instance at 12:15 GMT (see Figure 7.16). The LibVMI and AWS CloudWatch API recorded processing times of about *93.23 m*s and *84.53* ms respectively. Since the recorded processing times are below 100 ms, the *Arbitrator* indicates a VM failure. However, the next sets of processing times recorded after 5 minutes shows values of an average of 300 ms. This shows the *vmiGuestRM* instance "recovered" during the said time. Such recovery is made possible by cloud virtualisation features such as snapshot, autoscaling, load balancing, etc. All other processing times are considered to be "acceptable" by the Arbitrator, and hence passed on to the Auditor.

Table 7.5: Classification of Availability and Reliability for the Test-Bed VMs

Instances [S_N]	**Availability**			**Reliability**		
	MTBF/MTTF (hr)	**MTTR** (hr)	[A_N]	**MTBF/MTTF** (hr)	**TIME** (hr)	[R_N]
vmiGuestFDS	3	0	1	3	3	0.36788
vmiGuestRM	1.15	0.30	0.7931	1.15	3	0.07363
vmiGuestSAR	3	0	1	3	3	0.36788
vmiGuestSET	3	0	1	3	3	0.36788

The computation of system availability and reliability is often a laborious and complex process requiring system observation of many months to a year, or even

longer. This period usually records several incidents of failure and recovery from failure, which facilitates a more easily acceptable analysis of system availability and reliability. We are however limited by time to conduct such a detailed process; the amount of data to analyse from the two evidence sources; and by the cost of using the cloud platform over a prolonged period. Our aim for this work is to provide information towards the assurance of cloud platform's availability and reliability. That said, we base our analysis on the data collected during 3 hours of the peak time for the AWS cloud platform, to mainly demonstrate that our methodology can provide relatively more information towards the assurance of cloud dependability to cloud *agents*.

In this context, we proceed to **Step 6**, where the availability, $[A_N]$ and reliability, $[R_N]$ of the Reactive Architecture VMs are derived and classified. A classification of $[A_N]$ and $[R_N]$ values with respect to their corresponding VMs is shown in Table 7.5. Using the *autoscaling* feature for AWS VMs, a *stop-start cycle* for recovering from failure is provided as less than three minutes. So in computing the availability, $[A_{RM}]$ of *vmiGuestRM* after its failure at 12:15 GMT, we consider the *MTTR* as *three minutes* instead of five minutes for our evidence collection frequency.

7.2.3 Analysis

We analyse the examined processing times from the *Evidence Collector* towards meaningfully assuring availability and reliability of deployed cloud-based systems using the presented cloud accountability method. The processing times for the set of VMs from the two evidence sources are compared to identify significant variance(s) that may exist between the two sets of processing times.

7: The *Auditor* compares the sets of processing times from the cloud evidence sources (i.e. VMI, CMS) to understand their relationship based on the SLA (i.e. acceptable metric range, $[AMR]_c^m$) of the cloud platform.
8: If any metric data of the two evidence sources violates the SLA's $[AMR]_c^m$, the Auditor assigns a value of *-1* to the trust value of the related VM, and then triggers a call to the Reactive Middleware (RM).

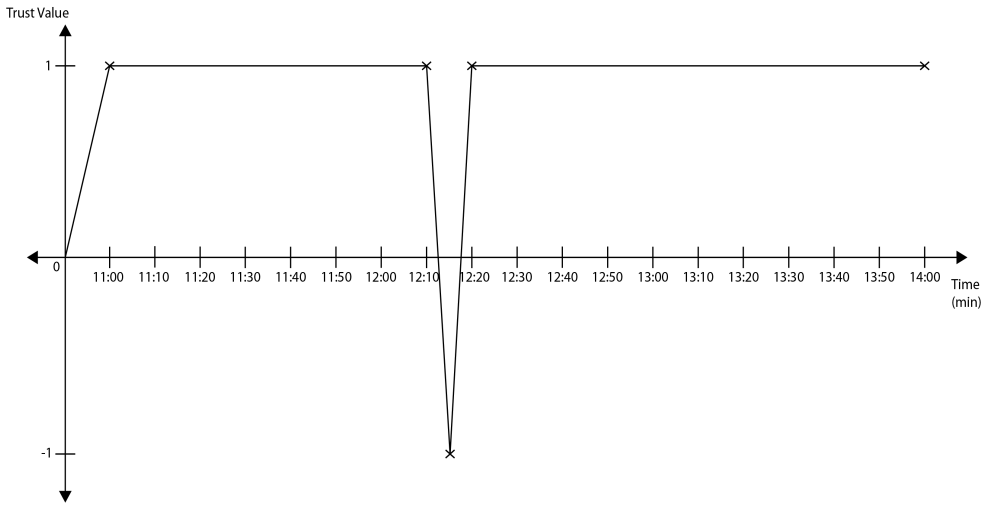


Figure 7.17: Evidence-based Trust Analysis of Reactive Middleware's VM

We also observed that the AWS CloudWatch API collected evidence at an average of *48.25 ms* faster than that of the LibVMI API. We believe that the implementation of the LibVMI API introduces some overhead cost which affects its processing time, and/or it is as a result of efficiency on the part of the AWS CloudWatch API. That said, comparing the failure incident of vmiGuestRM at 12:15 GMT recorded by both the AWS CloudWatch API and LibVMI APIs, with the AWS CloudWatch generated line graph (refer to Figure 7.12) shows a supportive recording. Here, the AWS CloudWatch line graph shows a drop in network data for both *NetworkIn* and *NetworkOut* metrics. However, this

drop in network data does not seem out of place. The oscillation of the network data in the line graph seems to follow a regular pattern. Furthermore, the *StatusCheckFailed*, *StatusCheckFailed_System*, and *StatusCheckFailed_Instance* metrics from the AWS CloudWatch generated line graph indicates no failure for the *vmiGuestRM* VM.

Here, our evidence-based trust analysis which has been assigning trust value of 1 to VMs with no violations to the SLA, and -1 to to VMs with violations, a resulting graph is generated as Figure 7.17. With the recorded failure incident of vmiGuestRM at 12:15 GMT, a trust value of -1 is assigned to this VM. A history of the trust values are made available to cloud agent for their consideration. As mentioned earlier, trust values are assigned to all VMs every five minute cycle to assure real-time monitoring.

11: The RM provides options such as requesting for "Service Credits" from the CSP using the *log* from (**9**).
12: The Auditor also computes the Net Present Value (NPV) of the cloud resources with respect to the identified processing times from (**7**), and/or violations from (**8**).

Since there is no justified reason according to the AWS SLA to request for "Service Credit" compensation, **Step 11** is not undertaken. At this point, the Auditor computes the net present value (NPV) of the Reactive Architecture's VMs as in **Step 12**. The formula for NPV varies slightly depending on the consistency with which returns are generated, and if the investment was a one-time event. Also, since each period generates returns in equal amounts and the observation time is relatively short, the mathematical expression for the NPV in our case is:

$$NPV = \frac{C}{(1+r)^i} - C_0 \tag{7.1}$$

where;

C - the expected cash flow per period (i.e. $ 96.82 per 3 hours for 5 AWS t2.micro VMs including data transfer cost, storage cost, and

VAT. *Exchange Rate*: £77.5602 as $1 to £0.801077 at 30/01/17),

r - the required rate of return. The recommended UK public service discount rate is 3.5%,

T - the number of periods over which the project is expected to generate income (i.e. 3 hours), and

C_0 - the initial investment (no initial investment is made here).

$$NPV = \frac{77.5602}{(1+0.035)^3} - 0 \tag{7.2}$$

$$NPV = £69.9548 \tag{7.3}$$

The positive NPV value from equation (7.3) indicates that the investment in the 5 AWS virtual machines is *desirable* subject to its prevailing *discount rate* over the three-hour period. Since there was no violation to the SLA, the NPV value indicates the value of the cloud investment after the said time.

7.2.4 Reporting

Generally, our examination and analysis of evidence using a wide range of analysis options (i.e. availability and reliability classification, evidence-based trust analysis, and line graphs from two evidence sources), combined with a widely accepted forensic process model, make our analysis considerably trustworthy. Here, our analysis provide relatively more detailed information than the state-of-the-art towards the assurance of dependability (i.e. availability, reliability) of the cloud-based Reactive Architecture.

```
<cas_log>
   <log_item id="user01">
      <user>System Engineer</user>
      <privilege>administrator</privilege>
      <publish_date>2017-01-30</publish_date>
      <duration>3 hours</duration>
      <description>Dependability Assurance of RA's AWS VMs.</description>
   </log_item>
   <log_item id="fail_vm_i-6d16b7ad">
      <name>vmiGuestRM</name>
      <instance_type>t2.micro</instance_type>
      <fail_time>12:15</fail_time>
      <fail_value_vmi>93.23</fail_value_vmi>
      <fail_value_aws>84.53</fail_value_aws>
      <avail>0.7931</avail>
      <rel>0.07363</rel>
      <description>Reactive Middleware VM fails ones.</description>
   </log_item>
   <log_item id="sys_analysis">
      <ci>95%</ci>
      <alpha>0.05</alpha>
      <pop_variance>reject</pop_variance>
      <sla_violation>no</sla_violation>
      <cash_flow>$96.82</cash_flow>
      <disc_rate>3.5%</disc_rate>
      <npv>£69.9548</npv>
      <npv_decision>desirable</npv_decision>
      <description>No AWS SLA violation but one failure</description>
   </log_item>
</cas_log>
```

Figure 7.18: CAS XML Log for Cloud Agents

9: All activities including VM users, duration of VMs observation, date, details of failed VMs, analysis of failure, etc. are *logged*.
10: The RM then sends a notification to the cloud agents when all VMs are "unavailable".
13: If there was no violation from (**8**), the Auditor assigns *1* to the trust value of the related VM, an analysis of the comparison in (**7**), and the NPV from (**12**) are saved as a *log*, $[l_N]$.
14: This log, $[l_N]$ is sent as a notification to the cloud agents as periodic reports (e.g. monthly, yearly, etc.).

The cloud agents are provided with a *log* of the analysis (i.e. comparison of the processing times, and the NPV). Refer to Figure 7.18 for an XML presentation

of the mentioned log. This log, together with the figures (including the evidence-based trust graph - Figure 7.17) and tables generated in the forensic process is provided to the cloud agents for assuring dependability of AWS resources. This activity is in line with **Steps 9**, **10**, **13**, and **14**.

To compare our analysis to a relevant benchmarking of the performance of cloud infrastructure, we realised that the closest work is the Standard Performance Evaluation Corporation (SPEC)'s Cloud IaaS 2016 Benchmark (see Section 2.3.3.2). SPEC's key benchmarking metrics are scalability, elasticity, and "mean instance provisioning time", and this performance evaluation is conducted over a period of "forty-six minutes". This evaluation period mentioned justifies our 3-hour period as a suitable time interval to observe cloud-based resources for analysis. That said, in a survey reported by [69] of the 40 largest CSPs, an average cloud service *availability* in 2010 was 99.948%, equivalent to 273 minutes of downtime per year. AWS EC2 (99.95% yearly) met their SLA but their S3 service (99.9% monthly) fell short. Based on the "downtime per week" availability from the "Nines of Availability" (refer to Figure 2.11), our 3-hour AWS EC2 peak time recording which was 94.828% (refer to Figure 7.5), has 2.8 hours of weekly downtime. This indicates about 99.0% availability (i.e. 2 nines) with a potential 3.65 days downtime per year. The reliability during the said time is 0.2943175.

7.2.5 Summary

We undertake our evaluation by implementing a prototype of the cloud accountability system (CAS) to facilitate the active monitoring of the Reactive Architecture (RA) components. Our evaluation is performed using a cloud-based test-bed in the Amazon Web Service (AWS) Elastic Cloud Compute (EC2) environment comprising 5 Linux VMs over a period of 3 hours. The 5 instances are classified into two main VMs; the "target VMs" of the RA components and the "monitoring VM" of the CAS. The Forensic Process Model (NIST SP800-86) is used to guide the collection, examination, analysis, and reporting of dependability metrics as digital evidence to assure dependability of the

AWS cloud environment. Here, we identified the failure of vmiGuestRM VM at 12:15 GMT that was not reported by the CSP to the cloud agents especially to the system engineers. The CAS reports relevant cloud related activities such as failures, conducted availability and reliability computations, evidence-based trust analysis, the net present value of cloud resource investment, and a log supported by graphical representations of our analysis. These logs can be used as: (1) evidence to claim compensation (i.e. "Service Credits"), and also (2) serve as a source of data for predicting dependability violations using a form of machine learning. With these in place, we are convinced that our *Objective 11* and *12* of Section 1.2.2 have been met, and that the cloud accountability method is sufficiently capable of being used to meaningfully assure availability and reliability of deployed systems in the cloud.

7.3 Reactive Architecture

In this section, cloud-ATAM is presented as a method for analysing and evaluating the trade-off of quality attributes for small-to-medium size cloud-based systems. The novelty of this method is identified and validated using a comparative study. These are in line with *Objectives 6* and *8* in Section 1.2.2. The cloud-ATAM has been used to design the Reactive Architecture based on the performance and availability quality attributes in Chapter 3. Here, we analyse and evaluate the Reactive Architecture.

This work focuses on analysing the Reactive Architecture using the defined two-staged approach (see Figure 7.19) of the cloud-ATAM. This approach is (1) stakeholder-centric, (2) elicits points of view from a more diverse and larger group of stakeholders, and (3) verifies and then builds on the results of the architecture design in Chapter 3.

Here, the cloud-ATAM uses a non-trivial set of scenarios to analyse the cloud-based architecture. A final report of the analysis results include a summary of the project drivers, the architectural approaches, a utility tree, the analysis of each chosen scenario, and important conclusions drawn. All these results

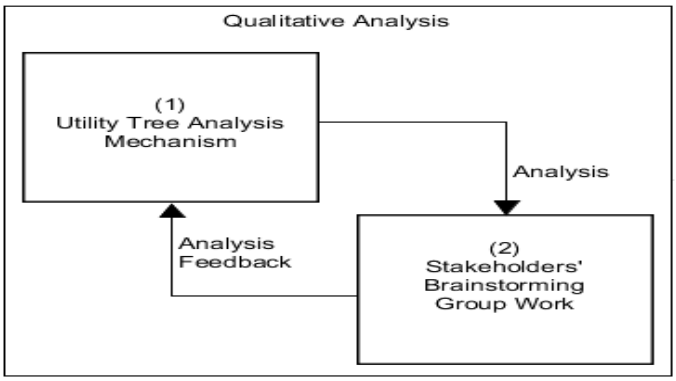


Figure 7.19: cloud-ATAM Two-Staged Evaluation Approach

are recorded visually, so stakeholders can verify the correct identification of the results. cloud-ATAM presents an enrichment in terms of coverage to ATAM in the form of the two-staged scenario-based analysis approach. The research question asked for this section is "what is the trade-off between availability and performance quality attributes identified by the cloud-ATAM for the cloud-based Reactive Architecture?"

The comparative study to validate cloud-ATAM is presented in Section 7.3.1. Also, the mentioned two-staged analysis approach defined in cloud-ATAM is respectively discussed in the following sections of the chapter: Section 7.3.2 and Section 7.3.3.

7.3.1 Comparative Study

The Architecture Trade-off Analysis Method (ATAM) has been presented as a very effective state-of-the-art architecture analysis method. This was established in a comparative study in Section 2.2.5.1. cloud-ATAM is derived from from ATAM. We argue that even though ATAM is very relevant as a method, it is overly general in its application, and hence ideal for analysing large architectures owned by large organisations. Here, its relevance and effectiveness may be challenged in analysing architectures of a smaller size, varying operational environments of architectures, number of stakeholders available, budget

constraints, etc.

To get a better appreciation of the differences, we conduct a comparative study of cloud-ATAM and ATAM software system architecture evaluation methods, based on a defined set of criteria. Considering the rapidly evolving nature of the cloud platform as a currently popular and suitable deployment environment for most software systems, we provide the main criteria for our comparative study as:

(a) **A goal of sensitivity and trade-off analysis**:- In a rapidly evolving environment, an efficient evaluation method should be capable of identifying the quality attributes that change relative to others. Also, it must be able to inform architectural decisions regarding an acceptable trade-off between quality attributes.

(b) **A focus on multiple quality attributes**:- In the described environment above, typically there will be multiple competing quality attributes. It is, however, relevant that an appropriate evaluation method can consider varying quality attributes.

(c) **A focus on applicable architecture size**:- In the cloud environment, there are several elements that are changing rapidly. This therefore necessitates that at least the size of the architecture being analysed be known and catered for. Also, it affords the methodology the opportunity to provide specialised features to that type of architecture. Such features can be the support of an ideal but specific number of stakeholders to undertake a set of activities.

(d) **Nature of method to guide design and analysis**:- An architecture analysis method should not be limited to post design activities, but present itself in a way as to facilitate design. This makes such a method useful and versatile for both design and analysis of cloud-based architectures.

(e) **Relevance of method for cloud environment in terms of complexity**:- Architecture analysis methods must have reasonably shorter processes and activities. The cloud environment hosts mainly small-to-medium size

architectures and are deployed by small organisations with fairly limited resources. A complex method requiring big budgets and many stakeholders may not be attractive.

(f) **Tool support**:- To reduce the introduction of errors by stakeholders during analysis, some form of automation or tool support will be beneficial.

Table 7.6: Comparative Study of ATAM and cloud-ATAM

Criteria	**ATAM**	**cloud-ATAM**
(a) Method goal	Sensitivity and trade-off analysis	Sensitivity and trade-off analysis
(b) Quality attributes	Multiple	Multiple
(c) Applicable architecture size	Any	Small-to-medium
(d) Nature of method	Architectural analysis	Architectural design and analysis
(e) Relevance of method for cloud environment	Complex and generic, hence well suited to large architectures	Fewer activities with good support especially for small-to-medium size architectures
(f) Tool support	No	No, but Reactive Architecture can be extended to support the method

7.3.1.1 Discussion of Comparative Study Results

The comparative study has been provided in Table 7.6. In this study, there are the same or similar features in terms of criteria (a), (b) and (f). This because both methods are fundamentally (i.e.Method goal, Quality attributes, and Tool support) the same. A slight distinction regarding criteria (f) (i.e.Tool support) is that cloud-ATAM's facilitating framework - Reactive Architecture - can be

extended to provide tool support. This framework supports cloud-based system engineering.

The distinguishing set of criteria can be identified as (c), (d) and (e). For criteria (c): (Applicable architecture size), ATAM is presented as generic method that can be used to analyse any size of architecture. Here, the nine-step method with several activities has been daunting for analysing especially small architectures. On the other hand, cloud-ATAM specialises on small-to-medium size architecture. Most architectures in the world fall under this classification. That said, even though this focus is narrow, its application has a wider potential. Criteria (d) looks at the nature of the methods with regards to their applicability to architectures in system engineering projects. As discussed earlier, ATAM is a popular method for architecture analysis. In this light, ATAM is limited to only analysing architectures whiles cloud-ATAM is used to design and analyse architectures. Finally, for criteria (e): (Relevance of method for cloud environment), ATAM is shown to be relatively complex and generic. ATAM is relatively complex and typically requires large organisations with large resources and experts to effectively conduct its analysis. This may not be particularly appealing to a greater number small and medium size organisations which are increasingly dominating the cloud environment. That said, cloud-ATAM presents relatively fewer steps and activities which does not compromise its effectiveness for design and analysis of cloud-based architectures.

The following section provides qualitative analysis of the Reactive Architecture using the utility tree analysis mechanism.

7.3.2 Utility Tree Analysis Mechanism

As discussed earlier in Chapter 3, the utility tree analysis mechanism is presented as a top-down mechanism for directly and efficiently translating the business drivers of a system into concrete quality attribute scenarios. For example, in an e-commerce system two of the business drivers might be stated as: “security is central to the success of the system since ensuring the privacy of our

customers' data is of utmost importance"; and "modifiability is central to the success of system since we need to be able to respond quickly to a rapidly evolving and very competitive marketplace." Before we can assess the architecture, these system goals must be made more specific and more concrete. Moreover, we need to understand the relative importance of these goals versus other quality attribute goals, to determine where we should focus our attention during the architecture evaluation. Utility trees help to prioritise quality goals.

Quality goals of a system are often presented as a set of system descriptions to facilitate modular system analysis. Typically the first job of an architecture analysis is to precisely elicit the specific quality goals against which the architecture will be judged. The mechanism that we use for this elicitation is the "scenario". Scenarios are applied not only to determine if the architecture meets a functional requirement, but also for further understanding of the system's architectural approaches and the ways in which these approaches meet the quality requirements such as performance, availability, modifiability, and so forth. They represent specific examples of current and future uses of a system. They are useful in understanding both vague development-time qualities (e.g. modifiability) and run-time qualities (e.g. performance, availability). In terms of run-time qualities, scenarios specify the kinds of operations over which performance needs to be measured, or the kinds of failures the system will have to withstand. The utility tree generated in this exercise with a set of scenarios based on the Reactive Architecture is shown in Figure 5.5.

The presented utility tree guides the remaining analysis process. It is important at this point to prioritise, and refine the Reactive Architecture's most important quality attribute goals. The utility tree presented starts with "Utility" as the "root node". This indicates the general "goodness" of the Reactive Architecture. The "second level" is constituted with the quality attributes of interest: "performance" and "availability". In the "third level", there are specific quality attribute refinements. From the "performance" quality attribute, we identify "data latency" and "transaction throughput" as relevant refinements. Such refinements are major determinants of performance. Also, "availability" is refined to "hardware failures" and "software failures". From this point, we are

Table 7.7: Prioritised Quality Attribute Scenarios

Quality Attribute Scenarios	**Scenario ID**	**Relative Ranking [X,Y]**	**Numbered Value [X,Y]**
Power outage at Availability Zone 1 requires traffic redirect to Availability Zone 2 in less than 5 seconds	A1	[L, M]	**[3, 2]**
Disk crash must have a backup that takes over in less than 3 seconds	A2	[H, L]	**[1, 3]**
Network failure is detected and recovered in 10 seconds	A3	[M, L]	**[2, 3]**
COTS/Third party software update with bug that causes failures is reverted to stable version in less than 5 seconds	A4	[M, M]	**[2, 2]**
Deliver change requests and reports in real-time	P1	[H, M]	**[1, 2]**
Reduce storage latency for users to 200 milliseconds	P2	[H, L]	**[1, 3]**
One system (e.g. Reactive Middleware) should not constitute a lag greater than 1 second	P3	[M, L]	**[2, 3]**
Accommodate over 500 queries per second	P4	[H, M]	**[1, 2]**

able to identify attribute goals as "quality attribute scenarios" that are concrete enough for "prioritisation" and "analysis". These "quality attribute scenarios" form the "leafs" of the utility tree. Here, the "hardware failures" (i.e. see "third level") refined from "availability" is further refined into "power outage at Availability Zone 1 requires traffic redirect to Availability Zone 2 in less than 5 seconds", "disk crash must have a backup that takes over in less than 3 seconds", and "network failure is detected and recovered in 10 seconds". These constitute specific "scenarios" that can be prioritised relative to each other and

Table 7.8: Prioritised Quality Attribute Scenarios. Ordered based on the importance of each scenario to the success of the Reactive Architecture **(X)**

No.	**Quality Attribute Scenarios**	**Scenario ID**	**Numbered Value**
1	Disk crash must have a backup that takes over in less than 3 seconds	A2	**1**
2	Deliver change requests and reports in real-time	P1	**1**
3	Reduce storage latency for users to 200 milliseconds	P2	**1**
4	Accommodate over 500 queries per second	P4	**1**
5	Network failure is detected and recovered in 10 seconds	A3	**2**
6	COTS/Third party software update with bug that causes failures is reverted to stable version in less than 5 seconds	A4	**2**
7	One system (e.g. Reactive Middleware) should not constitute a lag greater than 1 second	P3	**2**
8	Power outage at Availability Zone 1 requires traffic redirect to Availability Zone 2 in less than 5 seconds	A1	**3**

also analysed.

The utility tree is prioritised based on "the importance of each scenario to the success of the Reactive Architecture **(X)**" and "the degree of perceived risk posed by the achievement of this node **(Y)**" (i.e.how easy the architecture teams feel this level of performance or availability will be to achieve). To facilitate the prioritisation process, we apply a relative rankings approach such as High (H), Medium (M), and Low (L). These will be assigned to the scenarios as a pair (i.e. [**H,M**]) to represent a "high" importance in terms of **(X)**, and medium level of perceived risk in terms of **(Y)**). Furthermore, we assign a numbered value such as (H) corresponds to 1, (M) corresponds to 2, and (L) corresponds to 3 (see

Table 7.9: Classified Quality Attribute Scenarios according to Types

Scenario Type	**Scenario Type ID**	**Quality Attribute Scenarios**	**Scenario ID**
Use Case	*USC1*	Deliver change requests and reports in real-time.	P1
	USC2	Reduce storage latency for users to 200 milliseconds.	P2
Growth	*GS1*	Disk crash must have a backup that takes over in less than 3 seconds.	A2
	GS2	Network failure is detected and recovered in 10 seconds.	A3
	GS3	COTS/Third party software update with bug that causes failures is reverted to stable version in less than 5 seconds.	A4
	GS4	One system (e.g. Reactive Middleware) should not constitute a lag greater than 1 second.	P3
	GS5	Accommodate over 500 queries per second.	P4
Exploratory	*ES1*	Power outage at Availability Zone 1 requires traffic redirect to Availability Zone 2 in less than 5 seconds.	A1

Table 7.7). This is relevant in making the ordering process easy by arranging the numbered values in an ascending order. With this method, the high priority quality attribute scenarios will be arranged from the top of the list or table to the low priority at the base. That said, we proceed with the analysis of the quality attribute scenarios based on the relative rankings of the importance of each scenario to the success of the Reactive Architecture **(X)** (see Table 7.8).

In *cloud-ATAM*, we use three types of scenarios: use case scenarios (these involve typical uses of the existing system and are used for information elici-

Table 7.10: Prioritised Quality Attribute Scenarios (Ordered)

No.	Quality Attribute Scenarios	Scenario ID	Numbered Value
1	Disk (i.e. data repository) crash must have a back-up that takes over in less than 3 seconds	A2	1
2	**Deliver change requests and reports in real-time**	**P1**	**1**
3	Reduce storage latency for users to 200 milliseconds	P2	**1**
4	Accommodate over 500 queries per second	P4	**1**
5	Network failure is detected and recovered in 10 seconds	A3	**2**
6	COTS/Third party software update with bug that causes failures is reverted to stable version in less than 5 seconds	A4	**2**
7	One system (e.g. Reactive Middleware) should not constitute a lag greater than 1 second	P3	**2**
8	Power outage at *Availability Zone 1** requires traffic redirect to *Availability Zone 2** in less than 5 seconds	A1	**3**

tation); growth scenarios (these cover anticipated changes to the system), and exploratory scenarios (these cover extreme changes that are expected to "stress" the system). These different types of scenarios are used to probe a system from different angles, optimising the chances of surfacing architectural decisions at risk. In this work, we consider some scenarios bordering "use case", "growth" and "exploration". We have provided an introduction to some "use case scenarios" in Appendix C.14. The scenarios are classified and shown in Table 7.9.

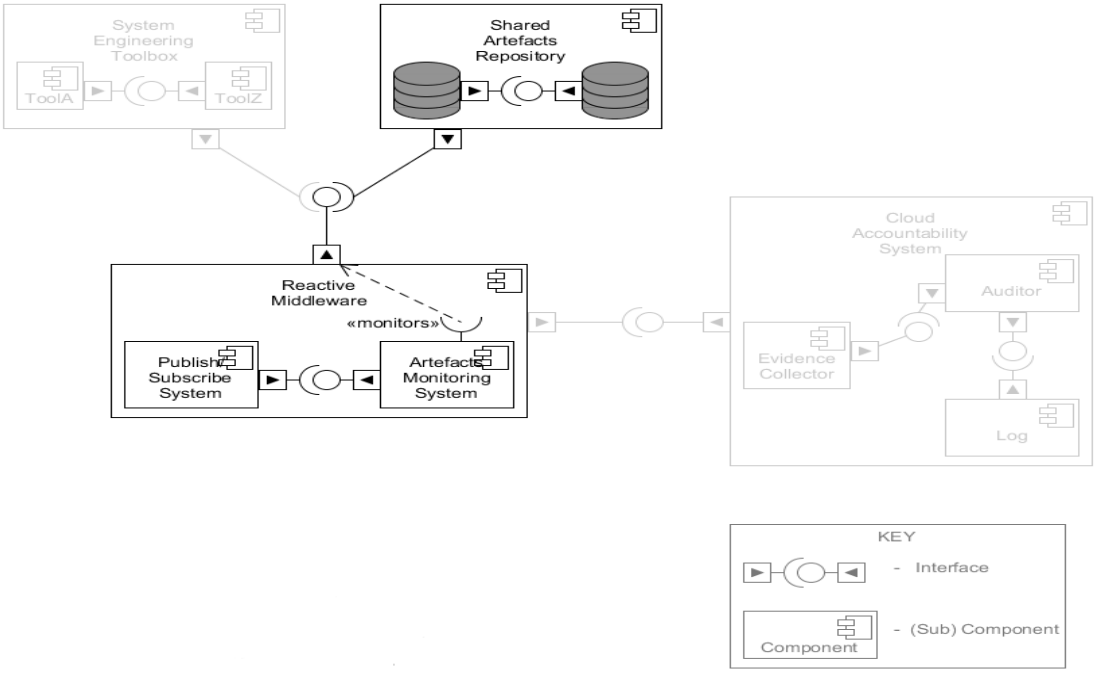


Figure 7.20: Component and Connector View of Reactive Architecture for Scenario (P1) Analysis

Table 7.11: Analysis of Sensitivities, Trade-offs, Risks & Non-Risks for the Utility Tree

Sensitivities:	* **S1:** Concern over network latency. * **S2:** Using a data-centric and client-server approach for the central repository can facilitate data integrity and consistency, but it makes the architecture sensitive to its faults and bottlenecks. * **S3:** Similarly, the central role played by the Reactive Middleware makes the architecture sensitive to faults, resource (i.e. CPU, memory) malfunctions or unavailability.
Trade-offs:	*** T1:** Availability (+) vrs Performance (-) vrs Reliability (-): defining a central artefacts repository makes artefacts readily available, but may be faced with bottlenecks when there are a burst of queries on the repository. *** T2:** Availability (+) vrs Performance (+): using APIs for component interfaces facilitate readily access to resources, and boosts performance. *** T3:** Availability (+) vrs Performance (-): client-server approach for the Reactive Middleware allows for multi-client service, but there can be an overwhelming network management performance constraint. *** T4:** Availability (+) vrs Performance (-) vrs Reliability (+): backing up the artefacts in the primary Shared Artefacts Repository allows for fail-over assurance and increased reliability, but the asynchronous back-up process can affect performance.
Risks:	*** R1:** Data integrity. *** R2:** The risk is that the Reactive Middleware and the Shared Artefacts Repository constitute a single point of failure.
Non-Risks:	*** N1:** The non-risk is the use of application programming interface (API) approach which should stay compatible. *** N2:** The independent communication connections should enable real-time data transfer.

Table 7.12: Analysis of Performance Scenario - **P1** - (see Table 7.11 for the description of S1, S2, T1, etc.) and (**C&C + API**: Component-and-connector architectural style and API, **SAR**: Shared Artefacts Repository, **RM**: Reactive Middleware, and **ICC**: Independent Communication Components)

Analysis of Architectural Approach using a Performance-related Scenario				
Scenario ID : *P1*	**Scenario:** *Deliver change requests and reports in real-time*			
Attribute(s)	*Performance*			
Environment	*Normal Operations*			
Stimulus	*Responsiveness to change events*			
Response	*real-time*			
Architectural Decisions	**Sensitivity**	**Trade-off**	**Risk**	**Non-Risk**
AD1 C&C + API	S1			N1
AD2				
AD3 Client-Server SAR	S2	**T1**	R1, R2	N1
AD4 Client-Server RM	S3	**T3**	R2	N1
AD5 Back-up	S1,S2	**T4**		N1
AD6 DS RM	S1		R1	N1
AD7 Schema-free-SAR			R2	
AD8 ICC	S1			N2

7.3.2.1 Analysis Process

The analysis process begins by considering a *high priority scenario* of the Reactive Architecture. From Table 7.8, all generated scenarios of the Reactive Architecture are identified and ordered based on their importance to the success of the architecture. These scenarios are considered to be of high priority in the analysis of the architecture. At this point, we consider **scenario P1** (i.e."Deliver change requests and reports in real-time" - see Table 7.10) for further analysis towards the identification of sensitivity points (i.e. trade-off and risk points).

Considering scenario P1, we first identify the related components of the Reactive Architecture. The interactions of such components will help us to identify and understand the sensitivity points. Now from scenario P1, we notice "*change requests and reports*" which constitutes the main feature of both the Reactive Middleware and the Shared Artefacts Repository. The Reactive Middleware provides facilities to stakeholders to submit "change request" affecting "high priority system artefacts" for consideration by change managers. Such artefacts are stored in the Shared Artefacts Repository. Also, the decision made by the change managers is delivered to the stakeholder as a report. Hence, the interaction between the Reactive Middleware and the Shared Artefacts Repository is relevant in our analysis at this point (see Figure 7.20).

From the interactions of the mentioned components, some sensitivity points (i.e. S1, S2, S3) were identified (see Table 7.11). A further assessment of these sensitivity points introduced trade-off points (i.e. T1, T2, T3, T4), and risk points (i.e. R1, R2). Also, some Non-Risk points (i.e. N1, N2) were identified. However, a Non-Risk point (i.e. N1) mitigates the trade-off point (i.e. T2) involving application programming interfaces (APIs). Here, no risks were identified to make the trade-off (i.e. T2) necessary for further consideration. Finally, an overview of our analysis is provided in Table 7.12.

After the analysis of the Reactive Architecture with the Utility Tree Analysis Mechanism, we continue the analysis process by soliciting the opinion of the system stakeholders on this analysis approach in the next section.

7.3.3 Stakeholders' Brainstorming Analysis Mechanism

At this point, the cloud-ATAM as well as the analysis conducted with the utility tree analysis mechanism are presented to the stakeholders, to brainstorm and provide their feedback on them. A stakeholders' brainstorming group work is organised between the team of system analysts (i.e. designers and analysts of the Reactive Architecture) and the relevant stakeholders (i.e. owners, users, software testers, database administrators, legal team, auditors, etc. of the Reactive Architecture).

7.3.3.1 Stakeholders' Brainstorming Group Work

In the stakeholders' group work, stakeholders numbering up to twenty-on (21) were specifically introduced to the Architecture Trade-off Analysis Methodology (ATAM) as the parent methodology, the cloud-based Reactive Architecture, and our utility tree analysis with the derived methodology - cloud-ATAM. To these three sections, stakeholders were asked to provide some answers to some questions. Here, a "questionnaire" is designed to facilitate the data gathering from the feedback of the stakeholders (see Appendix C.15).

We considered the following questions:

- **SECTION 1**: Presentation of ATAM
  (a) The overview of architecture evaluation and ATAM were presented reasonably well?
  (b) Quality attributes (i.e. availability, performance, etc.) play a critical role in architecture evaluation?
  (c) Scenario-based architecture evaluation methods are adequate in analysing software architecture of varying sizes?
  (d) ATAM presents a matured/convincing approach to architectural evaluation?
- **SECTION 2**: Reactive Architecture

(a) The Reactive Architecture was clearly presented?

(b) The requirements of the Reactive Architecture are representative enough?

(c) The presented constraints and focal quality attributes are relevant to the architecture?

(d) The architecture components, their relationships, and initial scenarios are useful in understanding the Reactive Architecture?

- **SECTION 3**: cloud-ATAM Evaluation and Results

(a) The cloud-ATAM was clearly presented?

(b) The quality attribute characterisation was well presented?

(c) The attribute-specific questions and the identified approaches provide adequate coverage of the quality attribute characterisation?

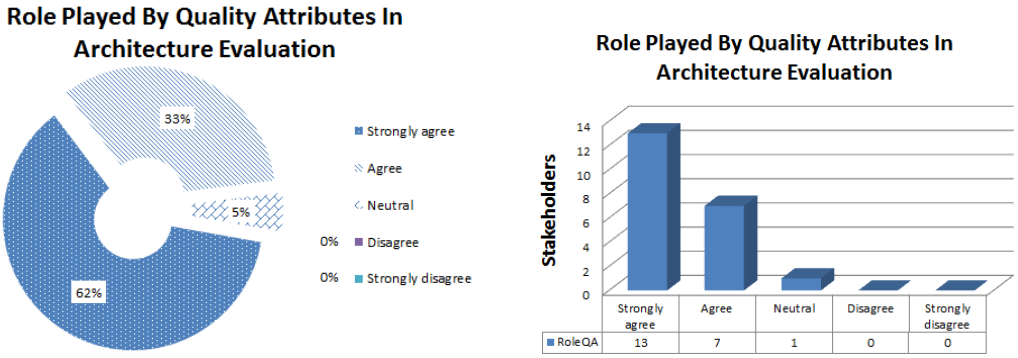
(d) The presented scenario (generated from the utility tree) was adequately analysed?

(e) The reasoning behind the analysis process was sound?

Stakeholders were encouraged to provide their answers in the range of the classification: (1) "Strongly agree", (2) "Agree", (3) "Neutral", (4) "Disagree", and (5) "Strongly disagree". However, some questions (such as questions 1 of both *Section 1*, and *Section 2*) only required either a "Yes" or "No" answer. They were provided the option to comment about each of the three sections of the questionnaire, and also to provide general comments.

7.3.3.2 Analysis of Stakeholders' Feedback

In our approach to analyse the feedback from the stakeholders, we consider each of the questions and their corresponding responses. We begin by taking a look at **question 1** from *Section 1*. Question 1 seeks to understand if *the overview of architecture evaluation and ATAM were presented reasonably well*. There was a *unanimous agreement* that the mentioned topics were reasonably well presented. This however provides a good basis for an adequate representation of the perception of the stakeholders.



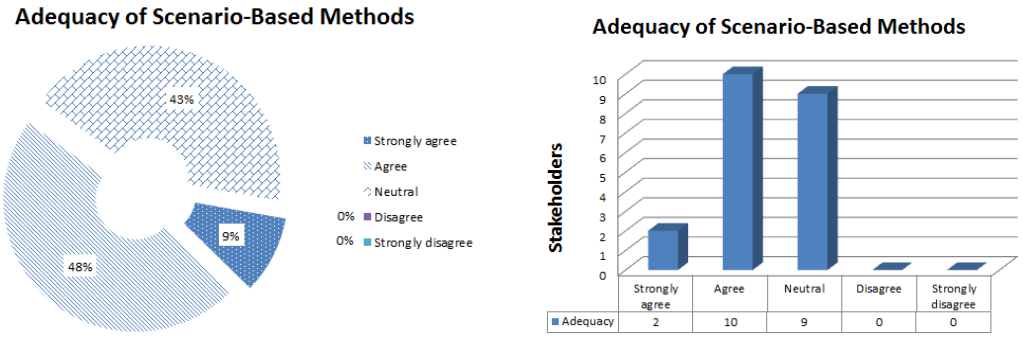
(a) Pie Chart (b) Bar Chart

Figure 7.21: The Role Played by the Quality Attributes in Architecture Evaluation

We proceed to **question 2** of *Section 1*, which seeks to identify if in the opinion of the stakeholders, *quality attributes play a critical role in architecture evaluation*. The feedback is shown graphically in Figure 7.21. This figure specifically shows a *pie chart* depicting a *percentage* distribution of the feedback, and a *bar chat* also showing the *frequency* of the feedback of stakeholders relative to this question. From Figure 7.21, *13* of the stakeholders (representing *62%*) indicated *strongly in agreement* that quality attributes play a critical role in architecture evaluation. Also, *7* of the stakeholders (representing *33%*) indicated that they *agree*, whiles *1* stakeholder (representing *5%*) was *neutral*. Here, *none* of the stakeholders either *disagreed* or *disagreed strongly* to our claim.

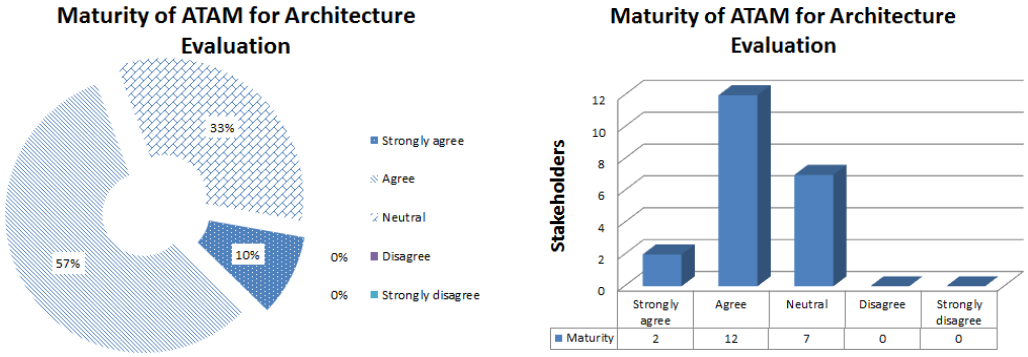
The responses to **question 3** (i.e. *adequacy of Scenario-Based Methods for Architecture Evaluation?*) of *Section 1* was generally positive (see Figure 7.22). Here, *2* of the stakeholders (representing *9%*) responded *strongly in agreement*, and *10* stakeholders (representing *48%*) *agree* to our claim. However, the remaining *9* stakeholders (representing *43%*) were *neutral* in their response to this question.

With **question 4** of *Section 1*, we seek to solicit the opinion of the stakeholders if "*ATAM presents a matured/convincing approach to architectural evaluation?*" (see Figure 7.23). We identified that *2* of the stakeholders representing *10%* *strongly agreed* to this claim, whiles *12* stakeholders representing *57%* *agree*.



(a) Pie Chart (b) Bar Chart

Figure 7.22: Adequacy of Scenario-Based Methods for Architecture Evaluation

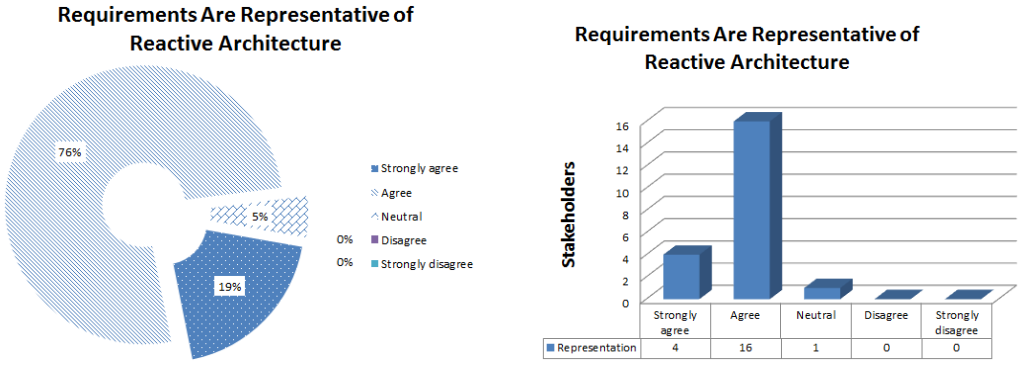


(a) Pie Chart (b) Bar Chart

Figure 7.23: Maturity of ATAM for Architecture Evaluation

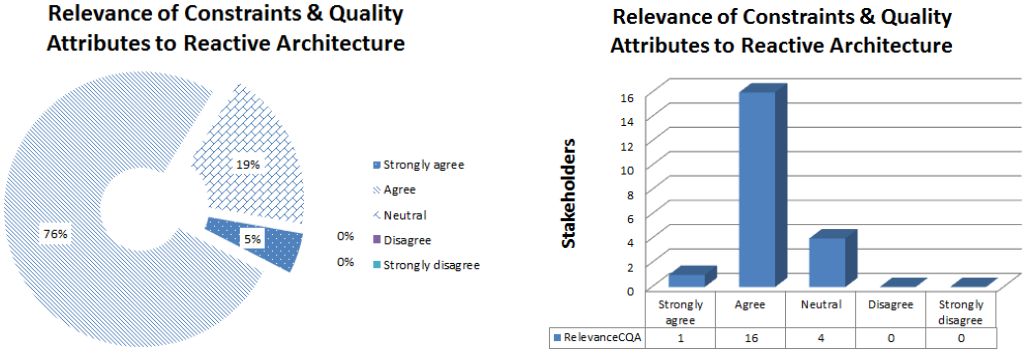
However, *7* stakeholders were *neutral*, and no one *disagreed* to this claim.

Considering the Reactive Architecture being analysed by **cloud-ATAM** in *Section 2*, we received very positive responses. With regards to: **question 1** - "*The Reactive Architecture was clearly presented?*", **question 2** - "*The requirements of the Reactive Architecture are representative enough?*", **question 3** - "*The presented constraints and focal quality attributes are relevant to the architecture?*", and **question 4** - "*The architecture components, their relationships, and initial scenarios are useful in understanding the Reactive Architecture?*" the responses are categorised respectively as:



(a) Pie Chart (b) Bar Chart

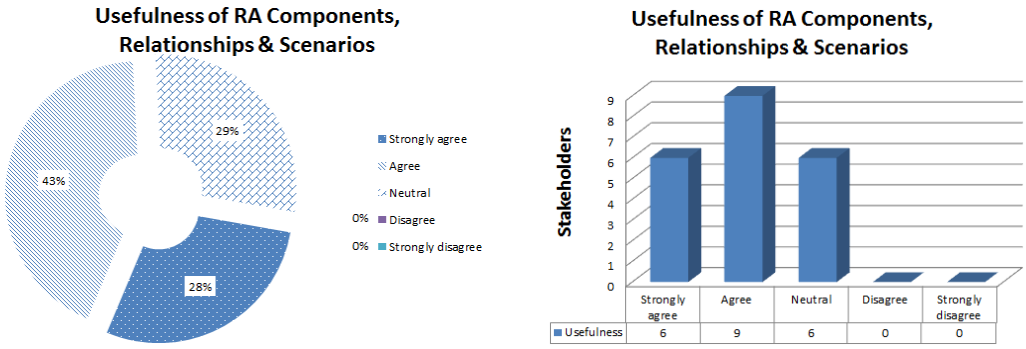
Figure 7.24: Representativeness of the Reactive Architecture's Requirements



(a) Pie Chart (b) Bar Chart

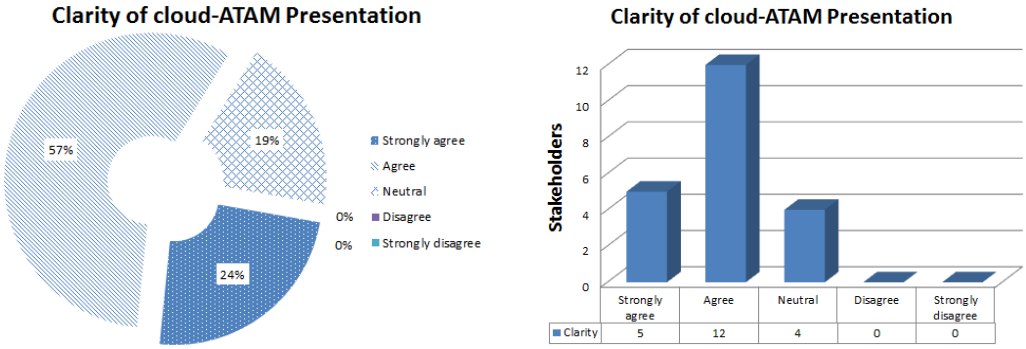
Figure 7.25: Relevance of the Reactive Architecture's Constraints and Quality Attributes

- *21 representing 100%* of the stakeholders thought that the Reactive Architecture was clearly presented,
- Strongly agree: *4 representing 19%*, Agree: *16 representing 76%*, Neutral: *1 representing 5%* (see Figure 7.24),
- Strongly agree: *1 representing 5%*, Agree: *16 representing 76%*, Neutral: *4 representing 19%* (see Figure 7.25),
- Strongly agree: *6 representing 28%*, Agree: *9 representing 43%*, Neutral: *6 representing 29%* (see Figure 7.26).



(a) Pie Chart (b) Bar Chart

Figure 7.26: Usefulness of Reactive Architecture's Components, Relationships and Scenarios

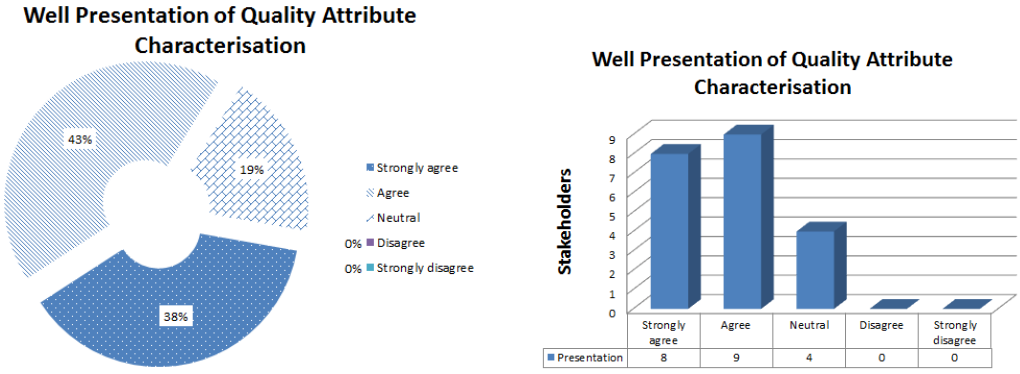


(a) Pie Chart (b) Bar Chart

Figure 7.27: Clarity of the Presentation of *cloud-ATAM*

In the final section, we seek to present cloud-ATAM, our evaluation and results, and then assess how the stakeholders appreciate our approach. Here, we identified that 24% (i.e. 5) of the stakeholders *strongly agree* that the presentation of the cloud-ATAM was clear (see Figure 7.27). Also to this question (i.e. **question 1**), *57%* (i.e. *12*) of the stakeholders *agree*. However, the remaining *29%* (i.e. *4*) were *neutral* in their response.

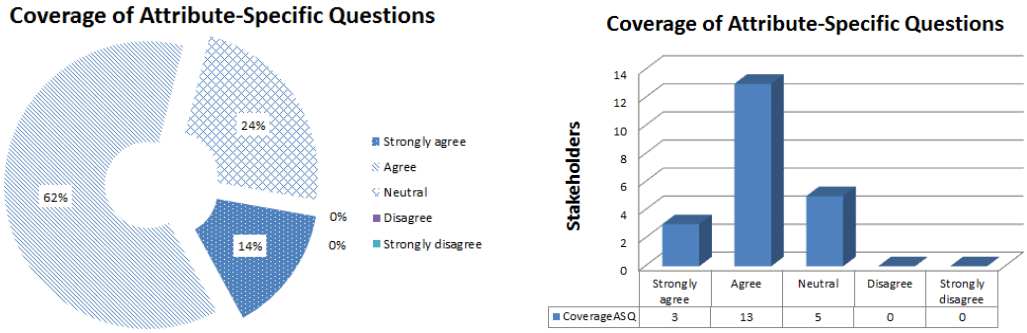
The second question in *Section 3* looks at how *well the presentation of quality attribute characterisation* was. The characterisation of the quality attributes (i.e. availability and performance), are very relevant and serves as the foundation



(a) Pie Chart (b) Bar Chart

Figure 7.28: Well Presentation of Quality Attribute Characterisation

of the analysis with cloud-ATAM. From Figure 7.28, we identify that *38%* (i.e. 8) of the stakeholders *strongly agree* and 43% (i.e. 9) of the stakeholders *agree* that the presentation went well. That said, the remaining *19%* (i.e. 4) of the stakeholders were *neutral*.

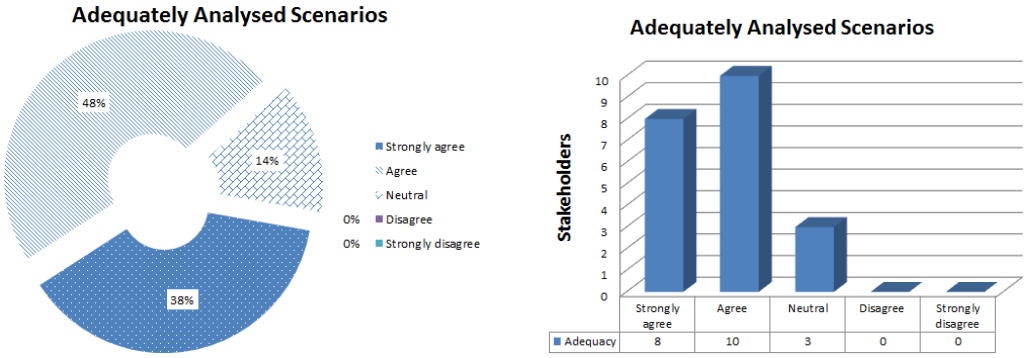


(a) Pie Chart (b) Bar Chart

Figure 7.29: Coverage of Attribute-Specific Questions

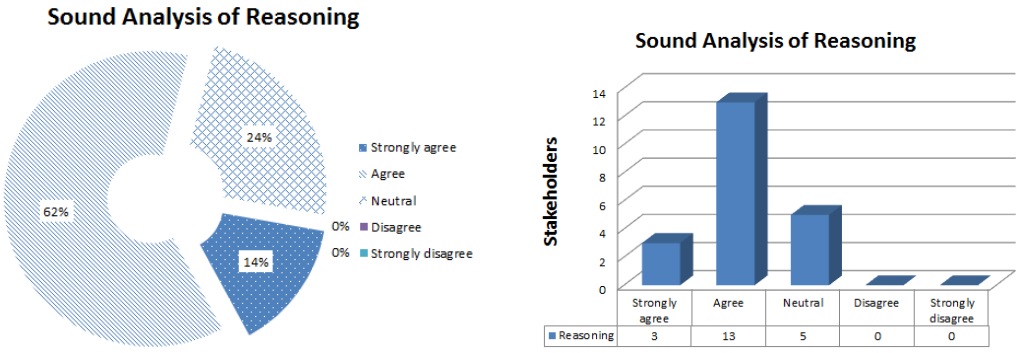
Focusing on the analysis of the Reactive Architecture with cloud-ATAM, we begin by building on the quality attribute characterisation by assessing the *coverage of attribute-specific questions*. From our assessment (see Figure 7.29), *14%* (i.e. *3*) of the stakeholders *strongly agree* whiles *62%* (i.e. *13*). Also, *24%* (i.e. *5*) of the stakeholders are *neutral*.

From the *attribute-specific questions* asked, we are able to understand the



(a) Pie Chart (b) Bar Chart

Figure 7.30: Adequately Analysed Scenarios of Reactive Architecture



(a) Pie Chart (b) Bar Chart

Figure 7.31: Sound Reasoning of *cloud-ATAM* Analysis

how the identified *quality attributes* will be achieved using the *architectural approaches* or *decisions* for the Reactive Architecture. In this process, use architecture-related *scenarios* to analyse the Reactive Architecture. At this point, we ask the stakeholders to assess the adequacy of the scenario-based analysis of the Reactive Architecture using the cloud-ATAM. The responses gathered are shown in Figure 7.30. Here, *38%* (i.e. *8*) of the stakeholders *strongly agree*, and *48%* (i.e. *10*) of the stakeholders *agree* that our analysis was adequately undertaken. That said, *14%* (i.e. *3*) are *neutral* to the claim.

Finally, we asked the stakeholders if *the reasoning behind the analysis process was sound* after identifying *sensitivity points*, *trade-off points*, and *risky points*

in the Reactive Architecture design. From Figure 7.31, the stakeholders who *strongly agree* with the soundness of the analysis were *14%* (i.e. *3*), and those who *agree* are *62%* (i.e. 13). The stakeholders who were *neutral* are *5* constituting *24%*.

7.3.4 cloud-ATAM Analysis Report

The cloud-ATAM delivers the main products: sensitivities, trade-offs, and architectural risks from the two-staged analysis approach (see Figure 7.19). Firstly, the Utility Tree Analysis Mechanism delivered the analysis products in Table 7.11. From Table 7.12, the cloud-ATAM completed a full cycle by linking the "architectural decisions" to the "quality attributes" (i.e. availability, performance), and back to the "business goals" of the Reactive Architecture. The results from Tables 7.11 and 7.12 indicate that the cloud-ATAM found some trade-offs (i.e. T1, T3, T4).

Also, the Stakeholders' Brainstorming Analysis Mechanism was used to assess the Utility Tree Analysis Mechanism approach. Here, the cloud-ATAM, and the Utility Tree Analysis Mechanism approach were presented to the stakeholders of the Reactive Architecture. They were expected to brainstorm and provide their feedback on the methodology and analysis. We then gathered the opinion of the stakeholders about the presentation and our analysis, using a facts gathering approach where a designed "questionnaire" was used. In the three sections of the questionnaire, we identified that the stakeholders were very appreciative of the information we provided to them, and generally supportive of our analysis process. Here, *77.818%* of the stakeholders on an average "strongly agreed" or "agreed" to all of our presentation and analysis. However, an average of *22.182%* (i.e. almost *5*) of the stakeholders chose to be "neutral" in all of the questions from the questionnaire. Their concerns were that:

(a) the description of ATAM was not clear,

(b) a concrete case study of ATAM would be useful in the presentation,

(c) the distinction between ATAM and cloud-ATAM was not clear,

(d) they wanted to know how quality attributes apply in real-world,

(e) if the scenario analysis is manually done or it is automated,

(f) the scenarios were not clear,

(g) they wanted to know the use of the Reactive Architecture, and

(h) whether the Reactive Architecture is a software and/or hardware.

Such concerns afforded the team of analysts the opportunity to respond by clarifying some aspects of our analysis and providing further information.

7.4 Conclusions

In this chapter, we have conducted evaluations for three main sections: the Reactive Middleware, the Cloud Accountability System, and the Reactive Architecture. We briefly present them below.

A summary of the main activities for evaluating the Reactive Middleware have been presented in Section 7.1.2.6. These are in line with *Objectives 1* and *2* presented in Section 1.2.2.

Also, the Cloud Accountability System facilitates the Cloud Accountability System. In this section, we have demonstrated the method by applying it to a cloud-based test-bed of the Reactive Architecture. Also, we have conducted an evidence-based trust analysis on the derived evidence for assuring the dependability of the cloud-based Reactive Architecture. With these two activities, we are convinced that our *Objective 11* and *12* of Section 1.2.2 have been met, and that the cloud accountability method is sufficiently capable of being used to meaningfully assure availability and reliability of deployed systems in the cloud.

We have also motivated the need for architecture evaluation methods suitable for the dynamic unpredictable cloud environments. In particular, we have presented an evaluation method - cloud-ATAM - derived from Architecture Trade-off Analysis Method (ATAM) for evaluating the availability and performance

quality attributes of a cloud-based Reactive Architecture (see *Objective 5* in Section 1.2.2). We have validated cloud-ATAM with a comparative study with ATAM (see *Objective 6* in Section 1.2.2). This methodology specifically presents a two-staged analysis approach (i.e. (1) Utility Tree Analysis Mechanism, and (2) Stakeholders' Brainstorming Analysis Mechanism) for analysing the cloud-based Reactive Architecture. Approaches (1) and (2) are for qualitative system analysis, whiles approach (3) is for quantitative system analysis. However, this methodology as well as the two-staged analysis approach are generic for analysing cloud-based architecture with particular focus on small-to-medium sized systems. This section also presents the results which led us to conclude that the cloud-ATAM is able to identify trade-offs between the availability and performance quality attributes for the Reactive Architecture.

Chapter 8

Conclusion and Future Work

This chapter summarises the contributions of the thesis in Section 8.1. We also show some threats to the validity of our work in Section 8.2, and some of the possible directions of future research in Section 8.3.

8.1 Conclusion

The research presented in this thesis makes several key contributions:

1. The design, development and evaluation of a novel Reactive Middleware, that supports a set of management guidelines for a high quality GSD change management and traceability. The middleware facilitates a novel change management and traceability process model (meets *Objective 1* presented in Section 1.2.2), within the context of quality management for GSD projects (meets *Objective 2*). An expert review panel process is conducted to assess the maturity of the process model (meets *Objective 3*). Also, an Airlock Control System case study is used to demonstrate the GSD management guidelines (meets *Objective 4*). Chapters 4 and 7 present this contribution.
2. The proposal of a novel methodology for the design and analysis of small-to-medium size cloud-based systems (meets *Objective 5*). This method considers the unpredictable character and rapidly evolving topology of the

cloud deployment environment, and its impact on dependability in the bespoke design of systems to be deployed to the cloud. The methodology targets systems that are classified within the range of small to medium size. A comparative study of the current state-of-the-art methods is initially undertaken to identify methods that present a high potential to remedy this challenge (meets *Objective 6*). The method is demonstrated by applying it to the design of the Reactive Architecture (meets *Objective 7*), and then an analysis of the quality attribute trade-off of the Reactive Architecture is conducted to meet *Objective 8*. This contribution is met in Chapters 5 and 7.

3. The design, development and evaluation of the Reactive Architecture, which is used for cloud-based system engineering (refer to *Objectives 7* and *8*). The Reactive Architecture presents critical components for system engineering such as the introduced Reactive Middleware, with a Shared Artefacts Repository, and a System Engineering Toolbox. Here, this contribution is satisfied in Chapters 3 and 7.

4. The proposal of a novel methodology for assuring cloud accountability in terms of dependability to meet *Objective 9*. A forensic analysis process is taken to guide the data collection, examination, evidence analysis, and reporting of information (meets *Objective 11*). This contribution is met in Chapters 6 and 7.

5. The design and development of the Cloud Accountability System to facilitate the cloud accountability methodology (meets *Objective 10* presented in Section 1.2.2). The Cloud Accountability System is used to conduct virtual machine introspection of some key components of the Reactive Architecture, where data is collected also from the Cloud Service Providers based on availability and reliability related metrics. Also, an evidence-based trust analysis of the reported information from the forensic process is conducted, to assure cloud users of the dependability of the cloud environment (meets *Objective 12*). This contribution is met in Chapters 6 and 7.

Table 8.1: Mapping High-Level Requirements to Components of Reactive Architecture

ID	High-Level Requirements	Reactive Architecture Components
R1	Artefacts independence	Shared Artefacts Repository (see Chapter 4)
R2	Supports globally distributed development	Reactive Middleware (see Chapter 4, 5)
R3	Ability to handle different and large numbers of changing artefacts	Reactive Middleware (see Chapter 4) and Shared Artefacts Repository (see Chapter 4, 5)
R4	Automated as far as possible	Reactive Middleware (see Chapter 4, 6)
R5	Diversity of tools	System Engineering Toolbox (Chapter 4)

We also introduced a set of high-level requirements for a framework that has the potential of addressing the state-of-the-art challenges of GSD: (1) effective information and knowledge sharing, (2) automation, and (3) diversity of tools. At this point we relate the components of our proposed Reactive Architecture with these high-level requirements in Table 8.1.

8.2 Limitations

8.2.1 Designing a Cloud-Based Architecture

This work has introduced cloud-ATAM for the design and analysis of small-to-medium size cloud-based architectures. This method has been presented as a qualitative architecture reasoning approach. This however introduces a threat to the validity of cloud-ATAM. This is so because, qualitative reasoning approaches are subjective and may not be widely accepted as a wholly accurate method. We are motivated by the quantitative reasoning approaches

provided by ATAM through the Attribute-Based Architectural Styles (ABASs). An example which is the Reliability Tri-modular Redundancy (RTmR) ABAS provides a means to design and analyse a system that focuses on providing the facility to quantitatively assess the trade-off between a set of quality attributes.

8.2.2 Ensuring Traceability with the Reactive Middleware

The Reactive Middleware introduces a change management and traceability (CM-T) process model. This process model is facilitated in terms of change management and traceability by the open services for lifecycle management (OSLC) approach. In this regard, the limitation of this process model is that the correctness of each consistency management stage is heavily reliant upon the correctness of trace links. OSLC is very effective in identifying artefacts and resources within artefacts. The dependence on trace links between artefacts (including these resources) raises the question of how CM-T process model could be more tolerant to errors introduced during trace creation, especially for a GSD service (i.e. change management and traceability-as-a-service: CM-TaaS) based on the rapidly evolving cloud platform. This is a significant issue considering that the current approach to creating trace links using the OSLC, by nature, is not likely to provide 100% accuracy. Thus, user intervention is required to ensure correct links are established prior to consistency management.

8.2.3 Cloud Accountability Analysis

The cloud accountability analysis undertaken in this thesis presents two main threats to validity. First, the evidence-based trust analysis conducted in this work can be perceived as simplistic even though it is considerably effective in assigning trust values to introspected virtual machines. This simple approach is preferred to make the analysis readily accessible and acceptable to cloud agents. We however take note that there are other established and widely applied evidence-based trust analysis approaches that can be accepted widely

in the niche research community.

Also, the reliability analysis was undertaken over a very limited time frame. Typically, reliability analysis is conducted over a protracted period of time (e.g. three months, six months, a year, etc.). The major constraint was the cost of using the cloud environment over a longer period.

8.2.4 Constitution of the Expert Panel

The choice of using an expert panel process to validate the change management and traceability (CM-T) process model has proved very useful in identifying some of the process model's potential strengths and weaknesses. We believe that the involvement of such a high calibre panel adds weight and rigor to our results. The high response rate and the many additional comments and contributions made, suggest that the experts to the task seriously. Also, the range of responses elicited from this relatively small group formed a good basis for us to guage how the CM-T process model might be viewed in practice.

The constitution of the expert panel is from colleagues at Newcastle University, researchers and practitioners from conferences attended, and other experts who were identified through their research or industrial work. These experts were selected exclusively based on their expertise. That said, the varying levels of the relationship between some of the experts and the author could be perceived as a source of bias. Despite some polarisation of views, there was relatively strong agreement that the requirement engineering process is in need of further support and hence, the CM-T process model has a high potential of enhancing this process.

8.3 Future Work

The work described in this thesis can be extended in a number of ways. We consider the use of repository mining and machine learning approaches for the

stock of development artefacts in the Shared Artefacts Repository, the automation support for the *cloud-ATAM* for architecture design and analysis, and the consideration of a larger set of dependability attributes for cloud accountability. We discuss these areas in order of relative importance. We complete this section by looking at a future case study.

8.3.1 Shared Artefacts Repository Mining and Machine Learning

In this thesis, and specifically in Chapters 2 and 4, we introduce a Shared Artefacts Repository with a Reactive Middleware as an important consideration for GSD as an effective information and knowledge-sharing mechanism. However, we realise that this repository has a large collection of artefacts spanning various software development life-cycle (SDLC) phases. This is a vital resource for system engineering as critical patterns and correlations can be drawn for process optimisation and efficiency. Such a collection can be *mined* to ascertain correlations between GSD team dynamics and the SDLC phases, the development behaviour of different GSD teams to understand the impact of culture and perception of authority on the SDLC phases, etc. Such correlations can be appropriately identified using the increasingly popular concept of *machine learning*. This will then align our work with the domain of data science for the identification of patterns which otherwise would be unknown.

8.3.2 Tool Support for the cloud-ATAM

In Chapter 3, we introduced the cloud-ATAM for designing and evaluating small-to-medium size cloud-based architectures. We then presented the seven derived steps of the cloud-ATAM, which are used to guide the *design* of the Reactive Architecture. To analyse the architectural approaches (i.e. Step 6 of cloud-ATAM), cloud-ATAM further provides a two-staged scenario-based analysis approach in a form of Utility Tree and Stakeholders' Brainstorming (see Chapter 7). We believe that the automation of the cloud-ATAM will help to

further make the design and analysis processes efficient. This is motivated by the partial tool support (i.e. Ex:SAAMTOOL) for the Software Architecture Analysis Method (SAAM) discussed in Section 2.2.5. Even though cloud-ATAM presents a lean methodology compared to ATAM, it is still considerably complex, and the use of a tool here will reduce inefficiencies (including those from human experts) that may be present. Furthermore, presenting this tool as an open source and cloud-based will enable a wider global accessibility, where more data can be obtained towards optimising cloud-ATAM.

With such a tool support, we can consider more quality attributes in the design and analysis phases of cloud-ATAM. This is important since multiple quality attribute analysis will provide a better representation of the quality of small-to-medium size architectures in their deployed cloud environments. It is obvious though that multiple quality attribute analysis will lead to more complex design and analysis processes. However, this automation process will be beneficial in taking away most of the workload by human-experts.

8.3.3 Cloud Accountability of Other Dependability Attributes

In Chapter 6, we identify an area that can be improved. The evaluation of the Reactive Architecture with the cloud accountability method was limited in terms of the 3-hour observation period. In this work, we argue that the Standard Performance Evaluation Corporation (SPEC)'s Cloud IaaS 2016 Benchmark used less than one hour for performance evaluation. However, an effective analysis of the Reactive Architecture's quality attributes (i.e. availability, reliability) could not be undertaken in our work. Even though, our availability and reliability analysis fit well with our objective of providing information to assure cloud agents, it fell short of a regular and acceptable analysis for the quality attributes. To further this work, an observation period of at least six months will be used for a more critical analysis of multiple quality attributes, and hence obtaining a larger set of data for effective analysis.

8.3.4 Future Case Study: Artificial Bee Colony Model-Inspired Traffic Light Control System

To apply cloud-ATAM to the design, analysis and evaluation of a more complex system, we plan to create a traffic light control system (TLCS). The TLCS is aimed at providing an optimised alternative to the coordination of vehicles and pedestrians at traffic lights and at a wide geographical location. This approach is motivated by the *artificial bee colony model*. The system works by networking traffic light systems. Each traffic light system is referred to as a *hive*, and each *hive* has a *queen bee* and a pool of *worker bees*. The *queen bee* is the local coordinating server at a traffic light location that determines the order of movement of vehicles and pedestrians based on priorities, and the *worker bees* are the vehicles and pedestrians. Priorities are given to emergency service vehicles, and to pedestrians at certain times of the day. The network of *queen bees* facilitate an intelligent coordination of traffic, as the state of traffic at individual *hives* are considered for an equitable and efficient traffic control mechanism. This mechanism can only be influenced at a control center. A facilitating framework is provided and deployed to the cloud to benefit from its elasticity and scalability. The *safety* and *security* of the coordination mechanism and the framework forms the focus for the design, analysis and evaluation with cloud-ATAM.

References

[1] Amazon EC2 Service Level Agreement, December 2015. Online at http://aws.amazon.com/ec2/sla/. 50, 54

[2] J. Abawajy. Establishing Trust in Hybrid Cloud Computing Environments. In *2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 118–125, Nov 2011. 63

[3] W. Abderrahim and Z. Choukair. Trust Assurance in Cloud Services with the Cloud Broker Architecture for Dependability. In *IEEE 17th International Conference on High Performance Computing and Communications (HPCC)*, pages 778–781, New York, NY, USA, Aug 2015. 62

[4] ActiveCollab. ActiveCollab: Powerful, Yet Simple Project Management Tool, 2017. Online at https://activecollab.com/. 24

[5] J. K. Adjei. Explaining the Role of Trust in Cloud Computing Services. *info*, 17(1):54–67, 2015. 67

[6] P. J. Ågerfalk, B. Fitzgerald, H. Holmström, B. Lings, B. Lundell, and E. Ó. Conchir. A Framework for Considering Opportunities and Threats in Distributed Software Development. In *Proceedings of the International Workshop on Distributed Software Development. Austrian Computer Society*, pages 47–61, August 2005. 2, 13

[7] M. Y. al Tarawneh, M. S. Abdullah, and A. B. M. Ali. A Proposed Methodology for Establishing Software Process Development Improve-

ment for Small Software Development Firms. *Procedia Computer Science*, 3:893 – 897, 2011. 27

[8] A. Ali Khan, J. Keung, S. Hussain, and K. E. Bennin. Effects of Geographical, Socio-cultural and Temporal Distances on Communication in Global Software Development during Requirements Change Management – A Pilot Study. In *Evaluation of Novel Approaches to Software Engineering (ENASE), International Conference on*, pages 159–168, Barcelona, Spain, April 2015. 1

[9] Amazon Web Service. Amazon Relational Database Service (Amazon RDS), 2014. Online at http://aws.amazon.com/rds/. 262

[10] Apache Software Foundation. Apache Subversion: Enterprise-class centralized version control for the masses, 2015. Online at http://subversion.apache.org/. 24

[11] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A View of Cloud Computing. *Commun. ACM*, 53(4):50–58, Apr. 2010. 3

[12] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, and M. Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. Technical report, Electrical Engineering and Computer Sciences; University of California at Berkeley; UC Berkeley Reliable Adaptive Distributed Systems Laboratory, 2009. 53

[13] N. Assawamekin, T. Sunetnanta, and C. Pluempitiwiriyawej. Ontology-based multiperspective requirements traceability framework. *Knowledge and Information Systems*, 25(3):493–522, Dec 2010. 19

[14] ASSEMBLA. Assembla: Enterprise Cloud Version Control, 2017. Online at https://www.assembla.com/home. 24

[15] Atlassian. Jira Software: The #1 software development tool used by agile teams, 2017. Online at http://www.atlassian.com/software/jira/. 24

[16] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. *Dependable and Secure Computing, IEEE Transactions on*, 1(1):11–33, Jan. 2004. 32

[17] Avritzer, Alberto and Paulish, Daniel and Cai, Yuanfang and Sethi, Kanwarpreet. Coordination Implications of Software Architecture in a Global Software Development Project. *Journal of Systems and Software*, 83(10):1881–1895, 2010. 2

[18] R. Bahsoon. Defining Dependable Dynamic Data-Driven Software Architectures. In *Proceedings of the IEEE International Conference on Information Reuse and Integration 2007, 13-15 August 2007, Las Vegas, Nevada, USA*, pages 691–694, 2007. 31

[19] J. M. Bass. Agile Method Tailoring in Distributed Enterprises: Product Owner Teams. In *2013 IEEE 8th International Conference on Global Software Engineering*, pages 154–163, Aug 2013. 18

[20] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley Professional, 3rd edition, 2012. 3, 31

[21] S. Beecham, N. Carroll, and J. Noll. A Decision Support System for Global Team Management: Expert Evaluation. In *Global Software Engineering Workshops (ICGSEW), IEEE Seventh International Conference on*, pages 12–17, Porto Alegre, Brazil, Aug 2012. 13

[22] S. Beecham, T. Hall, C. Britton, M. Cottee, and A. Rainer. Using an Expert Panel to Validate a Requirements Process Improvement Model. *Journal of Systems and Software*, 76(3):251–275, June 2005. 26, 29

[23] A. Behl and K. Behl. An Analysis of Cloud Computing Security Issues. In *World Congress on Information and Communication Technologies (WICT)*, pages 109–114, Trivandrum, India, Oct 2012. 3, 61

[24] P. Bengtsson and J. Bosch. Scenario-Based Software Architecture Reengineering. In *Software Reuse. Proceedings of the Fifth International Conference on*, pages 308–317, Victoria, BC, Canada, Jun 1998. 3, 39

[25] P. Bengtsson and J. Bosch. Architecture Level Prediction of Software Maintenance. In *Proceedings of the Third European Conference on Software Maintenance and Reengineering*, pages 139–147, Amsterdam, Netherlands, March 1999. 40

[26] P. Bengtsson, N. Lassing, J. Bosch, and H. van Vliet. Architecture-Level Modifiability Analysis (ALMA). *Journal of Systems and Software*, 69(12):129 – 147, 2004. 3, 39

[27] Berger, Olivier and Labbene, Sabri and Dhar, Madhumita and Bac, Christian. Introducing OSLC, an Open Standard for Interoperability of Open Source Development Tools. *ICSSEA*, 2011. 21

[28] J. Bergey, M. Fisher, L. Jones, and R. Kazman. Software Architecture Evaluation with ATAM in the DoD System Acquisition Context. Technical Report CMU/SEI-99-TN-012, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, September 1999. 3, 40

[29] K. Bernsmed. Accountable Health Care Service Provisioning in the Cloud. In *Utility and Cloud Computing (UCC), IEEE/ACM 7th International Conference on*, pages 902–907, London, United Kingdom, Dec 2014. 60

[30] S. Bibi, Y. Hafeez, M. S. Hassan, Z. Gul, H. Pervez, I. Ahmed, and S. Mazhar. Requirement Change Management in Global Software Environment using Cloud Computing. *Journal of Software Engineering and Applications*, 7(8):694–69, 2014. 14

[31] M. Biro. *Open Services for Software Process Compliance Engineering*, pages pp 1–6. Springer International Publishing, Cham, 2014. SOFSEM 2014: Theory and Practice of Computer Science: 40th International Conference on Current Trends in Theory and Practice of Computer Science,

Nový Smokovec, Slovakia, January 26-29, 2014, Proceedings, LNCS 8327. 21

[32] F. Bobot, J.-C. Filliâtre, C. Marché, and A. Paskevich. Why3: Shepherd Your Herd of Provers. In *Boogie 2011: First International Workshop on Intermediate Verification Languages*, pages 53–64, Wroclaw, Poland, 2011. 260

[33] C. Boldyreff, D. Nutter, and S. Rank. Active Artefact Management for Distributed Software Engineering. In *Proceedings 26th Annual International Computer Software and Applications*, pages 1081–1086, 2002. 22

[34] C. Boldyreff, D. Nutter, and S. Rank. Architectural Requirements for an Open Component and Artefact Repository System within GENESIS. In C. Gacek and B. Arief, editors, *In Proceedings of the Open Source Software Development Workshop, Newcastle upon Tyne, UK*, pages 176–196, February 2002. 22

[35] J. Bosch and P. Molin. Software Architecture Design: Evaluation and Transformation. In *IEEE Engineering of Computer Based Systems Symposium*, pages 4–10, Nashville, TN, USA, 1999. 39

[36] N. Boucké, D. Weyns, K. Schelfthout, and T. Holvoet. Applying the ATAM to an Architecture for Decentralized Control of a Transportation System. In *Second International Conference on Quality of Software Architectures, QoSA 2006, Västerås, Sweden, June 27-29, 2006 Revised Papers*, pages 180–198, 2006. 42

[37] B. Bruegge, A. H. Dutoit, and T. Wolf. Sysiphus: Enabling Informal Collaboration in Global Software Development. In *2006 IEEE International Conference on Global Software Engineering (ICGSE'06)*, pages 139–148, Oct 2006. 23

[38] D. Catteddu and G. Hogben. Cloud Computing Information Assurance Framework. *European Network and Information Security Agency (ENISA)*, 2009. 61

[39] Z. Chen, F. Han, J. Cao, X. Jiang, and S. Chen. Cloud computing-based forensic analysis for collaborative network security management system. *Tsinghua Science and Technology*, 18(1):40–50, Feb 2013. 58

[40] M. Chiregi and N. J. Navimipour. A new method for trust and reputation evaluation in the cloud environments using the recommendations of opinion leaders' entities and removing the effect of troll entities. *Computers in Human Behavior*, 60:280 – 292, 2016. 64

[41] M. B. Chrissis, M. Konrad, and S. Shrum. *CMMI Guidlines for Process Integration and Product Improvement*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003. 27, 30

[42] J. Cleland-Huang. Requirements traceability - when and how does it deliver more than it costs? In *14th IEEE International Conference on Requirements Engineering (RE 2006), 11-15 September 2006, Minneapolis/St.Paul, Minnesota, USA*, page 323, 2006. 19

[43] P. Clements, R. Kazman, and M. Klein. *Evaluating Software Architectures : Methods and Case Studies*. SEI series in software engineering. Addison-Wesley, Boston, San Francisco, Paris, 2002. x, 35, 36, 38

[44] Cloud Security Alliance. Cloud Controls Matrix (CCM), February 2017. Online at https://cloudsecurityalliance.org/group/cloud-controls-matrix/. 61

[45] Cloud Security Alliance Global Enterprise Advisory Board. CSA Global Enterprise Advisory Board: State of Cloud Security 2016, 2016. Online at https://www.scribd.com/document/332060881/CSA-GEAB-State-of-Cloud-Security-2016. 2

[46] L. Cocco, K. Mannaro, and G. Concas. A Model for Global Software Development with Cloud Platforms. In *Software Engineering and Advanced Applications (SEAA), 38th EUROMICRO Conference on*, pages 446–452, Cesme, Izmir, Turkey, Sept 2012. 13

[47] Codenvy. SaaS Developer Environment, May 2014. Online at https://codenvy.com/products. 51

[48] B. Costa, P. F. Pires, F. C. Delicato, and P. Merson. Evaluating a Representational State Transfer (REST) Architecture: What is the Impact of REST in My Architecture? In *2014 IEEE/IFIP Conference on Software Architecture*, pages 105–114, April 2014. 45, 46

[49] C. Costa, J. Figueiredo, L. Murta, and A. Sarma. TIPMerge: Recommending Experts for Integrating Changes Across Branches. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, pages 523–534, New York, NY, USA, 2016. ACM. 25

[50] D. Damian and D. Moitra. Guest Editors' Introduction: Global Software Development: How Far Have We Come? *Software, IEEE*, 23(5):17–19, Sept 2006. 13

[51] Daniela S. Cruzes, Nils B. Moe and Tore Dybå. Communication between Developers and Testers in Distributed Continuous Agile Testing. In *2016 IEEE 11th International Conference on Global Software Engineering (ICGSE)*, pages 59–68, Aug 2016. 18

[52] T. A. DeLong, D. T. Smith, and B. W. Johnson. Dependability Metrics to Assess Safety-Critical Systems. *IEEE Transactions on Reliability*, 54(3):498–505, Sept 2005. 33

[53] R. G. Dewar, L. M. MacKinnon, R. J. Pooley, A. D. Smith, M. J. Smith, and P. A. Wilcox. The OPHELIA Project: Supporting Software Development in a Distributed Environment. In *International Association for Development of the Information Society*, pages 568–571, 2002. 48

[54] H. Do, S. Elbaum, and G. Rothermel. Supporting Controlled Experimentation with Testing Techniques: An Infrastructure and Its Potential Impact. *Empirical Softw. Engg.*, 10(4):405–435, October 2005. 22

[55] Duarte, A. M. D. and Duarte, D. and Thiry, M. TraceBoK: Toward a Software Requirements Traceability Body of Knowledge. In *2016 IEEE 24th International Requirements Engineering Conference (RE)*, pages 236–245, Sept 2016. 19

[56] T. Dyba. An Instrument for Measuring the Key Factors of Success in Software Process Improvement. *Empirical Software Engineering*, 5(4):357–390, 2000. 29

[57] N. Ehsan, A. Perwaiz, J. Arif, E. Mirza, and A. Ishaque. CMMI / SPICE Based Process Improvement. In *2010 IEEE International Conference on Management of Innovation Technology*, pages 859–862, June 2010. 28

[58] K. El Emam and A. Birk. Validating the ISO/IEC 15504 Measure of Software Requirements Analysis Process Capability. *IEEE Trans. Softw. Eng.*, 26(6):541–566, June 2000. 29

[59] K. El Emam and N. H. Madhavji. An Instrument for Measuring the Success of the Requirements Engineering Process in Information Systems Development. *Empirical Software Engineering*, 1(3):201–240, 1996. 29

[60] Event-B Collaborators. Event-B and the Rodin Platform, 2014. Online at http://www.event-b.org/. 260

[61] C. Everett. Cloud Computing - A Question of Trust. *Computer Fraud & Security*, 2009(6):5 – 7, June 2009. 52

[62] W. Fan and H. Perros. A novel trust management framework for multi-cloud environments based on trust service providers. *Knowledge-Based Systems*, 70:392 – 406, 2014. 65

[63] B. Gallina, K. Padira, and M. Nyberg. Towards an ISO 26262-compliant OSLC-based Tool Chain Enabling Continuous Self-Assessment. In *2016 10th International Conference on the Quality of Information and Communications Technology (QUATIC)*, pages 199–204, Sept 2016. 30

[64] S. K. Garg, S. Versteeg, and R. Buyya. SMICloud: A Framework for Comparing and Ranking Cloud Services. In *the Fourth IEEE International Conference on Utility and Cloud Computing*, UCC, pages 210–218, Washington, DC, USA, 2011. IEEE Computer Society. 61

[65] D. Garlan. Software Architecture: A Roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, pages 91–101, New York, NY, USA, 2000. ACM. 3, 31

[66] D. Garlan and D. Perry. Software Architecture: Practice, Potential, and Pitfalls. In *Proceedings of the 16th International Conference on Software Engineering*, ICSE '94, pages 363–364, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press. 31

[67] D. Garlan and M. Shaw. An Introduction to Software Architecture. Technical report, Carnegie Mellon University, Pittsburg, PA 15213, USA, 1994. Advances in Software Engineering and Knowledge Engineering, Vol I. 31

[68] A. Gehani, G. F. Ciocarlie, and N. Shankar. Accountable Clouds. In *IEEE International Conference on Technologies for Homeland Security (HST)*, pages 403–407, Waltham, MA, USA, Nov 2013. 57

[69] L. Gillam, B. Li, and J. O'Loughlin. Benchmarking Cloud Performance for Service Level Agreement Parameters. *International Journal of Cloud Computing*, 3(1):3–23, 2014. PMID: 58828. 61, 187

[70] GitHub Inc. GitHub: A Better Way To Work Together, 2017. Online at https://github.com/. 23

[71] Goknil, Arda and Kurtev, Ivan and van den Berg, Klaas. Tool Support for Generation and Validation of Traces Between Requirements and Architecture. In *Proceedings of the 6th ECMFA Traceability Workshop*, ECMFA-TW '10, pages 39–46, New York, NY, USA, 2010. ACM. 19

[72] O. Gotel, J. Cleland-Huang, J. H. Hayes, A. Zisman, A. Egyed, P. Grünbacher, A. Dekhtyar, G. Antoniol, and J. Maletic. *The Grand*

Challenge of Traceability (v1.0), pages 343–409. Springer London, London, 2012. 19

[73] Gotel, O. C. Z. and Finkelstein, C. W. An Analysis of the Requirements Traceability Problem. In *IEEE International Conference on Requirements Engineering*, pages 94–101, Colorado Springs, CO, USA, Apr 1994. 19

[74] S. M. Habib, S. Hauke, S. Ries, and M. Mühlhäuser. Trust as a Facilitator in Cloud Computing: a Survey. *Journal of Cloud Computing: Advances, Systems and Applications*, 1(1):19, 2012. 52

[75] S. M. Habib, S. Ries, and M. Muhlhauser. Cloud Computing Landscape and Research Challenges Regarding Trust and Reputation. In *Symposia and Workshops on Ubiquitous, Autonomic and Trusted Computing*, UIC-ATC '10, pages 410–415, Washington, DC, USA, Oct 2010. IEEE Computer Society. 54

[76] B. Hadley, A. Hume, R. Lindberg, and K. Obraczka. Phantom of the Cloud: Towards Improved Cloud Availability and Dependability. In *Cloud Networking (CloudNet), IEEE 4th International Conference on*, pages 14–19, Niagara Falls, ON, Canada, Oct 2015. 61

[77] A. Haeberlen. A Case for the Accountable Cloud. *ACM SIGOPS Operating Systems Review*, 44(2):52–57, Apr. 2010. New York, NY, USA. 3

[78] S. Hande and S. Mane. An Analysis on Data Accountability and Security in Cloud. In *Industrial Instrumentation and Control (ICIC), 2015 International Conference on*, pages 713–717, Pune, India, May 2015. 52

[79] I. U. Haq, R. Alnemr, A. Paschke, E. Schikuta, H. Boley, and C. Meinel. Distributed trust management for validating sla choreographies. In P. Wieder, R. Yahyapour, and W. Ziegler, editors, *Grids and Service-Oriented Architectures for Service Level Agreements*, pages 45–55, Boston, MA, 2010. Springer US. 65

[80] T. Heath and C. Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Morgan & Claypool, 1st edition, 2011. 21

[81] Heinrich-Heine University. The ProB Animator and Model Checker, 2014. Online at http://www.stups.uni-duesseldorf.de/ProB/index.php5/. 260

[82] A. Hendre and K. Joshi. A Semantic Approach to Cloud Security and Compliance. In *Cloud Computing (CLOUD), 8th International Conference on*, pages 1081–1084, New York, NY, USA, June 2015. IEEE. 3, 61

[83] J. Herbsleb and D. Moitra. Global Software Development. *Software, IEEE*, 18(2):16–20, Mar 2001. 12, 13

[84] J. D. Herbsleb. Global Software Engineering: The Future of Socio-technical Coordination. In *Future of Software Engineering*, FOSE '07, pages 188–198, Washington, DC, USA, 2007. IEEE Computer Society. 13

[85] J. D. Herbsleb and A. Mockus. An Empirical Study of Speed and Communication in Globally Distributed Software Development. *IEEE Trans. Softw. Eng.*, 29(6):481–494, June 2003. 12

[86] H. Holmström, B. Fitzgerald, P. J. Ågerfalk, and E. Ó. Conchúir. Agile Practices Reduce Distance in Global Software Development. *IS Management*, 23(3):7–18, 2006. 2, 13

[87] M. Hölzl, A. Rauschmayer, and M. Wirsing. Software-Intensive Systems and New Computing Paradigms. In *Engineering of Software-Intensive Systems: State of the Art and Research Challenges*, pages 1–44. Springer-Verlag, Berlin, Heidelberg, 2008. 31

[88] E. Hossain, M. A. Babar, and J. Verner. *How Can Agile Practices Minimize Global Software Development Co-ordination Risks?*, pages 81–92. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. 19

[89] J. Huang and D. M. Nicol. Trust mechanisms for cloud computing. *Journal of Cloud Computing: Advances, Systems and Applications*, 2(1):9, Apr 2013. 66

[90] Y. Huo, Y. Zhuang, and S. Ni. Fuzzy Trust Evaluation Based on Consistency Intensity for Cloud Services. *Kybernetes*, 44(1):7–24, 2015. 68

[91] K. Hwang, S. Kulkareni, and Y. Hu. Cloud Security with Virtualized Defense and Reputation-Based Trust Mangement. In *2009 Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing*, pages 717–722, Dec 2009. 63

[92] IBM Rational DOORS. Rational Team Concert: IBM Rational DOORS family, 2017. Online at http://www-01.ibm.com/software/awdtools/doors/. 24

[93] IBM Rational DOORS Next Generation. IBM Rational Requirements Composer, 2017. Online at https://jazz.net/products/rational-requirements-composer/. 24

[94] A. Iliasov, L. Laibinis, E. Troubitsyna, D. E. Adjepon-Yamoah, and A. Romanovsky. Refinement-based Approach to Co-engineering Requirements and Formal Models. Technical Report CS-TR-1456, Newcastle University, Newcastle-upon-Tyne, UK, March 2015. 157

[95] A. Iliasov, I. Lopatkin, and A. Romanovsky. The SafeCap Project on Railway Safety Verification and Capacity Simulation. In *Software Engineering for Resilient Systems*, volume 8166 of *Lecture Notes in Computer Science*, pages 125–132. Springer Berlin Heidelberg, 2013. 260

[96] A. Iliasov, P. Stankaitis, and D. Adjepon-Yamoah. Static verification of railway schema and interlocking design data. In T. Lecomte, R. Pinger, and A. Romanovsky, editors, *Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification*, pages 123–133, Cham, 2016. Springer International Publishing. 260

[97] A. Iliasov, P. Stankaitis, and D. E. Adjepon-Yamoah. Event-B and Cloud Provers. In *Proceedings of the Automated Reasoning Workshop – Bridging the Gap between Theory and Practice*, ARW 2015, pages 11–12, Birmingham, UK, April 2015. 260

[98] A. Iliasov, P. Stankaitis, D. E. Adjepon-Yamoah, and A. Romanovsky. A Rodin Plug-in for Constructing Reusable Schematic Lemmas. In *6th Rodin User and Developer Workshop*, volume 6, pages 5–6, Linz, Austria, May 2016. 260

[99] A. Iliasov, P. Stankaitis, D. E. Adjepon-Yamoah, and A. Romanovsky. Rodin Platform Why3 Plug-In. In *Proceedings of the 5th International ABZ Conference 2016 ASM, Alloy, B, TLA, VDM, Z, ABZ*, volume 5, Linz, Austria, May 2016. 260

[100] D. C. Ince. *Introduction to Software Quality Assurance and Its Implementation*. McGraw-Hill, Inc., New York, NY, USA, 1995. x, 15, 16

[101] International Standards Organisation. ISO/IEC 19761:2011 Software Engineering - COSMIC: a Functional Size Measurement Method, 2011. Online at http://www.iso.org/iso/catalogue_detail.htm?csnumber_54849. 47

[102] International Standards Organisation. ISO/IEC/IEEE International Standard - Systems and Software Engineering – Software Life Cycle Processes. *ISO/IEC/IEEE 12207:2017(E) First edition 2017*, pages 1–157, Nov 2017. 23, 24

[103] Y. Jadeja and K. Modi. Cloud Computing - Concepts, Architecture and Challenges. In *International Conference on Computing, Electronics and Electrical Technologies (ICCEET) 2012*, pages 877–880, Tamil Nadu, India, March 2012. 2, 51

[104] H. Jayathilaka, C. Krintz, and R. Wolski. Response Time Service Level Agreements for Cloud-hosted Web Applications. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, SoCC '15, pages 315–328, New York, NY, USA, 2015. ACM. 53

[105] T. Joachims. *Text categorization with Support Vector Machines: Learning with many relevant features*, pages 137–142. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998. 20

[106] L. Jones and A. Lattanze. Using the Architecture Trade-off Analysis Method to Evaluate a Wargame Simulation System: A Case Study. Technical Report CMU/SEI-2001-TN-022, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2001. 42

[107] Julian M. Bass. Scrum Master Activities: Process Tailoring in Large Enterprise Projects. In *2014 IEEE 9th International Conference on Global Software Engineering*, pages 6–15, Aug 2014. 18

[108] Y. Kaeri, Y. Manabe, C. Moulin, K. Sugawara, and J. P. A. Barthes. A Cloud-Based Support System for Offshore Software Development. In *Broadband, Wireless Computing, Communication and Applications (BWCCA), 2010 International Conference on*, pages 315–319, Nov 2010. 49

[109] P. Kaur and S. Sharma. Agile Software Development in Global Software Engineering. *International Journal of Computer Applications*, 97(4):39–43, July 2014. 1

[110] R. Kazman, M. Barbacci, M. Klein, S. J. Carrière, and S. G. Woods. Experience with Performing Architecture Tradeoff Analysis. In *Proceedings of the 21st International Conference on Software Engineering*, ICSE '99, pages 54–63, New York, NY, USA, 1999. ACM. 42

[111] R. Kazman, L. Bass, M. Webb, and G. Abowd. SAAM: A Method for Analyzing the Properties of Software Architectures. In *Proceedings of the 16th International Conference on Software Engineering*, ICSE '94, pages 81–90, Los Alamitos, CA, USA, May 1994. IEEE Computer Society Press. 3, 38

[112] R. Kazman, L. Bass, M. Webb, and G. Abowd. Saam: A method for analyzing the properties of software architectures. In *Proceedings of the*

16th International Conference on Software Engineering, ICSE '94, pages 81–90, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press. 38, 39

[113] R. Kazman, M. Klein, and P. Clements. ATAM: Method for Architecture Evaluation. Technical Report CMU/SEI-2000-TR-004 ESC-TR-2000-004, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, August 2000. 38

[114] K. Kent, S. Chevalier, T. Grance, and H. Dang. Guide to Integrating Forensic Techniques into Incident Response. Technical Report SP 800-86, Gaithersburg, MD, United States, 2006. 59

[115] A. Khan, S. Basri, and P. Dominic. A Propose Framework for Requirement Change Management in Global Software Development. In *Computer Information Science (ICCIS), 2012 International Conference on*, volume 2, pages 944–947, Kuala Lumpur, Malaysia, June 2012. 1, 14, 18

[116] A. A. Khan, S. Basri, and P. Dominc. A proposed framework for communication risks during rcm in gsd. *Procedia - Social and Behavioral Sciences*, 129:496 – 503, 2014. 2nd International Conference on Innovation, Management and Technology Research. 2

[117] K. Khan, A. Khan, M. Aamir, M. Khan, S. Zulfikar, A. Bhutto, and T. Szabist. Quality Assurance Assessment in Global Software Development. *World Applied Sciences Journal*, 24(11):1449–1454, 2013. 14

[118] N. Khan, W. L. Currie, V. Weerakkody, and B. Desai. Evaluating Offshore IT Outsourcing in India: Supplier and Customer Scenarios. In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track 8*, volume 8 of *HICSS '03*, pages 239.1–, Washington, DC, USA, January 2003. IEEE Computer Society. 13

[119] A. Khatoon, Y. H. Motla, M. Azeem, H. Naz, and S. Nazir. Requirement Change Management for Global Software Development using Ontology.

In *IEEE 9th International Conference on Emerging Technologies (ICET)*, pages 1–6, Islamabad, Pakistan, Dec 2013. 14

[120] B. Kitchenham, S. L. Pfleeger, B. McColl, and S. Eagan. An empirical study of maintenance and development estimation accuracy. *Journal of Systems and Software*, 64(1):57 – 77, 2002. 29

[121] Kommeren, Rob and Parviainen, Päivi. Philips Experiences in Global Distributed Software Development. *Empirical Software Engineering*, 12(6):647–660, 2007. 2

[122] Kotonya, Gerald and Sommerville, Ian. *Requirements Engineering: Processes and Techniques*. Wiley Publishing, 1st edition, 1998. 19

[123] K. Kourai and K. Nakamura. Efficient VM Introspection in KVM and Performance Comparison with Xen. In *Proceedings of the 2014 IEEE 20th Pacific Rim International Symposium on Dependable Computing*, PRDC '14, pages 192–202, Washington, DC, USA, 2014. IEEE Computer Society. 59

[124] S. A. Kumar and T. A. Kumar. Study the Impact of Requirements Management Characteristics in Global Software Development Projects: an Ontology Based Approach. *International Journal of Software Engineering & Applications*, 2(4):107, 2011. 2, 13

[125] R. Lai and N. Ali. A Requirements Management Method for Global Software Development. *AIS: Advances in Information Sciences*, 1(1):38–58, 2013. 18

[126] A. Lamersdorf and J. Munch. TAMRI: A Tool for Supporting Task Distribution in Global Software Development Projects. In *Global Software Engineering, 2009. ICGSE 2009. Fourth IEEE International Conference on*, pages 322–327. IEEE, 2009. 24

[127] J. Lamps, I. Palmer, and R. Sprabery. WinWizard: Expanding Xen with a LibVMI Intrusion Detection Tool. In *Cloud Computing (CLOUD),*

2014 IEEE 7th International Conference on, pages 849–856, Anchorage, Alaska, June 2014. 59

[128] J. Lansing and A. Sunyaev. Trust in Cloud Computing: Conceptual Typology and Trust-Building Antecedents. *SIGMIS Database*, 47(2):58–96, June 2016. 52

[129] N. Lassing, P. Bengtsson, H. van Vliet, and J. Bosch. Experiences with ALMA: Architecture-Level Modifiability Analysis. *Journal of Systems and Software*, 61(1):47 – 57, 2002. 3, 40

[130] N. Lassing, D. Rijsenbrij, and H. van Vliet. On Software Architecture Analysis of Flexibility, Complexity of Changes: Size Isn't Everything. In *Proceedings of the Second Nordic Software Architecture Workshop (NOSA '99)*, Ronneby, Sweden, 1999. 3, 38, 39

[131] S. Lauesen and O. Vinter. Preventing Requirement Defects: An Experiment in Process Improvement. *Requirements Engineering*, 6(1):37–50, 2001. 29

[132] M. A. Law. Using Net Present Value as a Decision-Making Tool. *Air Medical Journal*, 23(6):28 – 33, 2004. 60

[133] J. Lee, S. Kang, H. Chun, B. Park, and C. Lim. Analysis of VAN-Core System Architecture - A Case Study of Applying the ATAM. In *Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing, 10th ACIS International Conference on*, pages 358–363, Daegu, Korea, May 2009. 42

[134] H. K. N. Leung. Slow change of information system development practice. *Software Quality Journal*, 8(3):197–210, 1999. 29

[135] LiHong, Xu and ShuTao, Sun and Qi, Wang. Text Similarity Algorithm Based on Semantic Vector Space Model. In *2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)*, pages 1–4, June 2016. 20

[136] P. Lous, M. Kuhrmann, and P. Tell. Is Scrum Fit for Global Software Engineering? In *2017 IEEE 12th International Conference on Global Software Engineering (ICGSE)*, pages 1–10, May 2017. 18

[137] J. Lyle and A. Martin. Trusted Computing and Provenance: Better Together. In *Proceedings of the 2nd Conference on Theory and Practice of Provenance*, TAPP'10, pages 1–1, Berkeley, CA, USA, 2010. USENIX Association. 3

[138] Mahmood Niazi, Mohamed El-Attar, Muhammad Usman and Naveed Ikram. GlobReq: A Framework for Improving Requirements Engineering in Global Software Development Projects: Preliminary Results. In *16th International Conference on Evaluation Assessment in Software Engineering (EASE 2012)*, pages 166–170, May 2012. 13, 18

[139] Y. Mao, X. Chen, and Y. Luo. HVSM: An In-Out-VM Security Monitoring Architecture in IaaS Cloud. In *Information and Network Security, ICINS - International Conference on*, pages 185–192, Jeju Island, Korea, Nov 2014. 59

[140] A. Martens, H. Koziolek, S. Becker, and R. Reussner. Automatically Improve Software Architecture Models for Performance, Reliability, and Cost Using Evolutionary Algorithms. In *Proceedings of the First Joint International Conference on Performance Engineering (Workshop on Software and Performance (WOSP) and the SPEC International Performance Evaluation Workshop (SIPEW))*, WOSP/SIPEW '10, pages 105–116, New York, NY, USA, 2010. ACM. 36

[141] R. Meli, A. Abran, V. T. Ho, and S. Oligny. On the Applicability of COSMIC-FFP for Measuring Software throughout its Life-cycle. In *11th European Software Control and Metrics Conference*, Munich, Germany, 2000. 47

[142] A. Mengist, A. Pop, A. Asghar, and P. Fritzson. *Traceability Support in OpenModelica using Open Services for Lifecycle collaboration (OSLC)*,

pages 823–830. Linkping University Electronic Press, 2017. This publication is part of the Horizon 2020 project: Integrated Tool chain for model-based design of CPSs (INTO-CPS), project/GA number 644047. 21

[143] Mercurial Inc. Mercurial Source Control Management, 2015. Online at https://mercurial.selenic.com/. 24

[144] Microsoft Research. Z3 - Efficient Theorem Prover, 2014. Online at http://rise4fun.com/Z3. 262

[145] M. Minna. *Application Management Requirements for Embedded Software*. Number VTT Publications 416 in VTT julkaisuja. Technical Research Centre of Finland, VTT Electronics, Finland, 1996. x, 16, 17

[146] Mohammad Abdur Razzak and Darja Šmite. Knowledge Management in Globally Distributed Agile Projects – Lesson Learned. In *2015 IEEE 10th International Conference on Global Software Engineering*, pages 81–89, July 2015. 18

[147] M. Mohite and S. Ardhapurkar. Overcast: Developing Digital Forensic Tool in Cloud Computing Environment. In *Innovations in Information, Embedded and Communication Systems (ICIIECS), 2015 International Conference on*, pages 1–4, Coimbatore, India, March 2015. 58

[148] G. Molter. Integrating SAAM in Domain-centric and Reuse-based Development Processes. In *Proceedings of the 2nd Nordic Workshop Software Architecture (N0SA '99)*, pages 1103–1581, Ronneby, Sweden, August 1999. 3, 39

[149] C. H. Montenegro, H. Astudillo, and M. C. G. lvarez. ATAM-RPG: A Role-Playing Game to Teach Architecture Trade-off Analysis Method (ATAM). In *2017 XLIII Latin American Computer Conference (CLEI)*, pages 1–9, Sept 2017. 42

[150] N. Narendra, K. Ponnalagu, N. Zhou, and W. Gifford. Towards a Formal Model for Optimal Task-Site Allocation and Effort Estimation in Global Software Development. In *2012 Annual Global Conference (SRII)*, pages 470–477, San Jose, CA, USA, July 2012. 13

[151] J. Nawrocki, M. Jasinski, B. Walter, and A. Wojciechowski. Extreme programming modified: Embrace requirements engineering practices. In *Proceedings IEEE Joint International Conference on Requirements Engineering*, pages 303–310, September 2002. 19

[152] O. Ngwenyama and P. Nielsen. Competing Values in Software Process Improvement: an Assumption Analysis of CMM from an Organizational Culture Perspective. *Engineering Management, IEEE Transactions on*, 50(1):100–112, Feb 2003. 29

[153] M. Niazi, C. Hickman, R. Ahmad, and M. Ali Babar. *A Model for Requirements Change Management: Implementation of CMMI Level 2 Specific Practice*, pages 143–157. Springer Berlin Heidelberg, Monte Porzio Catone, Italy, June 2008. x, 17, 18, 27, 29

[154] M. Niazi and S. Shastry. Critical Success Factors for the Improvement of Requirements Engineering Process. In *Proceedings of the International Conference on Software Engineering Research and Practice, SERP '03, June 23 - 26, 2003, Las Vegas, Nevada, USA, Volume 1*, pages 433–439, 2003. 27, 29

[155] M. Niazi, D. Wilson, and D. Zowghi. Critical Barriers for Software Process Improvement Implementation: An Empirical Study. In *IASTED International Conference on Software Engineering, part of the 22nd Multi-Conference on Applied Informatics*, pages 389–395, Innsbruck, Austria, February 2004. 27, 29

[156] M. Niazi, D. Wilson, and D. Zowghi. Implementing Software Process Improvement Initiatives: An Empirical Study. In *Proceedings of the 7th*

International Conference on Product-Focused Software Process Improvement, (PROFES), pages 222–233, Amsterdam, The Netherlands, June 2006. 27, 28, 29

[157] E. Ò. Conchir, H. Holmström Olsson, P. J. Ågerfalk, and B. Fitzgerald. Benefits of Global Software Development: Exploring the Unexplored. *Software Process: Improvement and Practice*, 14(4):201–212, 2009. 1

[158] Object Management Group Inc. Common Object Request Broker Architecture (CORBA), 2014. Online at http://www.omg.org/spec/CORBA/. 49

[159] Object Management Group Inc. Object Management Group: We Set The Standard, 2014. Online at http://www.omg.org/. 49

[160] Object Management Group Inc. Open Services for Lifecycle Collaboration (OSLC), 2017. Online at http://open-services.net/. 21, 30

[161] N. C. Olsen. The Software Rush Hour. *IEEE Software*, 10(5):29–37, Sept 1993. x, 14, 15

[162] M. Ouedraogo and H. Mouratidis. Selecting a Cloud Service Provider in the Age of Cybercrime. *Computers & Security*, 38:3–13, 2013. 61

[163] A. Pakhira and P. Andras. *Leveraging the Cloud for Large-Scale Software Testing A Case Study: Google Chrome on Amazon*, pages 252–279. IGI Global, Hershey, PA, USA, 2013. 50

[164] Y. Pan and N. Hu. Research on Dependability of Cloud Computing Systems. In *Reliability, Maintainability and Safety (ICRMS), 2014 International Conference on*, pages 435–439, Guangzhou, China, Aug 2014. 32

[165] B. Parno, J. Howell, C. Gentry, and M. Raykova. Pinocchio: Nearly Practical Verifiable Computation. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 238–252, May 2013. 67

[166] A. Patidar and U. Suman. A Survey on Software Architecture Evaluation Methods. In *Computing for Sustainable Global Development (INDIACom), 2015 2nd International Conference on*, pages 967–972, New Delhi, India, March 2015. 37

[167] P. S. Pawar, M. Rajarajan, S. K. Nair, and A. Zisman. Trust model for optimized cloud services. In T. Dimitrakos, R. Moona, D. Patel, and D. H. McKnight, editors, *Trust Management VI*, pages 97–112, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. 64

[168] Pernille Lous, Marco Kuhrmann and Paolo Tell. Is Scrum Fit for Global Software Engineering? In *Proceedings of the 12th International Conference on Global Software Engineering*, ICGSE '17, pages 1–10, Piscataway, NJ, USA, 2017. IEEE Press. 18

[169] D. E. Perry and A. L. Wolf. Foundations for the Study of Software Architecture. *ACM SIGSOFT Softw. Eng. Notes*, 17(4):40–52, Oct. 1992. 31

[170] Petra Heck and Andy Zaidman. Horizontal Traceability for Just-in-Time Requirements: The Case for Open Source Feature Requests. *Journal of Software: Evolution and Process*, 26(12):1280–1296, 2014. 20

[171] Pfleeger, S.L. and Atlee, J.M. *Software Engineering: Theory and Practice*. Prentice Hall, 2010. 19

[172] F. Pinto, U. Kulesza, L. Silva, and E. Guerra. Automating the Assessment of the Performance Quality Attribute for Evolving Software Systems: An Exploratory Study. In *2015 48th Hawaii International Conference on System Sciences*, pages 5144–5153, Jan 2015. 45

[173] R. Poisel, E. Malzer, and S. Tjoa. Evidence and Cloud Computing: The Virtual Machine Introspection Approach. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)*, 4(1):135–152, 2013. 59

[174] J. Portillo-Rodriguez, A. Vizcaino, C. Ebert, and M. Piattini. Tools to Support Global Software Development Processes: A Survey. In *2010 5th IEEE International Conference on Global Software Engineering*, pages 13–22, Aug 2010. 24

[175] R. Prikladnicki, J. Audy, and R. Evaristo. A Reference Model for Global Software Development: Findings from a Case Study. In *Global Software Engineering, ICGSE '06. International Conference on*, pages 18–28, Florianopolis, Brazil, Oct 2006. 1

[176] Project Management Institute. Adoption of the Project Management Institute (PMI) Standard: A Guide to the Project Management Body of Knowledge (PMBOK Guide)-2008 (4th edition). *IEEE P1490/D1, May 2011*, pages 1–505, June 2011. 29

[177] I. M. Putrama, K. T. Dermawan, G. R. Dantes, and K. Y. E. Aryanto. Architectural Evaluation of Data Center System using Architecture Tradeoff Analysis Method (ATAM): A Case Study. In *2017 International Conference on Advanced Informatics, Concepts, Theory, and Applications (ICAICTA)*, pages 1–6, Aug 2017. 42

[178] A. Rajan and T. Wahl. *CESAR: Cost-efficient Methods and Processes for Safety-relevant Embedded Systems*. Number 978-3709113868. Springer, 2013. 21

[179] M. Ramachandran and V. Chang. Recommendations and Best Practices for Cloud Enterprise Security. In *Cloud Computing Technology and Science (CloudCom), IEEE 6th International Conference on*, pages 983–988, Washington, DC, USA, Dec 2014. IEEE Computer Society. 3, 61

[180] N. Ramasubbu and R. K. Balan. Globally Distributed Software Development Project Performance: An Empirical Analysis. In *Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software*

Engineering, ESEC-FSE '07, pages 125–134, New York, NY, USA, 2007. ACM. 2, 12

[181] N. Ramasubbu, M. Cataldo, R. K. Balan, and J. D. Herbsleb. Configuring Global Software Teams: A Multi-company Analysis of Project Productivity, Quality, and Profits. In *33rd International Conference on Software Engineering*, ACM ICSE '11, pages 261–270, New York, USA, 2011. 12

[182] Regional Computer Forensics Laboratory Program. Annual Report for Fiscal Year 2011, 2011. Online at http://www.rcfl.gov/downloads/documents/RCFL_Nat_Annua1 11.pdf. 57

[183] T. Rosqvist, M. Koskela, and H. Harju. Software Quality Evaluation Based on Expert Judgement. *Software Quality Journal*, 11(1):39–55, May 2003. 29

[184] B. Roy and T. N. Graham. Methods for Evaluating Software Architecture: A Survey. Technical Report 2008-545, Queen's University at Kingston, School of Computing Technical Report, Ontario, Canada, April 2008. 37

[185] K. Ruan, J. Carthy, T. Kechadi, and I. Baggili. Cloud Forensics Definitions and Critical Criteria for Cloud Forensic Capability: An Overview of Survey Results. *Digital Investigation*, 10(1):34–43, 2013. 58

[186] J. Rushby. *An Evidential Tool Bus*, pages 36–36. Springer, Berlin, Heidelberg, 2005. 48

[187] N. Santos, K. P. Gummadi, and R. Rodrigues. Towards Trusted Cloud Computing. In *Proceedings of the Conference on Hot Topics in Cloud Computing*, HotCloud '09, Berkeley, CA, USA, 2009. USENIX Association. 3

[188] J. Sauvé, R. Rebouças, A. Moura, C. Bartolini, A. Boulmakoul, and D. Trastour. *Business-Driven Decision Support for Change Management:*

Planning and Scheduling of Changes, pages 173–184. Springer, Berlin, Heidelberg, October 2006. 14

[189] A. Selvaraj and S. Sundararajan. Evidence-based trust evaluation system for cloud services using fuzzy logic. *International Journal of Fuzzy Systems*, 19(2):329–337, Apr 2017. 68

[190] S.-O. Setamanit. *A Software Process Simulation Model of Global Software Development (GSD) Projects*. PhD thesis, Portland, OR, USA, 2007. Portland State University AAI3294665. 2, 13

[191] F. Shaikh and S. Haider. Security Threats in Cloud Computing. In *Internet Technology and Secured Transactions (ICITST), International Conference for*, pages 214–219, Abu Dhabi, United Arab Emirates, Dec 2011. 3, 61

[192] F. Sharevski. Digital Forensic Investigation in Cloud Computing Environment: Impact on Privacy. In *Systematic Approaches to Digital Forensic Engineering (SADFE), Eighth International Workshop on*, pages 1–6, Pok Fu Lam, Hong Kong, Nov 2013. 58

[193] S. Sharma, P. Kaur, and U. Kaur. Communication Understandability Enhancement in GSD. In *International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)*, pages 28–33, Greater Noida, India, Feb 2015. 1

[194] H. Shuijing. Data Security: The Challenges of Cloud Computing. In *the Sixth International Conference on Measuring Technology and Mechatronics Automation*, pages 203–206, Zhangjiajie, China, Jan 2014. 52

[195] G. Sibiya, T. Fogwill, H. Venter, and S. Ngobeni. Digital Forensic Readiness in a Cloud Environment. In *IEEE African Conference (AFRICON) on Sustainable Engineering for a Better Future*, pages 1–5, Pointe-Aux-Piments, Mauritius, Sept 2013. 58

[196] J. Siegel and J. Perdue. Cloud Services Measures for Global Use: The Service Measurement Index (SMI). In *Annual Service Research and Innovation Institute (SRII) Global Conference*, pages 411–415, San Jose, CA, USA, July 2012. 61

[197] Software Freedom Conservancy. GIT:Distributed Is The New Centralized, 2015. Online at http://git-scm.com/. 23

[198] Standard Performance Evaluation Corporation. SPEC^{TM}Cloud IaaS 2016, April 2016. Online at http://www.spec.org/cloud_iaas2016/index.html. 55

[199] F. Stefan, P. Heidl, and P. Lutz. *Reviewing Product Line Architectures: Experience Report of ATAM in an Automotive Context*, volume 2290. Springer, 2001. 42

[200] R. Sultana, F. Jan, A. Mateen, and A. Adnan. Empirical and Qualitative Studies by Analyzing Requirement Issues in Global Software Development (GSD). *International Journal of Managment, IT and Engineering*, 2(1):6–18, 2012. 18

[201] D. Sun, G. Chang, Q. Guo, C. Wang, and X. Wang. A Dependability Model to Enhance Security of Cloud Environment Using System-Level Virtualization Techniques. In *Pervasive Computing Signal Processing and Applications (PCSPA), First International Conference on*, pages 305–310, Harbin, China, Sept 2010. 3, 61

[202] S. Suneja, C. Isci, E. de Lara, and V. Bala. Exploring VM Introspection: Techniques and Trade-offs. *SIGPLAN Not.*, 50(7):133–146, Mar. 2015. 59

[203] P. Szwed, I. Wojnicki, S. Ernst, and A. Gowacz. Application of New ATAM Tools to Evaluation of the Dynamic Map Architecture. In *Multimedia Communications, Services and Security*, volume 368 of *Communications in Computer and Information Science*, pages 248–261. Springer Berlin Heidelberg, 2013. 42

[204] M. Tang, X. Dai, J. Liu, and J. Chen. Towards a trust evaluation middleware for cloud service selection. *Future Generation Computer Systems*, 74:302 – 312, 2017. 65, 69

[205] C. P. Team. CMMI for Development, Version 1.3. Technical Report CMU/SEI-2010-TR-033, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2010. 27

[206] B. Tekinerdogan. ASAAM: Aspectual Software Architecture Analysis Method. In *Proceedings. Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA 2004)*, pages 5–14, June 2004. 43, 46

[207] R. H. Thayer. Software System Engineering: A Tutorial. *Computer*, 35(4):68–73, Apr. 2002. 31

[208] The Apache Software Foundation. Apache Maven Project, 2017. Online at http://maven.apache.org/. 24

[209] The Eclipse Foundation. Orion: Open Source Platform for Cloud Based Development, 2014. Online at http://www.eclipse.org/orion/. 51

[210] The International Software Benchmarking Standards Group. ISBSG Repositories, 2007. Online at http://isbsg.org/. 48

[211] G. Theocharis, M. Kuhrmann, J. Münch, and P. Diebold. *Is Water-Scrum-Fall Reality? On the Use of Agile and Traditional Development Practices*, pages 149–166. Springer International Publishing, Cham, 2015. 18

[212] S. R. Tilley and T. Parveen. *Software Testing in the Cloud - Migration and Execution*. Springer Briefs in Computer Science. Springer, 2012. 50

[213] TPTP. Thousands of Problems for Theorem Provers, 2014. Online at www.tptp.org/. 260

[214] D. Trastour, M. Rahmouni, and C. Bartolini. *Activity-Based Scheduling of IT Changes*, pages 73–84. Springer, Berlin, Heidelberg, June 2007. 14

[215] E. N. Urwin, C. C. Venters, D. J. Russell, L. Liu, Z. Luo, D. E. Webster, M. Henshaw, and J. Xu. Scenario-Based Design and Evaluation for Capability. In *2010 5th International Conference on System of Systems Engineering*, pages 1–6, June 2010. 43

[216] V. Viji Rajendran and S. Swamynathan. Hybrid model for dynamic evaluation of trust in cloud services. *Wireless Networks*, 22(6):1807–1818, Aug 2016. 64

[217] S. Vinoski. CORBA: Integrating Diverse Applications within Distributed Heterogeneous Environments. *Communications Magazine, IEEE*, 35(2):46–55, Feb 1997. 62

[218] D. Šmite and J. Borzovs. Managing Uncertainty in Globally Distributed Software Development Projects. Technical report, Computer Science and Information Technologies, Scientific Papers, University of Latvia, 2008. 13

[219] L. Wagner. Extreme Requirements Engineering. *Cutter IT Journal*, 14(12):34–38, 2001. 19

[220] W. Wang, G. Zeng, D. Tang, and J. Yao. Cloud-dls: Dynamic trusted scheduling for cloud computing. *Expert Systems with Applications*, 39(3):2321 – 2329, 2012. 66

[221] Y. Wang, S. Chandrasekhar, M. Singhal, and J. Ma. A limited-trust capacity model for mitigating threats of internal malicious services in cloud computing. *Cluster Computing*, 19(2):647–662, Jun 2016. 68

[222] H. Wei and P.-L. Qiao. Reliability Assessment of Cloud Computing Platform Based on Semiquantitative Information and Evidential Reasoning. *Journal of Control Science and Engineering*, 2016. 3

[223] W. Wetekamp. Net Present Value (NPV) as a Tool Supporting Effective Project Management. In *Intelligent Data Acquisition and Advanced*

Computing Systems (IDAACS), IEEE 6th International Conference on, volume 2, pages 898–900, Prague, Czech Republic, Sept 2011. 60

[224] P. S. Weygant. *Clusters for High Availability: A Primer of HP Solutions*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 2001. x, 56

[225] P. A. Wilcox, C. R. Russell, M. J. Smith, and A. D. Smith. A Corba-Oriented Approach to Heterogeneous Tool Integration; OPHELIA. In *the 9th European Software Engineering Conference, and 11th ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 1–5, Helsinki, Finland, September 2003. 48

[226] F. Yucalar and S. Z. Erdogan. A Questionnaire Based Method for CMMI Level 2 Maturity Assessment. *Journal of Aeronautics & Space Technologies/Havacilik ve Uzay Teknolojileri Dergisi*, 4(2), 2009. 27

[227] J. Zou, Y. Wang, and K. J. Lin. A Formal Service Contract Model for Accountable SaaS and Cloud Services. In *IEEE International Conference on Services Computing*, pages 73–80, Miami, Florida, USA, July 2010. 57

[228] J. Zou, Y. Wang, and M. A. Orgun. Modeling Accountable Cloud Services. In *IEEE International Conference on Web Services*, pages 353–360, Anchorage, AK, USA, June 2014. 57

Appendix A: Expert Review of Change Management and Traceability Process Model

This appendix contains the results of the feedback analysis from the expert review for the *change management and traceability process model* discussed in Chapter 4. This analysis is presented here as bar graphs, and classified under seven (7) sections of the developed questionnaire.

A.1 Overview of CM-T Process Model

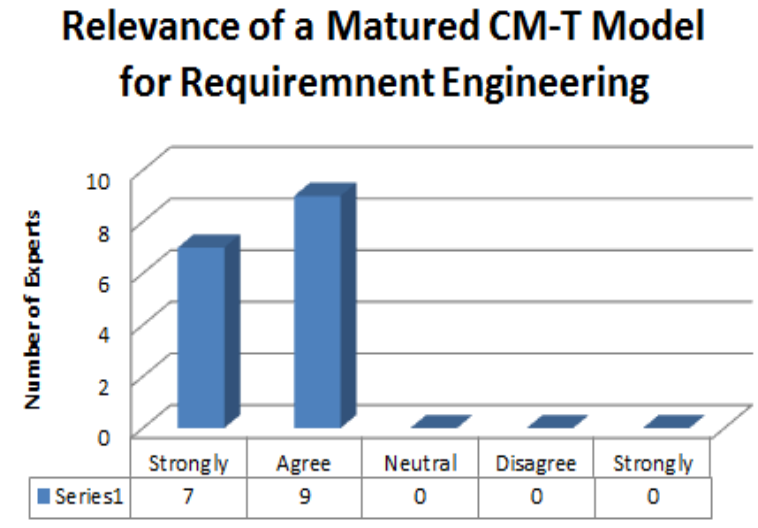
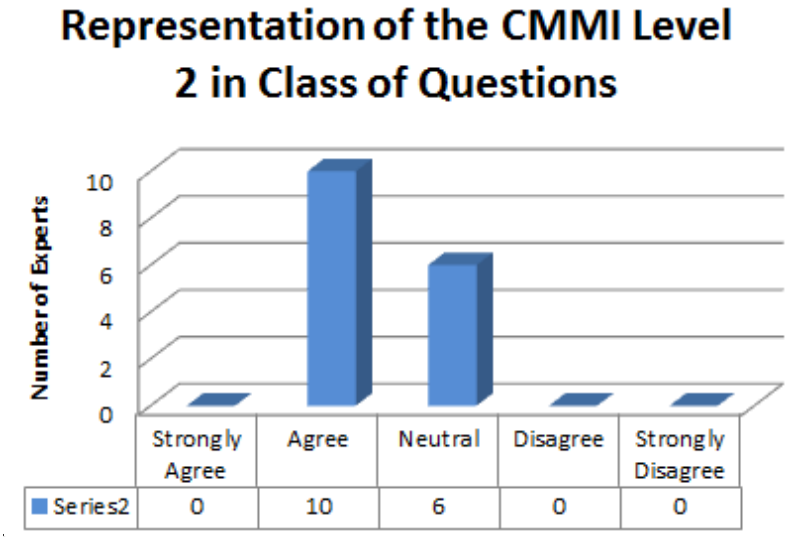
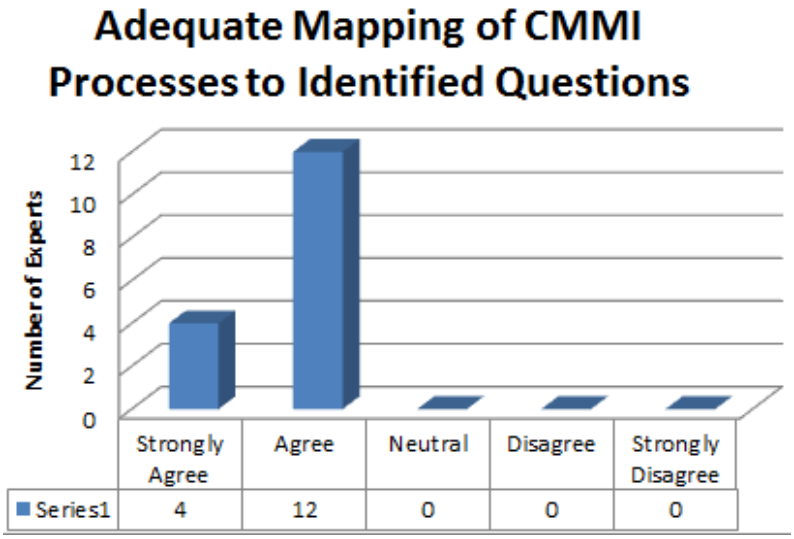


Figure 1: (A.1) Overview of CM-T Process Model

A.2 Adherence to CMMI Characteristics



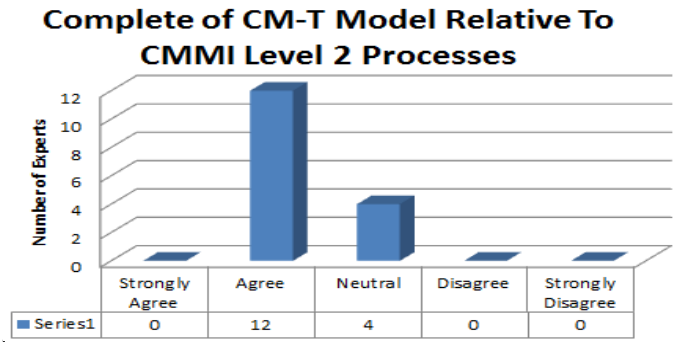
(a) The classification of questions is representative of the CMMI Level 2 goal?



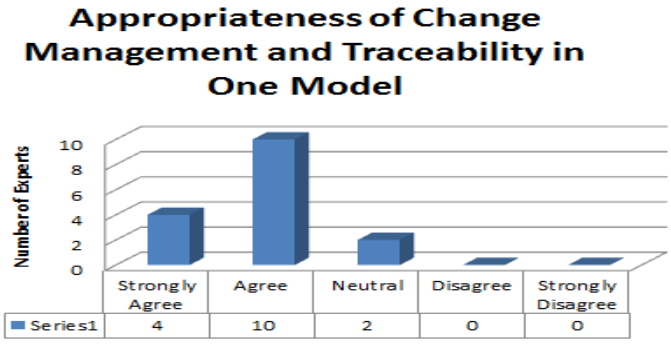
(b) The CMMI processes have been adequately mapped to the identified questions?

Figure 2: (A.2) Adherence To CMMI Characteristics

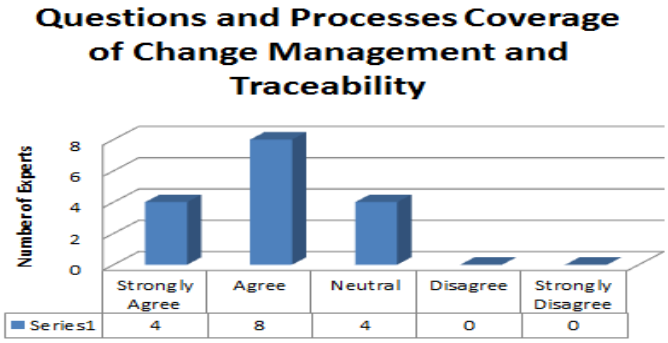
A.3 Limited Scope



(a) How complete is the CM-T model relative to the CMMI Level 2 processes?



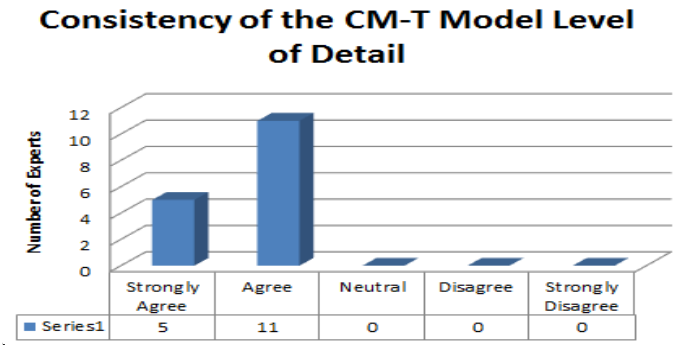
(b) How appropriate is it to include change management processes and traceability processes in one model?



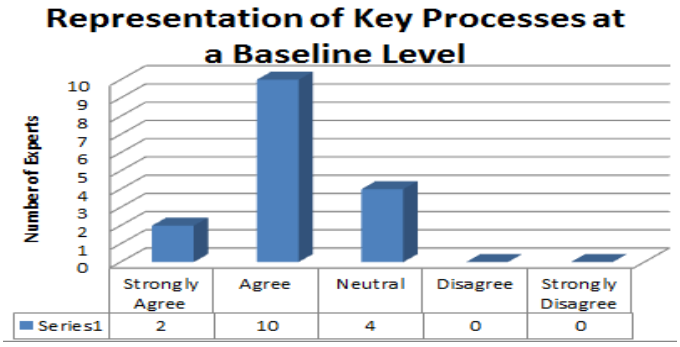
(c) How well do the questions and assigned processes cover the key activities in change management and traceability of requirements?

Figure 3: (A.3) Limited Scope

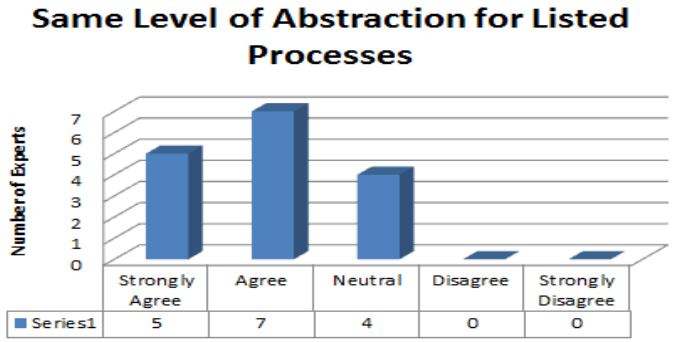
A.4 Consistency



(a) How consistent is the level of detail given within the CM-T model?



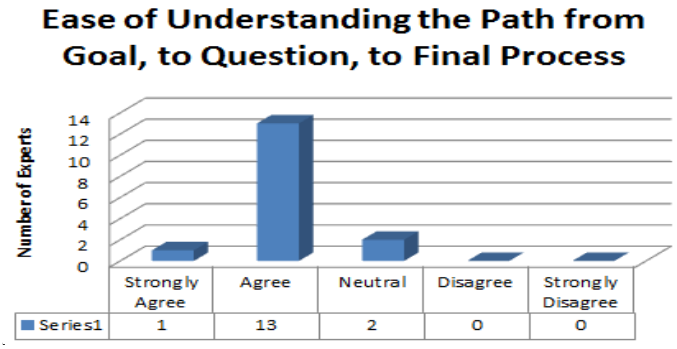
(b) All key processes are represented (at a baseline level)?



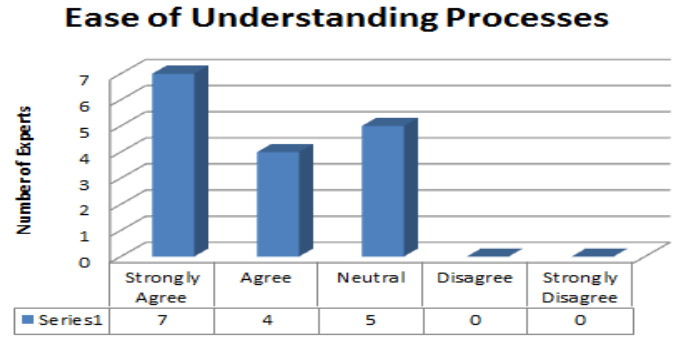
(c) All processes listed are at a similar level of abstraction?

Figure 4: (A.4) Consistency

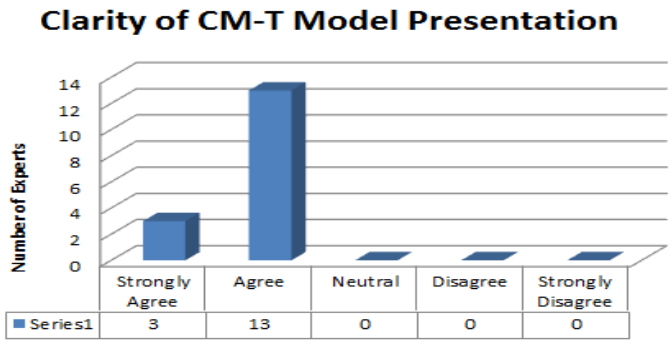
A.5 Understandability



(a) How easy is it to understand the path from initial goal, to question, to final process?



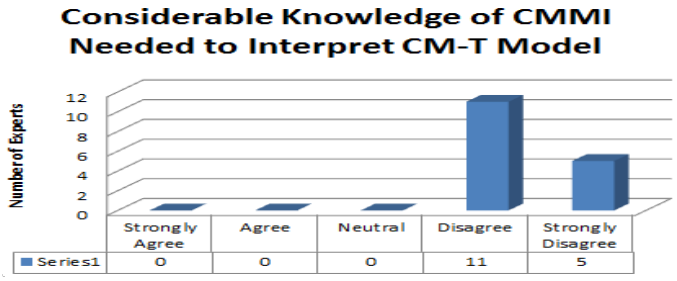
(b) Each individual process is easy to understand (i.e. they are clearly defined and unambiguous)?



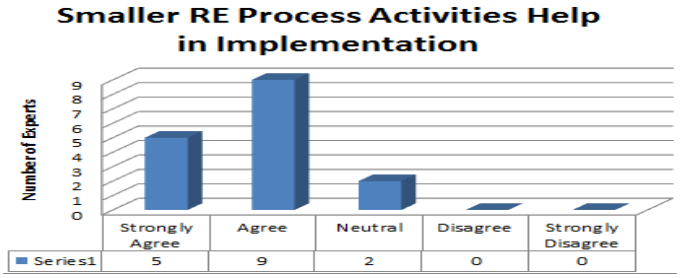
(c) How clear is this presentation of the model?

Figure 5: (A.5) Understandability

A.6 Ease of Use



(a) Do you think that a considerable amount of prior knowledge of CMMI is needed to be able to interpret the CM-T model?



(b) Dividing the RE process into smaller activities in this way will help practitioners to implement the process?

Figure 6: (A.6) Ease of Use

A.7 Verifiability

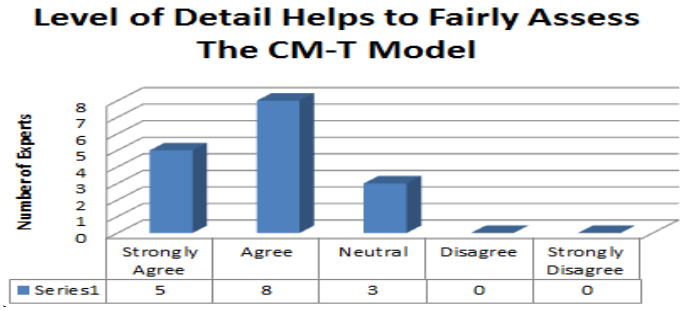


Figure 7: (A.7) Verifiability - How clear is this presentation of the model?

A.8 Constitution of the Expert Panel

Table 2: (A.8) The Expert Panel

Name of Participant	Current Institution	Position/Relevant Experience
1. M. Mehr (Ph.D.)	School of Computer Science, Newcastle University, UK	Researcher (expert in RE methods and Security)
2. S. Alajrami (Ph.D.)	Praqma, Norway	DevOps Consultant, and trained SPICE Assessor
3. R. Ebrahimy (Ph.D.)	DTU, Denmark	Post-Doc (expert in RE methods)
4. D. M. Dias (Ph.D.)	SIGMA Consult, Germany	IT Business Analyst and Programmer
5. R. Materre (Ph.D.)	School of Computer Science, Newcastle University, UK	Post-Doc (expert in RE methods)
6. S. F. Shahandashti (Ph.D.)	Department of Computer Science, University of York, UK	Lecturer (expert in RE methods and Security)
7. L. L. Bastos	Accenture, Newcastle, UK	Software Engineer and trained ISO 9001 Auditor
8. P. B. Mahama	Blue Oak System Ltd., Ghana	Quality Manager, IT Business Analyst - requirements, and Programmer
9. E. Dadzie	IT Systems Quality Control, United States Department of Agriculture, USA	Quality Manager and SPICE Assessor
10. E. Toreini (Ph.D.)	School of Computer Science, Newcastle University, UK	Post-Doc (expert in RE methods)
11. R. Ahmed	Department of Computer Science, Sulaimani Polytechnic University, Iraq	Lecturer (expert in RE methods and Security)
12. Z. Wen (Ph.D.)	School of Informatics, University of Edinburgh, UK	Post-Doc (expert in RE methods)
13. M. Dzandu	School of Computer Science, University of Reading, UK	Ph.D. Student and Lecturer (expert in RE methods)
14. Anonymous	TalkTalk, UK	SCRUM Master and Quality Manager
15. Anonymous	School of Computer Science, Tallinn University, Estonia	Senior Lecturer (expert in RE methods)
16. Anonymous	Institute for Applied Software Systems Engineering, Clausthal University of Technology, Germany	Senior Research Associate (expert in RE methods)

Appendix B: Reactive Middleware Implementation Details

B.9 Package Diagram of the Airlock Control System Case Study

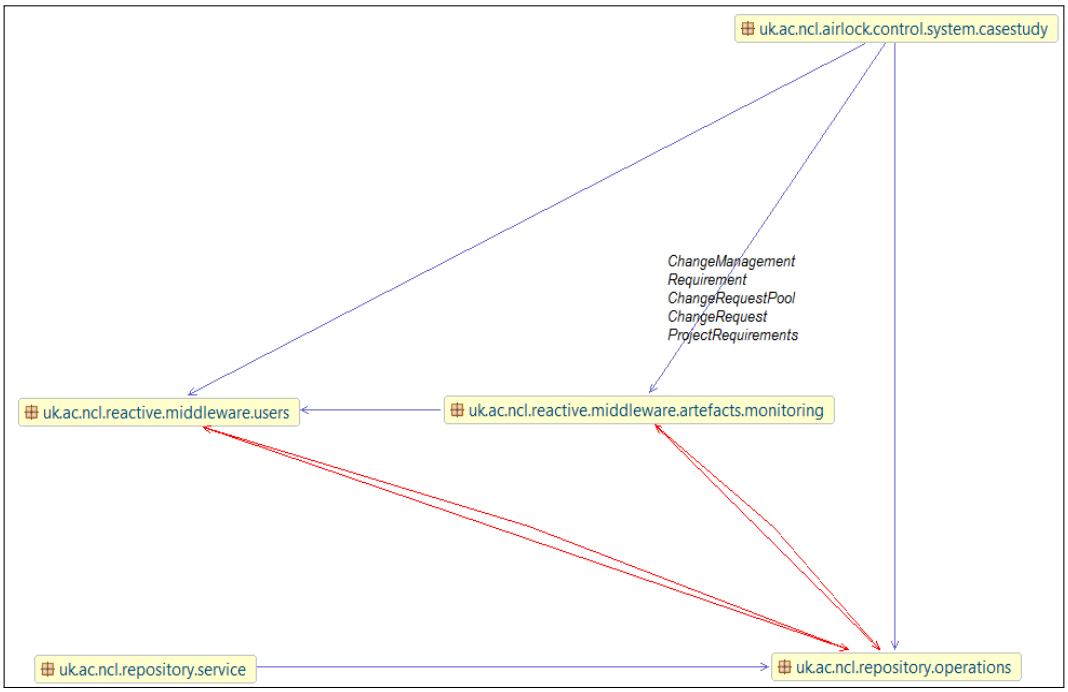


Figure 8: (B.9) Package Diagram of the Airlock Control System Case Study

B.10 Class Diagram of the Airlock Control System Case Study

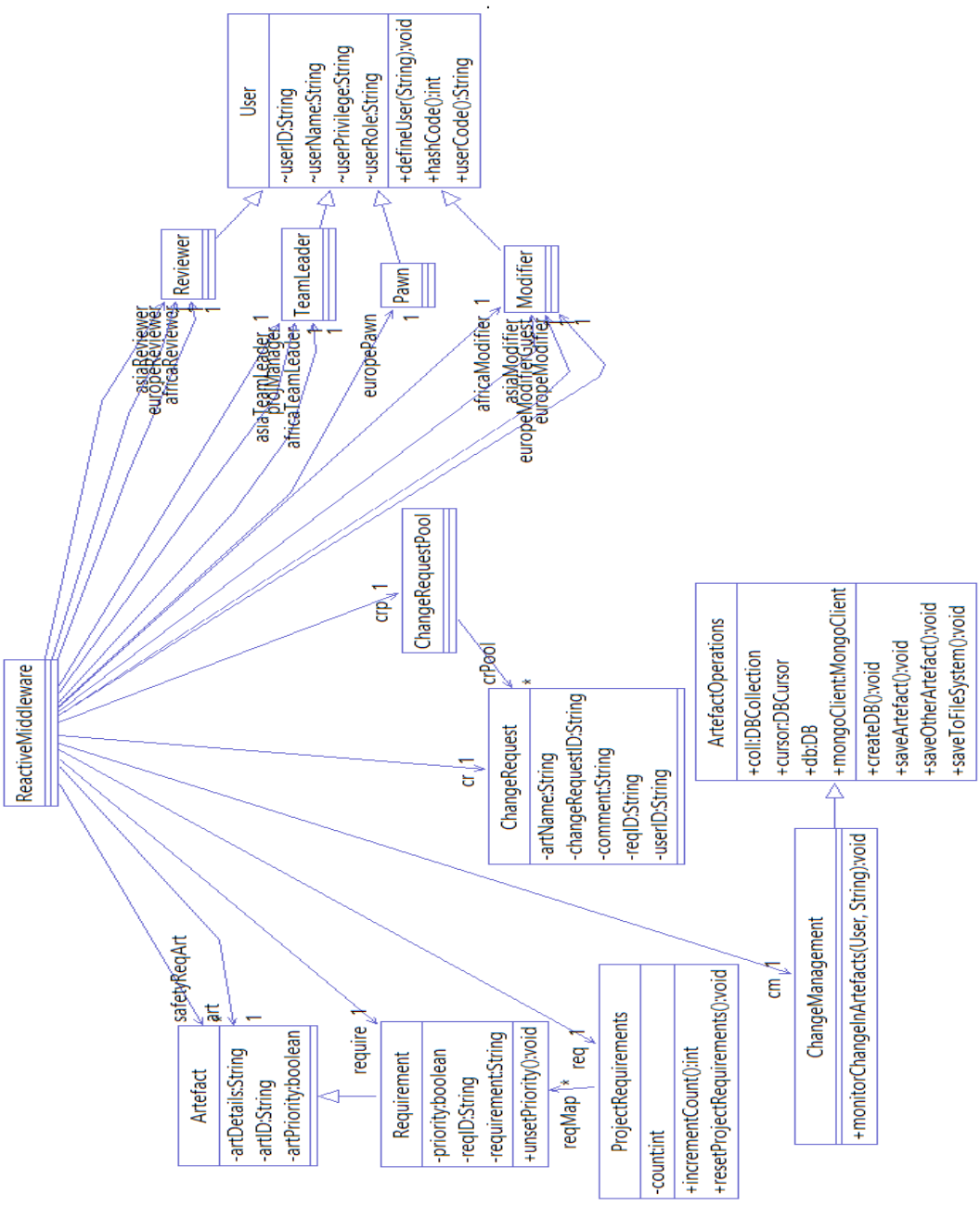


Figure 9: (B.10) Class Diagram of the Airlock Control System Case Study

B.11 Interaction Diagram of the Artefacts Monitoring System

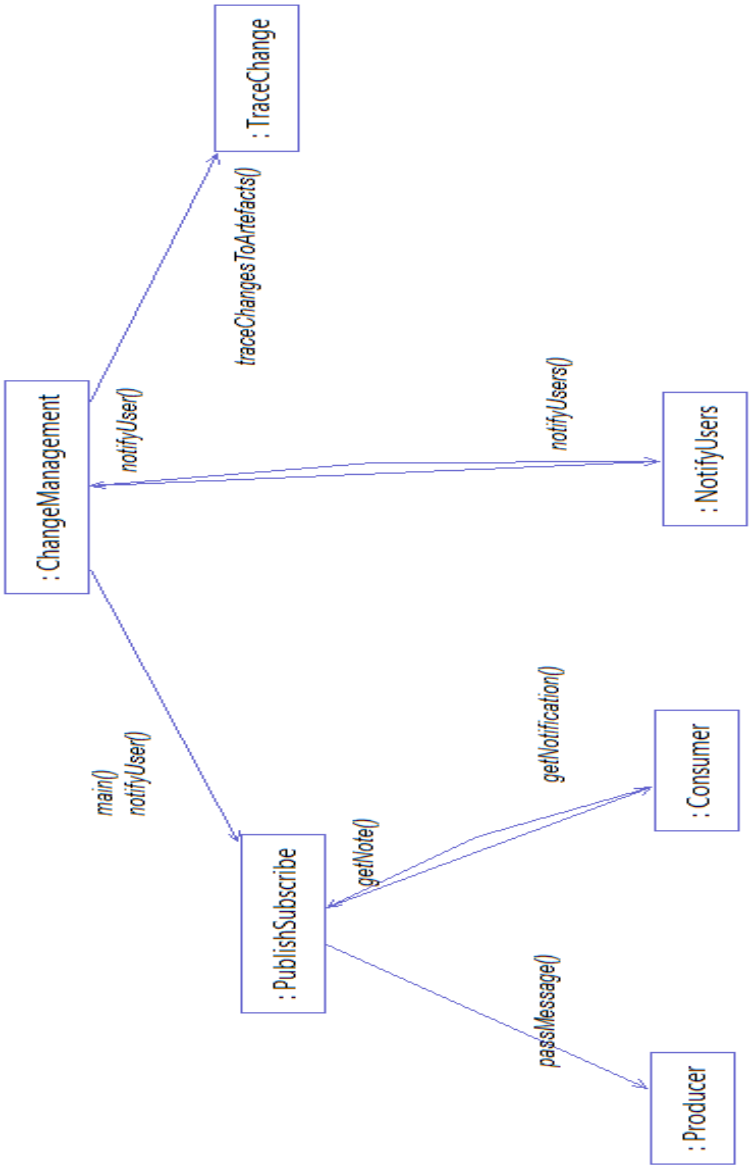


Figure 10: (B.11) Interaction Diagram of the Airlock Control System Case Study

B.12 Applying the System Engineering Toolbox to Formal Verification

The Toolbox system has been introduced in [99], [96], [98]. The Toolbox has been developed as a web service that is deployed to the Amazon Web Service' Elastic Cloud Compute's (EC2) environment. This system has been applied to formal verification involving our model checking web service and theorem proving web service. These applications are discussed below.

Model Checking Tool

In this project, a model-checker called ProB [81] is wrapped as a web service based on the REST protocol to become part of the architecture toolbox. A client plug-in created for the SafeCap IDE [95] sends specifications of railway signalling models for verification on the developed web service. The web service is deployed on Tomcat Server 6.0. The main CRUD (Create, Read, Update and Delete) operation used is the POST to receive REQUESTS from client IDEs as new entries of data (JSON file) into the web service. Another operation, GET is used to facilitate a RESPONSE with POST to return the verification results to the client.

Theorem Provers on the Cloud

In this work [97], we have created a theorem prover tool as part of our architecture toolbox and a plugin that connects the Rodin IDE [60] to this tool. The Rodin Platform supports modelling in the Event-B specification language and features a set of automated provers (pp, npp and AtelierB provers) as well an interactive proving environment. The tool is using the Why3 software [32] that brings together a collection of some well-known theorem provers (Alt-Ergo, Z3, Yices, Vampire, SPASS, etc.). The web service is hosted on the Amazon AWS cloud. The plugin maps Event-B mathematical language into the Why3 notation, the tool uses Why3 to implement subsequent translation into TPTP [213] and SMT-LIB formats compatible with a wide range of existing provers.

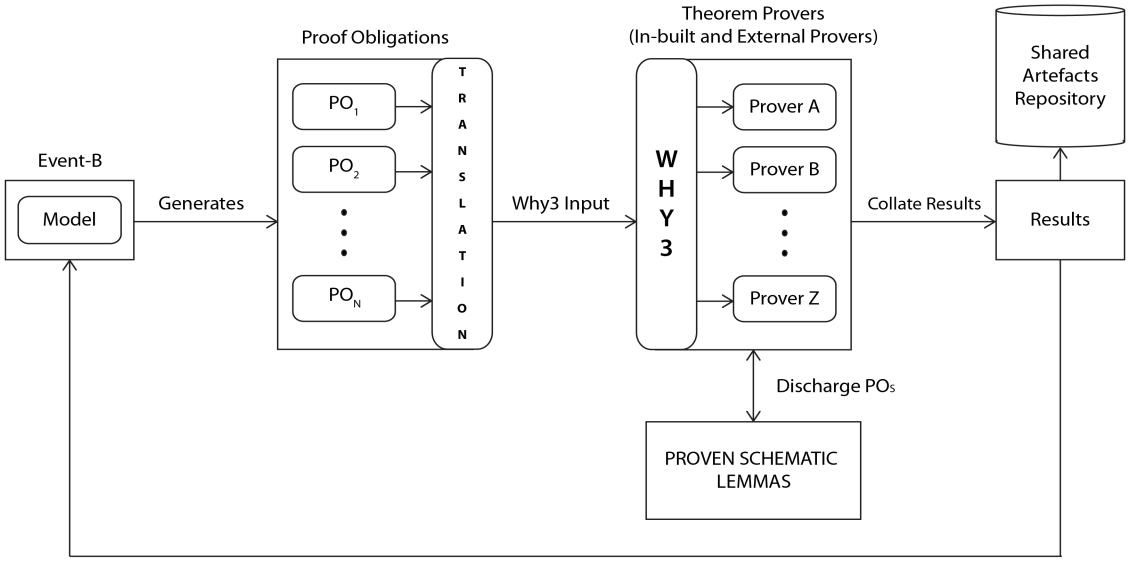


Figure 11: (B.12)Theorem Provers ToolBox Interactions in the AWS ECS Cloud

The main steps of this cloud-based verification tool-chain (see overview from Figure 11) are: (1) a client generates n verification conditions; (2) these are sent, individually to a scalable, cloud-based service; (3) each verification condition spawns, through a private sub-service, n prover instances; since provers are services themselves and the proof load is evenly distributed over physical nodes of the cloud; (4) prover results are collated and, if necessary, some prover instances are terminated before they complete; and (5) an adjudicated response is communicated back to the client.

Table 3: (B.13) Performance Benchmark

Model	**Total POs**	**Open, built-in**	**Open, built-in FA**	**Open, built-in + z3($c \cdot *$)**
prime15r3	625	281	18	201
paxos3a3	348	121	4	27
fishers	82	14	0	14

There are two significant points presented by this project. First is the fact that a prover is internally accessed as a service to make use of cloud elasticity in resource scaling. Second is the prover managing facility that distributes verification conditions to prover services.

The connection with the Rodin IDE provides an immediate access to hundreds of formal specifications containing many thousands of verification conditions. It also significantly strengthens the Platform (i.e. *Open, built-in + FA*) proving capability as illustrated in Table 3. In one of the models, *fishers*, the addition of the service makes the proof completely automatic; even more impressive is the fact that 14 previously undischarged proofs included two long interactive proofs which, originally, took several days to complete. For the case of *paxos3a3* model, the service proofs all but 4 POs which are genuinely challenging and require manually setting up an induction scheme. Finally, model *prime15r3* had 18 proof obligations (POs) undischarged due to a combination of incomplete axiomatisation of the Event-B language and a fairly short prover time-out hard-wired into the Platform. The last column of the table gives the performance of a scenario made of a single, though quite capable prover, - z3 [144].

We have also created and experimented with a Shared Artefact Repository (see Section 4.2.3) as part of our Reactive Architecture. The cloud-based theorem prover service keeps a detailed record of all artefacts for every proof attempt in the shared artefact repository. These artefacts are mainly proof obligations, supporting lemmas and translation rules. Provisions are made to obfuscate sensitive proof obligations. The repository is a relational database service running on the Amazon AWS cloud [9]. Here, all queries from the prover service are made as HTTP requests using the repository's URI.

Appendix C: Requirements and Sample Use Cases of the Reactive Architecture

C.13 Requirements

We present the requirements of the Reactive Architecture below:

(a) The Reactive Architecture must store all artefacts created in all of its components.

(b) It must monitor and trace all changes to these artefacts to inform system developers.

(c) The Reactive Architecture must support at least 20 users concurrently.

(d) The Reactive Architecture must provide capacity to scale quickly to accommodate changing demands of system developers, and failures.

(e) The Reactive Architecture must enable heterogeneous access and analysis operations on saved artefacts.

(f) All saved artefacts must be backed up asynchronously to facilitate roll-back of artefacts.

(g) Critical systems that manage developers and artefacts must not constitute a single point of failure which will affect the uptime of the system and the Reactive Architecture.

(h) The toolbox must facilitate sequential and parallel execution of tools in a workflow manner.

(i) The Reactive Architecture must provided a high capacity and dedicated channel to coordinate real-time analysis on artefacts for local client computers and on remote cloud environment.

(j) The Reactive Architecture must gather dependability metrics from several virtual machines, and perform a synchronous analysis of these metrics.

(k) Security mechanisms must not degrade defined performance threshold. Specifically, response time for create, delete, update, and display artefact operations should not exceed 5 seconds at peak cloud period and less than 1 second during off-peak period.

(l) The Reactive Architecture must provide high performance and availability to allow it to keep up with the sturdy stream of data and operations on artefacts from the system engineering processes.

C.14 Use Cases

To provide a clarification of the functional requirements of the Reactive Architecture, we present some of its use cases. The set of use cases presents possible sequences of interactions between the components of the Reactive Architecture, and with the clients of the architecture in the cloud environment. These identified interactions or activities in the use cases have significance to the clients of the Reactive Architecture. Some of such use cases are: client management (i.e. registration, collaboration, etc.), how to add tools to the toolbox, saving artefacts, sharing artefacts, upload and download artefacts, notification to stakeholders resulting from change to artefacts, and repository back-up or failover support. These use cases are briefly introduced below.

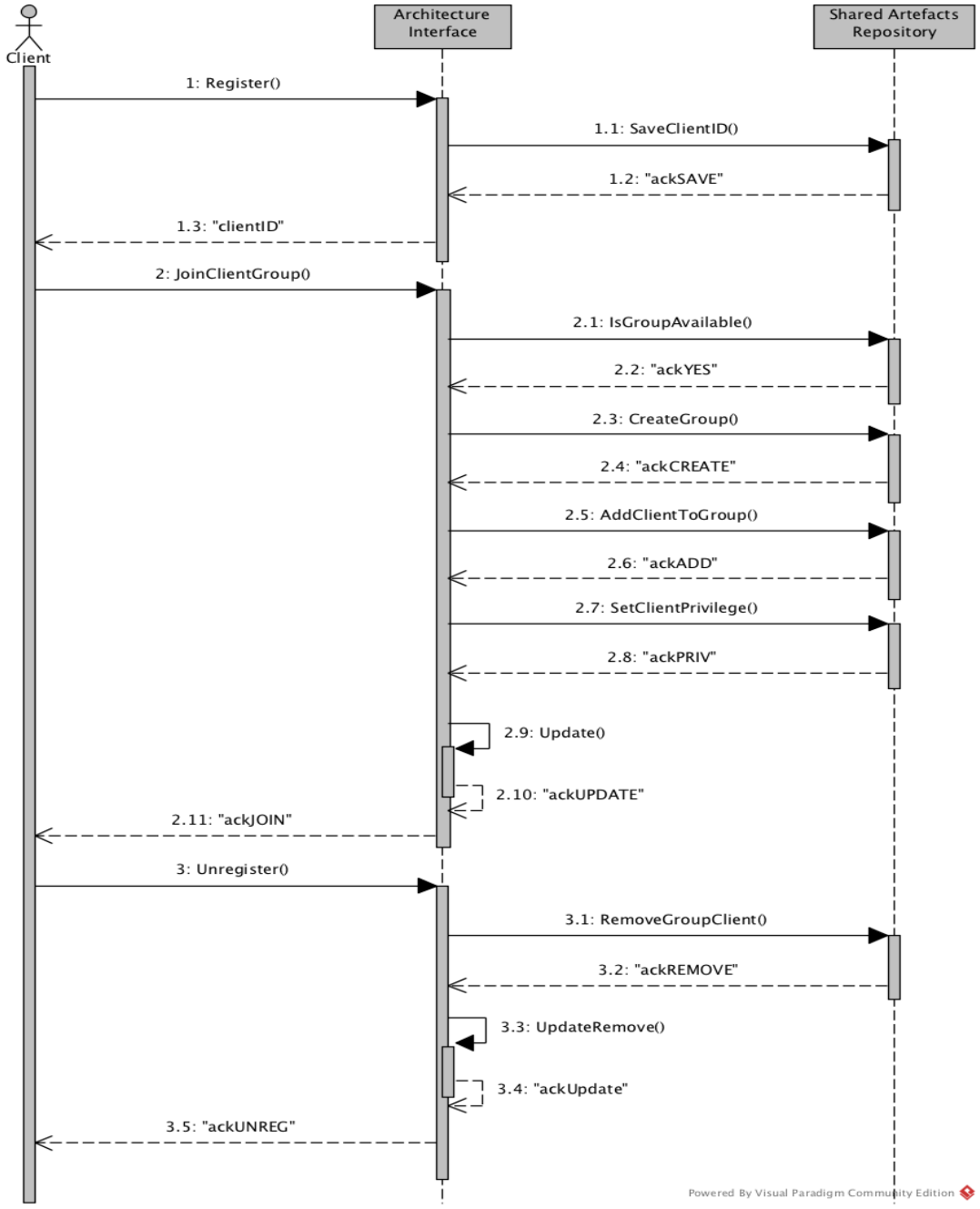


Figure 12: (C.14) UML Sequence Diagram: Client Management

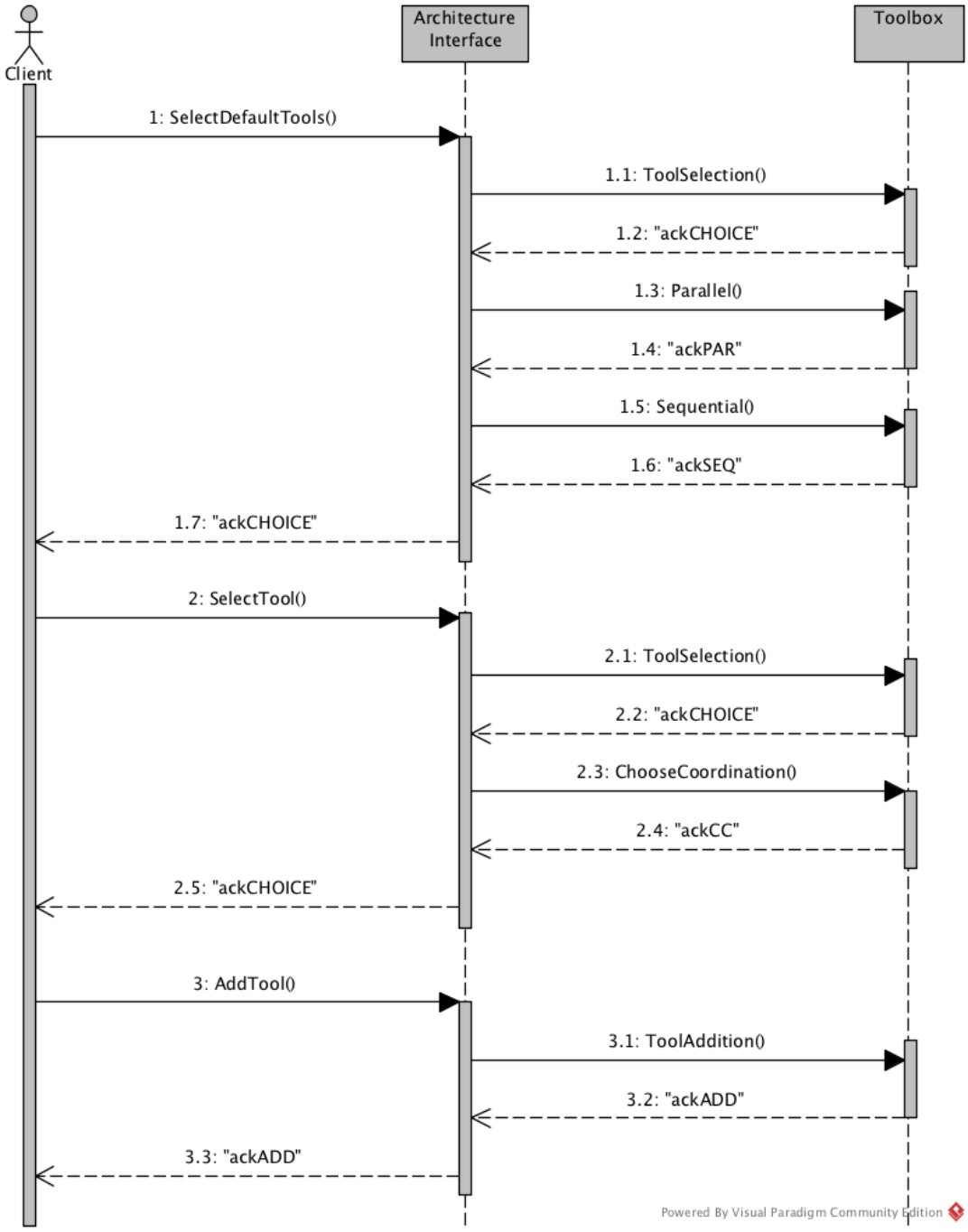


Figure 13: (C.15) UML Sequence Diagram: Adding Tools to the Supporting Toolbox

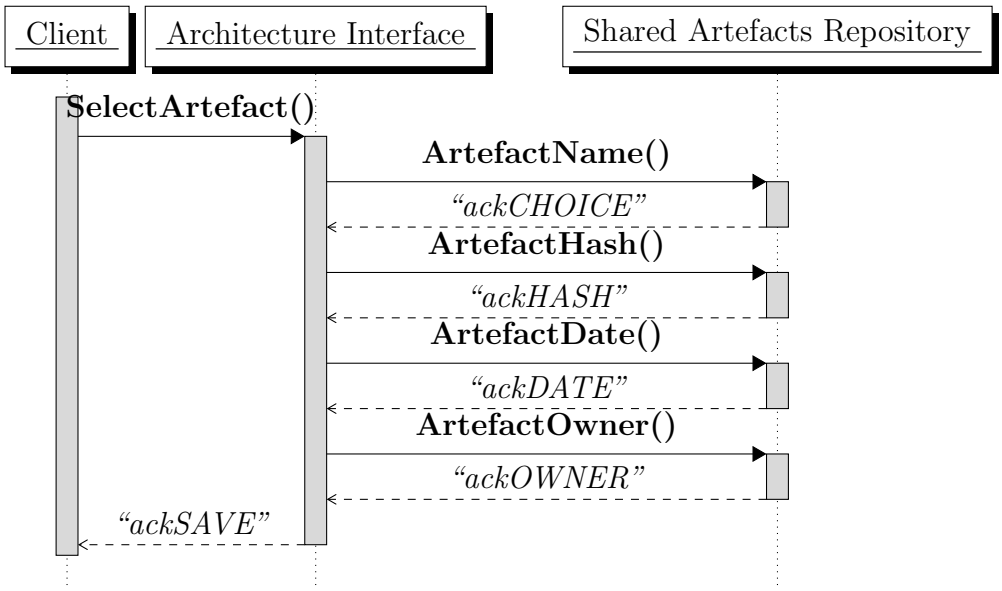


Figure 14: (C.16) UML Sequence Diagram: Saving Artefacts

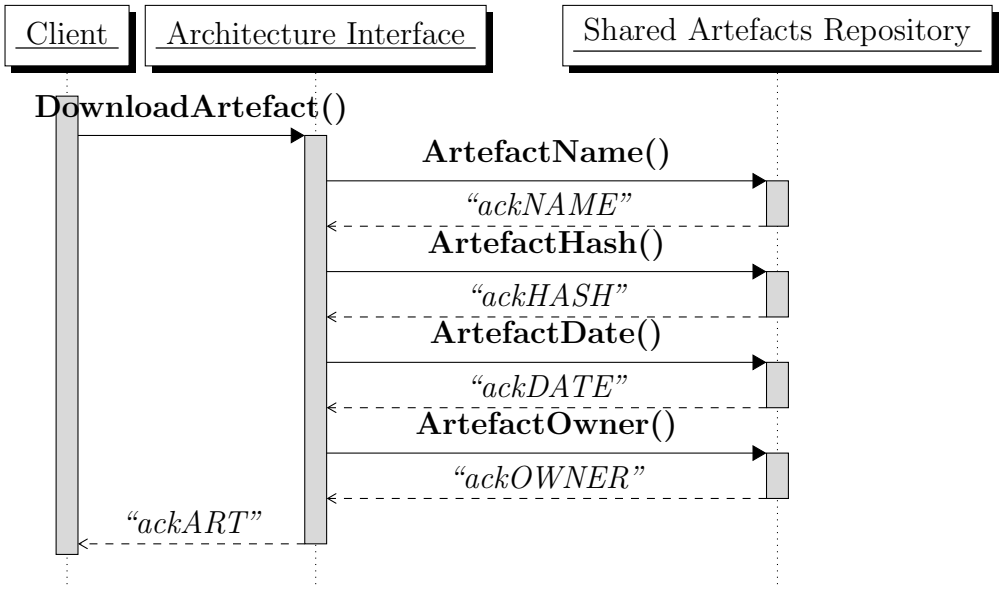


Figure 15: (C.17) UML Sequence Diagram: Download Artefact

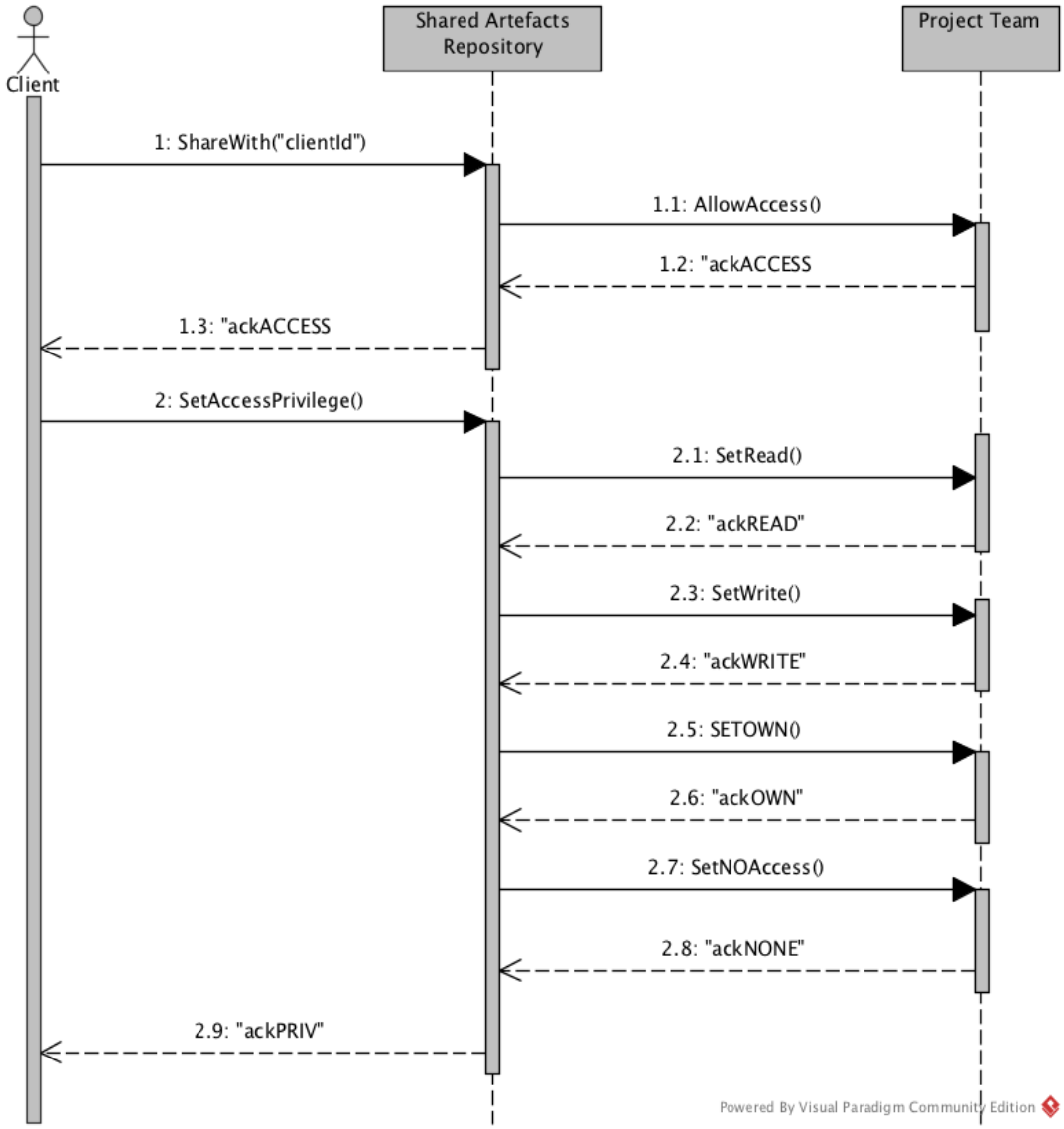


Figure 16: (C.18) UML Sequence Diagram: Sharing Artefacts

C.14.1 Client Management

Every client (i.e. system developer) of this architecture should register to use the Reactive Architecture. All transactions with the architecture or other clients will bare a unique identification for authentication. In the case of the formation

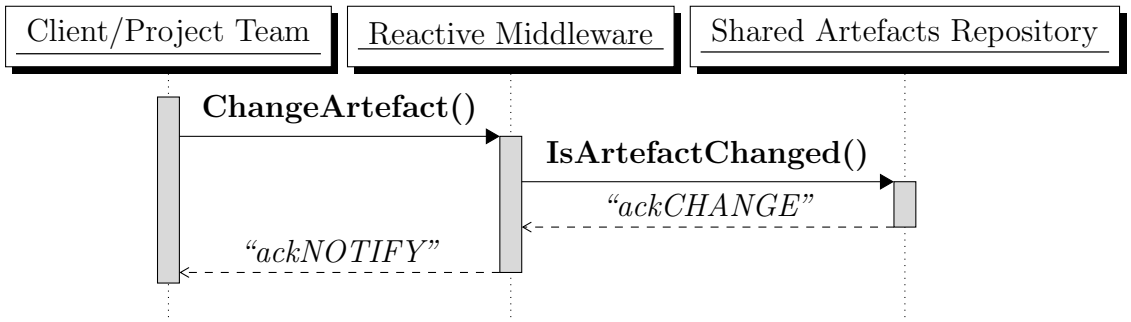


Figure 17: (C.19) UML Sequence Diagram: Change Management of Artefacts

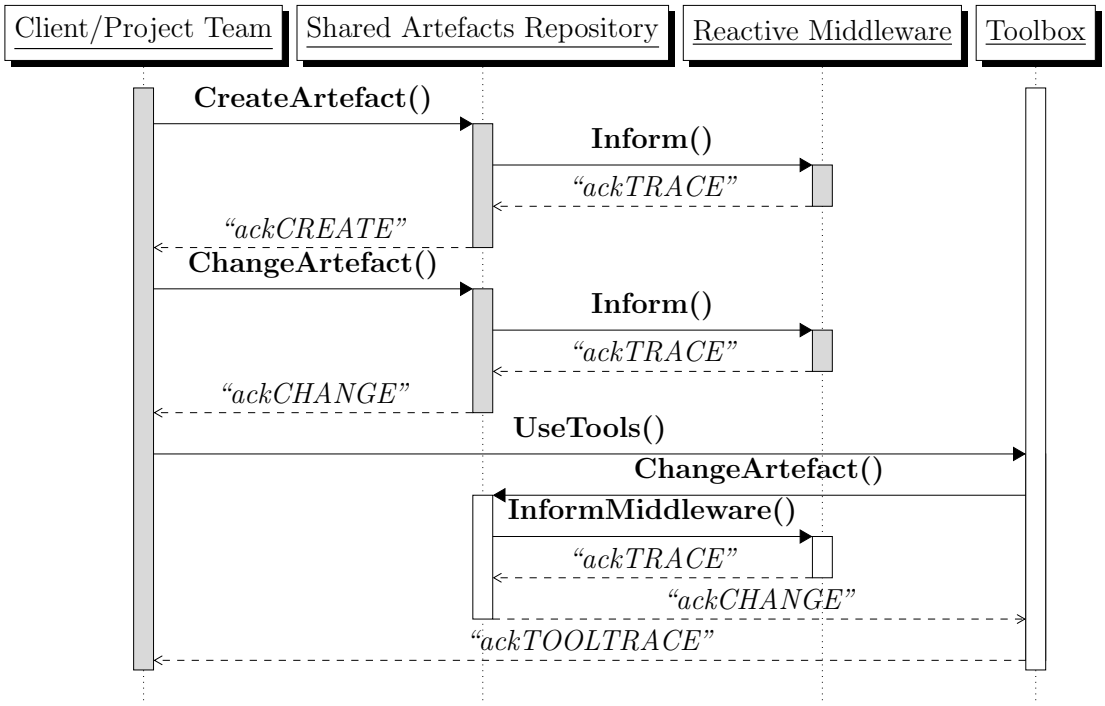


Figure 18: (C.20) UML Sequence Diagram: Traceability of Artefacts

of a collaborating group for a project, the group leader will *invite* or *add* system developers to a group using their unique identifications. In such a situation, a group name will be provided including the client identification of the team creator/leader. The client identification also helps to track the activities of system developers in a project, which is necessary for *traceability*. It must be mentioned that clients can switch from local system development to cloud-

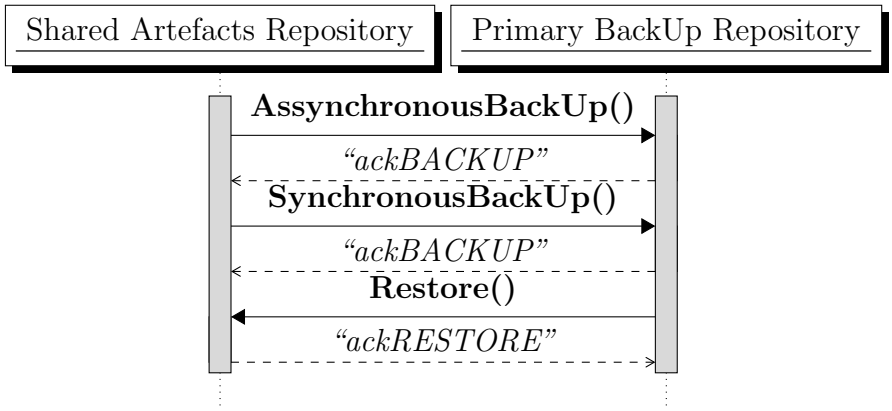


Figure 19: (C.21) UML Sequence Diagram: Primary Repository Back-Up

based development provided by the Reactive Architecture. Figure 12 provides a sequence diagram that illustrates the introduced use case.

C.14.2 Adding Tools to the Supporting Toolbox

The Toolbox provides a set number of tools by default. These tools are classified based on the development phase they support, and there are different versions to also support back-ward compatibility. However, client can add new tools to the Toolbox (see Figure 13). Also, some of the tools can be composed either in a *parallel* or *sequential* order. This facilitates a form of workflow among the tools.

C.14.3 Saving Artefacts

Most activities with the Reactive Architecture generate a form of artefact. These artefacts are saved to the Shared Artefacts Repository (see Figure 14). External artefacts have to be saved by the system developer. Here, the client selects the artefacts on his local computer, provide details about the artefacts and the client identification, in order to save those artefacts.

C.14.4 Downloading Artefacts

System developers can download artefacts from the Shared Artefacts Repository. This activity requires the system developer to either provide the name of the artefact or select the artefact from a list of all permissible artefacts (see Figure 15). The downloaded artefact is stored on the local computer of the client.

C.14.5 Sharing Artefacts

Saved artefacts can be shared by the system developer who created them, with other collaborating system developers. Here, the creator of the artefact provides the client identifications of the collaborating developers, which are then assigned to the artefacts to allow access. The system developer also assigns varying levels of permissions (i.e. read, write, own) to the artefacts for the collaborating developers. The collaborating developers can now access these artefacts, but within the restrictions of their permissions (see Figure 16).

C.14.6 Change Management of Artefacts

Artefacts such as model specifications and their composing elements are monitored to observe changes made to them. So whenever changes are made to an artefact, a notification is triggered to all collaborating system developers who have permissions (through subscription) to the artefact. Here, a Publish/Subscribe mechanism is used to distribute these notifications (see Figure 17).

C.14.7 Traceability of Artefacts

The traceability scenario is directly linked to the artefact change management scenario presented above. All changes to artefacts are traced. The tracing process identifies all "subscribers" to an artefact, name of system engineering team, name of artefact, unique identification of the artefacts (i.e. name and hash) that are derived or dependent on the artefacts to be traced, and all

associated artefacts (see Figure 18). Here, "subscribers" to an artefact can either be a client or tool. These are the primary change agents.

C.14.8 Primary Back-Up Repository

A back-up repository is provided for the Shared Artefacts Repository. The group leader(s) set the frequency of back-up. The frequency options are for asynchronous and synchronous back-ups. Synchronous back-up is preferred to guarantee up to date or real-time back-up, but it introduces latency as it interrupts the operations of the Shared Artefacts Repository. Asynchronous back-up provides timely and off-peak back-up of all the contents of the Shared Artefacts Repository. The main function of the Back-up Repository is to provide fail-over support for the Shared Artefacts Repository (see Figure 19).

C.15 Questionnaire for cloud-ATAM Stakeholders' Brainstorm

CSR Newcastle University

Participant Identification Number: …….. (To be assigned by the Researcher)

Name of Researcher: **David Ebo Adjepon-Yamoah**

1. I confirm that I have read and understand the study description for this study. I have had the opportunity to consider the information, ask questions and have had these answered satisfactorily.	☐
2. I understand that my participation is voluntary and that I am free to withdraw at any time, without giving any reason.	☐
3. I understand that any information given by me will be anonymised and may be used in future reports, articles or presentations by the research team.	☐
4. I agree to take part in the above study.	☐

Click here to enter text.	Click here to enter text.	*Click here to enter text.*
Name of Participant (optional)	Date	Signature
Click here to enter text.	Click here to enter text.	*Click here to enter text.*
Researcher	Date	Signature

Study description:
In this study, we argue that the existing architecture evaluation methods have limitations when assessing architectures interfacing with unpredictable environments such as the Cloud. The unpredictability of this environment is attributed to the dynamic elasticity, scale, and continuous evolution of the cloud topology. As a result, architectures interfacing such unpredictable environments are expected to encounter many uncertainties. It is however, important to focus on, and present holistic approaches combining aspects of both dynamic and static analysis of architecture dependability attributes. This exercise introduces a derived methodology - *cloud-ATAM* – from the Architecture Trade-off Analysis Methodology (ATAM) for evaluating the *trade-off* between multiple dependability quality attributes (i.e. *availability* and *performance*) of a cloud-based Reactive Architecture.

In this *stakeholders' group work*, you will be introduced to ATAM, the Reactive Architecture and our analysis with the derived methodology - *cloud-ATAM*. To these three (3) sections, you will be asked to provide some answers to some questions. We seek to solicit your impression/opinion of our study based on your varied expertise. Thank you for taking time to participate in our study!

Centre for Software Reliability, School of Computing Science, Newcastle University, Newcastle Upon Tyne, NE1 7RU
Tel: +44 (0) 191 222 7997 • E-mail: csr@ncl.ac.uk • Web: www.ncl.ac.uk/csr 26 July 2016

CSR Newcastle University

Participant Identification Number: …….. (To be assigned by the Researcher)

Stakeholder's Expertise

	None	Basic	Advanced	Academic Exposure	Industrial Exposure
1. Expertise in Architecture Design	☐	☐	☐	☐	☐
2. Expertise in Architecture Analysis and/or Evaluation	☐	☐	☐	☐	☐
3. Expertise in ATAM	☐	☐	☐	☐	☐

SECTION 1: Presentation of ATAM

4. The overview of *architecture evaluation* and *ATAM* were presented reasonably well?

Yes ☐ **No** ☐

	Strongly agree	Agree	Neutral	Disagree	Strongly disagree
5. Quality attributes (i.e. availability, performance, etc.) play a critical role in architecture evaluation	☐	☐	☐	☐	☐
6. Scenario-based architecture evaluation methods are adequate in analysing software architecture of varying sizes	☐	☐	☐	☐	☐
7. ATAM presents a matured/convincing approach to architectural evaluation	☐	☐	☐	☐	☐

8. Do you have any comments?

CSR Newcastle University

Participant Identification Number: …….. (To be assigned by the Researcher)

SECTION 2: Reactive Architecture

9. The Reactive Architecture was clearly presented? **Yes** ☐ **No** ☐

		Strongly agree	Agree	Neutral	Disagree	Strongly disagree
10.	The requirements of the Reactive Architecture are representative enough	☐	☐	☐	☐	☐
11.	The presented constraints and focal quality attributes are relevant to the architecture	☐	☐	☐	☐	☐
12.	The architecture components, their relationships, and initial scenarios are useful in understanding the Reactive Architecture	☐	☐	☐	☐	☐

13. Do you have any comments?

SECTION 3: cloud-ATAM Evaluation and Results

		Strongly agree	Agree	Neutral	Disagree	Strongly disagree
14.	The cloud-ATAM was clearly presented	☐	☐	☐	☐	☐
15.	The quality attribute characterisation was well presented	☐	☐	☐	☐	☐
16.	The attribute-specific questions and the identified approaches provide adequate coverage of the quality attribute characterisation	☐	☐	☐	☐	☐
17.	The presented scenario (generated from the utility tree) was adequately analysed	☐	☐	☐	☐	☐
18.	The reasoning behind the analysis process was sound	☐	☐	☐	☐	☐

Participant Identification Number: …….. (To be assigned by the Researcher)

19. Do you have any comments?

20. Do you have any general comments?

CONTACT: Thank you for your participation in this study. If you have further questions about the study, please contact David Ebo Adjepon-Yamoah (*d.e.adjepon-yamoah@ncl.ac.uk*). In addition, if you have any concerns about any aspect of the study, you may contact Prof. Alexander Romanovsky, Room 11.10, CSR, Claremont Tower, School of Computing Science, Newcastle University, Email: *alexander.romanovsky@ncl.ac.uk*.

Appendix D: Cloud Accountability System Implementation Details

D.16 Java Code Snippet for Metrics Data Collection on AWS

Listing 1: Snippet of Java Code for Collecting "CPUUtilization" Metric using AWS/EC2 CloudWatch API

```
1  private static GetMetricStatisticsRequest request(final
2     String instanceId) {
3          final long twentyFourHrs = 1000 * 60 * 60 * 24;
4          final int oneMin = 1 * 60;
5          return new GetMetricStatisticsRequest()
6                      .withStartTime(new Date(new Date()
7                      .getTime()- twentyFourHrs))
8                      .withNamespace("AWS/EC2")
9                      .withPeriod(oneMin)
10                     .withDimensions(new Dimension()
11                     .withName("InstanceId")
12                     .withValue(instanceId))
13                     .withMetricName("CPUUtilization")
14                     .withStatistics("Average", "Maximum")
15                     .withEndTime(new Date()); }
16
17 private static GetMetricStatisticsResult result(
18         final AmazonCloudWatchClient client,
19         final GetMetricStatisticsRequest request) {
20         return client.getMetricStatistics(request); }
```