Continuous Evaluation Framework for Software Architectures

An IoT Case

Dalia Sobhy

Supervisor: Dr. Rami Bahsoon
Dr. Leandro Minku

School of Computer Science
University of Birmingham

A thesis submitted to the University of Birmingham for the degree of
Doctorate of Philosophy

June 2019

To Allah, my loving parents, family, and friends . . .

Acknowledgements

First and foremost, my enormous gratitude and thanks go to Allah for his ever blessings. It is with immense gratitude that I acknowledge all the people who have helped and inspired me during my PhD studies.

Special thanks to my supervisors: Dr. Rami Bahsoon and Dr. Leandro Minku, for their continuous support and dedication, patience and motivation. Their guidance helped me in every aspect of my research work. Their motivation and care have motivated me to keep on track. This work would not have been possible without their sound advice and encouragement.

I wish to thank, my advisors, Prof. Rick Kazman and Dr. Tao Chen, for their guidance during my research and studies. With their enthusiasm and rigorous attitude of scholarship, they showed me how to develop and enrich my work.

Special thanks to the members of my Thesis Group who took time to provide guidance, especially Sara, Faisal, Carlos, and Maria. I would also like to thank Dr. Dave Parker for his timely comments, and Dr. Per Kristian Lehre for his insightful and encouraging comments. Their discussions and suggestions have greatly guided my research and helped in paving the way.

Acknowledges are given to all the administrative staff of the School of Computer Science for their great support, kindness and welcome throughout the PhD course, especially Sarah Brookes. Thanks also to Peter Hancox, Dave Parker and Steve Vickers for their role as Research Students Tutor, as well as Jon Rowe and Andrew Howes for their role as Head of School. Thanks to the Research Committee for their support and listening, especially Achim Jung.

Many thanks also to the people I met during my research visits. Thanks to Microsoft Research summer programs in Russia and UK, which provided me with the opportunity to meet new researchers, especially, Dr. Judith Bishop, Dr. Nilanjan Banerjee, Dr. Luca Mottola, Dr. Carlo Alberto, and Dr. Katja Hofmann. The discussions in these programs had helped me a lot in developing my research. The experience that I also gained during those programs was very worthwhile and everyone there was very welcoming and always willing to help.

I would also like to express my sincere gratitude to the organizers and tutors of the summer PhD school of software architecture, in Leiden, Holland. A special thanks to Dr. Patricya Lago, Dr. Paris Avgeriou, and Dr. Philippe Kruchten. This summer school has advanced my work by providing a clear linkage between academic studies and industrial cases.

I am indebted to my colleagues, for making our room a convivial place to work. I wish to thank them also for their continuous help and support throughout my years of study, especially Noha Ghatwary, Ingy Sarhan, and Noha Seddik.

Last and most importantly, I wish to thank my parents, for their endless support, unconditional love and guidance. They were always standing by my side. A special thanks goes to my Grandma, because of her care and continuous prayers. I would also like to express my gratitude to my brother, family, and friends for their tolerance, inspiration and support through all these years.

Abstract

Context. Design-time evaluation is essential to build the initial software architecture to be deployed. However, experts' design-time assumptions are unlikely to remain true indefinitely in systems characterized by scale, heterogeneity, and dynamism (e.g. IoT). Experts' design-time decisions can be thus challenged at run-time. A continuous architecture evaluation that systematically intertwines design-time and run-time evaluation is necessary. However, the literature lacks examples on how continuous evaluation can be realized and conducted.
Objective. This thesis proposes the first continuous architecture evaluation framework.
Method. The framework is composed of two phases: design-time and run-time evaluation. The design-time evaluation enables the necessary initial step of system design and deployment. Run-time evaluation assesses to what extent the architecture options adopted at design-time and other potential options, perform well at run-time. For that, the framework leverages techniques inspired by finance, reinforcement learning, multi-objective optimisation, and time series forecasting. The framework can actively track and proactively forecast the performance of architecture decisions and detect any detrimental changes. It can then inform deployment, refinement, and/or phasing-out decisions. We use an IoT case study to show how continuous evaluation can fundamentally guide the architect and influence the outcome of the decisions. A series of experiments is conducted to demonstrate the applicability and effectiveness of the framework.
Results. The design-time evaluation was able to evaluate the architecture options under uncertainty and shortlist candidates for further refinement at run-time. The run-time evaluation has shown to be effective. In particular, it enabled a significant improvement in overall quality (about 40-70% better than reactive and state-of-the-art approaches in some scenarios), with enhanced architecture's stability. It was also shown to be scalable and robust to various noise levels. In addition, it provides the architect with flexibility to set a monitoring interval to profile the quality of candidates and has parameters that enable the architect to manage the trade-off between architecture stability and learning accuracy.
Conclusion. The proposed continuous evaluation framework could potentially aid the architect in evaluating complex design decisions in dynamic environments.

Table of contents

List of figures xv

List of tables xx

Nomenclature xxiii

1 Introduction 1

1.1 Aim and Objectives . . . 3
1.2 Research Questions . . . 4
1.3 Research Methodology . . . 5
1.4 Thesis Contributions . . . 6
1.5 Thesis Storyline . . . 7
1.6 Publications . . . 12
1.7 Thesis Roadmap . . . 13

2 Evaluation of Software Architectures under Uncertainty: A Systematic Literature Review 16

2.1 Introduction . . . 17
2.1.1 Preliminaries and Basic Concepts . . . 19
2.1.1.1 Architecture Design Decisions . . . 19
2.1.1.2 Software Architecture . . . 19
2.1.1.3 Quality Attribute . . . 20
2.1.1.4 Uncertainty in Architecture Evaluation . . . 20
2.1.1.5 Stakeholder . . . 20
2.2 Systematic Literature Review Process . . . 20
2.2.1 SLR Protocol . . . 21
2.2.2 Inclusion and Exclusion Criteria . . . 21
2.2.2.1 Inclusion Criteria . . . 21

2.2.2.2 Exclusion Criteria 22
2.2.3 Search Strategy 22
2.2.3.1 Keyword Selection 23
2.2.3.2 Bibliographical Sources 24
2.2.4 Search Execution 25
2.2.5 Quality Assessment 25
2.2.6 Data Extraction process 27
2.3 Overview of the included studies 27
2.3.1 Distribution of Studies over Publication Channels 27
2.3.2 Distribution of Included Studies Through the Years 28
2.3.3 Citation Rate of Included Studies 30
2.4 Data Extraction Results 32
2.4.1 Classification Framework 32
2.4.2 Design-Time Architecture Evaluation 34
2.4.2.1 Utility-Driven 34
2.4.2.2 Economics-Driven 40
2.4.2.3 Relevance to Continuous Evaluation 42
2.4.3 Run-time Architecture Evaluation 42
2.4.3.1 Utility-driven 42
2.4.3.2 Learning-driven 46
2.4.3.3 Relevance to Continuous Evaluation 48
2.4.4 Limitations of the Review 48
2.5 Relevant Fields 53
2.6 Related Reviews 53
2.7 Discussion and Recommendations for Future Research 54
2.7.1 Research Area Maturation 55
2.7.2 Leveraging Existing Approaches To Develop A Continuous Evaluation Framework 56
2.7.3 Finding The Necessary Ingredients For Developing A Continuous Evaluation Framework 58
2.8 Conclusion 59

3 IoT as a Motivating Case Study 61
3.1 IoT challenges 61
3.2 Introducing the IoT case: *iTransport* 62

3.3 A hypothetical scenario 64
3.4 iTransport in the context of continuous architecture evaluation 66
3.4.1 Perform a systematic design-time evaluation 67
3.4.2 Complement the design-time decisions using a run-time architecture evaluation approach 69
3.4.3 Use forecasting analytics 71
3.5 Conclusion 72

4 Economics-driven Design-time evaluation 73
4.1 Introduction 74
4.2 Background 76
4.2.1 Cost-Benefit Analysis Method (CBAM) 76
4.2.2 Real Options Analysis 76
4.3 Architecture Diversification as a Real Options in the context of iTransport . 78
4.3.1 Options 79
4.3.2 Goal 79
4.4 The Economics-driven Design-time Evaluation Approach 80
4.4.1 Initiate and Diversify 81
4.4.2 Elicit Benefits and Costs 82
4.4.2.1 Step 1: Eliciting the benefits of the diversified architecture options 82
4.4.2.2 Step 2: Eliciting the costs of diversified architecture options 83
4.4.3 Evaluate 84
4.4.3.1 Step 1: Anticipating the utility values and calculating the system value 86
4.4.3.2 Step 2: Calculating the likely rise and fall of payoff with *DAO* 86
4.4.3.3 Step 3: Calculating the option value of exercising a *dao* . 87
4.5 Evaluation and Discussion 90
4.5.1 Experimental Setting 90
4.5.2 Evaluation 90
4.5.2.1 Case 1: Energy Consumption as a Concern 90
4.5.2.2 Case 2: Response Time and Network Usage as Concerns 96
4.5.3 Discussion 100
4.6 Conclusion 102

5 Reactive Run-time evaluation 104
5.1 Introduction . . . 105
5.2 Background . . . 106
5.2.1 Reinforcement Learning . . . 106
5.2.2 Time-Decayed Function . . . 106
5.2.3 Change Detection Approaches . . . 107
5.2.4 Multi-Objective optimisation . . . 108
5.3 Research Questions . . . 109
5.4 Proposed Approach . . . 111
5.4.1 Diversified Architecture Options (DAO) . . . 111
5.4.2 The Proposed Approach In The Context Reinforcement Learning . 113
5.4.3 Stakeholder Involvement versus Automated Management . . . 113
5.4.4 Evaluating The Diversified Architecture Options . . . 114
5.4.5 Detecting Significant Changes Over Time . . . 119
5.4.6 Selecting The Architecture Option With The Optimal Trade-offs . . 121
5.5 Data Synthesis And Analysis . . . 125
5.6 Experimental Evaluation . . . 127
5.7 Further Analysis Of The Proposed Approach . . . 134
5.7.1 Impact Of Frequency Of Monitoring . . . 134
5.7.2 Robustness To Noise . . . 138
5.7.3 Scalability Of The Proposed Approach . . . 139
5.7.4 Impact Of The Parameter θ On The System's Stability . . . 141
5.7.5 Impact Of The Parameter α On The System's Stability . . . 143
5.8 Illustration Of Some Insights The Approach Can Provide . . . 145
5.9 Discussion . . . 148
5.10 Threats to Validity . . . 149
5.10.1 Threats to Internal Validity . . . 149
5.10.2 Threats to External Validity . . . 149
5.11 Conclusion . . . 151

6 Proactive Run-Time Architecture Evaluation 152
6.1 Introduction . . . 153
6.2 Background . . . 155
6.2.1 Online learning models for stationary environments . . . 156
6.2.1.1 k-Nearest Neighbours (kNN) . . . 156

6.2.1.2 Perceptron 156
6.2.1.3 Stochastic Gradient Descent (SGD) Multiclass 157
6.2.2 Online learning models for non-stationary environments 157
6.2.2.1 Fast Incremental Model Trees with Drift Detection (FIMTDD) 157
6.2.2.2 Fading Target Mean (FTM) 158
6.2.2.3 Additive Ensemble (AddExp) 158
6.3 Proactive Architecture Evaluation Approach 159
6.3.1 Forecast and Learn 160
6.3.2 Detect Based on Forecast and Learn 165
6.3.3 Select based on Forecast and Learn 168
6.4 Experimental Evaluation 168
6.4.1 Data Synthesis and Experimental Environment 168
6.4.2 Forecasting Ability of the Proposed Proactive Approach 169
6.4.3 Usability and Efficiency of the Proactive Approach 178
6.5 Discussion 191
6.6 Threats to Validity 192
6.6.1 Threats to Internal Validity 192
6.6.2 Threats to Construct Validity 192
6.6.3 Threats to External Validity 193
6.7 Conclusion 193

7 Reflections, Future Directions, and Conclusion Remarks 195
7.1 Reflections 195
7.1.1 How the research questions have been addressed 195
7.1.2 Applicability and beneficiaries of the framework 199
7.2 Future Directions 201
7.3 Closing Remarks 205

Appendix A List of Included Studies in Systematic Literature Review 206

Appendix B Utility and Binomial trees 210
B.1 Case 1: Energy Consumption as a Concern 210
B.2 Case 2: Response Time and Network Usage as Concerns 210

Appendix C Other Evaluation Results For The Proactive Approach 231
C.1 Plots for RQ4.1.2 231

C.1.1 Learning Parameters 231
C.2 Additional Experimental Evaluation 231
C.2.1 Analysis of Static, Predefined, and Random Approaches Overall Behaviour: 233
C.2.2 Analysis (Best/Worst Case Scenario for Normal): 234
C.2.3 Analysis (Best/Worst Case Scenario for Case 1) 237
C.2.4 Analysis (Best/Worst Case Scenario for Case 3) 239

References **245**

List of figures

1.1 Thesis Roadmap. 15

2.1 Search Execution. 26
2.2 Distribution of the publication types. 28
2.3 Distribution of the publication types. 30
2.4 The proposed classification of architecture evaluation approaches 34
2.5 Distribution of the included studies over the research area maturity classification model (The number of studies for each maturation stage are placed between parenthesis and maturity distribution are shown in percentage). . . 56

3.1 The flow diagram of *iTransport* application [1]. 64

4.1 Steps of Classical CBAM [2, 3]. 77
4.2 A typical binomial tree structure [4]. It uses u (an estimation for the stock increase), d (an estimation for the stock decrease), p (probability that stock increases), $(1-p)$ (probability that stock decreases) and S (system value of the option) to generate the tree from the final level (i.e. timestep 3) to the first level (i.e. timestep 0). 78
4.3 The design-time architecture evaluation approach. 80
4.4 An illustration of the utility trees (left side) and binomial trees (right side) for a TwoPeriod tree. 87
4.5 Utility Tree of all options for Case 1. 93
4.6 Binomial Tree of dao_1 for Case 1. 94
4.7 Binomial Tree of dao_1 for Case 1. 95
4.8 Utility Tree of dao_4 for Case 2 (Best Scenario). 97
4.9 Binomial Tree of dao_4 for Case 2 (Best Scenario). 98

5.1 A representation for the confidence interval statistical technique used by [5] as the change detection test. 108
5.2 An illustrative example of Pareto front and application of the knee point strategy. In the example, the knee points inside the dotted box have better chances to win (i.e. provide a more balanced trade-off between the two objectives). 109
5.3 Steps of the approach, where the design-time evaluation (Chapter 4) forms the initial design decisions and run-time evaluation complements it. 111
5.4 An overview of the operational procedures of run-time evaluation approach. 112
5.5 An overview of the level of stakeholder involvement and automated management in the design-time and run-time evaluation approach. 114
5.6 Sample of changing environmental conditions facing *iTransport* (i.e. input for one of the *DAO*). *Note that the red circles denote some examples of accidental spikes.* 126
5.7 The Initial Experiment Configuration for iFogsim. 128
5.8 The results of assessing design-time and run-time evaluation. 129
5.9 The evaluation of the four approaches' decision-making process under strict quality constraints. 132
5.10 An illustration of the input environmental conditions for the 120 timesteps including network latency, smart camera's latency, power load on the devices, change detection, and switching occurrence, as well as the output exponential benefit of informed-selection approach for strict quality constraints and prioritized ranking score for the best case scenario as an example. *The red squares represent change detections that did not lead to switching DAO, and green circles represent change detection followed by dao switching.* 135
5.11 The % Relative Error of monitoring every T' intervals for each *dao*. 137
5.12 The % relative error and execution time of monitoring every T' intervals for four scenarios. 137
5.13 The behaviour of all the approaches after the application of varying noise levels on the data. 139
5.14 The execution time for large number of *DAO* for every timestep. 141
5.15 Exponential Benefit of the informed-selection approach using different values for the stability parameter θ. 142
5.16 The impact of varying α values on the informed-selection approach. 144

5.17 The evaluation of embedding diversification to the architecture on the decision-making (positive impact (Case 1) and negative impact (Case 2)) as compared to the default case (6 original *DAO* in Table 3.1). 147

6.1 Steps of the continuous evaluation approach, where the design-time evaluation (Chapter 4) forms the initial design decisions and proactive run-time evaluation complements it by adopting time series forecasting. 159
6.2 An overview of the operational procedures of proactive run-time evaluation approach. 165
6.3 The MAE over time for 3 quality attributes to determine the number of training examples for kNN Learner. 173
6.4 The RMSE over time for 3 quality attributes to determine the number of training examples for kNN Learner. 173
6.5 Average MAE of *DAO* for 3 quality attributes for varying forecast timesteps. 177
6.6 Average RMSE of *DAO* for 3 quality attributes for varying forecast timesteps. 178
6.7 The selected *DAO* by reactive/proactive ($t+1$ and average forecasts) approach for **Case 2** for the *best* scenario. 187
6.8 The evaluation of the five approaches' decision-making process for **Case 2** ($t+1$, Average Forecasts) for *best/worst* scenarios. 188

B.1 Utility Tree of all options for Case 1. 211
B.2 Binomial Tree for dao_3 (Energy Consumption Concern, i.e. Case 1). 212
B.3 Binomial Tree of dao_4 for Case 1. 212
B.4 Binomial Tree of dao_5 for Case 1. 213
B.5 Binomial Tree of dao_6 for Case 1. 213
B.6 Binomial Tree of dao_7 for Case 1. 214
B.7 Binomial Tree of dao_8 for Case 1. 214
B.8 Binomial Tree of dao_9 for Case 1. 215
B.9 Utility Tree of options dao_1, dao_2 and dao_3 for Case 2 (Best Scenario). . . 216
B.10 Utility Tree of options dao_5 to dao_7 for Case 2 (Best Scenario). 217
B.11 Utility Tree of options dao_8 and dao_9 for Case 2 (Best Scenario). 218
B.12 Binomial Tree of dao_1 for Case 2 (Best Scenario). 218
B.13 Binomial Tree of dao_2 for Case 2 (Best Scenario). 219
B.14 Binomial Tree of dao_3 for Case 2 (Best Scenario). 219
B.15 Binomial Tree of dao_5 for Case 2 (Best Scenario). 220
B.16 Binomial Tree of dao_6 for Case 2 (Best Scenario). 220

B.17 Binomial Tree of dao_7 for Case 2 (Best Scenario). 221
B.18 Binomial Tree of dao_8 for Case 2 (Best Scenario). 221
B.19 Binomial Tree of dao_9 for Case 2 (Best Scenario). 222
B.20 Utility Tree of dao_1 to dao_3 for Case 2 (Worst Scenario). 223
B.21 Utility Tree of dao_4 to dao_6 for Case 2 (Worst Scenario). 224
B.22 Utility Tree of dao_7 to dao_9 for Case 2 (Worst Scenario). 225
B.23 Binomial Tree for dao_1 for Case 2 (Worst Scenario). 226
B.24 Binomial Tree for dao_2 for Case 2 (Worst Scenario). 226
B.25 Binomial Tree for dao_3 for Case 2 (Worst Scenario). 227
B.26 Binomial Tree for dao_4 for Case 2 (Worst Scenario). 227
B.27 Binomial Tree for dao_5 for Case 2 (Worst Scenario). 228
B.28 Binomial Tree for dao_6 for Case 2 (Worst Scenario). 228
B.29 Binomial Tree for dao_7 for Case 2 (Worst Scenario). 229
B.30 Binomial Tree for dao_8 for Case 2 (Worst Scenario). 229
B.31 Binomial Tree for dao_9 for Case 2 (Worst Scenario). 230

C.1 The error (MAE and RMSE) over time for 3 quality attributes to determine the number of training examples for kNN Learner for **t+5**. 231
C.2 The error (MAE and RMSE) over time for 3 quality attributes to determine the number of training examples for kNN Learner for **t+8**. 232
C.3 The error (MAE and RMSE) over time for 3 quality attributes to determine the number of training examples for kNN Learner for **t+15**. 232
C.4 The selected *DAO* by reactive/proactive ($t+1$) approach for **Normal** for the *best* scenario. 235
C.5 The evaluation of the five approaches' decision-making process for **Normal** ($t+1$, Average Forecasts) for the *best/worst* scenario. 236
C.6 The selected *DAO* by reactive/proactive (Average Forecasts) approach for Case 1 for the *best* scenario. 237
C.7 The evaluation of the five approaches' decision-making process for **Case 1** (Average Forecasts) for the *best/worst* scenario. 238
C.8 The evaluation of the five approaches' decision-making process for **Case 1** ($t+1$, $t+5$, $t+8$, and $t+15$) for the *worst* scenario. 240
C.9 The selected *DAO* by proactive approach ($t+1$, $t+5$, $t+8$, $t+15$, and average forecasts) for **Case 1** for the *worst* scenario. 241

C.10 The evaluation of the five approaches' decision-making process for **Case 3** ($t+1$,$t+5$) for the *best/worst* scenario. 242
C.11 The evaluation of the five approaches' decision-making process for **Case 3** (Average Forecasts) for the *best/worst* scenario. 243
C.12 The selected *DAO* by reactive/proactive approach for Case 3 ($t+1$, $t+5$, and average forecasts) for the *best* scenario. 244

List of tables

2.1 Summary of Search Results and Included Studies from each database. *Note that there are 14 studies generated from the snowballing process in addition to the 26 studies from the databases below, making a total of 40 included studies. Further, some studies may be duplicated in more than one database.* 25
2.2 Data Extraction Criteria. 27
2.3 Distribution of included studies along with the publication channels. 29
2.4 An overview of citation rate of included studies. 30
2.5 Featuring the most cited studies above 100 citations. 31
2.6 Representative Contributions for Design-time Architecture Evaluation. . . . 43
2.7 Representative Contributions for Design-time Architecture Evaluation (Continued). 44
2.8 Representative Contributions for Run-time Architecture Evaluation. 47
2.9 Summary of Contributions for Design-time Architecture Evaluation approaches with other categories. 49
2.10 Summary of Contributions for Run-time Architecture Evaluation approaches with other categories. 50
2.11 Summary of Contributions for Architecture Evaluation approaches with respect to approaches to evaluation. 50

3.1 Possible Diversified Architecture Options for *iTransport* application. The diversity in each *dao* can refer to using fixed and/or mobile fog devices for data collection capability; using different cloud providers and heterogeneous fog devices for data processing capability. 65

4.1 Possible Diversified Architecture Options for *iTransport* application. The diversity in each *dao* can refer to using fixed and/or mobile fog devices for data collection capability; using different cloud providers and heterogeneous fog devices for data processing capability. 79
4.2 Quality Attribute Responses 91
4.3 Costs of diversified architecture options 91
4.4 Option and Net values of diversified architecture options for Energy Consumption Concern (i.e. Case 1). *Note that the dao_i with the highest net value is highlighted in gray*. 96
4.5 Option and Net values of diversified architecture options for Response Time and Network Usage concerns (i.e. Case 2 for Best case scenario). *Note that the dao_i with the highest net value is highlighted in gray*. 99
4.6 Option and Net values of diversified architecture options for Response Time and Network Usage Concerns (i.e. Case 2 for Worst case scenario). *Note that the dao_i with the highest net value is highlighted in gray*. 100

5.1 Aggregate Functions. The Max, Min and $\sum$ operations are over all architecture decisions that are connected to each other in the specified way (parallel or sequence). 116
5.2 Simulation Parameters for *iTransport*. 127
5.3 The number of change detection and switching for informed-selection approach for varying α values. 145
5.4 The evaluation of embedding diversification to the architecture with respect to the number of change detection and switches. 146

6.1 An Illustration of the Process of Creating Training Examples to Train the Forecasting Models. The table illustrates what training example is produced at each timestep, and what forecast can be provided at each timestep, when $\eta_{min} = 1$. The value of "-" in a cell represents the absence of a certain input attribute or forecast at a given timestep. A training example can only be created when the true value of all of of its input and output attributes has already been observed. 162
6.2 Simulation Parameters for *iTransport* 169
6.3 Evaluation for kNN learner for 6 *DAO* (2, 6, and 14 input attributes). 171
6.4 Average ranking and p-value for determining the statistical difference between the use of 2, 6, and 14 input attributes. 172

6.5 The statistical difference between the learners measured using p-value with respect to each quality attribute. 174
6.6 Evaluation for Learners for 6 *DAO*. 176
6.7 The number of switches, mean exponential benefit and mean cost for Reactive and Procative informed-selection approach for short-term forecasts (i.e. $t+1$). The best cases are highlighted in green. 180
6.8 The average rank of approaches and statistical difference between them measured using p-value with respect to the exponential benefit. *The top ranked approach is highlighted in orange.* 183
6.9 The number of switches, mean exponential benefit and mean cost for Static-, Predefined-, and Random-Selection approaches. 183
6.10 The number of switches, mean exponential benefit and mean cost for Reactive and Procative informed-selection approach for $t+h$. The best cases are highlighted in green. 184

A.1 Studies with respect to Research Area maturation level. 206
A.2 Studies included in the review. 207
A.3 Studies included in the review (Continued). 208
A.4 Studies included in the review (Continued). 209

C.1 The parameters for the single and ensemble learners. 232

Nomenclature

List Of Abbreviations

ALMA	Architecture-Level Modifiability Analysis
APTIA	Analytic Principles and Tools for the Improvement of Architectures
aSQA	Architecture Software Quality Assurance
ATAM	Architectural Trade-off Analysis Method
ATMIS	Architectural Tradeoff Method using Implied Scenario
CBAM	Cost-Benefit Analysis Method
CDT	Change Detection Tests
CPASA	Continuous Performance Assessment of Software Architecture
CPM	Change-Point Methods
DACAR	Decision-centric software Architecture evaluation Method
HT	Hypothesis Tests
IoT	Internet of Things
MOP	Multi-Objective Optimisation
MPT	Modern Portfolio Theory
NPV	Net Present Value
NSGA-II	Non-dominated Sorting Genetic Algorithm-II
PASA	Performance Assessment of Software Architecture

QA	Quality Attribute
QoS	Quality of Service
QuaDAI	Quality-Driven Architecture Derivation and Improvement
ROA	Real Options Analysis
SAAM	Software Architecture Analysis Method
SBAR	Scenario-Based Architecture Re-engineering
SHT	Sequential Hypothesis Tests
SLR	Systematic Literature Review
SQUASH	Systematic Quantitative Analysis of Scenarios' Heuristics

List Of Symbols

α	Confidence level
η	Number of training examples
η_{min}	Minimum number of training examples for enabling forecasts
$\hat{\mu}_{dao_i}(t)$	Forecast exponential benefit of a *dao* varying over time
$\hat{\sigma}_{dao_i}(t)$	Standard deviation based on forecast exponential benefit of a *dao* varying over time
$\hat{B}_{dao_i}$	Forecast benefit of a *dao* varying over time
$\hat{L'}_{\lambda}(dao_i,t)$	Forecast Marginal loss of non-dominated *dao* varying over time
$\hat{q}_{dao_i}(t)$	Forecast for Normalised Quality of a *dao* varying over time
$\hat{q}_{dao_i}(t+h)$	Forecast for Normalised Quality of a *dao* varying over time for further ahead timesteps
λ	A pre-defined parameter that controls the relative importance between $c_{dao_i}(t)$ and $\mu_{dao_i}(t)$
$\|Q\|$	Number of quality attributes
$\|\lambda\|$	Number of λ values

$\lvert v \rvert$	Number of variety of costs
$\lvert DAO \rvert$	Number of diversified architecture options
$\mu_{dao_i}(t)$	Exponential Benefit of a *dao* varying over time
v	Variety of costs
$\sigma^2_{dao_i}(t)$	Exponential Variance of a *dao* varying over time
$\sigma_{dao_i}(t)$	Exponential Standard Deviation of a *dao* varying over time
τ_q	Error threshold per quality attribute
θ	Relative Importance of the past
ζ	Number of input attributes
B_{dao_i}	Benefit of a dao_i
$B_{dao_i}(t)$	Benefit of a *dao* varying over time
c_{dao_i}	Operating cost of a dao_i
$c^v_{dao_i}(t)$	Costs of each variety v per *dao* (e.g. deployment cost, leasing cost, *etc*)
$c_{dao_i}(t)$	Cost of a *dao* varying over time
d_{ka}	Architecture decision d for capability k implemented using component a
DAO	A set of diversified architecture options
dao_i	i diversified architecture option
dao_{curr}	Current *dao*
$e_q(t)$	Forecast error per timestep
h	Number of further ahead timesteps
$L'_\lambda(dao_i,t)$	Marginal loss of non-dominated *dao* varying over time
$L_\lambda(dao_i,t)$	Loss Function shows how (un)desirable a *dao* is
NV_{dao_i}	Net value of a dao_i

$O_d(t)$	The likely fall of payoff from implementing a dao_i at time t
$O_u(t)$	The likely rise of payoff from implementing a dao_i at time t
O_{dao_i}	Option Price
p	Risk-adjusted probability
Q	The set of quality Attributes such as Response time, network usage, *etc*
q	A quality Attribute such as Response time
$q'_{dao_i}(t)$	Normalised Quality of a *dao* varying over time
$q'_{dao_i}(t+h)$	Normalised Quality of a *dao* varying over time for further ahead timesteps
$q_{dao_i}(t)$	Quality of a *dao* varying over time
r	Risk-free Interest Rate
S_{dao_i}	System value of a dao_i
$S_{dao_i}(t)$	System value after implementing dao_i causing incremental improvement/degradation at time t
S_{ini}	Initial system value
sc_{dao_i}	Switching cost of a dao_i
u, d	System value (corresponding to stock price) benefiting/being hurt from dao_i i.e. value rise/fall
V_q	The impact of deploying a dao_i on a particular quality of interest
w_q	Weight of quality q

Chapter 1

Introduction

Architectures of complex and scalable real-time systems are dynamic in nature and exhibit numerous uncertainties in their operation [6, 7]. This dynamism limits the effectiveness of design-time architecture evaluation approaches. In particular, Internet of Things (IoT) environments [8, 7] are fundamentally different from classical ones for which most architecture evaluation methods, such as Cost-Benefit Analysis Trade-off Method (CBAM) [9], Architecture Analysis Trade-off Method (ATAM) [2], and Architecture-level modifiability analysis (ALMA) [10], were advanced. These systems are data-driven, characterised by their scale, hyper-connectivity, dynamism, and uncertainty in operation (with a constant stream of new devices, new services, and fluctuations in quality of service (QoS) provisions) [11]. Furthermore, IoT has constrained devices, which are mobile, variable in processing and computational power, and in some cases with limited network connectivity [6, 7].

We acknowledge that design-time evaluation [9, 2, 10] is a necessary step to build fundamental architecture decisions when designing a software system. However, its effectiveness is highly influenced by the expertise of the evaluators, their knowledge of the domain, and the usage contexts for which the architecture is conceived [12, 13]. Valuation and stakeholders' perception on cost and value of the candidate architecture solution can suffer from under/over estimations. They also rely on assumptions that are unlikely to hold true indefinitely in dynamic, scalable and uncertain environments, such as IoT. Dynamicity, multi-tenant nature, hyper connectivity, environmental changes, and uncertainty are fundamental properties of non-stationary architectures, which are difficult to conceive and cater for in design-time evaluation only.

In this context, design-time evaluation tends to be subjective and incomplete. Widely used architecture evaluation methods [9, 2, 14] fundamentally lack mechanisms for aligning operations with development when the architecture evaluation is conducted. Therefore, a more continuous approach to software architecture evaluation is necessary to *complement* design-time evaluation. We define *continuous* software architecture evaluation as *an evaluation of the software architecture that begins at the early stages of the development and is periodically performed throughout the lifetime of the software system.* Continuous evaluation is composed of two phases: *design-time* and *run-time* evaluation. In particular, design-time evaluation can be used to support the necessary initial system design and deployment. After that, run-time evaluation can be used to assess to what extent the architecture options created at design-time, as well as other potential architecture options, perform well at run-time. This enables architects to make informed decisions on potential changes to the architecture, so that its performance remains good over time.

Intertwining and interleaving run-time evaluation with design-time has the potential to change ad hoc and "trial and error" practices for architecting complex, scalable, and dynamic software systems. Consider the case where architects embed *design diversity* [15] into their solutions in an attempt to meet quality requirements under uncertainty and to mitigate risks and Service Level Agreement violations. Diversification [15] encompasses design decisions and architecture tactics that can be used to adapt the system to unforeseen changes [16]. For a given concern, architecture diversity "spices" the architecture with a variety of design decisions and strategies (e.g. the choice of data collection and processing strategies and tools), which can better cope with uncertainties at run-time. Diversified design decisions, whether planned or accidental [17], can be expensive; their behaviour and value can be best evaluated at run-time. However, there are no systematic continuous software architecture evaluation methods that intertwine design-time and run-time evaluation.

Continuous architecture evaluation has been introduced and discussed as an integral phase within modern software development processes (e.g., agile [18], DevOps [19], and continuous delivery [20]). This is an enabler for rapid response to operational, environmental, and requirements changes. It has also been advocated as an essential appraisal and quality assurance practice that symbiotically aligns development and operations as a measure for responding to dynamism, unpredictability, and operational uncertainties. Despite the widespread acceptance of continuous architecture evaluation as a concept, the literature requires further examples on how continuous evaluation can actually be realized and conducted. There are few research efforts (e.g. [21, 18, 13]) which explicitly mention continuous architecting and assessment, while some others implicitly adopt it (e.g. [19]).

As an example, Pooley and Abdullatif [18] define continuous assessment as the continuous production of performance evaluation tests. Despite their work being one of the few approaches that explicitly provides continuous assessment, it lacks run-time monitoring and forecasting of the performance of architecture decisions. In such cases architecture is, at best, a modelling tool, which may (or may not) be applicable in dynamic environments. Bersani et al. [13] propose a continuous architecting approach that aids the designers in easing the static analysis of the architecture, but lacks dynamicity and is domain-specific (i.e. for streaming applications). Further, though DevOps [19] advocates continuous integration as part of the continuous development process, it lacks a systematic mechanism for evaluating the artefacts generated out of such systems. Thus, novel continuous evaluation methods could aid the prior methods by providing extra insights about the behaviour of candidate architectures to improve decision-making. In particular, they can benefit from further analysis in terms of dynamic tracking and forecasting of architecture decisions' performances, and automated management of cost-benefit trade-offs.

1.1 Aim and Objectives

The complexity in reasoning about large scale environments, such as IoT, are hard to be evaluated at design-time only. These architectures can employ diversification, as an example, to respond to IoT challenges, handle uncertainties and hence improve the performance of an IoT application. However, these diversified architectures introduce many design trade-offs. Therefore, an approach that dynamically and continuously learns and forecasts the behaviour of diversified architecture decisions is necessary. To this regard, the major aim of this thesis is to address the *lack of systematic continuous architecture evaluation framework that intertwines design-time and run-time evaluation to handle complex architectures and improve the decision-making, in dynamic and uncertain environments, such as IoT*. This can be done by addressing the following problems of existing work:

- **Problem 1:** The lack of a systematic design-time evaluation approach that can quantify the long-term value of evaluating complex architecture decisions, such as diversity in design.

- **Problem 2:** The lack of a run-time evaluation approach that can systematically and dynamically track the benefits of architecture design decisions, while providing the architect with flexible tuning for learning parameters and taking into account the trade-offs between architecture's stability and learning accuracy.

- **Problem 3:** The lack of a proactive run-time evaluation approach that can aid in determining how well the architecture design decisions will behave in the future and proactively deal with non-stationary scenarios.

Accordingly, this thesis has four objectives:

- **Objective 1:** Perform a systematic literature review that discusses the state-of-the-art of architecture evaluation approaches under uncertainty and hence provides the necessary guidelines to develop a continuous architecture evaluation framework.
- **Objective 2:** Propose a systematic economics-driven design-time evaluation approach as an initial step to shortlist the candidate architecture decisions, as well as, rank the options under investigation.
- **Objective 3:** Complement the design-time decisions using an informed run-time architecture evaluation approach that can monitor and learn the cost-benefit trade-offs of architecture decisions over time.
- **Objective 4:** Propose a proactive evaluation approach that forecasts the future potentials of architecture design decisions and hence improves the decision-making support of (2) and (3).

1.2 Research Questions

The objectives outlined in Section 1.1 will be addressed by answering the following research questions:

- **RQ1:** What is the state-of-the-art in software architecture evaluation under uncertainty and to what extent continuous architecture evaluation is used? *(Addressing Objective 1)*
- **RQ2:** To what extent design-time architecture evaluation methods that adopt economics-driven principles, such as real options analysis, can systematically evaluate complex design decisions, such as diversification, in the face of operational uncertainties in dynamic environments? *(Addressing Objective 2)*
- **RQ3:** How can run-time architecture evaluation using cost-benefit analysis and machine learning techniques complement the design-time one to handle uncertainty? *(Addressing Objective 3)*
- **RQ4:** How can the use of forecasting analytics improve the state of run-time architecture evaluation? *(Addressing Objective 4)*

1.3 Research Methodology

In this thesis, we leverage techniques from classical research methodologies, inspired by [22] to conduct the research. The methodology adopted is divided into five steps as follows:

1. **Identifying thesis problems:** The initial step is to acquire knowledge about the domain (i.e. Evaluating Software Architectures for uncertainty). For that, we have conducted a systematic literature review that covers the state-of-the-art and practices of software architecture evaluation approaches. The review provides a holistic view about the research progress for the domain along with open problems and gaps. As the understanding of the problem is advanced, the major problem that this thesis aims to address is the *lack of continuous architecture evaluation approaches to handle complex and dynamic architectures*. The problem is also formulated through a set of research questions.

2. **Identifying thesis objectives:** Motivated by the prior problem, this thesis aims to propose a *novel continuous architecture evaluation framework* to reason about architecture decisions operating under uncertainty. To achieve the main objective of thesis, we contribute to the following:

   2.1. A systematic economics-driven design-time evaluation approach that reasons about architecture decisions and quantifies their long-term value to cope with operational uncertainties.

   2.2. A run-time evaluation approach for tracking the benefits of architecture design decisions and evaluating their cost-benefit trade-offs. This approach in turn can complement the design-time decisions (2.1) for further refinements.

   2.3. A proactive evaluation approach that forecasts the future potentials of architecture design decisions and hence improves the decision-making support of (2.1 and 2.2).

3. **Designing and developing thesis contributions:** To achieve the prior objectives, this thesis extends CBAM [23] and binomial real options analysis [24–26] to contribute an economics-driven design-time evaluation approach to suit for the complexity and dynamicity of software architectures. The binomial real options analysis can aid in long-term valuation of architecture decisions (such as diversity in design) under uncertainty. Moreover, this thesis leverages reinforcement learning techniques [27]

to dynamically profile the benefits of architecture design decisions along with multi-objective optimisation approaches [28, 29] to assess the cost-benefit trade-offs of decisions. Finally, it demonstrates what benefits proactive approaches adopting time series forecasting [30–32] can render to architecture evaluation.

4. **Demonstrating thesis contributions:** This thesis adopts the internet of things paradigm for demonstrating the usefulness of the proposed continuous evaluation approach. We consider the case of architecting for IoT (i.e. *iTransport* application) as a motivating scenario to show how continuous evaluation, leveraging cost benefit analysis, machine learning and forecasting analytics, can be realised and conducted for handling IoT challenges. In particular, software architects are challenged by the heterogeneity, scale, and high dynamism in IoT environments. For this purpose, we consider the case where the architect has embedded design diversity [33] to the architecture aiming to improve the application's performance. Further, the stakeholders could have different QoS priorities due to variation in contexts. The uncertainties associated with these decisions require novel ways for evaluation. In this context, the use of a continuous evaluation approach could aid the architect in determining the long-term value of diversification and analysing the cost-benefit trade-offs. It can also provide the architect with flexibility to adjust the trade-offs between architecture's stability and learning accuracy. As such, the approach can inform deployment, refinement and/or phasing out of (diversified) architecture decisions.

5. **Evaluating thesis contributions:** To evaluate each of the demonstrated contributions, a set of controlled experiments are conducted derived from the *iTransport* application, introduced in Chapter 3. In particular, we have used qualitative and quantitative experimental evaluations to highlight on the extra insights which the continuous evaluation approach would provide to reasoning of diversification in IoT design.

1.4 Thesis Contributions

This thesis makes a number of novel contributions towards the objective of providing a continuous architecture evaluation framework for architectures operating in dynamic and uncertain environments. In particular, the thesis investigates how leveraging on the principles of options theory, cost-benefit analysis, machine learning, and time series forecasting can lead to a pragmatic technique that can provide extra insights about the behaviour of candidate architectures to improve the decision-making.

The major contributions are the following:

- **A systematic literature review that covers the state-of-the-art of software architecture evaluation approaches under uncertainty:** We review existing work on architecture evaluation under uncertainty. We also provide a classification framework, which aided us in categorising and understanding the current state-of-the-art. The objective of this review is to determine the design-time and/or run-time evaluation approaches, the list of major challenges of architecture evaluation approaches, and how they handle uncertainties. These challenges contribute to the design of our novel continuous evaluation framework.

- **A design-time architecture evaluation approach that adopts an economics-driven approach:** This contributes to an architecture-centric model, which is built on a well established software architecture method, namely, CBAM. We have used diversification in design as an example of architecture significant design decision that is taken by the architect to design for more dependable provision in the face of uncertainty.

- **A novel reactive run-time architecture evaluation method:** We propose a novel run-time architecture evaluation method, suited to systems that exhibit uncertainty and dynamism in their operation. The method uses machine learning and cost-benefit analysis at run-time to continuously profile the architecture decisions for their added value that trace back to the design decisions. The method provides continuous assessment for these decisions; it can inform deployment, refinement and/or phasing out decisions.

- **A novel proactive run-time architecture evaluation approach:** We provide continuous learning as built-in mechanisms for proactive evaluation that complement the reactive one. The approach shows how proactive approaches leveraging the time series forecasting analytics can fundamentally guide the continuous evaluation of software architectures and influence the outcome of the decisions (i.e. potential impact on decision-making).

1.5 Thesis Storyline

A systematic literature review we conducted discussing the state-of-the-art for design-time and run-time architecture evaluation approaches indicates that despite the uncertainties in dynamic environments, there is still a lack of examples and approaches on how to conduct continuous architecture evaluation. To overcome this lack, we aim to answer *RQ1: What is* *the state-of-the-art in software architecture evaluation under uncertainty and to what extent continuous architecture evaluation is used?* we report on a systematic literature review that consolidates different approaches that can assist in providing the necessary guidelines to develop continuous evaluation approaches. Though there are some methods (e.g. [21, 18, 13]) which explicitly mention continuous architecting and assessment, while others implicitly adopt it (e.g. [19]), these approaches can benefit from further investigations in terms of how continuous evaluation could leverage novel techniques to dynamically track and forecast the architecture decisions, and also automatically manage the trade-offs.

Architecture diversification is a common practice, when architecting software for systems at scale, dynamism, and uncertainty in their operations, to improve the QoS (e.g. performance). Our framework investigates this phenomenon and formulates the problem of architecture diversity from economics-driven and run-time perspectives. In particular, this thesis contributes to *a systematic and continuous architecture evaluation framework that reasons about complex architecture design decisions under uncertainty, such as diversity, which in turn improves decision-making*.

IoT architecture is a beneficiary distributed paradigm for which continuous evaluation can be conducted due to their inherent properties that are characterised by scale, complexity, heterogeneity, dynamism and uncertainty in operation [8, 7]. Uncertainties can be attributed, for example, to hardware or software faults affecting an IoT device, thermal drifts or ageing effects in sensors, resource-constrained and/or mobile devices, and fluctuations in QoS [8, 30, 34]. We have extended an IoT case study proposed in [35] (i.e. used diversification to handle IoT challenges) and developed various scenarios to motivate and evaluate the need for the continuous evaluation framework.

Design-time evaluation is a prerequisite for continuous architecture evaluation framework. However, based on our review, the current economics-driven design-time evaluation approaches (e.g. [36–43]) have not been used before to deal with cost-benefit trade-offs in dynamic environments, such as IoT. They have also not been validated as a technique to determine the long-term value of diversified architecture decisions. Further, a design-time economics-driven approach could be a potential solution to evaluate diversified architecture decisions. To overcome this lack, we aim to answer *RQ2: To what extent design-time architecture evaluation methods that adopt economics-driven principles, such as real options analysis, can systematically evaluate complex design decisions, such as diversification, in the face of operational uncertainties in dynamic environments?*

For that, we will explore the binomial real options analysis [24–26] and necessary approaches to value it. CBAM has proven to be effective in valuing the costs and benefits of architecture design decisions (e.g. [23, 44, 45]). However, our design-time approach will further extend it using Real Options. So why Real Options? It provides an analysis paradigm that emphasises the value-generating power of flexibility under uncertainty. An option is the right, but not the obligation, to make an investment decision in accordance to given circumstances for a particular duration into the future, ending with an expiration date [24]. Our economics-driven approach has the following benefits: (1) It extends the CBAM with binomial real options analysis to quantify and visualise the *long-term* value of engineering diversification into the solution space at design-time in the face of operational uncertainties; (2) It can be used to shortlist the candidate architecture decisions for run-time evaluation and hence reduce deployment costs. We use the motivating IoT case study to demonstrate the usability, the strengths, and limitations of our design-time evaluation approach.

The proposed economics-driven design-time evaluation is essential to build the initial software architecture to be deployed. However, it relies on human experts (i.e. based on their knowledge and confidence level), who can only *partially* forecast the fitness and the extent to which an architecture can cope with operational uncertainties, unanticipated usage scenarios, and emergent behaviours. Therefore, this requires run-time evaluation, which leads to *RQ3: How can run-time architecture evaluation using cost-benefit analysis and machine learning techniques complement the design-time one to handle uncertainty?*

To answer RQ3, we propose a *reactive run-time evaluation approach* inspired by self-adaptive systems [46]. Our approach makes the following contributions to the research literature: (1) A run-time approach for tracking the benefits of architecture design decisions using machine learning; (2) An approach inspired by multi-objective optimisation to evaluate the cost-benefit trade-offs of architecture decisions at run-time. In particular, the approach is able to profile situations where options can be more effective and provide continuous updates on their value potentials. Specifically, our approach adopts an *exponential time-decay function* inspired by reinforcement learning [27]. This function enables us to track the current benefit of a diversified architecture option by weakening the effect of old data (i.e. emphasise on the recent versus past observations). Our reactive run-time evaluation approach also adopts *change detection tests* to check whether the benefit of the diversified architecture option currently being used is getting significantly worse [5]. If it is significantly worse, a method inspired by the multi-objective optimisation literature [28, 29] is adopted to identify the diversified architecture option with the optimal trade-off between cost and benefit. Based on the profiling of options over time, it is also possible to determine which ones are not fit for the purpose, and hence could be phased out.

By run-time evaluation, we envision several potential scenarios of use to capture the dynamic behaviour of the selected design decisions in relation to the quality attributes in question:

- Simulation can be a useful alternative to experimentation with real environments: IoT environments are often large scale: an architecture can embed large numbers of heterogeneous and diverse things, sensors and devices supporting some design decisions. As it would be prohibitively expensive to configure the architecture and to test these decisions exhaustively before deployment, the use of simulation can be useful for assisting the architect in what-if analysis prior to deployment, stress-testing the architecture with inputs that can go beyond the ones encountered in normal operation, abstract the analysis, and demonstrate the potential for scaling it.

- This approach can also work if run-time data of a given configuration is available. The architect will need to instrument the system with mechanisms for monitoring, logging, and profiling quality attributes of interest and design decisions supporting these attributes. This can be particularly useful for cases where the system is already deployed and further refinements are envisioned. Learning from the log of operation for example can be useful for profiling and analysing the likely technical and value potentials of these decisions at run-time, whether diversified or not.

- A third scenario can also be possible, where the approach can be integrated in continuous development paradigms, such as DevOps, where run-time information from the operation side can provide feedback for development.

Each of the mentioned scenarios would require separate treatment and reporting to show how the approach can be applied. The scope of this thesis is concerned with the case of using simulation to support the run-time analysis. Nevertheless, many of the observations can be applicable for the other cases. In particular, we adopt the *iFogSim* [35] tool. *iFogSim* builds on Cloudsim [47]; it provides the architect with the freedom of hierarchically composing fog devices, clouds, and data streams to simulate the technical and value potentials of selected decisions using normal usage and stress tests in relation to quality attributes of interest. We implement the motivating IoT case study using the iFogsim tool to simulate the qualities of interest of each architecture decision over time. Accordingly, a series of experiments are conducted to demonstrate the applicability and effectiveness of the continuous evaluation framework using the case of architecting for IoT.

To evaluate the reactive approach, we conduct a set of controlled experiments that aim at providing a better understanding of the influence of the stability parameters related to exponential decay function and change detection tests on the proposed approach. This better understanding can guide software architects in the decision of which values to use for these parameters. We have also provided another series of experiments to demonstrate the usefulness of the approach as compared with three baseline architecture evaluation methods [4, 48, 7]. The scalability and robustness to noise are other necessary factors to validate the applicability of the approach, which we experimented as well.

To summarise, the proposed reactive run-time evaluation approach assesses the architecture design decisions through the following: (i) evaluating, complementing, and refining design-time decisions with run-time ones; (ii) tuning the relative importance of present/past of data for knowledge accumulation about an architecture which can aid in learning the behaviour of architecture decisions over time; (iii) efficiently assessing the value of architecture decisions at different monitoring intervals; (iv) allowing the architect to tune the sensitivity of the approach to changes, and therefore how stable/unstable it will be over time. However, in other contexts, the reactive learning may suggest wrong decisions and recommend unnecessary switches due to the lack of knowledge about the future benefit of candidate architecture decisions. Taking into account the future potential of diversified architecture option is important to provide a more informative evaluation. This leads to *RQ4: How can the use of forecasting analytics improve the state of run-time architecture evaluation?*

To answer RQ4, we propose *a novel proactive run-time evaluation approach*. This approach contributes the following: (i) aid the architect in continuously learning about the architecture decisions; (ii) help in forecasting how well they will behave in the future; (iii) proactively deal with non-stationary scenarios. For this purpose, the proactive evaluation approach uses continuous *time series forecasting* [30–32] as built-in mechanisms to complement reactive run-time evaluation with proactive run-time evaluation. In this way, the proactive approach does not take into account only the past likely performance of architecture decisions, but also their future potentials, better informing run-time architecture decisions. We report on a comparative study involving several forecasting models to illustrate the impact of the forecasting analytics on the evaluation and decision-making. We also provide the architect with recommendations that are grounded on experimentation and evidence. We finally compare the proposed approach against the reactive approach and the classical architecture evaluation approaches [4, 48, 7].

We have chosen diversified design decisions in IoT as a beneficiary to demonstrate the usefulness and applicability of continuous evaluation in the context of highly dynamic and non-stationary environments. Nevertheless, the approach could be applied to other software systems operating in dynamic environments, such as volunteer computing, microservices, service-oriented architectures, and so forth.

The scope of our work can benefit the relation between requirements (mainly non-functional requirements) and architecture. Though the contribution leverages configurable architecture to better meet the requirements, we do not adapt the requirements themselves but we alternate between architecture design strategies and decisions, when possible to meet the requirements.

This thesis acknowledges that extending and evaluating the framework in real setting in the future calls for further treatment and investigations related to practicality. Therefore, this thesis concludes by reflecting on some suggestions related to applicability of the framework in practice. Further, it points out some open problems which could further enrich the thesis's contributions. These open problems include (but are not limited to): the extension of currently adopted linear aggregation modelling of qualities of architecture decisions to consider their dependencies, the use of the framework to value the architecture's sustainability [49], the application of the framework in other contexts (e.g. volunteer computing, microservices, service-oriented architectures, *etc*), moving it towards decentralisation, using transfer learning to improve the evaluation, and the development of a tool-support, *etc*.

1.6 Publications

The work presented in thesis has been partially/completely derived from the following list of papers published during the PhD. This thesis must be considered as the definitive reference of details and ideas, presented in these publications.

- D. Sobhy, R. Bahsoon, L. Minku, R. Kazman, Diversifying software architecture for sustainability: A value-based perspective, Proceedings of 2016 the European Conference on Software Architecture (ECSA).

- Sobhy, D., Minku, L., Bahsoon, R., Chen, T. & Kazman, R. (2018, March). Run-time Evaluation of Architectures: A Case Study of Diversification in IoT. Journal of Systems and Software (Second Review Cycle).

- Sobhy, D., Minku, L., Bahsoon, R., & Kazman, R. (2019, April). Continuous and Proactive Software Architecture Evaluation: An IoT Case. ACM Transactions on Software Engineering and Methodology (TOSEM) (Review Cycle).

- Sobhy, D., Bahsoon, R., Minku, L., & Kazman, R.. Evaluating Software Architectures under Uncertainty: A Systematic Literature Review. IEEE Transactions on Software Engineering (TSE) (To be submitted).

1.7 Thesis Roadmap

We will illustrate the thesis roadmap via Figure 1.1 and as follows:

- **Chapter 2:** conducts a systematic literature review on the state-of-the-art and practices of software architecture evaluation under uncertainty. The objective of this review is to provide a solid background on architecture evaluation domain. In particular, it identifies the gaps of the current architecture evaluation approaches and motivates the need for a continuous evaluation approach. Further, it highlights the inadequacies in the design-time and run-time evaluation approaches and demonstrates the properties required in developing a continuous evaluation approach. This chapter is derived from [50].

- **Chapter 3**: introduces the IoT case study *iTransport* and motivates the need for the continuous evaluation framework. This chapter first provides an overview of IoT challenges, followed by the *iTransport* case study, diversity as a solution for handling uncertainty and then the *iTransport* design trade-offs. It further discusses the current problems faced by architects while evaluating for IoT applications. It finally shows how the proposed continuous evaluation framework could be used to improve the current evaluation practices. This chapter is partially derived from [51, 52].

- **Chapter 4:** presents the first part of the proposed framework. Particularly, this chapter introduces the use of CBAM and binomial real options analysis to quantify the long-term value of diversified architecture decisions at design-time. It also demonstrates the usability of the approach through the *iTransport* application and different scenarios of context. Accordingly, it shortlists the candidate architecture decisions for run-time evaluation. This chapter is partially derived from [14].

- **Chapter 5:** proposes the reactive run-time evaluation approach which is inspired by reinforcement learning and multi-objective optimisation to complement and evaluate the diversified design-time decisions of Chapter 4. A set of controlled experiments are conducted to provide systematic guidance for the architect on how to evaluate the architecture decisions at run-time and tune the input parameters to best benefit from evaluation. We have also compared the approach against three baseline and state-of-the-art approaches. This chapter is derived from [51].

- **Chapter 6:** proposes the final component of the proposed framework. In particular, this chapter demonstrates steps of the proactive evaluation approach, which adopts time series forecasting analytics. A series of experiments is conducted to demonstrate the applicability and effectiveness of the approach. We also provide the architect with recommendations on how to best benefit from the approach through choice of learners, input parameters, *etc*, grounded on experimentation and evidence. This chapter is derived from [52].

- **Chapter 7:** concludes the thesis with a discussion of the main findings and remarks with respect to the thesis's research questions. We also provide directions that this research can take, in the future.

- **Appendix A**: presents the list of included studies used by the systematic literature review in Chapter 2.

- **Appendix B**: plots the utility and binomial trees generated by the design-time approach to perform the evaluation of architecture options introduced in Chapter 4.

- **Appendix C**: plots the rest of the experimental evaluation figures generated by the proactive approach and provides additional evaluation results, introduced in Chapter 6. It also tabulates the experimental parameters by the proactive approach.

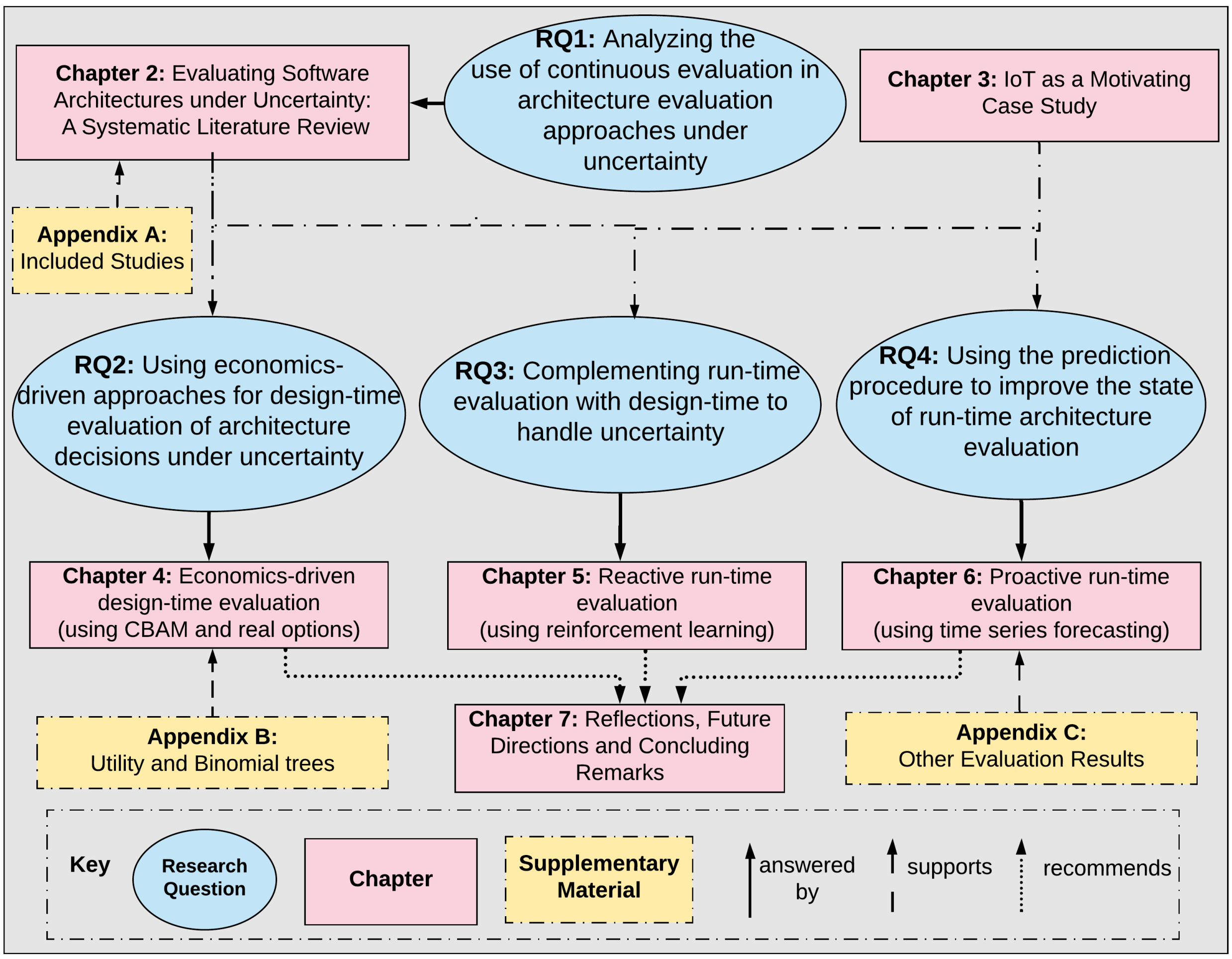

RQ1: Analyzing the use of continuous evaluation in architecture evaluation approaches under uncertainty
Chapter 2: Evaluating Software Architectures under Uncertainty: A Systematic Literature Review
Chapter 3: IoT as a Motivating Case Study
Appendix A: Included Studies
RQ2: Using economics-driven approaches for design-time evaluation of architecture decisions under uncertainty
RQ3: Complementing run-time evaluation with design-time to handle uncertainty
RQ4: Using the prediction procedure to improve the state of run-time architecture evaluation
Chapter 4: Economics-driven design-time evaluation (using CBAM and real options)
Chapter 5: Reactive run-time evaluation (using reinforcement learning)
Chapter 6: Proactive run-time evaluation (using time series forecasting)
Appendix B: Utility and Binomial trees
Chapter 7: Reflections, Future Directions and Concluding Remarks
Appendix C: Other Evaluation Results
Key
Research Question
Chapter
Supplementary Material
answered by
supports
recommends

Figure 1.1 Thesis Roadmap.

Chapter 2

Evaluation of Software Architectures under Uncertainty: A Systematic Literature Review

Context. Evaluating software architectures in dynamic and uncertain environments raises a number of new challenges, which require continuous approaches. We define continuous software architecture evaluation as an evaluation of the software architecture that begins at the early stages of the development and is periodically performed throughout the lifetime of the software system. Continuous evaluation has been called by different names, such as run-time, dynamic, continuous, *etc.*, along with assessment and analysis. The common characteristic between these approaches is that they start at design-time (even if they do not mention that explicitly) and continue to evaluate the architecture decisions during the life-time of the system by observing environmental conditions. However these approaches are still few, limited, and lack maturity.
Objective. In this review, we attempt to survey all these efforts and provide a unified terminology and perspective on the subject. In particular, we review the existing architecture evaluation methods under uncertainty, which may implicitly or explicitly adopt continuous approaches. Through this survey and associated analysis we provide a set of guidelines for the elements needed to develop and conduct a continuous architecture evaluation approach.
Method. We conducted a systematic literature review to identify and analyse architecture evaluation approaches under uncertainty including continuous and non-continuous, covering work published between 1990-2018. We examined each approach and provided a classification framework for this field. We present an analysis of the results and provide insights regarding open challenges, which will pave the way for future research. Our review and

classification framework contribute to the core principles of evaluating software architectures under uncertainty.
Results. Most of the existing approaches focus on either design-time evaluation or run-time evaluation. In particular, several approaches select the optimal candidate at design-time and then deploy it at run-time without further evaluation. Other approaches make implicit use of run-time evaluation through monitoring one or more qualities of interest. However, these solutions typically lack an explicit linkage between design-time and run-time. Their run-time evaluation of architectures and forecasting capability tend to be limited.
Conclusion. There is a lack of systematic guidance on how continuous architecture evaluation can be realised or conducted and few examples of their application. To remedy this lack, and as a contribution of this chapter, we present a set of necessary ingredients for continuous architecture evaluation.

2.1 Introduction

Architecture evaluation is a milestone in the decision-making process. It aims at justifying the extent to which architecture design decisions meet a system's quality requirements and their trade-offs, particularly in the face of operational uncertainties and changing requirements. The evaluation can aid in early identification and mitigation of design risks; the exercise is typically done in an effort to save integration, testing and evolution costs [53]. Examples of seminal work in this area include Architecture Tradeoff Analysis Method (ATAM) [2], and Cost Benefit Analysis Method (CBAM) [9].

Software architectures that operate in dynamic and non-stationary environments (e.g., IoT and cloud applications) require a fundamental shift in the way evaluations are conducted. This is due to unforeseen factors that may affect the evaluation, including (but not limited to), fluctuations in QoS, multi-tenancy, hyper-connectivity, sensor ageing effects, *etc* [54, 8, 34].

Though existing design-time evaluation approaches promise to evaluate flexibility in architectures under uncertainty and their responses in enabling change [2, 9, 10], in contexts of highly dynamic environments these approaches tend to be limited because there may be emerging scenarios where the architect cannot rely solely on design-time evaluation. Such scenarios require a run-time evaluation to inform and calibrate the design-time decisions. In this context, a more continuous approach would benefit the evaluation process. We define *continuous* software architecture evaluation as *an evaluation of the software architecture that begins at the early stages of the development and is periodically performed throughout the* *lifetime of the software system.* Continuous evaluation is performed either continuously or sporadically covering either one feature (e.g. QoS) or multiple features.

There have been many research studies aimed at evaluating software architectures to deal with uncertainty which may implicitly or explicitly adopt continuous approaches (e.g. DevOps [19]). The field has attracted a wide range of researchers and practitioners. However, continuous evaluation has not been viewed as a key area within software architecture research. We still lack a clear vision regarding the elements of a continuous software architecture evaluation approach.

In past years, many research studies have reviewed design-time architecture evaluation methods (e.g. [12, 55, 56]), while some have attempted to review run-time methods without addressing them from the context of continuous architecture evaluation (e.g. [57, 46, 58–60]). In particular, to date there is no systematic literature review for software architecture evaluation approaches under uncertainty which may implicitly or explicitly adopt continuous approaches. A systematic literature review (SLR) is a methodological mean to aggregate empirical studies, to systematically investigate a research topic, answer specific research questions, and finally determine the gaps and research directions for the research topic [61–63].

The objective of this study is to (i) provide a basic classification schema with which to categorise software architecture evaluation approaches under uncertainty; (ii) categorise the current design-time and run-time approaches for evaluating software architectures based on this schema; (iii) determine the necessary guidelines for developing a continuous evaluation approach; (iv) point out current gaps and directions in software architectures operating under uncertainty with respect to design-time and run-time evaluation for future research. Concretely, we aim to provide answers for the thesis's first research question: **RQ1: What is the state-of-the-art in software architecture evaluation under uncertainty and to what extent continuous architecture evaluation is used?** This research question is further divided into following:

- RQ1.1: How can the current research on software architecture evaluation under uncertainty be categorised and what are the current state-of-the-art approaches with respect to this categorisation? *The goal is to provide a categorisation of existing architecture evaluation approaches under uncertainty and classify the state-of-the-art approaches under this categorisation.*

- RQ1.2: What are the actions taken by these architecture evaluation approaches to deal with uncertainty? *The aim of this question is to demonstrate and discuss how the* *existing approaches deal with uncertainty and whether these actions can contribute to developing more continuous approaches.*

- RQ1.3: What are the current trends and future directions in software architecture evaluation under uncertainty and their consideration for continuous evaluation? *This question aims to show how we can benefit from the existing approaches to draw inspiration on the essential requirements and address the pitfalls when developing a continuous evaluation approach.*

The chapter is structured as follows: Section 2.1.1 identifies and explains the necessary concepts to ease the understanding of the review. Section 2.2 demonstrates the systematic literature review process, Section 2.3 provides an overview of the included studies from the chronological and distribution perspectives. Section 2.4 categorises the included studies with respect to a classification framework and presents the limitations of review. The related reviews are discussed in Section 2.6. New trends and research directions are discussed in Section 2.7. Finally, Section 2.8 concludes the work.

2.1.1 Preliminaries and Basic Concepts

In this section, we list descriptions of the main concepts used in this review to ease the analysis.

2.1.1.1 Architecture Design Decisions

The foundation of an architecture is in the set of design decisions taken [64–66]. The architects define the possible set of candidate architectures to serve a particular concern and then, based on their experience and knowledge, they choose the best candidate [67]. For example, in an IoT application, the architect could prefer processing the data in the cloud rather than the fog devices to improve energy consumption. However, this design decision could have a negative impact on performance. This motivates the need for software architecture evaluation.

2.1.1.2 Software Architecture

In the literature, software architecture is defined in many ways. In our work, we use the definition introduced by ISO/IEC/IEEE 42010:2011: "the fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution". This definition is complementary to [68, 69] and later ones [70].

In this context, a software architecture represents the abstractions for a software system by defining its structure, behaviour, and key properties [69]. These include software components (i.e. processing and computational elements), connectors (i.e. interaction elements), and their relation to the environmental conditions [68, 70].

2.1.1.3 Quality Attribute

We adopt the definition introduced by the IEEE Standard for Software Quality Metrics [71], where a quality attribute is "a characteristic of software, or a generic term applying to quality factors, quality sub-factors, or metric values". Examples of quality attributes are performance, reliability, energy consumption, availability, security, and so forth.

2.1.1.4 Uncertainty in Architecture Evaluation

A common issue in architecture evaluation is the presence of uncertainty. In architecture evaluation and decision-making, uncertainty is the lack of full knowledge about the outcomes of deploying the architecture options [45]. For instance, the architects may be uncertain about the effect of a proposed software architecture on benefit (e.g. performance, availability, *etc*) and cost. Uncertainty also may arise due to unpredictable situations in dynamic applications, such as IoT. For instance, sensors' ageing effects, the varying internet connectivity and mobility of sensors, fluctuations in QoS and so forth [8, 34, 72, 7].

2.1.1.5 Stakeholder

We adopt the notion used by ISO/IEC/IEEE 42010:2011: "an individual, team, organisation, or classes thereof, having an interest in a system". In this context, stakeholders are any people who have a stake in the success of the architecture, and of any systems that are derived from the architecture. So this could include customers, programmers, testers, reusers, architects, integrators, users, managers, *etc*. An architect is just one stakeholder among many, whose needs are less important (and hence lower priority) than the needs of many of the other stakeholders.

2.2 Systematic Literature Review Process

In this section, we will discuss the SLR protocol, how the systematic review process has been carried out, and finally the existing architecture evaluation approaches with respect to criteria and review objectives.

2.2.1 SLR Protocol

We have followed the systematic literature review guidelines and procedures [63] and the work of [56] to develop our review protocol. In particular, the protocol identifies the objectives of the review, the necessary background, research questions, inclusion and exclusion criteria, search strategy, data extraction and analysis of gathered data. One author has developed the review protocol and then the outcome has been revised by other authors to limit bias. The review objectives, background, and the research questions are discussed in Section 2.1, whereas other procedures are described below.

2.2.2 Inclusion and Exclusion Criteria

Initially, we needed to set up a criteria to aid in the search process and filtration of irrelevant studies. We considered English papers published in peer-reviewed journals, conferences, and workshops from 1990-2018. This time frame was chosen because one of the earliest well-known architecture evaluation approaches (e.g. SAAM [73]) was published in 1994. We excluded studies that do not have software architecture evaluation as one of its main contributions. We also excluded editorials, position paper, abstract, keynote, opinion, tutorial summary, panel discussion, or technical reports, panels and poster sessions. Moreover, we found that some studies are duplicated in different versions that appear as books, journal papers, conference and workshop papers. In this context, we included only the latest and most complete version. We provide a summary of the inclusion and exclusion criteria below. Publications are included if they cover all the inclusion criteria in Section 2.2.2.1, and publications are excluded if they fit any of the exclusion criteria in Section 2.2.2.2.

2.2.2.1 Inclusion Criteria

- Studies published between 1990-2018.
- Studies in the domain of software architecture evaluation. In particular, the study should include a software architecture evaluation method as one of its contributions.
- Studies focus on architecture-centric evaluation (i.e. focusing on the architecture decisions).
- Studies provides a quantitative evaluation for software architectures.

2.2.2.2 Exclusion Criteria

- Studies not related to the core objective of architecture evaluation [1].
- Studies that are non-peer reviewed.
- Studies not written in English and not accessible in full-text.

2.2.3 Search Strategy

The search strategy was performed to identify the studies through the following:

1. Applying an initial search to determine the current systematic reviews and mapping studies, and hence identifying significant related studies.
2. Using the concept of "quasi-gold" standard, as introduced by Zhang and Babar [74], where we performed a manual scan for the most well-known venues of the software architecture and the software engineering domains to cross-check the automated search results.
3. Performing several trials using different combinations of keywords derived from the main objectives of the review (i.e. automated search from recognised bibliographical data sources).
4. Performing an additional search to manually check and analyse the related references (snowballing) [75] to ensure that we did not miss any important study and hence guarantee a representative set of studies.

All the prior procedures aided us in defining valid search strings along with other procedures discussed in Section 2.2.3.1. For the venues, we manually searched the following:

– International Conference on Software Engineering (ICSE).

– International Conference on Software Architecture (ICSA) [2].

– European Conference on Software Architecture (ECSA).

[1]Some contributions fundamentally apply architecture evaluation at run-time, but these approaches did not mention that explicitly. In particular, in the area of models@run-time, self-managed systems, and self-adaptive systems, their evaluation can be an after-thought made through profiling configurations and recommending alternatives. Therefore, we have discussed closely related contributions to demonstrate how these approaches conduct evaluations and capture uncertainty.

[2]Formerly the Working IEEE/IFIP Conference on Software Architecture (WICSA)

Our manual search included the title, keywords, and abstract of each publication. After finishing the manual and automatic searches, we checked the differences between the results to guarantee the most appropriate coverage of the domain. We found that all the manual results were a subset of the automatic results (i.e. meeting the "quasi-gold" standard).

2.2.3.1 Keyword Selection

As mentioned above, we used both automatic and manual search. In the automatic search, we tried several keywords on search engines of electronic bibliographical sources. Manual search is not a practical procedure as it retrieves thousands of results, which is difficult to manually filter. However, we still performed a manual search (to meet the "quasi-gold" standard [74]) to ensure that we used the most suitable search queries.

One of the main challenges identified through our automatic search is a lack of well-defined terminology in this domain. Though we see that several "self-*" systems may implicitly incorporate some principles of a typical architecture evaluation approach. Further, searching in a very large domain such as self-adaptive systems, self-managed, self-awareness, *etc* would mislead the searching process and would not serve the main objective of this review. Therefore, to solve this problem and avoid missing any relevant studies, we used some generic keywords in the search query of automatic search (e.g. "run-time", "dynamic", *etc*). This led to retrieving some studies that were actually relevant to our search. We have also performed a manual search for the studies, which could seem to be a run-time architecture evaluation approach. To obtain our search query, we applied the following procedures:

1. Extract the major keywords from the objectives of review and main research topics.
2. Determine and try different spellings, related terms and synonyms for major keywords, if applicable.
3. Use the "advanced" search option to insert the complete search query and filter by date, if the bibliographical source allows for that (Section 2.2.3.2).
4. Pilot various combinations of search keywords in test queries.
5. Validate the results of (4) with "quasi-gold" standard.

From our pilot testing, we found that the notion of "continuous" architecture evaluation is used in different forms in the context of software architecture and software engineering with other closely-related alternative terms, such as run-time and dynamic. This is because the term "continuous" is not clearly defined. We also incorporated additional keywords which may implicitly refer to continuous evaluation, such as design-time and static (i.e. the state-of-the-art approaches for architecture evaluation). Furthermore, in other contexts, architecture evaluation is interpreted as architecture assessment or architecture analysis. Therefore, we tried to consider these related keywords in our search query and used them in an interchangeable manner.

The search query is composed of five major terms, Continuous **AND** Software **AND** Architecture **AND** Evaluation **AND** Uncertainty. To generate the main search query, we used the alternate keywords listed above. This is performed by connecting these terms through logical OR as follows:
(design-time **OR** run-time **OR** design time **OR** runtime **OR** static **OR** dynamic **OR** continuous) **AND** Software **AND** (architecture **OR** architectural) **AND** (Evaluation **OR** analysis **OR** assessment) **AND** Uncertainty

2.2.3.2 Bibliographical Sources

The selected databases present the most important and highest impact journals and conference proceedings. They also provided us with the ability to perform expert search with a variety of Boolean operations and limit the search on the Title, Abstract and Keywords fields and time frame, which returned more relevant results as compared to searching all the fields. For instance, this allowed us to use Boolean "OR" to try different spellings and synonyms, and use Boolean "AND" to link the major keywords (e.g. software AND architecture AND evaluation).
The electronic bibliographical sources used include:

– IEEE Xplorer (http://ieeexplore.ieee.org/Xplore/)

– ACM digital library (http://portal.acm.org/)

– SpringerLink (http://www.springerlink.com/)

– ScienceDirect (http://www.sciencedirect.com/)

– GoogleScholar (http://scholar.google.com/)

Note that we included Google Scholar as there are some of works in software architecture evaluation (e.g. ATAM), which were not retrieved in the first four databases. Our pilot tests showed that the final 100-200 results from Google Scholar were more relevant to other domains rather than architecture evaluation. To ensure the reliability of results, we limited the Google Scholar results to 1000.

Table 2.1 Summary of Search Results and Included Studies from each database. *Note that there are 14 studies generated from the snowballing process in addition to the 26 studies from the databases below, making a total of 40 included studies. Further, some studies may be duplicated in more than one database.*

Database	**Search Results**	**# Included Studies**
IEEE Xplorer	994	8
ACM digital library	2108	6
SpringerLink	999	3
ScienceDirect	524	2
GoogleScholar	1000	7
Other		
Snowballing Process	349	14
Total		40

2.2.4 Search Execution

In this stage, we executed the search process in Figure 2.1, realising the procedures in Section 2.2.3. Initially, we performed manual search (13 results) to determine the set of studies to be compared with automatic search list (i.e. "quasi-gold" standard). After that, we searched through all the search engines and bibliographical sources mentioned in Section 2.2.3.2 using search queries created in Section 2.2.3.1. All the search engines provided the option to save the results to excel spreadsheets, except for Springer which exports only the first 999 relevant results and ScienceDirect which does not have that option and hence a manual scan was performed. We then filtered the 5,230 primary studies using title, abstract, full-text (when needed), inclusion and exclusion criteria. We also snowballed the primary studies [75], where we scanned the list of references for the primary studies and the citations to add related works (14 results), which were not identified by the bibliographical engines. In the end we included 40 studies. The search results and number of included studies from each database and snowballing process are listed in Table 2.1.

2.2.5 Quality Assessment

To assess the quality of the findings, we adopted similar quality criteria to the ones used by [56]. The following criteria show the credibility of an individual study when analysing the results:

1. The study provides evidence or theoretical reasoning for their experimental evaluation and data analysis rather than relying on non-justified or ad hoc statements.

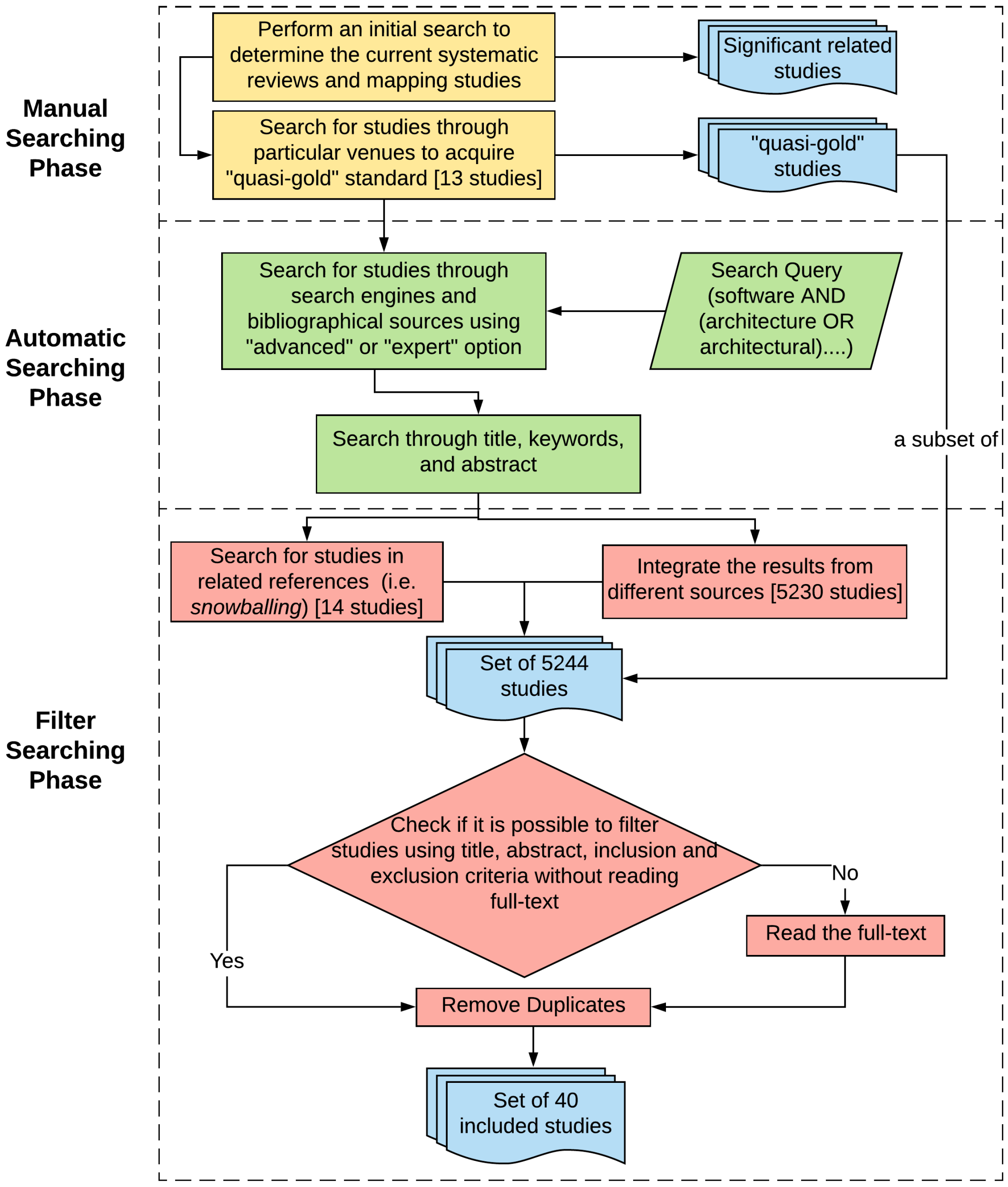

Manual Searching Phase
Perform an initial search to determine the current systematic reviews and mapping studies
Significant related studies
Search for studies through particular venues to acquire "quasi-gold" standard [13 studies]
"quasi-gold" studies
Automatic Searching Phase
Search for studies through search engines and bibliographical sources using "advanced" or "expert" option
Search Query (software AND (architecture OR architectural)....)
Search through title, keywords, and abstract
a subset of
Filter Searching Phase
Search for studies in related references (i.e. snowballing) [14 studies]
Integrate the results from different sources [5230 studies]
Set of 5244 studies
Check if it is possible to filter studies using title, abstract, inclusion and exclusion criteria without reading full-text
No
Read the full-text
Yes
Remove Duplicates
Set of 40 included studies

Figure 2.1 Search Execution.

Table 2.2 Data Extraction Criteria.

Extracted Data	**Description**
Study Identification	Unique ID for the study
Bibliographical references	Author, title, publication type, source and year
Study Type	Book, journal paper, conference paper, workshop paper
Study Focus	Main area and study objectives
Strengths and Limitations	Identified strengths and limitations of the approach and its application and its potentials for future directions

2. The study describes the context in which the research was conducted.

3. The design and implementation of the research is mapped to the study objectives.

4. The study provides full description of their data collection process.

To guarantee the credibility of studies, we checked all the primary studies against the above criteria, where we included only the studies that met each of the four criteria.

2.2.6 Data Extraction process

In this process, we performed a thorough scan for the 40 included papers to extract the relevant data, which were managed by Excel spreadsheets and bibliographical management tool BibTeX. The data extraction for the 40 studies was driven by the form depicted in Table 2.2 and the classification framework in Section 2.4.1. For the data analysis, we investigated the extracted data with respect to their relationships. The results of this process is given in the subsequent sections. The list of included studies are presented in Table A.2, A.3, and A.4.

2.3 Overview of the included studies

Here we provide an overview of the included studies with respect to their distribution along publication channels, over the years, and their ranks.

2.3.1 Distribution of Studies over Publication Channels

Most of the included studies (i.e. 40 studies) were published in the most well-known and prominent journals and conferences. In Table 2.3, we provide an overview of the included studies with respect to their publication channels and the number of studies per channel. We have checked the included studies against the criteria for quality assessment and confirmed

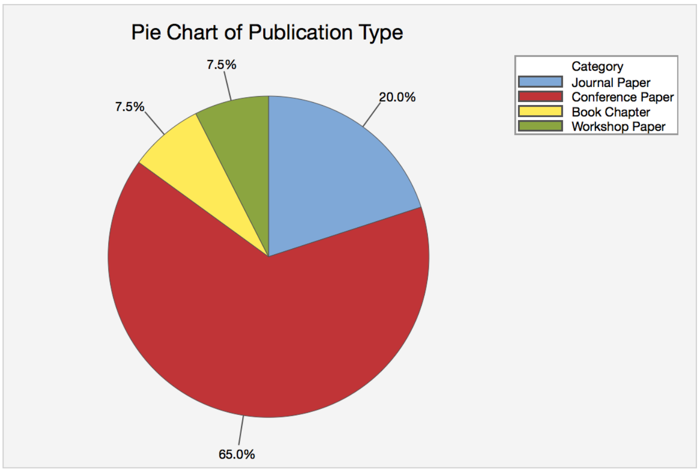


Figure 2.2 Distribution of the publication types.

that they indeed fulfil the quality criteria introduced in Section 2.2.5. We have also plotted the distribution of the included studies related to the publication channel (i.e. conference, journal, *etc*) in Figure 2.2. From these results, we found that there are a significant number of studies published in conferences (about 65%), followed by a smaller number of studies (20%) in journals. There are limited studies published in workshops (roughly 7.5%) and books (about 7.5%). This indicates that architecture evaluation approaches are still presented in conferences, and some of them have matured and been published through books and journals. These results suggest that architecture evaluation research is still maturing and hence it is attractive for researchers to develop new approaches.

2.3.2 Distribution of Included Studies Through the Years

By analysing the studies by year of publication, as depicted in Figure 2.3, we observe an increasing trend in the area of software architecture evaluation starting from 2003 till 2013 (with some oscillation). Though it may seem that interest in architecture evaluation has decreased in the past four years, there were recent studies that provided new architecture evaluation approaches. For instance, Abrahão and Insfran [76] reported a comparative analysis for ATAM and a model-driven method named Quality-Driven Architecture Derivation and Improvement (QuaDAI) method. There are also many approaches which either implicitly or explicitly adopt continuous evaluation through continuous quality control (e.g. [77, 78]), continuous development (e.g. DevOps [19]), continuous testing [79], *etc*. We excluded these studies because our main goal is to survey the continuous architecture evaluation approaches

Table 2.3 Distribution of included studies along with the publication channels.

Publication Channel	**No. of Studies**
IEEE International Conference on Software Engineering (ICSE)	8
International Conference on Software Architecture (ICSA)	5
Software Engineering for Self-Adaptive Systems (SEAMS)	4
Journal of Systems and Software (JSS)	3
Book	3
IEEE Transactions on Software Engineering (TSE)	2
IEEE Internet Computing	1
Software Quality Journal	1
Empirical Software Engineering	1
European Conference of Software Architecture (ECSA)	1
IEEE International Conference on Software Maintenance (ICSM)	1
ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)	1
IEEE International Conference on Autonomic Computing (ICAC)	1
ACM/SPEC International Conference on Performance engineering (ICPE)	1
International Conference on Software Reuse (ICSR)	1
International Conference on Quality of Software Architectures (QoSA)	1
IEEE International Conference and Workshops on Engineering of Computer-Based Systems (ECBS)	1
International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)	1
IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)	1
International Workshop on the Economics of Software and Computation (ESC)	1
IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOC)	1
Total	40

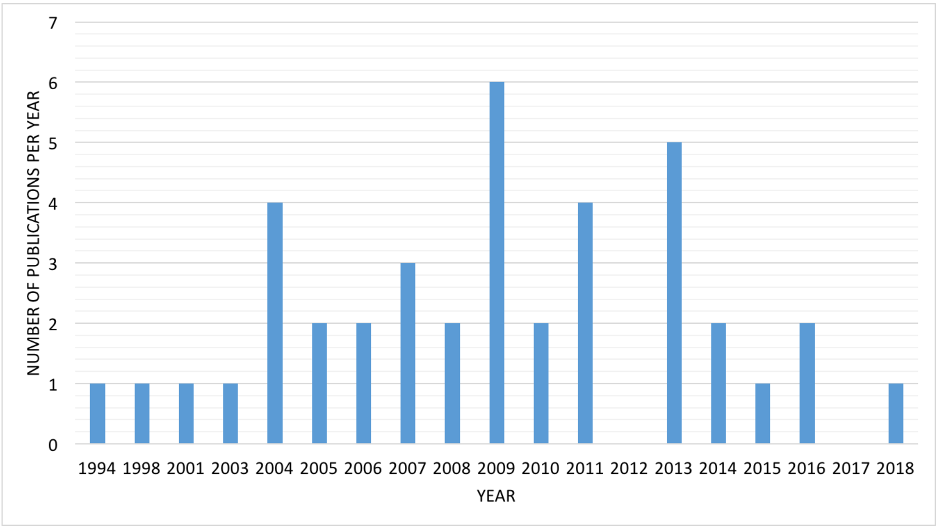


Figure 2.3 Distribution of the publication types.

Table 2.4 An overview of citation rate of included studies.

Cited by	<10	10-50	50-100	>100
Number of Studies (Total = 40)	5	17	5	13

from a high-level (architecture) perspective rather than low-level (code level) and including these studies could affect the analysis.

2.3.3 Citation Rate of Included Studies

We list in Table 2.4 the citation rate for the included studies, which was obtained from Google Scholar[3]. The citation rate is not meant for comparing studies; instead we use it to provide a rough estimate of the quality of papers. In particular, five studies were cited by fewer than 10 sources. Two of them were cited in 2004 and 2010 and hence we do not expect that they will be cited further, whereas the others are relatively new. Almost 45% of the studies (17 publications) were cited by 10-50 other sources, and five studies were cited 50-100 times. Thirteen studies have very high rates with more than 100 citations and the first ranked study was cited almost 1448 times. This shows that the included studies are, in general, highly cited, which signifies their quality and impact. In Table 2.5, we present the most cited publications. The first study is a book, and the remainder are journal and conference papers.

[3]http://www.googlescholar.com

Table 2.5 Featuring the most cited studies above 100 citations.

Rank	Ref	Author(s)	Year	Title
1	[80]	R. Kazman, M. Klein, P. Clements and others	2003	Evaluating software architectures
2	[73]	R. Kazman, L. Bass, G. Abowd, & M. Webb	1994	SAAM: A method for analyzing the properties of software architectures
3	[10]	P. Bengtsson, N. Lassing, J. Bosch, and H. Vliet	2004	Architecture-level modifiability analysis (ALMA)
4	[81]	R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, & G. Tamburrelli	2011	Dynamic QoS management and optimization in service-based systems
5	[21]	I. Epifani, C. Ghezzi, R. Mirandola, & G. Tamburrelli	2009	Model evolution by run-time parameter adaptation
6	[9]	R. Kazman, J. Asundi, & P. Clements	2001	Quantifying the costs and benefits of architectural decisions
7	[82]	P. Bengtsson,J. Bosch	1998	Scenario-based software architecture reengineering
8	[83]	G. Tesauro	2007	Reinforcement learning in autonomic computing: A manifesto and case studies
9	[84]	S. Cheng	2004	Rainbow: cost-effective software architecture-based self-adaptation
10	[85]	T. Al-Naeem, I. Gorton, and M. Babar	2005	A quality-driven systematic approach for architecting distributed software applications
11	[86]	N. Esfahani, E. Kouroshfar, & S. Malek	2011	Taming uncertainty in self-adaptive software
12	[87]	L. Zhu, A. Aurum, I. Gorton, & R. Jeffery	2005	Tradeoff and sensitivity analysis in software architecture evaluation using analytic hierarchy process
13	[88]	R. Calinescu & M. Kwiatkowska	2009	Using quantitative analysis to implement autonomic IT systems

2.4 Data Extraction Results

This section aims to provide answers for the first and second research question: *RQ1.1: How can the current research on software architecture evaluation under uncertainty be categorised and what are the current state-of-the-art approaches with respect to this categorisation?*; *RQ1.2: What are the actions taken by these architecture evaluation approaches to deal with uncertainty?* Our analysis of research topics addressed in each study and the systematic reviews and surveys found in literature (e.g. [12, 89, 56, 58, 60]) helped us in developing the following classification framework. This classification aided us in filtering, mapping, and understanding the architecture evaluation domain. We also discuss how the included evaluation approaches deal with uncertainty.

2.4.1 Classification Framework

Next, we will explain in detail the criteria presented in Figure 2.4.

1. **Stage of Evaluation:** The evaluation could occur at *design-time* and/or *run-time*. Design-time evaluation occurs before system deployment, whereas the run-time evaluation approaches use run-time and/or simulated data (e.g. QoS) to capture the dynamic behaviour of architecture decisions under uncertainty and use such information to profile or evaluate design decisions.

2. **Approaches to Evaluation:** The architecture evaluation methods can be categorised as *utility-driven*, *economics-driven*, and *learning-driven*.

   2.1. **Utility-driven:** We define utility-driven architecture evaluation methods as methods which adopt utility functions for decision-making. Utility functions are used in two contexts. First, it is a measure of the user's satisfaction with respect to a set of quality attributes. Second, it can be used to provide a stakeholder's preferences over a set of quality attributes, which is called a *Weighted Utility function*.

   2.2. **Economics-driven:** We define economics-driven architecture evaluation methods as the methods which quantitatively evaluate the impact of architecture decisions on quality attributes of interest using finance-inspired methods. In most cases, economics-driven approaches evaluate the architectures at design-time.

   2.3. **Learning-driven:** We define learning-driven architecture evaluation methods as methods which adopt machine learning techniques to improve the evaluation. These types of approaches are related to run-time architecture evaluation.

3. **Level of Stakeholder Involvement in Evaluation:** Full, partial, non (i.e. autonomous). As mentioned earlier, a stakeholder is a person or organisation who is involved in the architecture design.

4. **Addressing quality attributes:** There are some evaluation methods which focus on a *single* or *multiple* quality attributes.

5. **Management of Trade-offs:** A common problem in selecting an optimal architecture decision is the management of trade-offs [90]. For example, an architecture decision concerning a sensor could provide high response time but with low energy efficiency. So one objective is to select an architecture decision that can satisfy both quality attributes. There are two types of trade-off management: *manual* and *automated*. Manual management denotes the adoption of tools or techniques that require human-intervention, whereas automated indicates the use of parametric models that automatically select and/or shortlist trade-off candidates.

6. **Quality Attributes Supported:** We categorised the quality attributes supported into: *general* and *specific*. For *general*, we consider the literature that discusses the support of any quality attribute, such as performance, availability, reliability, *etc*. For instance, some studies propose generic methods that accept any quality attribute. However, there are others that focus on *specific* quality attributes (e.g. performance only).

7. **Dealing with uncertainty:** In the research literature there are approaches that deal with *explicit* or *implicit* uncertainty. *Explicit* approaches are those that consider uncertainty to be a main focus whereas other methods which do not mention uncertainty, but their tools and techniques could be used to handle uncertainties (i.e. *implicit*).

8. **Quality attribute monitoring and treatment:** This criterion is relevant to run-time evaluation approaches where quality attribute values are either determined through run-time monitoring or through prediction. Similarly for the treatment, there are two types [91, 92, 58]: *reactive* and *proactive*. A reactive approach triggers a switch after experiencing a drop in performance, a goal violation, *etc*. A proactive approach switches architecture options without experiencing a drop in performance; instead it is based on predictions that a significant change in performance may occur in the near future.

In Section 2.4.2 and 2.4.3, we aim to provide answers for the review's research questions mentioned earlier. We classify the architecture evaluation approaches to: design-time and

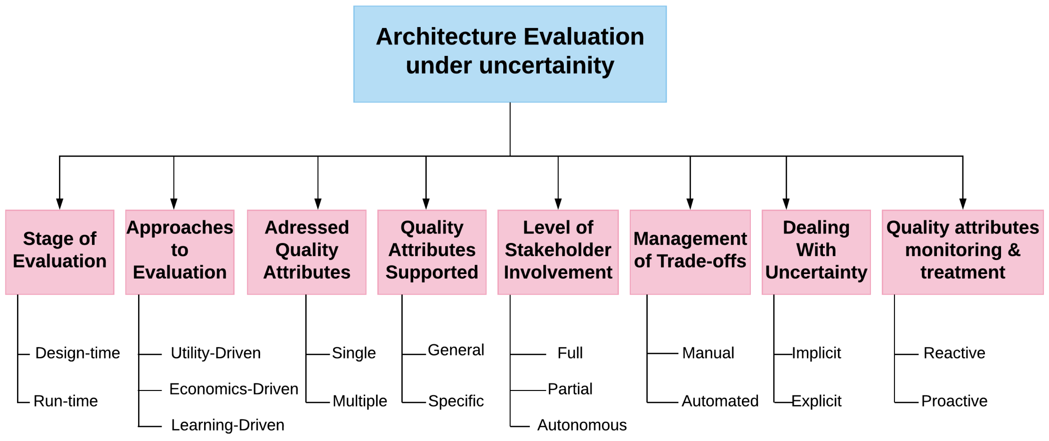


Figure 2.4 The proposed classification of architecture evaluation approaches

run-time. In each classification, we further classify and explain the existing architecture evaluation approaches with respect to classification framework (answering RQ1.1, introduced in Section 2.1). We also discuss the actions taken by these architecture evaluation approaches to deal with uncertainty (answering RQ1.2, introduced in Section 2.1).

2.4.2 Design-Time Architecture Evaluation

The design-time evaluation of software architectures aims to enable a proper specification of the problem which is the first step on that path of finding the suitable architecture decision. Documented efforts on systematic design-time architecture evaluation approaches are best linked to the seminal work of [73, 2, 9, 10]. These approaches focus on selecting design decisions and styles that are best fit for quality requirements of interest and their trade-offs. Table 2.6 and 2.7 summarise the included studies related to design-time architecture evaluation approaches with respect to the proposed classification. Next, we will discuss the relevant studies to utility-driven and economics-driven.

2.4.2.1 Utility-Driven

Within design-time evaluation, there are utility-driven approaches, which are guided by scenarios and/or parametric models.

Utility-driven Approaches guided by Scenarios The foundation of most architecture evaluation approaches rests on scenarios [12, 89, 55, 56]. These approaches use quantitative evaluation to determine the fitness of operational quality attributes. They elicit from stakeholders the utilities of architecture decisions and their effect on quality attributes of interest. Some of the scenario-based approaches have been validated and used in industry over the past decades [12].

- **Software Architecture Analysis Method (SAAM) [73]:** is the first well-known architecture evaluation method that aimed to reify quality attributes via a set of scenarios as a means to evaluate architecture design decisions under concern and identify risks in an architecture. It assesses the extent to which the architecture meets the quality attributes of interest. It was originally used for assessing modifiability, but it has been applied for other quality attributes, such as portability and extensibility. SAAM takes as input: business goals, software architecture description, and quality requirements that illustrate the interaction between stakeholders and the system being analysed. It then maps between scenarios and architecture components to assess anticipated changes to the system. This mapping can also be employed to estimate the amount of effort needed to handle these changes. The SAAM does not explicitly deal with trade-offs between quality attributes. The lack of trade-off management has contributed to the evolution of Architectural Trade-off Analysis Method.

- **Architectural Trade-off Analysis Method (ATAM) [80]:** is the most popular architecture evaluation method. It is an evolved version of SAAM. Unlike SAAM, the ATAM focuses on a comprehensive evaluation of quality attributes rather than just concentrating on modifiability, portability, and extensibility. ATAM is a generic design-time architecture evaluation method that uses scenarios to assess the value of architecture design decisions. Specifically, it aims to reveal the degree to which an architecture will meet its quality requirements (e.g. availability, security, usability, and modifiability), and the interaction between those goals through trade-off analysis.

  Uncertainties and risks, linked to the deployment, are implicitly discussed and mitigated through envisioning a set of scenarios, taking the form of use case, growth, and exploratory scenarios [2, 93, 80] as defined by the ATAM. A use case scenario reveals how stakeholders envision the system usage. A growth scenario illustrates planned and foreseen refinements to the architecture, whereas exploratory scenario help to probe the extent to which the architecture can adapt to future changes (e.g. functionality upgrades, new quality attribute requirements). Hence, the evaluation and its conclusions are highly dependent on the choice of these scenarios. The ATAM defines and records the risks that may threaten the achievement of quality attribute goals. These include architecture decisions leading to subsequent problems in some quality attributes *(risks)*, architecture decisions where a slight alteration results in significant impact on quality attribute responses *(sensitivity points)*, and the simultaneous effect of a single decision on multiple quality attributes *(trade-offs)* [94]. ATAM focuses on the risks and benefits of architecture decisions and does not explicitly consider cost.

- **Cost-Benefit analysis method (CBAM) [9]:** is an architecture evaluation method that extends ATAM to provide cost/benefit analysis of architecture design decisions. The CBAM was created to "develop a process that helps a designer choose amongst architectural options, during both initial design and its subsequent periods of upgrade, while being constrained to finite resources" [3]. Although the CBAM uses cost/benefit information to value the architecture design decisions and to justify their selection, this method is unable to dynamically profile the added value of architecture decisions, which is essential for applications operating in uncertain environments (such as IoT). It only deals with uncertainty through a set of scenarios, similar to ATAM.

- **Scenario-Based Architecture Re-engineering (SBAR) [82]:** is another scenario-based architecture evaluation method that uses different techniques to assess the quality attributes of interest and implicitly deal with uncertainty: scenarios, simulation, and mathematical modelling. For example, if a quality attribute is concerned with development and design-time properties, such as maintainability and reusability, scenario-based techniques can be best utilised. Scenario-based analysis can be still used for behavioural and run-time properties, such as performance and fault-tolerance, simulation and/or mathematical models can better provide meaningful insights and can complement scenario-based ones. A major concern in SBAR is its use of impractical assumptions. For instance, to address the reusability concern, the architect has to define all the scenarios related to the reuse of parts of the architecture, which is not feasible.

- **Architecture-Level Modifiability Analysis (ALMA) [10]:** Unlike ATAM and CBAM, ALMA focuses on a single quality attribute, and hence it does not consider trade-offs. It utilises probabilities to determine the likelihood of the impact of scenarios at the software architecture level with respect to modifiability concerns (e.g. maintenance cost prediction and risk assessment).

- **Systematic Quantitative Analysis of Scenarios' Heuristics (SQUASH)** [95]: is a systematic quantitative method for scenario-based value, risk, and cost analysis. The method focuses on evaluating the relative benefits of proposed scenarios in early stages of architecting. The method extends some steps from CBAM by providing extensive evaluations of the internal structure of the scenarios to predict the quality attributes of architecture decisions. In this context, the approach relies more on stakeholders than CBAM and hence it may not be easy to apply in practical settings.

- **Analytic Principles and Tools for the Improvement of Architectures (APTIA) [96]:** is an architecture improvement method that combines existing architecture evaluation methods (such as ATAM, CBAM, *etc.*) through: quality attribute models, design principles in the form of tactics, scenario-based quality attribute elicitation and analysis, explicit elicitation of the costs and benefits of architecture decisions from stakeholders, and the use of architecture documentation templates. It also adds new steps to the analysis. Particularly, it identifies design decisions linked to the analysis rather than stating their future problems. It was able to aid the team of architects to propose architecture alternatives for a complex system and in a short period of time.
- **Architectural Tradeoff Method using Implied Scenario (ATMIS) [97]:** is an extension of ATAM through the adoption of Implied Scenarios for security testing [98]. The main aim of this approach is to apply trade-off analysis between security and any other quality attribute through the use of implied scenarios.

ATAM, CBAM, ATMIS, SQUASH and APTIA partially capture uncertainty, although they do not operate at run-time. However, they suffer from the limitation of design-time analysis (i.e. high reliance on stakeholders). ATMIS is also specifically tailored for security. ALMA is similar to ATAM and CBAM, in the context of taking more utility-driven perspective for the evaluation. It aids in performing architecture evaluation more systematically than SBAR. Scenario-based approaches do not provide explicit management for uncertainties, and include manual tools/techniques which may not be effective at run-time. Further, some of these approaches handle trade-offs manually (ATAM, CBAM, ATMIS, and APTIA) or do not consider the trade-offs at all (SAAM, SBAR, SQUASH, and ALMA).

Summary: Scenario-based evaluation approaches can be described as best-effort, where the evaluators' expertise, choice of stakeholders *etc.*, are all factors that influence the evaluation. In particular, these approaches heavily rely on human inputs and expert judgement. These processes can thus suffer from subjectivity, bias and can never be complete.

Utility-driven Approaches guided by Parametric models: The previous scenario-driven approaches used simplistic mathematical models and relied heavily on stakeholders for the elicitation of scenarios and on expert evaluators for the impact of these scenarios on quality attributes. Here, we will discuss approaches that assess architecture decisions using parametric models.

– Analytic Hierarchy Process (AHP) [99] is a mathematical modelling tool used in dealing with complex decision-making. AHP has been used in two contexts for architecture evaluation: managing trade-offs and determining the relative importance of scenarios and decisions. Zhu et al. [87] adopted AHP to explicitly determine the trade-offs being made and the relative size of these trade-offs. It has been used with CBAM to determine the relative importance of scenarios through pair-wise comparisons [100]. It relies on eliciting the benefits and costs from stakeholders, and hence suffers from the same limitations of scenario-based approaches. ArchDesigner [85] is an architecture framework that first adopts AHP to elicit from stakeholders their preferred architecture decisions. It then uses Integer programming to determine the optimal architecture decision, which satisfies conflicting stakeholder quality goals subject to project constraints such as time and cost. LiVASAE [101] (a lightweight value-based architecture evaluation technique) attempts to measure the uncertainty level using AHP and also provides three simplified evaluation steps as compared to the CBAM. All these approaches rely on stakeholders for evaluating the candidate architecture decisions as well as their benefits and costs.

– Various methods have adopted utility theory to shortlist the candidate architectures operating under uncertainty, such as [102, 44, 45]. Osterlind et al. [102] used utility theory to balance quality attributes against each other to obtain the best possible architecture. GuideArch [44] is an architecture framework that explicitly models the uncertainty of architecture decisions using fuzzy logic to rank and determine the optimal architecture decision. However the use of fuzzy logic cannot be empirically evaluated and adjusted. Letier et al [45], designed a method, based on GuideArch and CBAM, to deal with uncertainty. Utility theory and Monte Carlo simulation were used to calculate the costs and benefits of candidate architecture decisions under uncertainty. The latter approach made an assumption that the probability distributions of model attributes are accurate; this may affect its applicability, particularly in dynamic environments.

– Grunske et al. [103] proposed a method to automate the trade-off management process using an evolutionary algorithm. The aim of the approach was to rank design decisions (architecture refactorings) by taking into consideration competing quality goals.

However, this was an initial attempt without a complete evaluation (i.e. it has not been applied on architecture evaluation methods).

Summary: Though all the prior approaches provide some management for uncertainties, they suffer from the same concerns: the high reliance on stakeholders for the elicitation of the relative importance (i.e. rank) of architecture decisions and their impact on quality attributes.

Other Approaches

- The architecture evaluation approach in [104] focuses on middleware and design pattern integration for developing adaptive self-managing architectures at design-time that are able to recover from failures. This approach suffers from the same limitation of design-time approach: the design-time patterns (i.e. decision) may not be able to handle the changing environmental conditions at run-time. Architecture Software Quality Assurance (aSQA) [105] is an evaluation method that uses metrics to determine the user's satisfaction towards prioritised quality requirements, especially in agile software projects. Despite it focuses on a single point of evaluation to lighten the evaluation process, yet it misses the main aim of evaluation (i.e. assess the impact of architecture decisions on quality attributes).

- Decision-centric software architecture evaluation method (DACAR) [106] assesses the architecture decisions made or to be made independently, rather than evaluating the whole architecture. The method could be potentially adopted in agile projects, since the architecture decisions could be evaluated as they appear in the process. However, the approach is not as flexible as scenario-based methods in obtaining the novel paradigms and significant change domains from stakeholders.

- Further, there is another scenario-based method which is different from the commonly used scenario-based architecture evaluation methods. The method is named Performance Assessment of Software Architecture (PASA) [107]. In PASA, the architect uses the architecture specification to form performance models. The generated models are then utilised to assess whether the performance objectives are met. ATAM uses scenarios to identify, prioritise and refine the most important quality attribute goals by building a utility tree, where each leaf in the tree represents a scenario. PASA instead employs scenarios in the form of UML and sequence diagrams to demonstrate how the software architecture will achieve the performance objectives. Continuous Performance Assessment of Software Architecture (CPASA) [18] is one the few explicit attempts of continuous evaluation. It is an extension of PASA, with an explicit focus on deployment in agile development process. It provides an interactive system that aids the architect in the automatic assessment of performance attributes through modelling of architecture decisions. They define "continuous" assessment as the production of continuous performance evaluation tests. Despite the attempts in PASA and CPASA to handle cost-benefit trade-offs, (i) the evaluation was incomplete; (ii) they are not using any run-time information to refine their architecture decisions; and (iii) the evaluation lacks run-time monitoring and forecasting of the performance of architecture decisions. In such cases architecture is, at best, a modelling tool, which may (or may not) be applicable in dynamic environments. Therefore, these approaches are still design-time evaluation approaches.

Summary: The prior approaches are either domain-specific or cannot be used to evaluate new paradigms. They also still rely on stakeholders for the elicitation of benefits and costs of architecture decisions.

2.4.2.2 Economics-Driven

Traditional cost-benefit analysis methods have been used to evaluate software. For instance, Cellini et al. [108] computed the net benefit of software through the deduction of total costs from total benefits. These attributes have been obtained from software architects through a group of questions (e.g. "what is the state of the world in the absence of the program ?"). CBAM [9] is a utility-driven architecture evaluation method that uses cost-benefit to assess the impact of architecture decisions on quality attributes of interest. This approach partially capture uncertainties which motivated the need to integrate some finance-inspired approaches into the software engineering field. Boehm [109, 110] was among the first to incorporate economics and finance theories to evaluate software design decisions. Examples of these approaches: Net Present Value (NPV) [111, 112], Modern Portfolio Theory (MPT) [113], and Real Options Analysis (ROA) [25]. In Table 2.7, we outlined the software architecture evaluation approaches, which adopted economics-driven principles (i.e. excluded the studies, which dealt with design evaluation (e.g. modularity in design), but were not scoped for high-level architecture evaluation, such as [38, 37, 36, 43], *etc*).

- **Net Present Value (NPV) [112, 111]:** is a popular approach used to value software. It values the software project by eliciting the probability of investing in an established discount rate or interest. A positive NPV indicates that it is financially beneficial to invest (i.e. deploy this architecture decision) and negative NPV is the opposite. It has been originally used in [114, 115]. However, NPV has been discouraged, because it ignores the value of the flexibility under uncertainty [38, 37, 36, 39, 40, 43].

- **Modern Portfolio Theory (MPT) [113]:** was first introduced by the Nobel prize winner Markowitz in 1950s. MPT aims to improve the decision-making process by allocating capital to a portfolio of diverse investment assets. MPT handles uncertainty through the distribution of capital among assets to minimise risk and maximise the returns. In particular, it provides a weighted combination (i.e. portfolio) of the assets, where the weight denotes the investor's share of capital in each asset. In this context, MPT seeks to demonstrate the rewards of having a diversified portfolio of assets. MPT is well-known in finance domain and has been also introduced in software engineering domain as a means to deal with uncertainties. In software architecture [116], it has been adopted with CBAM to determine which portfolio of architecture decisions will deliver value by considering sustainability dimensions. Although this approach explicitly deals with uncertainty, yet it provides a short-term value. It does not embed flexibility as real options analysis.

- **Real Options Analysis (ROA) [25]:** provides an analysis paradigm that emphasises the value-generating power of flexibility under uncertainty. An option is the right, but not the obligation, to make an investment decision in accordance to given circumstances for a particular duration into the future, ending with an expiration date [24]. Real options are typically used for real assets (non-financial), such as a property or a new product design. ROA treats uncertainty as an option which may provide future opportunities to the project, which could be exercised when it provides a high option value. On the contrary, MPT specifically deals with financial assets and considers uncertainty as a risk that should be minimised.

  Real options theory has been discussed thoroughly in the literature in various domains, including systems design and engineering [36], COTS-centric development [38, 43], architecture patterns [42], software design and refactoring [37], and software architecture [41, 117]. Baldwin et al. [36] enact the use of real options in systems design and engineering. They emphasise the contribution of modularity on system designs in the form of real options. Similarly, Sullivan et al. [39] propose an options-based analysis approach to evaluate the use of the spiral model in software development [37]. The goal of their approach is to deal with uncertainty by providing alternative options. The decision for an option is based on knowledge from prior phases of the software development. Erdogmus et al. [38] investigate the means of valuing the strategic flexibility in software development using real options. The main objective is to study the economic incentive of choosing a favourable COTS centric strategy in a project. According to their results, real options theory is preferred over NPV analysis, as the latter ignores the value of the flexibility in COTS-centric projects. The approach in [42], a variety of different architecture patterns are valued by CBAM along with real options with respect to their relative impacts on system quality goals. Tansey et al. integrated COCOMO II as the software-development cost model along with ROA to quantify the potential business value in service-based applications. Moving to software architecture context in specific, Bahsoon et al. [41] were the first to use real options analysis along with CBAM to measure the architecture's stability. They then used their method to value scalability in distributed architectures [117].

2.4.2.3 Relevance to Continuous Evaluation

As mentioned earlier, a continuous architecture evaluation approach starts at design-time and continues to operate at run-time. Therefore, the first component should be a design-time architecture evaluation. The approach should be able to cope with operational uncertainties and dynamicity, which may be encountered at run-time. Therefore, a systematic design-time evaluation approach that explicitly deals with uncertainty rather than either relying on ad hoc evaluation or implicit mitigation of uncertainty is necessary.

2.4.3 Run-time Architecture Evaluation

By run-time evaluation, we refer to approaches that use run-time and/or simulated data (e.g. QoS data) to capture the dynamic behaviour of architecture decisions under uncertainty and to use such information to profile and evaluate design decisions. Table 2.8 summarises the run-time architecture evaluation methods studied.

2.4.3.1 Utility-driven

In software architecture evaluation, utility functions are commonly used to select the optimal architecture option. This approach has also been adopted to determine the stakeholder's preferences towards quality attributes of interest. Therefore, it is utilised as a way to model trade-offs between quality attributes. Utility functions have been used at run-time for self-adaptive and self-managed systems, such as [86, 118–126].

– Heaven et al. [120] reported on an approach tailored for self-managed software systems. The approach provides the following features: high-level task planning, architecture

Table 2.6 Representative Contributions for Design-time Architecture Evaluation.

Study	Approaches to Evaluation	Addressing QA	QA supported	Level of stakeholder involvement	Management of trade-offs	Dealing with uncertainty
[73]	Utility-driven	Multiple	Specific (Modifiability, Portability, Extensibility)	Full	No Support	Implicit
[80]	Utility-driven	Multiple	General	Full	Manual	Implicit
[9]	Utility-driven	Multiple	General + Specific (Cost)	Full	Manual	Implicit
[10]	Utility-driven	Single	Specific (Modifiability)	Full	No Support	Implicit
[82]	Utility-driven	Multiple	Specific (Performance, Fault-tolerance, Maintainability, Reusability)	Full	No Support	Implicit
[95]	Utility-driven	Multiple	General + Specific (Cost)	Full	No Support	Implicit
[96]	Utility-driven	Multiple	General + Specific (Cost)	Full	Manual	Implicit
[87]	Utility-driven	Multiple	General + Specific (Cost)	Full	Manual	Explicit
[85]	Utility-driven	Multiple	General + Specific (Cost)	Partial	Automated	Implicit
[103]	Utility-driven	Multiple	General + Specific (Cost)	Partial	Automated	Implicit
[101]	Utility-driven	Multiple	General + Specific (Cost)	Full	Manual	Explicit
[104]	Utility-driven	Multiple	Specific (dependability, reliability and maintainability)	Full	No Support	Implicit
[18]	Utility-driven	Multiple	General + Specific (Cost)	Full	Manual	Implicit
[100]	Utility-driven	Multiple	General + Specific (Cost)	Full	Manual	Explicit
[105]	Utility-driven	Multiple	General + Specific (Cost)	Partial	Automated	Explicit
[97]	Utility-driven	Multiple	General + Specific (Security)	Full	Manual	Explicit
[102]	Utility-driven	Multiple	General + Specific (Cost)	Partial	Automated	Explicit
[106]	Utility-driven	Multiple	General	Full	No Support	Implicit
[44]	Utility-driven	Multiple	General + Specific (Cost)	Partial	Automated	Explicit
[45]	Utility-driven	Multiple	General + Specific (Cost)	Partial	Automated	Explicit

Table 2.7 Representative Contributions for Design-time Architecture Evaluation (Continued).

Study	**Approaches to Evaluation**	**Addressing QA**	**QA supported**	**Level of stakeholder involvement**	**Management of trade-offs**	**Dealing with uncertainty**
[41]	Economics-driven	Multiple	Specific (Stability) + Specific (Cost)	Partial	Manual	Explicit
[117]	Economics-driven	Multiple	Specific (Scalability) + Specific (Cost)	Partial	Manual	Explicit
[42]	Economics-driven	Multiple	General + Specific (Cost)	Partial	Manual	Explicit
[116]	Economics-driven	Multiple	General + Specific (Cost)	Partial	Manual	Explicit

configuration and reconfiguration, and component-based control. Their approach uses weighted utility functions to represent quality attributes and determine the total utility of configurations by taking into account reliability and performance concerns. Esfani et al. [86] proposed an approach that elicits from stakeholders their beliefs regarding uncertainty with respect to attributes such as network bandwidth. In particular, the stakeholders provide an estimate for the range of uncertainty with respect to the expected level of input variation. The approach also quantifies the uncertainty through profiling by comparing the actual values with estimates from stakeholders and hence provides probability distributions for the variation in data collection. After that the overall uncertainty is computed using fuzzy math. Veritas [124] is another utility-driven approach which adopts utility functions to guide the test adaptation process as part of a run-time testing framework.

– Cooray et al. [121] proposed a proactive approach, which continuously updates reliability predictions in response to environmental changes. The approach has proved its efficiency in adapting the system before it experiences a significant performance drop. However, the approach does not consider cost and suffers from scalability issues.

– Recently, the approach in [126] proposes an architecture evaluation approach inspired by CBAM [9] (mentioned in Section 2.4.2) for run-time decision-making in self-adaptive systems that considers benefits and costs of decisions. The approach adopts a weighted utility measure of the qualities that the adaptation decisions can provide to the stakeholders. Although this approach seems to provide continuous evaluation, it requires additional elements, such as online machine learning techniques, and extra experimental evaluation for applicability and efficiency.

Models@run.time [127, 128] includes built-in mechanism for evaluating the behaviour of software systems through continuous monitoring, planning, and model transformation. However, the effort was not discussed from the architecture evaluation angle. In particular, the authors state that "models of the functional and/or non-functional software behaviour are analysed at run-time, in order to select system configurations that satisfy the requirements" [129]. Models@run.time operates on the assumption that possible run-time configurations have already been evaluated and encoded in the system, where evaluation can be an afterthought through profiling configurations and recommending alternatives. It aims to reevaluate requirements satisfaction while the system is evolving [130].

– In the spirit of models@run.time, several approaches which are architecture-centric have been discussed in the context of self-adaptive and managed architectures [131, 132, 46, 133, 59]. Examples of these approaches include [118, 21, 134, 81, 135, 123, 121], which formally analyse their architectural models. The Rainbow framework [118] uses Markov processes to compute the expected aggregated impact of each strategy on each quality attribute. It requires high human intervention to determine the effects of strategies with respect to quality attributes (i.e. predefined probabilities) [136]. Epifani et al. [21] proposed a utility-driven approach leveraging a Discrete Time Markov Chain approach and Bayesian estimators to provide continuous automatic verification of requirements at run-time and support failure detection and prediction. Their approach does not consider multiple quality attributes, switching cost, and variance in run-time data. Additionally, Morin et al. [134] integrated model-oriented and aspect-oriented techniques to support run-time adaptation. The latter applies offline analysis by using synthetic data to analyse and evaluate the quality attributes of a group of system configurations. Ghezzi et al's [123] method is one of the few that complement design-time with run-time analysis. At design-time, the approach integrates goal-refinement methodologies with Discrete Markov Time Chains to determine all possible execution paths for the goal. At run-time, it exploits utility functions to measure the utility of paths, based on assumptions. For example, the utility for a 5ms response time is 1 and so forth. Given these assumptions, a hill climbing algorithm is used to select the optimal goal. We will discuss [81, 135, 121] in Section 2.4.3.2.

– Different from models@run.time, Yang et al. [119] proposed a utility-driven approach that extends the scenario-based approaches (e.g. ATAM, CBAM) and profiles the run-time information to better manage the QA trade-offs. It aims to improve decision-making and handle the uncertainty which may be better managed at run-time. In particular, their approach determines the potential QA trade-off points, designs the adaptive architecture decisions, and finally deploys their system on a middleware platform to collect run-time information. Though the latter approach is one of the few attempts to extend scenario-based approaches at run-time, it lacks the ability to learn over time and hence cannot forecast the future potentials of architecture decisions.

Summary: Generally, the major problems of the prior approaches are: (i) the high reliance of stakeholders for utility estimations, which is subject to their experience; (ii) the utility functions are hard to define; (iii) there is complexity and uncertainty in the quantification of utility values. This motivated the need to integrate learning techniques to learn over time and hence improve the analysis (discussed in next section).

2.4.3.2 Learning-driven

"The effectiveness of model-based reasoning about the properties of a software system depends on the accuracy of the models used in the analysis" [129]. For example, some models may become obsolete due to evolution in the software architecture. The same applies to the use of utilities for evaluation and decision-making. Therefore, machine learning could be adopted to better enhance the evaluation through profiling the observations of the system properties over time, as in the following studies [83, 137, 138, 122].

– In the context of using reinforcement learning techniques, Tesauro et al. [83] integrated queuing policies with reinforcement learning, forming a hybrid approach to enhance the dynamic resource-allocation decision-making process in data centres. The approach suffers from scalability and performance overhead. A reinforcement learning online planning technique was used by Kim et al. [137] to improve a robot's operation with respect to changes in the environment, by dynamically discovering the appropriate adaptation plans. However, it does not continuously evaluate the cost-effectiveness of architecture decisions over time. These approaches [83, 137] tend to be domain-specific. Further, Calinescu et al. [138] proposed initial attempts for the use of Bayesian learning and ageing coefficients to update the model parameters, where the ageing coefficients may be a useful element for a continuous evaluation approach. Because it may then allow the architect to tune the sensitivity of approach to present/past observations. Though their work had potential, it was still work-in-progress (i.e. initial evaluation for the approach has been performed and hence it requires further analysis).

Table 2.8 Representative Contributions for Run-time Architecture Evaluation.

Study	Approaches to Evaluation	Addressing QA	QA supported	Level of stakeholder involvement	Management of trade-offs	Dealing with uncertainty	QA Monitoring &treatment
[84]	Utility-driven	Multiple	General + Specific (Cost)	Autonomous	Automated	Explicit	Reactive
[83]	Utility-driven Learning-driven	Single	Specific (Performance)	Autonomous	No Support	Explicit	Reactive
[88]	Utility-driven	Multiple	Specific (Performance, Energy Consumption)	Autonomous	Automated	Implicit	Reactive
[137]	Utility-driven Learning-driven	Multiple	General	Autonomous	Automated	Explicit	Reactive
[120]	Utility-driven	Multiple	Specific (Reliability , Performance)	Partial	No Support	Implicit	Reactive
[119]	Utility-driven	Multiple	General + Specific (Cost)	Partial	No Support	Implicit	Reactive
[138]	Utility-driven Learning-driven	Multiple Performance)	Specific (Reliability,	Autonomous	No Support	Explicit	Reactive
[81]	Utility-driven	Multiple	Specific (Reliability, Performance)	Partial	Automated	Explicit	Reactive
[86]	Utility-driven	Multiple	General	Partial	Automated	Explicit	Reactive
[122]	Utility-driven Learning-driven	Multiple	General + Specific (Cost)	Autonomous	Automated	Explicit	Reactive
[124]	Utility-driven	Single	Specific (Energy Consumption)	Partial	No Support	Explicit	Reactive
[21]	Utility-driven	Single	Specific (Reliability)	Partial	No Support	Implicit	Proactive
[123]	Utility-driven	Multiple	General	Partial	No Support	Explicit	Proactive
[121]	Utility-driven	Multiple	Specific (Reliability, Efficiency)	Autonomous	Automated	Implicit	Proactive
[139]	Utility-driven Learning-driven	Multiple	Specific (Performance)	Autonomous	No Support	Explicit	Proactive
[126]	Utility-driven	Multiple	General + Specific (Cost)	Autonomous	Automated	Explicit	Reactive

– FUSION [122] is another learning-driven approach that adopts a machine learning algorithm named Model Trees Learning to tune the adaptation logic towards unpredictable triggers, rather than using static analytical models. It also uses utility functions to determine the benefit of models in question. The major benefit of FUSION is its ability to learn over time and improve the adaptation actions due to the promising learning accuracy. However, FUSION has the following limitations: (i) it is specifically tailored to feature modelling; and (ii) it only detects goal violations, i.e. constraints, but does not have the ability to check if the current architecture option is getting worse.

– Moreno et al. [139] recently propose a proactive latency-aware adaptation approach that constructs most of the Markov Decision Processes offline through stochastic dynamic programming. Their method focuses on optimising the latency of adaptation action based on forecasts, without considering the cost of architecture decisions and multiple stakeholder concerns.

Summary: As far as we know, the majority of run-time evaluation approaches rely on models for the analysis, which may be subject to scalability and complexity concerns. For that, these approaches have adopted some machine learning algorithms, such as Reinforcement learning, to update their models at run-time. Despite their potential, these approaches suffer from the following limitations: (i) they assume that the quality data about architecture decisions is available at every timestep, which may not be true in non-stationary environments such as IoT; and (ii) they lack the capability for checking whether the current architecture decision is getting worse.

2.4.3.3 Relevance to Continuous Evaluation

The most important component in a continuous evaluation approach is the run-time approach to be included. Some of the above approaches seem to provide important elements for a run-time approach in terms of providing learning techniques. These techniques could aid the architect in predicting the impact of architecture decisions on quality attributes under different scenarios of interest. However, non of the above approaches was explicitly used in the context of software architecture evaluation.

2.4.4 Limitations of the Review

Though this review was developed following the typical systematic literature review methodology [61–63], there are some limitations that require clarification:

Table 2.9 Summary of Contributions for Design-time Architecture Evaluation approaches with other categories.

	Category		**Representative Contributions**
Design-time	Addressing QA	Single	[10]
		Multiple	[73, 80, 9, 95, 82] [96, 87, 101, 100, 103] [85, 102, 45, 41, 18] [116, 117, 42, 106]
	QA Support	General	[80, 9, 95, 96, 87] [103, 101, 100, 102, 44] [45, 41, 116, 18, 106]
		Specific (Cost)	[9, 95, 96, 87, 85] [103, 101, 18, 100, 105] [102, 44, 45, 41, 117] [42, 116]
		Specific (Other QA)	[73, 10, 82, 104, 97] [41, 117]
	Level of stakeholder Involvement	Full	[73, 80, 9, 95, 101] [82, 10, 96, 87, 104] [100, 18, 97, 106]
		Partial	[103, 85, 102, 45, 41] [117, 42, 116, 44, 105]
	Management of Trade-offs	Manual	[80, 9, 96, 87, 101] [18, 100, 97, 41, 117] [42, 116]
		Automated	[85, 103, 105, 102, 44, 45]
		No Support	[73, 10, 82, 106, 95, 104]
	Dealing with Uncertainty	Implicit	[73, 80, 9, 82, 10] [95, 96, 85, 103, 104] [18, 106]
		Explicit	[87, 101, 100, 102, 45] [41, 105, 97, 117, 42] [44, 116]

Table 2.10 Summary of Contributions for Run-time Architecture Evaluation approaches with other categories.

	Category		**Representative Contributions**
Run-time	Addressing QA	Single	[83, 124, 21]
		Multiple	[84, 137, 88, 120, 119] [138, 81, 86, 122, 123] [121, 139, 126]
	QA Support	General	[84, 137, 119, 86, 122] [123, 121, 139, 126]
		Specific (Cost)	[84, 119, 122, 126]
		Specific (Other QA)	[83, 21, 120, 138, 81, 124]
	Level of stakeholder Involvement	Partial	[120, 119, 81, 86, 124, 21, 123]
		Autonomous	[84, 83, 88, 137, 138] [122, 121, 139, 126]
	Management of Trade-offs	Automated	[84, 137, 88, 81, 86] [122, 121, 126]
		No Support	[83, 120, 119, 138, 124] [21, 123, 139]
	Dealing with Uncertainty	Implicit	[88, 120, 119, 21, 121]
		Explicit	[84, 83, 137, 138, 86] [122–124, 81, 126, 139]
	QA Monitoring and Treatment	Reactive	[84, 83, 119, 88, 120] [137, 81, 138, 86, 122] [124, 126]
		Proactive	[21, 123, 121, 139]

Table 2.11 Summary of Contributions for Architecture Evaluation approaches with respect to approaches to evaluation.

Stage of Evaluation	**Approaches to Evaluation**	**Representative Contributions**
Design-time	Utility-driven	[73, 80, 9, 95, 82, 96, 10, 103, 87, 101, 100, 85] [104, 18, 97, 102, 44, 45, 105, 106]
	Economics-driven	[41, 117, 42, 116]
Run-time	Utility-driven	[84, 83, 120, 21, 88, 81, 119, 121, 122] [123, 86, 124, 126, 138, 139]
	Learning-driven	[83, 137, 138, 122, 139]

- The main threats to validity in this SLR is the selection bias when including the studies and extracting the data. To resolve that in terms of determining the relevant studies a research protocol (Section 2.2) was conducted. We applied this protocol to set out the objectives of the review, the necessary background, the research questions, inclusion and exclusion criteria, search strategy, data extraction and analysis of gathered data. The SLR protocol was arranged by one author and then revised by other authors to verify and evaluate the research questions and whether the search queries map to the review objectives and research questions. They also checked the relevance between data to be extracted and research questions.

- Several junior and senior researchers (with up to 15-30 years of experience in architecture evaluation) assessed and reviewed the SLR. They provided feedback which reduced the bias of the formalisation of the protocol, due to the selection of search keywords. There is still a risk of missing some related studies. This could occur in cases where software architecture evaluation keywords are not standardised and clearly identified. For instance, continuous evaluation is defined under different terms, such as continuous, run-time, dynamic, *etc*. Therefore, we made an agreement with each other about the definitions of unclear keywords. In some cases it was difficult to elaborate how the authors of reviewed studies interpreted terms such as continuous or run-time or dynamic (Section 2.2.3.1). In this context, we tried our best to include all the related terms that imply continuity. However, we cannot guarantee completeness.

- We also used a data extraction form to select information for answering research questions hence improving the consistency of data extraction (Section 2.2.6). To ensure that the findings and results were credible, we conducted a quality assessment on related studies (Section 2.2.5).

- The limited number of included studies might open a question about the completeness and coverage of the review, as compared to other SLRs (e.g. [135]). But the objective of this review was to focus on a specific goal, i.e. the state-of-the-art in software architecture evaluation approaches under uncertainty and to what extent continuous software architecture evaluation approaches are used. This results in a narrowed scope for the review. This is analogous to the case of [60] that conducted a review focusing on methods that handle multiple quality attributes in architecture-based self-adaptive systems (54 included studies), and [140] that studied the variability in quality attributes of service-based software systems (46 included studies). The narrow scope of SLRs explains the limited number of search results and included studies. We believe that the relevant studies to the research topic were indeed included. Further, the quality of conferences, journals, and books of the included studies ensures the significance of the analysis.

– In our search execution, some relevant studies may have not been shown in the search results of the bibliographical sources. This may be due to the fact that automated searches depend on the quality of the search engine. On the other hand, the selected bibliographical sources are considered the largest and most significant sources for conducting SLRs and the most used ones in software architecture and software engineering [56, 60]. We also performed manual and automated searches through the most popular venues for software architecture and software engineering [60]. Consequently, we are confident that the included studies are the most relevant and important ones and others are unlikely to be missed.

– We applied our search on meta-data (i.e. abstract, title, and keywords) only and some studies might have used architecture evaluation as a part of their proposed work without mentioning that explicitly in abstract, title, and keywords. Since the authors identify the meta-data of their studies, therefore, our included studies depend on the quality of the bibliographical digital sources in classifying and indexing studies.

– Since the self-adaptive and self-managed domain is large, we did our best to include studies which show architecture evaluation as part of their approach. In particular, we added studies from a list of 5244 papers (the output of the search process in Figure 2.1) through search databases and a snowballing process, in addition to some manual search. However we may have missed some works unintentionally.

– Furthermore, a common threat to validity is the fact that there are some criteria—such as dealing with uncertainty and management of trade-offs—where the paper's authors do not explicitly mention whether they are addressed or not. In this context, we attempted to infer these criteria. Similarly, a common concern in the run-time approaches is that, in most cases, "the proactiveness or reactiveness of the approaches are not explicitly discussed and it can only be inferred from the adaptation strategies" [60]. Accordingly, we made our best effort to infer the reactiveness and proactiveness of the examined approaches.

2.5 Relevant Fields

There are other techniques outside the area of software architecture and/or software engineering, which have inspired our proposal for developing a continuous architecture evaluation framework. In particular, a continuous architecture evaluation approach could benefit from control theory [141]. For example, the approach may use a feedback system, which is analogous to the feedback control system in control theory [141]. Control theory has primitives to monitor the observations of the system, detect its problems and then react. In the context of continuous evaluation, the approach can use primitives of control theory for system monitoring and change detection. However, novel ways are necessary to provide faster and more optimal reactions in case of system failure.

The requirements-driven approaches have been adopted as a way to perform qualitative analysis in handling non-functional requirements trade-offs. However, it has not been discussed from the architecture evaluation perspective. The traditional requirements-driven approaches such as object-oriented modelling [142] and structured analysis [143] are not suitable for evaluation because of the following: (i) the inability to handle the rising complexity of software systems; and (ii) the lack of dealing with non-functional requirements. Requirements-driven approaches are "concerned with the use of goals for eliciting, elaborating, structuring, specifying, analysing, negotiating, documenting, and modifying requirements" [144]. The main idea of requirements-driven approaches is starting with the high-level goal of system, and then keep refining it (i.e. decompose into sub-goals). They often do not explicitly evaluate these requirements in relation to the architecture and its design options.

2.6 Related Reviews

There are several surveys in the software architecture community, which focus on reviewing architecture evaluation methods. However, to the best of our knowledge, they did not explicitly address architecture evaluation methods from both perspectives: design-time and run-time. Some reviews analysed the use of specific quality attributes for evaluation, such as reliability and availability prediction methods for software architectures [145]. Others focused on practices that may relate to architecture evaluation, such as decision-making techniques for software architecture design [146], software architecture optimisation methods [135], and architecture design rationale [147].

In the area of design-time architecture evaluation, there are many studies, such as [12, 55, 56]. For instance, Dobrica et al. [12] focused on surveying the most popular methods, such as ATAM [2], CBAM [9], and ALMA [10]. Babar et al. [89] provided a framework for classifying design-time software architecture evaluation methods and a comparative analysis for the scenario-based approaches in specific in [148]. Roy et al. [55] extended the previous reviews and considered most of the design-time evaluation methods at that time. The authors in [56] systematically reviewed and classified architecture evaluation methods from the architecture evolution perspective.

Other surveys focused on run-time methods, such as self-adaptive systems [46, 58, 60], self-managed systems [57], and models@run-time [59]. From [57, 46, 58–60], we found that none of the studies explicitly demonstrated the use of run-time architecture evaluation principles. And none of the works have examined continuous software architecture evaluation. This is surprising because some research studies implicitly provide the elements for a continuous evaluation approach. Our survey bridges this gap by rethinking architecture evaluation and providing classifications that can do the following: (i) help architects to conduct the evaluation in continuous settings by determining the elements of a continuous evaluation approach; (ii) help in identifying common approaches for this type of evaluation; (iii) identify common concerns for systems that can benefit from this type of evaluation; (iv) point out the strengths and weaknesses of these types of approaches.

2.7 Discussion and Recommendations for Future Research

Based on the SLR, it is clear that the area of software architecture evaluation has received a lot of attention in the past decades. Nevertheless, the results demonstrate some observations which could potentially lead to future research. In particular, the SLR has identified several gaps in relation to architecture evaluation under uncertainty with respect to decisions which relate to designing dynamic and complex systems, such as IoT, cloud, and volunteer computing. In this context, this section aims to address the third question: RQ1.3: What are the current trends and future directions in software architecture evaluation under uncertainty and their consideration for continuous evaluation? *This question aims to show how we can benefit from the existing approaches to draw inspiration on the essential requirements and address the pitfalls when developing a continuous evaluation approach.* In Section 2.7.1, we present the architecture evaluation research area maturation stages and classification. We then highlight the important objectives that should be accomplished by the research community to advance this research area (Section 2.7.2 and 2.7.3).

2.7.1 Research Area Maturation

In this systematic review, we aim to investigate the extent to which architecture evaluation under uncertainty and the consideration for continuous evaluation have matured as a discipline. For this purpose, we examine the included studies with respect to the Redwine-Riddle model [149]. The latter provides six stages for technology (research area) maturity. These stages are [149]:

1. *Basic Research:* investigating the ideas and concepts; and providing a clear articulation of problem's scope.
2. *Concept Formulation:* presenting a comprehensive evaluation of solution approach through seminal paper or a demonstration system.
3. *Development and Extension:* preliminary using the ideas and extending the general approach to a broader solution.
4. *Internal Enhancement and Exploration:* extending the general approach to solve real problems in other research areas.
5. *External Enhancement and Exploration:* creating a broader group and involving them in decision-making to provide a substantial evidence of value and applicability.
6. *Popularisation:* showing production-quality, providing supported versions, as well as marketing and commercialising the technology.

Initially, one author has classified the 40 included studies against Redwine-Riddle model, and the outcome was revised independently by other authors. Discussions and agreements were carried out in cases of discrepancies between the authors' categorisations. Figure 2.5 shows the results of classification. It is clear that almost 80% of the studies are still in early maturity stages (Basic Research and Concept Formulation), whereas almost 20% have been extended to broader problem domains and applied in practice. Among those approaches that are already adopted by industry, none of them are deployed at run-time; they only focus on design-time evaluation. In particular, maturity has only been proven for design-time approaches, such as ATAM and CBAM. This explains why 5% (2) of approaches are still in the popularisation stage. Table A.1 depicts the studies with respect to area maturity level.

We have seen some examples of continuous evaluation that are either implicit, partial, or explicit, such as CPASA and DevOps. However, these research efforts have not demonstrated and documented how to adapt those practices in the evaluation of software architectures

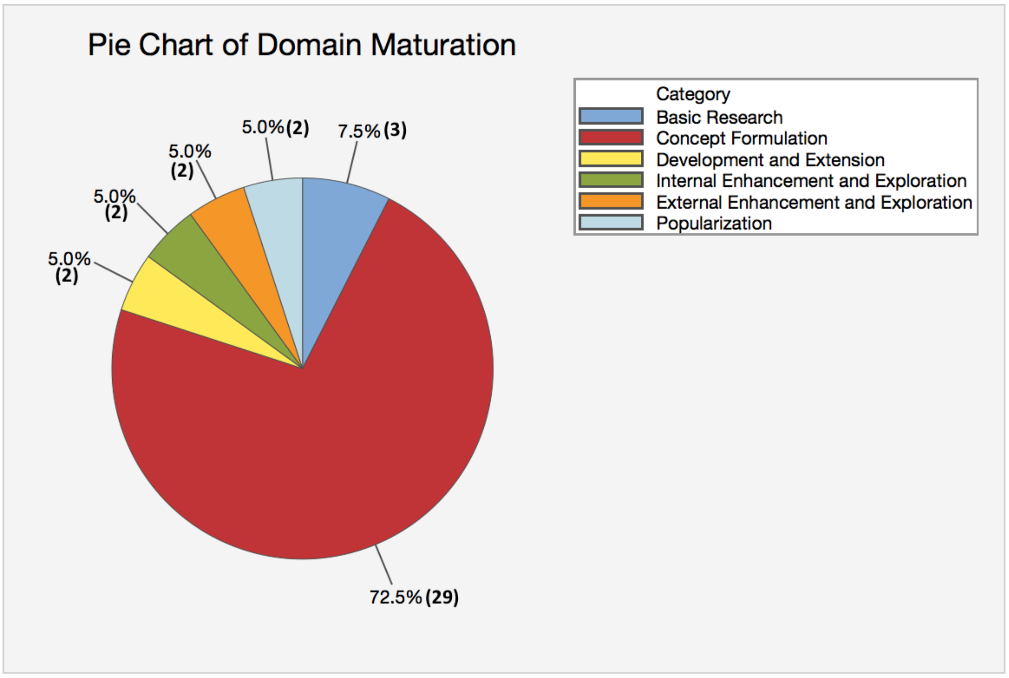


Figure 2.5 Distribution of the included studies over the research area maturity classification model (The number of studies for each maturation stage are placed between parenthesis and maturity distribution are shown in percentage).

under uncertainty. Therefore, to mature the architecture evaluation research area with continuous evaluation approaches, we need a set of guidelines, tools, systematic procedures, acceptance from (and case studies with) real-world organisations, and shared benchmarks across companies for best practices. In this thesis, the systematic evaluation practice has been chosen as part of avenue to fill in this gap. The presence of systematic methods could support the evaluation, where integrating these methods with development life-cycle can help to accelerate the maturity of that practice and transient from ad hoc attempt to discipline attempt.

2.7.2 Leveraging Existing Approaches To Develop A Continuous Evaluation Framework

Having done this SLR, we observe that elements from different approaches could be combined to develop a continuous software architecture evaluation framework. For instance, the CBAM [9] is a scenario-based design-time evaluation method, which determines the influence of architecture decisions on the cost-benefit trade-offs. The CBAM provides an implicit mitigation for uncertainty through different types of scenarios. However, this type of evaluation approach would not be suitable for the emerging technologies and paradigms, such as IoT and cloud-based systems. Further, diversification [150] is an important design principle that is often used to architect dynamic and complex systems, such as IoT and cloud [151, 152]. But the majority of evaluation techniques do not provide the means to either value or evaluate for diversification. In this context, a systematic design-time evaluation approach that can deal with complex architecture design decisions (such as diversity) and handle uncertainties is necessary. This is an important foundation of a continuous evaluation framework.

Based on the existing approaches, we infer that there is a lack of well-documented, real-world examples for economics-driven approaches in the context of design-time evaluations (Table 2.6 and 2.11). In particular, these approaches ([36–42, 117]) have not been used to deal with cost-benefit trade-offs in dynamic environments, such as IoT. Further, they have not been explored from the perspective of forecasting the long-term value of architecture decisions to determine whether the complex design decisions, such as diversity in design [150], can handle uncertainties that can be attributed to dynamic changes in the environment.

We believe that economics-driven approaches, such as real options analysis [25], could be combined with CBAM to support the analysis. Real Options Analysis is one of the few design-time techniques that can embed flexibility under uncertainty. Therefore, it can aid the architect in predicting the impact of architecture decisions on quality attributes of interest. It can also shortlist the candidate options for deployment at run-time and thus reduce unnecessary costs. This is still very much a research area that requires further investigation in the context of design-time evaluation, as an initial stage for continuous evaluation.

We found that most of the architecture evaluation approaches focus on design-time (about 60% of the approaches) and less on run-time (about 40% of the approaches). Evaluation approaches also tend to focus on development (i.e. mostly human-centric activities) and lack a consolidated approach that integrates design-time and run-time considerations. On the contrary, in the context of architecting and evaluating dynamic and complex systems, a more continuous approach that starts at the early stages of development and continues to evaluate the architecture options during the lifetime of the system at run-time is necessary to cope with operational uncertainties, such as high fluctuations in QoS, sensor ageing effects, *etc*.

2.7.3 Finding The Necessary Ingredients For Developing A Continuous Evaluation Framework

Modern software system environments, such as IoT, cloud, volunteer computing, and microservices, are a challenge for existing software architecture evaluation methods. Such systems are largely data-driven, characterised by their dynamism, unpredictability in operation, hyper-connectivity, and scalability. Properties, such as performance, delayed delivery, and scalability, are acknowledged to pose great risk and are difficult to evaluate at design-time only. Therefore, a run-time evaluation approach is necessary to complement design-time analysis. This run-time stage should be able to handle different sources of uncertainty and evaluate complex design-time decisions. In this regard, we need to determine the necessary ingredients for this run-time stage.

One interesting observation is that just 25% of run-time approaches address cost as a concern (Table 2.8 and 2.11). Since the management of cost-benefit trade-offs is essential in dynamic environments [8], cost will highly influence the value of architecture decisions.

Based on the results of our review (Table 2.8), most of the run-time approaches handle uncertainty either by checking goal violations or providing some probabilistic estimations. However, in contexts of highly dynamic environments such as [34, 7, 8, 153, 154], this is not sufficient. Even if the currently deployed architecture decision is not violating any goal, this does not mean that it has good performance. For example, in some cases, an architecture decision is meeting its quality constraints but it is providing poor performance. In this context, a change detection test is a necessary component in a continuous evaluation framework to determine significant drifts in the architecture decisions. This type of test can provide the architect with the flexibility of adjusting the sensitivity to changes. Therefore, determining the type of test and its efficiency could be a potential future direction.

Moreover, most of the existing run-time methods rely on historical data or online data to perform the evaluation, but they do not consider the age of data. Therefore, embedding some ageing parameters to emphasise the relative importance of older versus more recent data could potentially improve the analysis [138]. Further investigations, related to the use of these parameters and how the architect could tune these parameters to enhance the evaluation are required (examined in Chapter 5).

From the run-time perspective (Table 2.8 and 2.11), it is clear that most of the current approaches (e.g. [138, 155, 122], *etc*) tend to be reactive when simplistic learning, partial or incomplete knowledge is used. Thus they may suggest incorrect decisions due to unexpected future environment changes and recommend unnecessary switches due to the lack of future knowledge about the candidate architecture decisions. This in turn may affect the architecture's stability and overall behaviour. To bridge the gap, further proactive approaches are necessary to improve the continuous evaluation process (addressed in Chapter 6).

In addition, our analysis shows that just 25% of the run-time approaches embed machine learning principles in the decision-making process. Using machine learning approaches in decision-making has shown great improvements to the decision-making (e.g. [156]). Therefore, another important element when developing a continuous architecture evaluation framework is leveraging machine learning techniques. In this thesis, we plan to investigate the use of machine learning techniques for architecture evaluation in Chapter 5 and 6.

The literature depicts that there are some approaches (e.g. [21, 123, 121, 139]) that are proactive in terms of failure prediction and recommending alternatives. These approaches may, however, experience scalability problems. Moreover, these approaches assume that the impact of architecture decisions on QoS is available at run-time, which is not always the case for uncertain environments such as IoT. To this end, novel solutions are required to determine how QoS monitoring challenges could be handled (discussed in Chapter 5).

There are methods (e.g. [21, 18]) that explicitly mention continuous architecting and assessment, and others that implicitly adopt it (e.g. [19]). These approaches can benefit from further investigations in terms of how continuous evaluation could dynamically track and forecast architecture decisions and automatically manage cost-benefit trade-offs.

2.8 Conclusion

Continuous evaluation has been discussed under different labels, such as run-time, dynamic, continuous, *etc*, along with assessment and analysis. The common characteristic among these efforts is that they start at design-time (even if they do not mention that explicitly) and continue to evaluate architecture decisions during the life-time of the system by observing environmental conditions. In this review we have attempted to unify these efforts. We performed a systematic literature review to examine existing architecture evaluation methods that deal with uncertainty either design-time or run-time. We also provided guidelines for the necessary elements to develop and conduct a continuous architecture evaluation approach. We both automatically and manually searched well-known venues for software architecture and engineering, other related systematic reviews and mapping studies, and significant bibliographical data sources. In addition we applied a snowballing process to collect our primary studies.

The results of our investigation are the following: (a) design-time architecture evaluation approaches garnered more attention than run-time ones, though the latter are increasingly important to handle the dynamism and increasing complexity in software systems; (b) there is a lack of examples on demonstrating how continuous evaluation approaches can realised and conducted; (c) few methods focus on managing trade-offs between benefits and costs at run-time; (d) few methods focus on adopting machine learning techniques to the evaluation; (e) most of the run-time approaches tend to be reactive (and may recommend unnecessary switches and hence increase deployment costs).

In summary, our main findings, listed in Section 2.7 and Table 2.6-2.11, call for a continuous software architecture evaluation method that can complement some of the included studies and aid the architect in assessing the architecture design decisions under uncertainty through the following: (i) analyse the use of continuous architecture evaluation in dynamic environments, such as IoT and cloud systems (discussed in Chapter 3); (ii) employ economics-driven approaches (i.e. forecasting the long-term value of complex architecture decisions), this is discussed in Chapter 4; (iii) adopt economics-driven principles in the design-time evaluation approach (the initial stage of a continuous evaluation approach) because it embeds flexibility under uncertainty (introduced in Chapter 4); (iv) perform additional research in analysing the use of machine learning techniques to improve architecture evaluation at run-time (the ongoing stage in a continuous evaluation approach), this is explored in Chapter 5 and 6; (v) explore how tuning the input parameters for the continuous evaluation (e.g. sensitivity to changes, monitoring intervals, the relative importance of present/past data) could affect the evaluation and what are the most suitable parameters to improve the decision-making (addressed in Chapter 5); (vi) investigate the development of proactivity in the architecture evaluation process (tackled in Chapter 6).

Chapter 3

IoT as a Motivating Case Study

Context. Dynamic and complex systems, such as IoT, are fundamentally different from classical ones for which most design-time architecture evaluation methods, such as CBAM and ATAM, were advanced. These systems are characterised by their heterogeneity, high dynamicity, and scale. Therefore, the dynamic nature of IoT requires us to combine design-time evaluation with run-time when evaluating architecture design decisions. In this context, continuous architecture evaluation frameworks, which integrate design-time and run-time evaluation, could be a potential aid for the architects to improve the decision-making, in dynamic environments such as IoT.
Objective. In this chapter, we aim to motivate the need for continuous evaluation and show how it goes beyond existing architecture evaluation approaches.
Method. We have used and extended an existing IoT (i.e. video surveillance) application from the literature, which was not addressed from the context of run-time architecture evaluation and adaptability under time-varying environment. Through the case study, we have shown the challenges facing the team of architects and how the proposed framework can aid in improving the architecture evaluation practices.
Conclusion. The challenges facing IoT applications require a continuous evaluation framework that can help the architect in evaluating and forecasting cost-benefit trade-offs of complex architecture decisions in IoT domain.

3.1 IoT challenges

Upon evaluating the IoT application, the architect is faced with several challenges that are rooted in the characteristics of the domain and the environmental conditions facing that domain. These challenges call for revisiting evaluation practices for software architectures. Let us focus the discussion on the challenges below ([8, 6, 11, 7]):

- *Heterogeneity*, having different types of things, such as static sensing (e.g. fixed sensors), mobile crowd-sensing (e.g. cellular-based, vehicle-based, and bike-based sensors), virtual (e.g. web services), and social sensing (e.g. share data across social networks like facebook);

- *High dynamism*, due to the presence of fixed and mobile things, and uncertainty of their resource demand and QoS provisions over time (i.e. service level objectives), varying energy consumption per device and their varied availability;

- *Scale*, where their ubiquitous, light, and mobile nature has led to the presence of, in some cases, millions of things.

Classically, design-time analysis is a commonplace for architecting software systems. However, the new era of applications, such as IoT [157], requires us to rethink the way we analyse and evaluate architectures of software systems [158, 8, 159]. Current architecture evaluation approaches, such as ATAM [2], CBAM [9], and ALMA [10], are not well designed to handle these emerging challenges. The dynamism, heterogeneity and scale of IoT naturally moves architecting practices towards architecting for uncertainty, where architecture design decisions are more complex to evaluate at design-time only. This is because it is difficult to anticipate their fitness and the extent to which they can cope with operational uncertainties and continuous changes in the structure, topology, and requirements of possible composition patterns. It leads to a need for a more continuous approach that intertwines design-time and run-time evaluation of architectures.

3.2 Introducing the IoT case: *iTransport*

In this thesis, we draw on an IoT application, documented in [35], to motivate the need of a framework that is applicable to domains like IoT. In particular, we are looking at domains that can benefit from the continuous evaluation framework, such as IoT. The basic concept of IoT is the interaction between a group of devices— "things"—such as sensors and actuators, over the Internet. Our IoT application extends Gupta et al. [35]'s application – an urban traffic monitoring system, named *iTransport*. Gupta et al. [35] used a video surveillance application to demonstrate the usefulness of their proposed cloud/fog simulator tool *iFogSim*. However, in their case study, the context of architecture evaluation and adaptability under time-varying environment have not been considered, which are addressed in our work. In a nutshell, iFogSim is a cloud/fog simulation environment; it can aid developers to simulate the impact of their application on qualities of interest. It forms the basis in our work to mimic the dynamics and uncertainty of cloud/fog environments, and their impact on qualities of interest in our case study. Further explanation related to our use of iFogSim, which differs from that of Gupta et al., can be found in Section 5.5. In addition, Gupta et al. [35] assume the presence of one application architecture (i.e. configuration). We have designed the multiple configurations with respect to the common architecture decisions of a video surveillance application.

The *iTransport* application provides online (for emergencies) and offline (for long-term prediction) analytics. It uses smart cameras, which are either fixed (attached to street lights and buildings) or mobile (attached to vehicles and bikes) to capture the traffic for accident avoidance and traffic management. For instance, the data generated from vehicles could be used for the following: monitor traffic patterns, inform drivers of traffic situations, and help governments determine appropriate solutions to traffic problems [160]. The *iTransport* application has 6 modules as seen in Figure 3.1: camera, motion detector, object detector, object tracker, accident storage, and emergency control. Smart cameras transmit raw video streams to the motion detector module, which then forwards the video in which motion was detected to the object detector module. The object detector module analyses the objects and detects any abnormal actions (i.e. car accidents). If it observes an accident, the emergency control searches for a nearby ambulance for notification. The data is then sent to the accident storage database to profile the accidents with respect to areas. This database could be stored in either fog or cloud, which will be evaluated in this thesis. The application automatically provides either "online analytic" functionality or "offline analytic" functionality, every 10 minutes based on user requirements. For instance, if "online analytic" functionality is invoked, then minimised response time and network usage are necessary, whereas if "offline analytic" is called, then the energy consumption is the main goal for optimisation.

When architecting *iTransport* application, the architects must address uncertainties due to heterogeneity of the things; the dynamicity of the things' behaviors and the dynamism of their composition. For instance, having different types of smart cameras (e.g. fixed and mobile) could potentially lead to varying QoS, due to their varied mobility, heterogeneity in the topology, memory, computational efficiency, *etc*. Further, the hyperconnectivity and increasing number of smart cameras have a significant impact on the overall behavior of *iTransport*. This could cause unpredictable and dynamic performance levels.

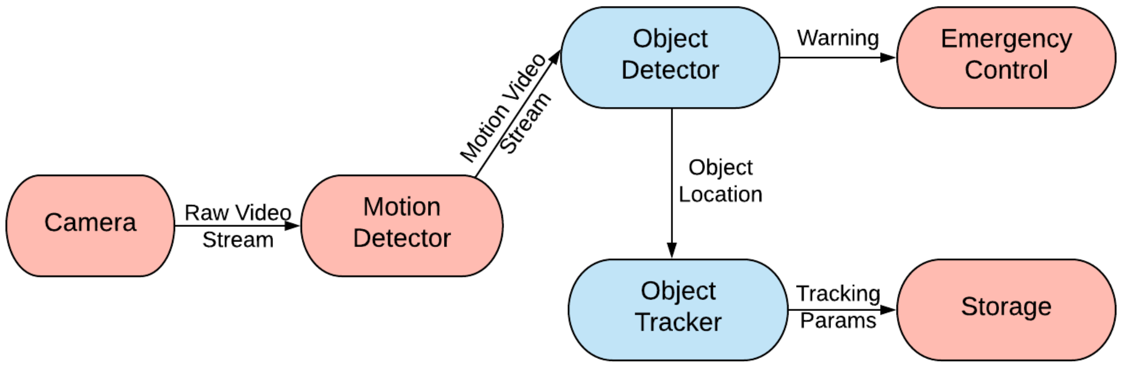


Figure 3.1 The flow diagram of *iTransport* application [1].

3.3 A hypothetical scenario

In the past decades, a stream of academic literature has touted the benefits of software diversity in design [150, 161, 162]. J. Kelly was among the first to speak about design diversity; defined as "the approach in which the hardware and software elements that are to be used for multiple computations are not copies, but are independently designed to meet a system's requirements" [163]. Diversity here refers to the generation of functionally equivalent versions of a software system, but implemented differently [164]. Historically, diversity is used to handle the uncertainties [165]. These are translated into quality-related objectives; improving dependability [33, 166, 167], reliability [168, 169], security and privacy [170, 171], robustness [172, 152], and adaptability [151].

Design diversity is a potential solution to embrace uncertainty through spicing the architecture by variety to improve the QoS of the architecture and its design decisions when deployed in uncertain and unpredictable environments, such as IoT [152, 151]. The intuition is that the architecture of software systems can maximise its benefits from varieties through the availability of multiple implementations of a specification in an attempt to better mitigate operational uncertainties in open and dynamic environments, such as IoT. In this context, design diversity poses itself as a key solution for architecting under uncertainty. Design diversification has the potential to mitigate risks in situations exhibiting uncertainty in operation, usage, *etc*. Therefore, we decided to perform a case study where we use our continuous evaluation framework to evaluate diverse software architectures.

Let us consider the case where the architects have employed design diversification strategies [15, 167, 16] to respond to the challenges and to handle uncertainties: the greater the uncertainty, the more architects have attempted to apply design diversification with the objective of improving the QoS in terms of performance and energy consumption. We contend that diversification means embedding in flexibility in handling operational changes. Since there is a variety of ways to diversify, each diversified architecture can be treated as a candidate option, which we denote by *dao* (i.e. diversified architecture option). The set of diversified architecture options are denoted by *DAO*. A *dao* implements a set of diversified decisions to meet some quality goals and trade-offs. Given a set of architecture decisions D, where a decision $d_{ka} \in D$. d_k denotes a particular capability, including connectivity, data collection, data management, *etc.* d_{ka} indicates the software architecture components and connections that implement this capability. For example, the architects decided to diversify the *data collection* capability (d_1), where video could be captured using fixed cameras (d_{11}), mobile cameras (d_{12}), or both (d_{13}). Another diversification decision is concerned with the *connectivity and processing* capability (d_2), where the things can connect, track, and process the captured video on the cloud (d_{21}) or both cloud and fog (d_{22}). So dao_1 comprises d_{11} and d_{21}, whereas dao_2 consists of d_{12} and d_{22} and so forth. Further, fixed d_{11} or mobile camera decisions d_{12} could have different software and hardware, which then affect their QoS over time. Processing the data from cloud d_{21} or fog d_{23} denote the use of diverse cloud/fog providers, which in turn provide uncertain QoS over time. In this context, dao_1, dao_2, dao_7, and dao_8 embed diversity through the use of diverse fixed/mobile devices and different cloud providers/heterogeneous fog devices. Table 3.1 depicts the selected options, with decisions designed for cloud, fog, mobile, or fixed.

Table 3.1 Possible Diversified Architecture Options for *iTransport* application. The diversity in each *dao* can refer to using fixed and/or mobile fog devices for data collection capability; using different cloud providers and heterogeneous fog devices for data processing capability.

Decision d_k / Option dao_i	Data Collection Capability (d_1)	Data Processing Capability (d_2)
1	Fixed (d_{11})	Cloud (d_{21})
2	Mobile (d_{12})	Cloud (d_{21})
3	Fixed and Mobile (d_{13})	Cloud (d_{21})
4	Fixed (d_{11})	Fog and Cloud (d_{22})
5	Mobile (d_{12})	Fog and Cloud (d_{22})
6	Fixed and Mobile (d_{13})	Fog and Cloud (d_{22})
7	Fixed (d_{11})	Fog (d_{23})
8	Mobile (d_{12})	Fog (d_{23})
9	Fixed and Mobile (d_{13})	Fog (d_{23})

3.4 iTransport in the context of continuous architecture evaluation

In *iTransport*, there are several design trade-offs concerning the critical QoS attributes (e.g., response time, energy consumption, network usage, *etc*) and cost, subject to constraints such as the predefined coverage and availability of the things. In the context of *iTransport*, we consider the operating cost to include the following: deployment cost (the expenses related to the infrastructure deployment in cloud/fog environment), execution cost (the computational costs of running the processing tasks on cloud/fog devices), and networking costs (related to the bandwidth requirements and associated expenses). For instance, data uploading cost from end devices/sensors and inter-nodal data sharing costs) [173]. Further, the switching costs in *iTransport* embrace the migration costs to/from the cloud/fog, thing's connectivity and other costs (if any). Nevertheless, the architect can include other costs. So the question here is "Will IoT architectures forward all streams of data to a centralised cloud, or will the scale and timeliness requirements of IoT applications require distributing storage and computing to the fog's devices?" [174]. The design trade-offs can inform the diversification design decisions and the deployment of *dao*.

Let us imagine a group of architects *Team Alice*, whom are assigned to design and evaluate the architecture of *iTransport*. With respect to design trade-offs introduced earlier, Team Alice have found that an ad hoc fashion would not serve the purpose. In particular, it appears that the previously assigned architects have chosen random *dao* based on their perception, without performing any systematic evaluation. On the contrary, Team Alice argue that the deployment of all candidate *DAO* or the ad hoc selection would lead to unnecessary costs. Therefore, Team Alice may:

1. Perform a systematic design-time evaluation as an initial step to shortlist the candidate options, as well as, rank the options under investigation (Section 3.4.1).
2. Complement the design-time decisions using an informed run-time architecture evaluation approach that can monitor and learn the cost-benefit trade-offs of diversified architecture options over time (Section 3.4.2).
3. Use forecasting analytics capabilities to predict the cost-benefit trade-offs of *DAO* and hence improve the decision-making process (Section 3.4.3).

3.4.1 Perform a systematic design-time evaluation

Team Alice observed that the common practice has been to select *dao* based on the context, for example, (i) Deploying dao_1 in cases where the energy consumption is the major concern to be optimised. Because data processing in the cloud may (not) incur lower energy consumption than in fog devices [175]; (ii) Selecting dao_4 to be continuously deployed when response time and network usage are the stakeholders' concerns, due to the presence of fog and cloud for data processing, which in turn may reduce the network congestion.
However, Team Alice are still uncertain about their decisions, due to the following *DAO* trade-offs:

- dao_1 uses fixed camera sensors to provide more stable and better response time to fulfil the pre-defined coverage. However, achieving the coverage for the scale of city highways using fixed cameras may incur much higher cost and static coverage. In contrast, the use of mobile crowdsensing (using smart vehicles) [11], as in dao_2, could be an alternative solution due to its low cost. But the mobile crowdsensing in dao_2 may be unstable in terms of response time and it can consume much more power at this scale due to the simultaneous transmission, processing, and remote execution of the images on the cloud. Further, availability in dao_2 is much more restricted than that of dao_1.

- When comparing with the case where cloud is used as the sole computation paradigm (in dao_1 to dao_3), the partial use of fog (in dao_4 to dao_6) could provide faster response time (e.g., for scenarios such as emergency notification and online analytics) and lower network usage, due to the offloading of computational load on the near by fog devices. But it may incur more energy consumption, given the large number of required fog things and the additional overhead that may be required to synchronise and store the processed information on the cloud (if any). In addition, the fog option needs to fulfil the constraints on the proximity of the thing to the fog and the availability of the fog. Further, the sole use of fog (in dao_7 to dao_9) could provide the best response time (i.e. fastest) as compared with the use of cloud (i.e. in dao_1 to dao_3) or partial use of fog (i.e. in dao_4 to dao_6). However, relying on fog only is strongly dependent on the devices, which are affected by their efficiency, mobility, and heterogeneity. They may also incur high energy consumption, similar to in dao_4 to dao_6. On the contrary, the presence of cloud (in dao_4 to dao_6) could be a backup in case fog devices were not working as expected.

From the architect's perspective, relying on an ad hoc fashion to evaluate the architecture options may cause undesirable results, because *iTransport* is a time-critical application and ignoring the uncertainties is not desirable. From the operator's perspective, it is even too expensive to implement all the *DAO* and then select between them at run-time. This is due to the need for licensing costs, maintenance costs, switching costs and so forth. Consequently, this calls for a systematic design-time evaluation that forecasts the cost-benefit trade-offs of each *dao*. Further, an approach that can value flexibility under uncertainty is necessary to demonstrate how the architecture option can "survive" under varying environmental changing conditions.

By taking into consideration the economic gains/losses, the architect may deploy a *dao* that seemed to initially deliver value, but then it turned out to be inefficient. This is due to the high reliance of expertise knowledge and neglecting the uncertainties in environments, such as IoT. This in turn may cause unnecessary losses and the need to switch to another *dao*. The design-time evaluation should consider the uncertainty and dynamicity in IoT when selecting the architecture options for deployment. In particular, it would aid the architect in quantifying the long-term value of diversified architecture options. This leads to our second research question: **RQ2:** *To what extent design-time architecture evaluation methods that adopt economics-driven principles, such as real options analysis, can systematically evaluate complex design decisions, such as diversification, in the face of operational uncertainties in dynamic environments?* For that, we adopt an economics-driven approach *binomial real options analysis* [176] that extends one of the most well known architecture evaluation techniques CBAM [9]. The problem with the state-of-the-art evaluation techniques, such as CBAM, is that it can shortlist the candidate architecture options without providing the long-term value of these options. More specifically, the CBAM evaluates the architecture options by performing classical cost-benefit analysis. On the contrary, the proposed design-time evaluation approach can predict and value the architecture's flexibility in handling uncertainties. The proposed design-time evaluation approach is introduced in Chapter 4.

Summary: A systematic economics-driven design-time evaluation is a necessary step to initially decide on the candidate diversified architecture options by determining their long-term value. This shortlisting would limit the number of architecture options for implementation and hence reduce costs.

3.4.2 Complement the design-time decisions using a run-time architecture evaluation approach

Team Alice have found that there are some scenarios where design-time decisions may fail to select the "suitable" options due to the run-time uncertainties and dynamics caused by various environmental factors, which emergently affects the benefit of the options. In particular, it is possible that, at run-time, their benefit deviates more than the expected value at design-time, for example:

- The design-time evaluation suggests dao_6 to be continuously deployed when response time and network usage are the stakeholders' concerns, due to its high expected benefit. Conversely, at run-time (i.e. output of simulator), the hyper-connectivity of mobile things and the high network latency affect its actual benefit in terms of response time and network usage, which was significantly lower than expected.

- Team Alice have decided to implement dao_1 in cases where response time and network usage is not a concern, aiming to improve the energy consumption in fog devices due (i.e rely on cloud providers rather than fog devices for data processing). However, the actual overall benefit (e.g. response time, network usage and energy consumption) was much worse than expected during run-time. This is because the sensors are still performing some processing to transmit the data to the cloud (i.e. there is high power load on the fixed sensors).

- Team Alice have prioritised the response time concern over the network usage and energy consumption. They selected dao_2 for deployment due to the use of mobile things, which may have lower impact on network congestion as compared with fixed ones. On the contrary, Team Alice have discovered that the actual benefit of dao_2 was much worse than expected, due to the presence of a very large number of mobile things (i.e equivalent to the deployment of lower fixed number of sensors), which resulted in high network usage. Therefore, dao_2 was almost violating the quality constraints in most of the cases.

The prior scenarios motivate the need for the run-time evaluation to capture conditions that may not be discovered at design-time. The approach can accumulate run-time information (i) discovering patterns related to the availability of mobile nodes and their connection to benefit improvement/degradation and value; (ii) the added value of switching to a diversified option and when that value will be realized, become optimal or cease to exist, considering various costs and load. The dynamism of the above cases makes it difficult for architects to solely evaluate the architecture based on design-time knowledge.

This leads to our third research question: **RQ3:** *How can run-time architecture evaluation using cost-benefit analysis and machine learning techniques complement the design-time one to handle uncertainty?* The run-time evaluation could be employed to *complement* the design-time decisions. Run-time knowledge can be particularly useful to suggest refinements for the diversified architecture options; informing when a *dao* should (not) be invoked; phasing-out a *dao* or suggesting a replacement, *etc*.

However, there are some scenarios, which Team Alice need to consider while evaluating the architecture options at run-time. For example:

- When evaluating the architecture options over time, the currently available change detection methods (e.g. [177, 125]) detect only QoS constraint violation. In particular, they lack the ability of checking whether the current deployed architecture option is getting worse.

- Due to the dynamcity and hyperconnectivity of IoT devices, in some contexts, the IoT's QoS data, such as response time, energy consumption, *etc*, are not always available.

- The fluctuation in QoS can be attributed to environment and network changes, such as noise in the network.

- The selection of *dao* is time-consuming, which may be a problem in case of IoT. For instance, the need to comply to standardisation [178] may have implications on performance and scalability of the system; this can be best understood and evaluated at run-time.

In this context, we contribute to a run-time architecture evaluation approach that gets inspiration from reinforcement learning [27] and multi-objective optimisation [28, 29] literature (introduced in Chapter 5). The proposed run-time architecture evaluation approach would improve the decision-making process by providing the following properties: (i) evaluate the set of diversified architecture options and ensure the selection of the *dao* with most balanced cost-benefit trade-off over time; (ii) the flexibility for the architects to adjust the relative importance of information (i.e. QoS); (iii) the approach's robustness to noise imposed on the data; (iv) the malleability of tuning the monitoring intervals, which could potentially save costs and handle cases where the data is not always available; (v) the ability to search for the optimal *dao* in a short period of time.

Summary: The proposed run-time architecture evaluation approach *complements* design-time decisions. In particular, it continuously profiles architecture decisions for their added value, identifies significant deviations from previously expected benefits, and automatically determines the architecture options with the most balanced cost-benefit trade-offs.

3.4.3 Use forecasting analytics

We name the run-time evaluation introduced in Section 3.4.2 as *reactive*, because it lacks the use of forecasts about the future, which is its main limitation. Team Alice have figured out that the reactive run-time evaluation approach suffers from some limitations. Though the reactive approach continuously helps the architect in understanding the past behaviour of the architecture decisions in question, such a reactive approach is limited. Even though it can monitor the past observed performance of architecture decisions, it cannot proactively reason about their future potentials. However, a decision that has worked well in the past may not necessarily work well in the future, given the potential changes and uncertainty underlying the environment where the system is embedded. Reactive approaches may ignore the future potentials of architecture decisions. For instance, the reactive approaches may cause the current decision to seem poor, even though it could become attractive in the near future. This could trigger unnecessary architecture adaptations that would cause the system to be unstable. Conversely, a decision that may not seem attractive based on the past could have better future potentials. Discarding that decision could lead to poor future performance. In summary, reactive approaches can lead to partially justified decisions, unnecessary adaptation, and increase development cost, while slowing down the operations. Therefore, there is a great need for an approach that has the ability to forecast the future potentials of diversified architecture options.

This leads to our fourth research question: **RQ4:** *How can the use of forecasting analytics improve the state of run-time architecture evaluation?* To answer this research question, we adopt time series forecasting algorithms to predict the future benefits of diversified architecture options in question. More specifically, our approach has the ability to: (i) forecast for further ahead timesteps how the *DAO* will react; (ii) select the *DAO* based on their future potentials. This in turn will improve the decision-making process and aid the architect in handling unforeseen scenarios. Chapter 6 presents the proactive architecture evaluation approach.

Back to the IoT context, so the question here is *do proactive approaches using forecasting analytics provide useful information for evaluation, which reactive approaches may miss?* By focusing on economic gains/losses, without forecasting which architecture decisions have the potential to materialise into future economic gains, IoT systems cannot keep up with the rate at which these potentials appear/disappear in dynamic IoT environments. In particular, by the time the reactive approach suggests an architecture decision in response to a detected problem (i.e. the current architecture is getting worse), the problem itself may cease to exist (i.e. the current architecture is improving). Therefore, a reactive approach can lead to missing opportunities for future economic gains. Similarly, without forecasting whether architecture decisions will continue to deliver value for a long time or not, unnecessary costs can materialise into economic losses for the IoT system when a decision is implemented. The opportunity for future economic gains which should have been enabled by this decision (i.e. recommended by the reactive approach) can disappear. In that situation, implementing such a decision is useless and incurs nothing but an economic loss. We aim to illustrate how proactive approaches using forecasting analytics can handle the prior scenarios, in contrast to design-time and reactive approaches (addressed in Chapter 6).

Summary: The use of forecasting analytics completes the continuous evaluation framework, as it has the ability to forecast and value the architecture options for further ahead timesteps. Henceforth, it would lower unnecessary costs (i.e. unnecessary adaptations) and improve the system's stability.

3.5 Conclusion

In this chapter, we sketched an IoT case study to motivate the need for the continuous architecture evaluation framework. Nevertheless, our framework has the potential to be applied to other systems exhibiting high dynamism and uncertainty in their operations, such as volunteer computing, microservices, *etc*. Through the case study, we have shown the challenges facing the team of architects and how the proposed framework can aid in improving the architecture evaluation practices.

The use of ad hoc fashion to evaluate the architecture options may cause undesirable results in time-critical applications, such as *iTransport*. This is because IoT applications are generally challenged by the following characteristics: heterogeneity, dynamism, and scale. This calls for a systematic economics-driven approach that can value flexibility under uncertainty by quantifying the long-term benefit of architecture options (introduced in the next chapter). In particular, this approach can aid the architect in selecting the suitable diversified architecture options for the varying environmental conditions. This approach could also shortlist the candidate options for run-time architecture evaluation.

Chapter 4

Economics-driven Design-time evaluation

Context. The first step for conducting a continuous architecture evaluation is performing design-time evaluation. It is expensive for the architect to deploy all the possible architecture decisions. In this context, a design-time approach is necessary to shortlist a set of architecture decisions to be deployed. In particular, the continuous evaluation framework requires a design-time evaluation approach that can evaluate complex design decisions operating in dynamic environments, such as IoT, and suggest a set of architecture decisions believed to be suitable for the application. There are many economics-driven design-time architecture evaluation approaches (e.g. [36–42, 117]). However, none of them have been used before to (1) evaluate diversification in design; (2) deal with cost-benefit trade-offs in dynamic environments, such as IoT.
Objective. In this chapter, we aim to demonstrate how an economics-driven approach could be used to assess and shortlist the candidate architecture decisions under uncertainty.
Method. We propose a novel extension of one of the widely used architecture trade-offs analysis methods (CBAM [9]) to architect for diversification and reason about its design trade-offs, costs, and benefits. The fundamental premise is that diversification embeds flexibility in an architecture [151], which can be useful for dynamic and uncertain environments. Therefore, it is beneficial to reason about this flexibility as part of a continuous architecture evaluation approach. The proposed economics-driven design-time evaluation method shows how CBAM along with binomial real options analysis could be used to evaluate diversification as a design decision in dynamic and uncertain environments, such as IoT. Real options theory provides an analysis paradigm that emphasises the value-generating power of flexibility under uncertainty.

Results. We can infer, from the results, that the architects' confidence level is important for evaluating the design-time decisions. For example, when the confidence level was high, the architects were able to estimate the expected impact of architecture options on the QoS. However, in other cases (i.e. low confidence level), the option values of architecture options were very close and hence it was hard for the architect to decide which one to deploy at run-time. Further, the results show the approach's ability to shortlist the candidate architecture options for run-time evaluation and deployment and hence avoid unnecessary costs.
Conclusion. The proposed method can potentially be used as a design-time approach within a continuous architecture evaluation framework.

4.1 Introduction

One of the commonly used practices for engineering software systems to deal with uncertainty is *design diversity* [150, 161, 162]; "the approach in which the hardware and software elements that are to be used for multiple computations are not copies, but are independently designed to meet a system's requirements" [163]. Let us consider the case where the architect has decided to embed diversity in the architecture to improve the QoS (e.g. performance) in a dynamic environment, such as IoT. The assumption here is that a diversified architecture has better potentials in meeting the requirements through diversifying the architecture design decisions within the architecture. In particular, it enables better support for dependable provision of quality and functionality within the system. This requires rethinking architecture design decisions by looking at their link to long-term value creation to determine whether the design decisions can handle uncertainties that can be attributed to dynamic changes in an uncertain environment, such as IoT. Here, we aim to demonstrate how options thinking [176] can be employed to evaluate diversity in design. An option is the right, but not the obligation to invest or take an action in the future of uncertain value.

This chapter aims to answer the second research question introduced in Chapter 1:
RQ2: To what extent design-time architecture evaluation methods that adopt economics-driven principles, such as real options analysis, can systematically evaluate complex design decisions, such as diversification, in the face of operational uncertainties in dynamic environments?

As aforementioned in Chapter 1, despite the accidental or planned presence of diversity in many modern architectures, research still lacks an understanding of its core fundamentals; and justification/evaluation of the worthiness of this exercise. Further, design diversity-based systems have been criticised for their high cost, because each software version has to be implemented, which increases the implementation cost [168]. This has opened a series of works dedicated to mechanisms to assure the contrast, such as [179, 150]. We argue that this will always be the ongoing dilemma. To this end, it is necessary to develop cost-benefit trade-off analysis for the diversified architecture decisions, which is why we use CBAM [9, 23] with Real Options principles [176].

In particular, we develop a framework that extends CBAM [9, 23] with an emphasis on diversified architecture options, their cost, and the value they add to the software. Unlike CBAM, our focus is on valuing the options which diversification can embed in the architecture and their corresponding value using binomial real options theory [24]. More specifically, we use this information as a way to reason and reflect about their added value. CBAM tends to quantify the extent to which an architecture decision meets the scenarios and its response level, whereas our method operates on the improvement (if any) on the quality attributes' responses once we consider diversification. This improvement's response can carry an added value, but with a price. The use of real options analysis can adequately help to quantify and visualise that response under uncertainty. The essence is in the future opportunities flexibility creates and its contribution to long-term value creation. Therefore, the key issue is how to select the best-fit diversified option according to the current goal. Our holistic view is reaching a set of options that can have a global impact on long-term value and hence be able to deal with uncertain and dynamic environments.
Our method makes the following contributions to the research literature:

- an economics-driven method that evaluates and quantifies the long-term value of engineering diversification into the solution space at design-time in the face of operational uncertainties;

- a design-time evaluation method that shortlists the candidate architecture decisions for run-time evaluation and hence reduce deployment costs.

This chapter is organised as follows: Section 4.2 provides background for CBAM and real options analysis, which are necessary to understand the economics-driven design-time evaluation approach. Section 4.3 describes architecture diversification as real options in the context of the motivating example *iTransport*. Section 4.4 then introduces the proposed economics-driven design-time evaluation approach. Section 4.5 provides a constructive evaluation and discussion for the method. Finally, Section 4.6 concludes the work.

4.2 Background

This section explains the background needed by our proposed approach to quantify the impact of applying diversified decisions on the system's quality attributes.

4.2.1 Cost-Benefit Analysis Method (CBAM)

The CBAM was created to "develop a process that helps a designer choose amongst architecture options, during both initial design and its subsequent periods of upgrade, while being constrained to finite resources" [3]. CBAM extends ATAM [2] with an explicit focus on the costs and benefits of the architecture decisions in meeting scenarios related to quality attributes. CBAM is illustrated in Figure 4.1. It starts with the determination of business goals, which are elicited from the stakeholders. It has typically been troublesome for stakeholders to elicit goals, due to their vagueness and varying levels of abstraction (this is outside of our scope of research). For interested readers, a review of the categorisation of business goals is found in [180]. An example of a typical goal could be to cut down maintenance and development costs, or to provide a highly reliable and secure system. These goals drive the architecture decisions D. They have two major implications [9, 3, 2]: *technical* and *economic*. The former are the quality attributes portraying the system, e.g., performance, usability, security, *etc*. These in turn influence the benefit of the architecture. The latter are the costs of implementation, maintenance, operation, deployment, configuration, and so forth.

4.2.2 Real Options Analysis

Real options analysis is a well-known paradigm used for strategic decision-making [181]. It emphasises the value-generating power of flexibility under uncertainty. An option is the right, but not the obligation, to make an investment decision in accordance to given circumstances for a particular duration into the future, ending with an expiration date [24]. Real options are typically used for real assets (non-financial), such as a property or a new product design. Furthermore, they can be categorised into call and put options. The former gives the right to buy an uncertain future valued asset for the strike price by a specified date, whereas the latter offers the option to sell that asset. Two of the most popular approaches widely used to value real options are *Black-Scholes formula* [182] and *Binomial option pricing model* [183, 25, 42]. We employed the Binomial option pricing model [183, 25, 42], a popular approach to value real options. The choice of this model is advocated as it gives the architect

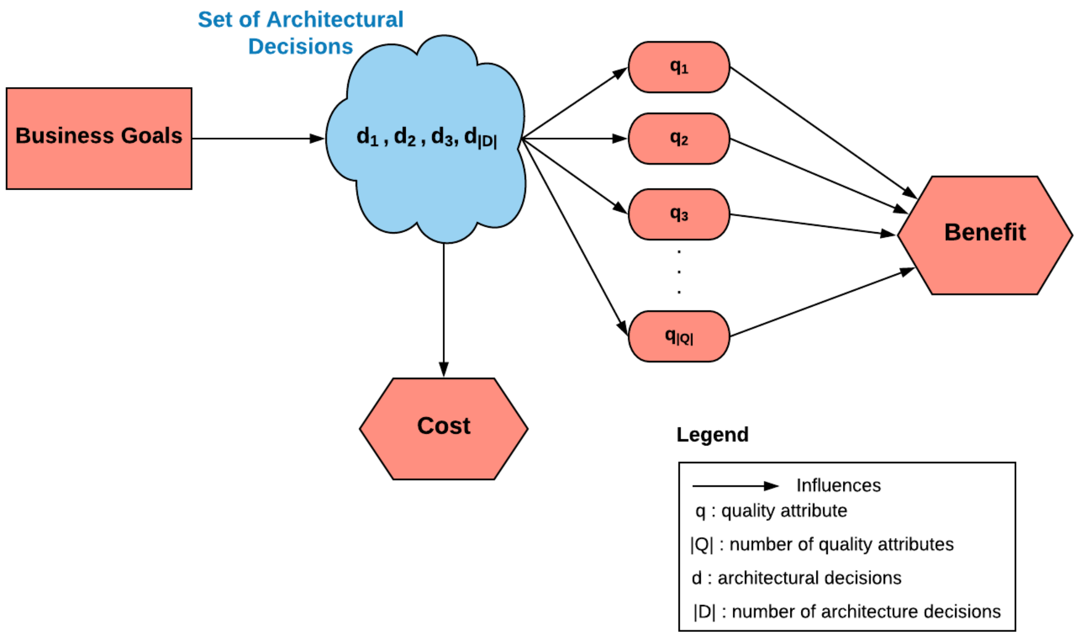

Set of Architectural Decisions
Business Goals
$d_1, d_2, d_3, d_{|D|}$
q_1
q_2
q_3
$q_{|Q|}$
Benefit
Cost
Legend
Influences
q : quality attribute
|Q| : number of quality attributes
d : architectural decisions
|D| : number of architecture decisions

Figure 4.1 Steps of Classical CBAM [2, 3].

the freedom to estimate the up and down in the value backed up by their experience. Whereas, Black and Scholes requires information about the volatility using history and the use of twin asset [37, 26], which is not always possible to have in practice. In this chapter, we have chosen the option to switch, wait, and abandon [184] to evaluate for diversification.

The Binomial option pricing model [183, 25, 4, 176, 42] estimates the value and price of options based on a binomial decision tree. The latter is ordinarily employed to model project flexibility [4]. The essence of the timing of architecture design decisions is homologous to an American option, where the option can be exercised any time until a given expiration date. In this context, the binomial tree with American option adopts a backward-looking strategy; forming the tree from the end to the beginning and checking at each decision point whether exercising the decision rule before the expiration of the option is optimal [42]. Figure 4.2 represents the structure of the tree for a given option. Each level of the tree represents a timestep. For example, if one timestep equals one month, the first level represents the present, level 1 represents one month into the future, level 2 represents 2 months into the future, and so on. The first node corresponds to the current market price, i.e. system value, S of the option. This system value has a probability p of rising to Su in the next timestep, and a probability of $(1-p)$ of going down to Sd. The u is the factor by which the stock price increases, whereas d is the factor by which the stock price decreases [42]. The market's

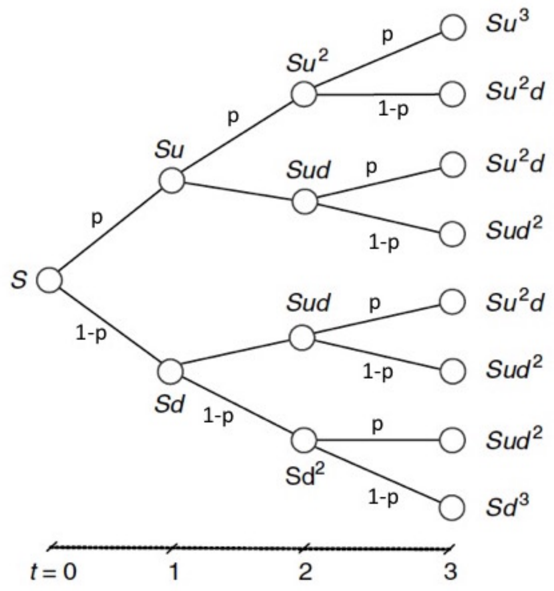


Figure 4.2 A typical binomial tree structure [4]. It uses u (an estimation for the stock increase), d (an estimation for the stock decrease), p (probability that stock increases), $(1-p)$ (probability that stock decreases) and S (system value of the option) to generate the tree from the final level (i.e. timestep 3) to the first level (i.e. timestep 0).

volatility, given by the experts, is used to compute the u and d coefficients [24]. Nodes Su and Sd are split again based on these probabilities, generating the potential values in the next timestep. The process of splitting nodes is repeated until we generate all desired levels of the tree. Once the potential system values covering the whole desired time period are determined, then the option value O, at each timestep, considering these system values is determined. The formulation of decision tree coefficients and examples of how to calculate the system and option value are given in section 4.4.

4.3 Architecture Diversification as a Real Options in the context of iTransport

In this section, we will introduce the notion of diversified options along with goal of the approach.

4.3.1 Options

We contend that diversification is likely to create decisions in the form of real options that can be exercised for value creation. An option which embeds architecture diversification is referred as a diversified architecture option (*dao*). For example, in *iTransport*, a *dao* could include d_{11}, d_{12}, and d_{21}. In particular, the architect has decided to diversify the type of things (d_1), which indicates the usage of fixed and/or mobile sensors (d_{13}), whereas performing all the computation in the cloud (d_{21}) forming dao_3. Other examples related to *DAO* were discussed in Section 3.3.

When evolving an existing system, the current implementation of the system has a direct ramification on the selection of a *dao*. It could provide an intuitive indication about whether the current system architecture needs to grow, alter, defer, *etc*. To exemplify, if the current system architecture is valued as having low long-term value, hence it is obvious that another *dao* should be employed instead. We will also demonstrate here the candidate architecture options of *iTransport* from Chapter 3 in Table 4.1.

Table 4.1 Possible Diversified Architecture Options for *iTransport* application. The diversity in each *dao* can refer to using fixed and/or mobile fog devices for data collection capability; using different cloud providers and heterogeneous fog devices for data processing capability.

Decision d_k / Option dao_i	Data Collection Capability (d_1)	Data Processing Capability (d_2)
1	Fixed (d_{11})	Cloud (d_{21})
2	Mobile (d_{12})	Cloud (d_{21})
3	Fixed and Mobile (d_{13})	Cloud (d_{21})
4	Fixed (d_{11})	Fog and Cloud (d_{22})
5	Mobile (d_{12})	Fog and Cloud (d_{22})
6	Fixed and Mobile (d_{13})	Fog and Cloud (d_{22})
7	Fixed (d_{11})	Fog (d_{23})
8	Mobile (d_{12})	Fog (d_{23})
9	Fixed and Mobile (d_{13})	Fog (d_{23})

4.3.2 Goal

The goal of our design-time approach is to help the architect to choose a *dao* that maximises the benefit of the system on some quality goals of interest. The selection shall consider both the cost and long-term value for these options, which is expected to provide insights on how to maximise it. More specifically, we propose an approach that can be used to evaluate a set of diversified architecture options from an economics-driven perspective under uncertainty.

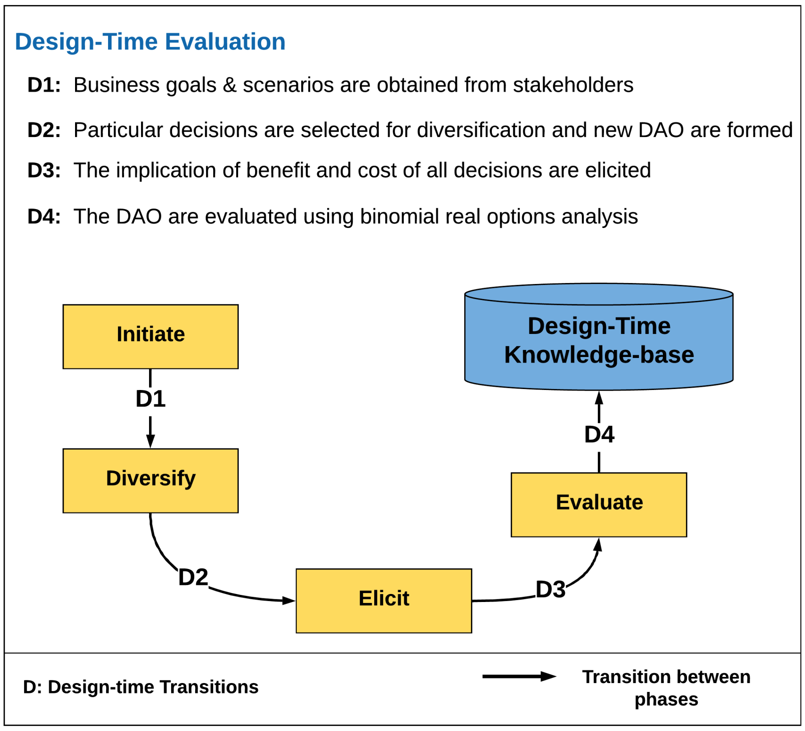

Design-Time Evaluation
D1: Business goals & scenarios are obtained from stakeholders
D2: Particular decisions are selected for diversification and new DAO are formed
D3: The implication of benefit and cost of all decisions are elicited
D4: The DAO are evaluated using binomial real options analysis
Initiate
D1
Diversify
D2
Elicit
D3
Evaluate
D4
Design-Time Knowledge-base
D: Design-time Transitions
Transition between phases

Figure 4.3 The design-time architecture evaluation approach.

The software architect can then choose the *dao* that provides the most balanced trade-off between long-term value (i.e. benefit) and cost.

4.4 The Economics-driven Design-time Evaluation Approach

Our proposed economics-driven design time evaluation approach builds upon the CBAM method for evaluating a set of diversified architecture options with real options theory, as illustrated in Figure 4.3. In our approach, we follow the steps of CBAM, found in [9]. The architect decides on some candidate diversified options. A key question to be answered in our method is *which dao will provide the most added value*. Despite the prominent outcome from having a variety of options, sometimes the value of an option in terms of cost exceeds its benefit. To this end, real-options theory is employed along with CBAM, for example, to provide design-time support for decisions that promote diversification and evaluate their long-term value.

4.4.1 Initiate and Diversify

In this step, we aim to select the business goals, scenarios of interest, and decisions. Based on that, some decisions are selected for diversification forming the diversified architecture options. We then elicit the relative importance of each quality attribute from the architects.

Step 1: Identifying diversified architecture options and attributes of interest

The analysis first aims at providing the system with typical business drivers, a number of scenarios of interest, and varying architecture decisions, which uses the formulation of CBAM. Our method focuses on quality attributes and their responses with respect to scenarios of interest, which are key tenets for *improving the long-term value of the system*. The business drivers encompass business goals, barriers, schedule, and cost constraints [23]. The treatment of business goals is implicit in CBAM and elicited from stakeholders. In CBAM, evaluation of design decisions and choices tend to align technical design decisions with their business potentials. Scenarios of interest are generally the system quality constraints which address a quality attribute. Moreover, *DAO* are the set of diversified architecture options that deal with these scenarios. Based on the business requirements, some critical capabilities are candidates for diversification. The combination of these diversified decisions form a *dao*. For instance, the candidate solutions of diversifying for performance are different from those for energy consumption, and so forth.

In our approach, exercising each single option *dao* can be formulated as a call option [24], with an exercise price and uncertain value. Our approach aims to maximise the benefit and minimise the cost of applying diversified options on a system's quality attributes over time, given the following information that must be specified:

- The set K of capabilities selected for diversification. Examples of capabilities are data collection, connectivity, processing, routing topology, and data management.
- The set D of architecture decisions. An architecture decision $d_{ka} \in D$ specifies an architecture component a to implement a given capability $k \in K$. For example, architecture decisions for the capability of data collection could be performed either through fixed, mobile, or fixed+mobile things.
- The set Q of qualities of interest. Response time and network usage are examples of quality attributes of interest.
- Each d_{ka} has a cost and quality. The cost of each architecture decision is c_{ka} and quality is q_{ka}, where ka identifies a decision $d_{ka} \in D$, c is a measure of cost, $q \in Q$ is a

measure of quality. In particular, each architecture decision d_{ka} will be associated to one measure of cost and $|Q|$ measures of quality.

- A set of diversified architecture options *DAO*, where $|DAO|$ represents the number of *DAO* and each $dao_i \in DAO$ is composed of a set of architecture decisions. For example, in *iTransport*, the architect has decided to diversify the type of things (d_1), which indicates the usage of fixed and/or mobile sensors (d_{13}), whereas performing all the computation (d_2) in the cloud (d_{21}) forming dao_3. Other examples of *DAO* are found in Table 4.1.

Step 2: Assessing the weight of each quality of interest

A ranking weight (w_q) must be chosen by the stakeholders to reflect the relative importance of each quality attribute to the benefit of the system, which should satisfy equation 4.1. Generally, stakeholders have diverse interests and quality needs [185], so there are several researches dedicated to reach a common understanding between stakeholders [186, 187]. This is outside of our scope.

$$\sum w_q = 1; \forall q : w_q \geqslant 0 \tag{4.1}$$

4.4.2 Elicit Benefits and Costs

Here, we aim to determine the benefits and costs accompanied by each *dao* from stakeholders.

4.4.2.1 Step 1: Eliciting the benefits of the diversified architecture options

Classically, any *dao* has an effect on a multitude of quality attributes. A *dao* will favour certain quality attributes and hurt others, which is represented in CBAM as a contribution score [9]. It shows how the architects "feel" about the impact of architecture decision on quality attributes of interest. In CBAM, it is theoretically rated using -1 to 1 scale, where ‘1’ resembles the best impact on quality attribute (e.g. high throughput value) and ‘-1’ means the opposite. Unlike CBAM, our approach adopts the *utility with quality* model, which infers the utility gained for quality attributes in question by investing (i.e. deploying) in this diversified architecture option. These utilities are elicited from stakeholders, which are input to the binomial trees. The utility gained from deploying a *dao* on a particular quality of interest is denoted by $V^q_{dao_i}$. The benefit of an architecture option B_{dao_i} is the weighted sum of utilities gained from deploying a *dao* on a particular quality of interest $V^q_{dao_i}$, where each quality attribute is assigned a ranking weight w_q (equation 4.2). For example, if the architect has assumed that dao_4 adds utility to the system through response time ($V^{RT}_{dao_i} = \$1800$) and network usage ($V^{NU}_{dao_i} = \$1200$), and the stakeholders set equal ranking weight for response time and network usage. Therefore, the benefit of dao_4 is equal to $w_{RT}V^{RT}_{dao_i} + w_{NU}V^{NU}_{dao_i} = 0.5 * 1800 + 0.5 * 1200 = \$1400$.

$$B_{dao_i} = \sum_{q \in Q} w_q V^q_{dao_i}; \forall q : V^q_{dao_i} \geqslant 0 \tag{4.2}$$

The $V^q_{dao_i}$ is elicited from the architects based on their knowledge about the expected *dao* behaviour. If the architects do not have enough knowledge about the application domain, they estimate $V^q_{dao_i}$ by looking at similar application architectures, domains, cross company data and publicly available benchmarks [26, 42].

4.4.2.2 Step 2: Eliciting the costs of diversified architecture options

Our consideration for the cost is situation dependent. As an example, the cost can relate to one or more dimensions of interest. This can include the cost of configuration, deployment, testing, maintenance, leasing, execution, *etc*. These costs can be estimated using parametric models, back-of-the-envelope estimation, and reliance on experts (i.e. architects and other stakeholders) [110, 188].

Classical CBAM uses the common measures for determining the costs, which involves the implementation costs only. Unlike CBAM, our approach embraces the switching costs between decisions, which is equivalent to primary payment required for purchasing a stock option, denoted by sc_{dao_i}. A switching cost could be the deployment cost of *dao*, which is considered once the architect decided to replace the current *dao*. This is in addition to the costs of configuration, maintenance, *etc* (as mentioned earlier), similarly to the exercise price, denoted by c_{dao_i}. The cost is computed using equation 4.3, where v considers a variety of costs (e.g. operating cost, leasing cost, *etc*).

$$c_{dao_i} = \sum c^v_{dao_i} \tag{4.3}$$

It is essential to note that CBAM implements the architecture decisions with high benefit and low cost [9]. On the other hand, we believe that some architecture decisions could provide high cost with low benefit initially or high cost with high benefit, but a much higher benefit in the long-term that outweighs the cost. To this extent, the long-term benefit is the key factor for the decisions' evaluation.

4.4.3 Evaluate

In real options analysis, the value of a project's cash flow is analogous to the stock price in a financial option [176]. We employ a different approach by encompassing the *utility with quality* (i.e. $V^q_{dao_i}$). Our *stock price* is translated to the system value S_{dao_i} and benefit B_{dao_i} (i.e. weighted sum of utility values), whereas the *exercise price* is the cost of implementing/deploying that *dao* (i.e. c_{dao_i}). The *volatility* is the architecture's fluctuation (i.e. uncertainty in QoS), which is represented using p (explained in Section 4.4.3.2). We consider the risk-free interest rate r to be a known market value by the architect [176]. The time until the opportunity disappears (i.e. date on which *dao* implementation decision must be taken) is equivalent to the *time to expiration*.

For our analysis, the real options will help us in deciding which *dao* to embed in the architecture and when. This is based on the value of the options created by the chosen *dao*. As aforementioned, diversification decisions can be formulated as options that can be evaluated in the presence of uncertainty. *DAO* supporting diversification can be valued as calls. The value of these calls provide the right without symmetric obligations to *switch* to a new *dao* if the exercise is favourable. Further, our approach embraces the option to *wait* to switch, where the rational investigation of an option and waiting for provocative outcome has a positive impact on switching options. Another use of options valuation is *abandoning* an option, if the current one is no longer favouring the system.

We have exploited the binomial pricing calculation approach proposed by [25, 183, 176, 42] to calculate the option value. The binomial model provides a visual representation for the options. It is a constructive aid aiming to show the suitable time slot for exercising an option. In other words, it indicates the cost-benefit of diversified options over time. For each level of the binomial tree, the up and down node values are important in determining the system value rise and fall, which is ultimately used to calculate the option value. Our method aims to determine the impact of applying each *dao* (i.e. utility) on the system quality attributes, which is computed at every time slot t, where $t = T$ indicates that the time equals to T unit time of interest. In the binomial tree [25, 183, 176], the option is exercised at time T. So, *OnePeriod* tree assumes that the option is exercised at time $T = OnePeriod$ (i.e. one tree level), *TwoPeriod* assumes that the option is exercised at $T = TwoPeriod$ (i.e. two tree levels), *etc*. We measure the benefit and costs (operating and switching) in dollars. Next, we will explain the steps of binomial real options evaluation along with Figure 4.4 and Algorithm 1.

ALGORITHM 1: EvaluateDAOUsingBinomialRealOptions()

Input: diversified architecture options DAO, number of DAO ($|DAO|$), System initial value S_{ini}, risk-free interest rate r, utility trees (i.e. $V^q_{dao_i}(t)$, which represents the increase/decrease in the quality attribute value over time), operating cost $c_{dao_i}(t)$, switching cost sc_{dao_i}, and $listOfPeriods$ which represents the number of exercise strategies

Output: net value NV_{dao_i}

```
1  for $i = 1$ to $|DAO|$ do
2      foreach $period \in listOfPeriods$ do
3          $T = period$
4          for $t = 0$ to $T$ do
               {Calculate the benefit and system value of $dao_i$ by forward passing through all the utility tree (upper and lower nodes)}
5              for $q = 1$ to $|Q|$ do
6                  $B_{dao_i}(t) = \sum_{q \in Q} V^q_{dao_i}(t)$
               end
7              $S_{dao_i}(t) = S_{ini} + B_{dao_i}(t)$
           end
           {Calculate the likely rise $O_u(t)$ and fall $O_d(t)$ of payoff with each $dao$ for the final node in tree (i.e. $t = T$)}
8          $O_u(t) = max(0, S_{dao_i}(t) - c_{dao_i})$ {Use $S_{dao_i}(t)$ from upper node}
9          $O_d(t) = max(0, S_{dao_i}(t) - c_{dao_i})$ {Use $S_{dao_i}(t)$ from lower node}
10         for $t = T - 1$ to $0$ do
               {Calculate the value rise factor $u$ and value fall factor $d$ }
11             $u = \frac{S_{dao_i}(t+1)}{S_{dao_i}(t)}$ {Use $S_{dao_i}(t+1)$ from upper node}
12             $d = \frac{S_{dao_i}(t+1)}{S_{dao_i}(t)}$ {Use $S_{dao_i}(t+1)$ from lower node}
               {Calculate the risk-adjusted probability $p$ }
13             $p = \frac{1+r-d}{u-d}$
               {Calculate the option value of exercising $dao_i$ }
14             $O_{dao_i}(t) = \frac{pO_u(t+1)+(1-p)O_d(t+1)}{1-r}$
           end
15         $O^{period}_{dao_i} = O_{dao_i}(0)$
       end
16     $O_{dao_i} = \sum_{period} O^{period}_{dao_i}$ {where $period \in listOfPeriods$
       {Calculate the net value of $dao_i$ }
17     $NV_{dao_i} = O_{dao_i} - sc_{dao_i}$
   end
```

4.4.3.1 Step 1: Anticipating the utility values and calculating the system value

As a start, the favourable and unfavourable outcomes at each time period are anticipated using a utility tree from the architects. A sample of a utility tree is shown in Figure 4.4 (left side), where each node corresponds to the utility gained from implementing dao_i on a particular quality attribute over timestep t (i.e. $V^q_{dao_i}(t)$). For instance, in the case of improvement in utility at time $t+1$, the elicited utility is placed in upper nodes (i.e. node **B** in Figure 4.4 (left side)), whereas for degradation in utility at time $t+1$, the elicited utility is placed in lower nodes (i.e. node **C** in Figure 4.4 (left Side)). The same process is repeated till we reach the final nodes of the tree (i.e. $t = T$). Each quality attribute has its own utility tree (i.e. utility tree for response time, utility tree for energy consumption, *etc*). We suggest reading subsection 4.5.2.1 for better illustration and discussion about the utility trees. The benefit of a *dao* over time $B_{dao_i}(t)$ is computed using equation 4.4.

$$B_{dao_i}(t) = \sum_{q \in Q} w_q V^q_{dao_i}(t) \tag{4.4}$$

Let $S_{dao_i}(t)$ be the system value after implementing a particular *dao* causing either incremental improvement or degradation at time t (line 4-7), which is equivalent to the uncertain stock price when modelling an American call option. It is evaluated with respect to the initial system value, denoted by S_{ini} and benefit gained from a *dao* over time t ($B_{dao_i}(t)$), as depicted in equation 4.5. In equation 4.5, we used equation 4.4 with the values (i.e. $V^q_{dao_i}(t)$) from the utility tree to compute $B_{dao_i}(t)$.

$$S_{dao_i}(t) = S_{ini} + B_{dao_i}(t) = S_{ini} + \sum_{q \in Q} w_q V^q_{dao_i}(t) \tag{4.5}$$

As seen in Figure 4.4 (right side), the system values at each timestep $S_{dao_i}(t)$ are used as a part of the construction of the binomial tree. In particular, they are used to calculate the likely rise and fall of payoff with *DAO*, as shown in the next step.

4.4.3.2 Step 2: Calculating the likely rise and fall of payoff with *DAO*

Let $O_u(t)$ be the likely rise of payoff at time t, i.e. rising option value from implementing a *dao*, whereas $O_d(t)$ is the likely fall of payoff , i.e. falling option value from implementing a *dao*. The $O_u(t)$ and $O_d(t)$ are computed using the same formula (i.e. equation 4.6). Except that system value of a dao_i at t=1 from the *upper* node (i.e. node B in Figure 4.4 (left side)) is used to compute $O_u(t)$ (line 8), whereas system value of a dao_i at t=1 from the *lower* node (i.e. node C in Figure 4.4 (left side)) is used to quantify $O_d(t)$ (line 9) and so

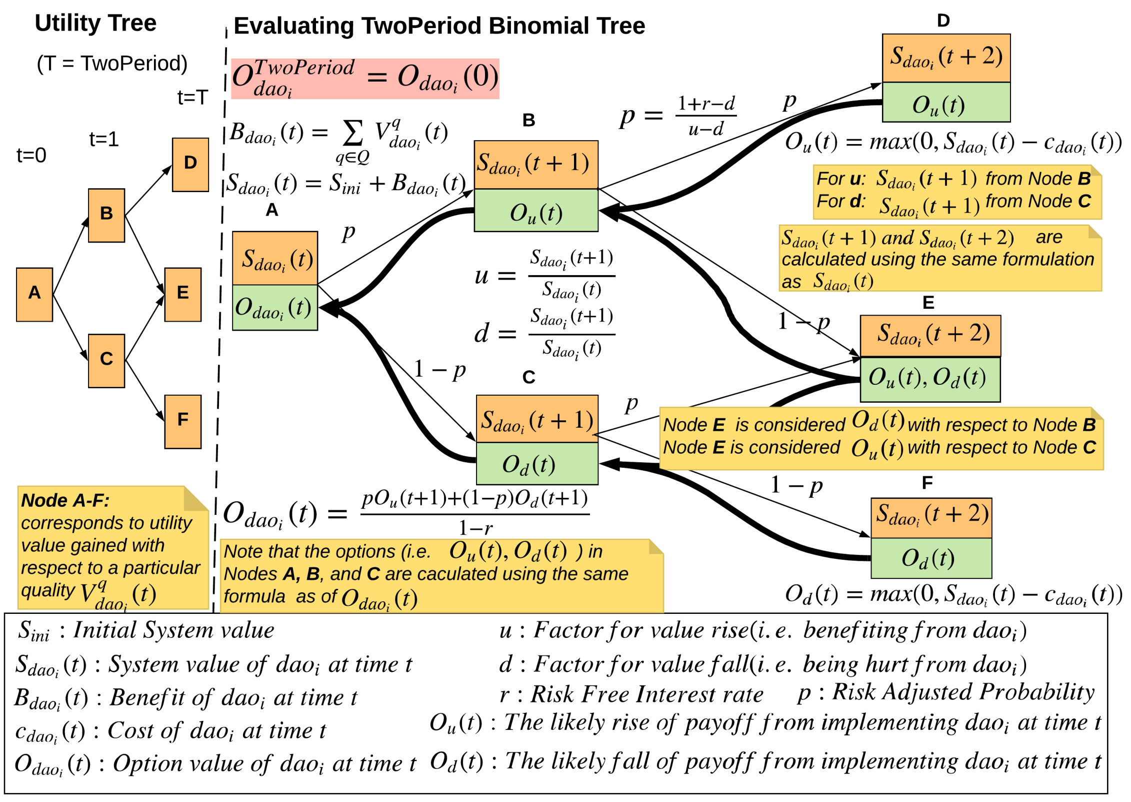

Utility Tree
(T = TwoPeriod)
t=0
t=1
t=T
A
B
C
D
E
F
Node A-F: corresponds to utility value gained with respect to a particular quality $V^q_{dao_i}(t)$
Evaluating TwoPeriod Binomial Tree
$O^{TwoPeriod}_{dao_i} = O_{dao_i}(0)$
$B_{dao_i}(t) = \sum_{q \in Q} V^q_{dao_i}(t)$
$S_{dao_i}(t) = S_{ini} + B_{dao_i}(t)$
$S_{dao_i}(t)$
$O_{dao_i}(t)$
$S_{dao_i}(t+1)$
$O_u(t)$
$O_d(t)$
$S_{dao_i}(t+2)$
$O_u(t), O_d(t)$
p
$1-p$
$p = \frac{1+r-d}{u-d}$
$u = \frac{S_{dao_i}(t+1)}{S_{dao_i}(t)}$
$d = \frac{S_{dao_i}(t+1)}{S_{dao_i}(t)}$
$O_u(t) = max(0, S_{dao_i}(t) - c_{dao_i}(t))$
For u: $S_{dao_i}(t+1)$ from Node B
For d: $S_{dao_i}(t+1)$ from Node C
$S_{dao_i}(t+1)$ and $S_{dao_i}(t+2)$ are calculated using the same formulation as $S_{dao_i}(t)$
Node E is considered $O_d(t)$ with respect to Node B
Node E is considered $O_u(t)$ with respect to Node C
$O_{dao_i}(t) = \frac{pO_u(t+1)+(1-p)O_d(t+1)}{1-r}$
Note that the options (i.e. $O_u(t), O_d(t)$) in Nodes A, B, and C are caculated using the same formula as of $O_{dao_i}(t)$
$O_d(t) = max(0, S_{dao_i}(t) - c_{dao_i}(t))$
S_{ini} : Initial System value
$S_{dao_i}(t)$: System value of dao_i at time t
$B_{dao_i}(t)$: Benefit of dao_i at time t
$c_{dao_i}(t)$: Cost of dao_i at time t
$O_{dao_i}(t)$: Option value of dao_i at time t
u : Factor for value rise(i.e. benefiting from dao_i)
d : Factor for value fall(i.e. being hurt from dao_i)
r : Risk Free Interest rate
p : Risk Adjusted Probability
$O_u(t)$: The likely rise of payoff from implementing dao_i at time t
$O_d(t)$: The likely fall of payoff from implementing dao_i at time t

Figure 4.4 An illustration of the utility trees (left side) and binomial trees (right side) for a TwoPeriod tree.

on. Besides, c_{dao_i} is the cost for maintaining/configuring/operating a *dao* (corresponding to exercise price), computed using equation 4.3. In our case, equation 4.6 is applied first at $t = T$, i.e. the final nodes in the tree, as seen in Figure 4.4.

$$O_{u,d}(t) = max(0, S_{dao_i}(t) - c_{dao_i}) \tag{4.6}$$

4.4.3.3 Step 3: Calculating the option value of exercising a *dao*

The option value reveals *at what time t it is favourable to take the decision*, i.e. exercise an option. It also illustrates the long-term value of a *dao*. In step 2, we have demonstrated the process of quantifying the option values (i.e. $O_u(t)$ and $O_d(t)$) at the final tree level. To compute the option value of exercising a diversified architecture option $O_{dao_i}(t)$ for other tree levels (i.e. from $t = T - 1$ to $t = 0$), we need first to compute the following coefficients u, d, r, and p. The system value benefiting from *dao* i.e. value rise, is denoted by the up factor u, whereas d is the down factor denoting the system value being hurt from *dao* i.e. value fall. In our approach, the u and d coefficients are computed from the expected system values, elicited from architects. However, there are other ways to calculate them in classical n-level binomial trees [183, 176]. To be consistent with the classical binomial tree, explained in Section 4.2.2, we assume that $S_{dao_i}(t+1) = uS_{dao_i}(t)$. For instance, at $t = 0$, $S_{dao_i}(1)$ in the *upper* node is equal to $uS_{dao_i}(0)$, whereas $S_{dao_i}(1)$ in the *lower* node is equivalent to $dS_{dao_i}(0)$. In this context, the u and d are computed using the same formula (i.e. equation 4.7), as seen in Figure 4.4 (right side). Except that $S_{dao_i}(t+1)$ is used from the *upper* node to compute u (line 11), whereas $S_{dao_i}(t+1)$ is taken from the *lower* node to determine d (line 12).

$$u,d = \frac{S_{dao_i}(t+1)}{S_{dao_i}(t)} \tag{4.7}$$

To compute the risk-adjusted probability, we use equation 4.8 (line 13). The risk-free interest rate r is an estimated theoretical concept, and a measure of an economy's long-run growth rate in a country [189]. The u and d should satisfy the following constraint: $d < 1 + r < u$ [42]. The latter is important, to avoid a negative risk-adjusted probability p [183].

$$p = \frac{1+r-d}{u-d} \tag{4.8}$$

Therefore, the option value is computed using equation 4.9 (line 14).

$$O_{dao_i}(t) = \frac{pO_u(t+1) + (1-p)O_d(t+1)}{1-r}, 0 \leq t \leq (T-1) \tag{4.9}$$

Since we adopt a backward-looking strategy for the quantification of option value, therefore, the option value of a *dao* for a period $O^{period}_{dao_i}$ is equal to option value at the first node of the tree (i.e. $O_{dao_i}(0)$), as seen in equation 4.10 (line 15). This is analogous to Figure 4.4.

$$O^{period}_{dao_i} = O_{dao_i}(0), where\ period \in listOfPeriods \tag{4.10}$$

Here, we will summarise the previous steps through Figure 4.4. Let us consider the case of having TwoPeriod tree as in Figure 4.4, we initially compute system value $S_{dao_i}(t)$ for all the nodes in the upper orange cells of nodes (A-F) using equation 4.5. Since we adopt backward-looking strategy for the quantification of option value, then the option value will be computed from final node to start node. For example, at $T = TwoPeriod$, we computed the $O_u(2)$ and $O_d(2)$ in lower green cells (at nodes D, E, and F) using equation 4.6. After that, we determined the corresponding u and d factors (equation 4.7), then p is computed using equation 4.8. Note that r is a known market value estimated from the architects. We then used the equation 4.9 to quantify the option value $O_{dao_i}(t)$. The process of calculating u, d and p to quantify the option value is repeated for $O_{dao_i}(1)$ and $O_{dao_i}(0)$ (i.e. till we reach the start node). The final option value $O_{dao_i}^{TwoPeriod}$ is at $t = 0$, which is computed using equation 4.10.

In our approach, we adopt a *compound option* strategy [190, 191]. This type of strategy implies that exercising an option in a time t enables further options in the upcoming times. For example, in the context of *iTransport*, the compound options mean that exercising an option related to energy consumption can enable further options to maintain that QoS even further. The premise is in finding a diversified architecture option that can deal with the energy consumption constraint, every time change occurs. This change could happen due to the heterogeneity of devices in computational power and excess load. In this context, at design-time, the architect attempts to find a *dao* that can "survive", i.e. maintain the required QoS, despite these uncertainties. Therefore, the option value will be calculated by adding the option value for several periods using equation 4.11 (line 16), after computing them separately, as shown in equation 4.10.

$$O_{dao_i} = \sum_{period} O_{dao_i}^{period}, where\ period \in listOfPeriods \tag{4.11}$$

For example, if we have OnePeriod, TwoPeriod, and ThreePeriod exercise strategies, then they are computed as in equation 4.12.

$$O_{dao_i} = O_{dao_i}^{OnePeriod} + O_{dao_i}^{TwoPeriod} + O_{dao_i}^{ThreePeriod} \tag{4.12}$$

We then compute the net value of each *dao* (NV_{dao_i}) to determine the "suitable" *dao* after switching using equation 4.13 (line 17). An option should be exercised only when the net value of exercise is positive [192].

$$NV_{dao_i} = O_{dao_i} - sc_{dao_i} \tag{4.13}$$

After quantifying the net value for each *dao*, we exclude the *DAO* with negative net values from the shortlist of *DAO* to be deployed, as this indicates that the costs of these options exceed its benefits and hence it is not favourable to deploy them. In this context, this would aid the architect in shortlisting the *DAO* for deployment as part of the continuous evaluation framework.

In the next section, we show a step-by-step example of how to construct a binomial tree and calculate the option values through *iTranport* case study.

4.5 Evaluation and Discussion

In this section, we aim to show how the economics-driven design-time approach can evaluate the architecture options for uncertainty in the presence of varying QoS constraints. Section 4.5.1 introduces the experimental settings, whereas Section 4.5.2 demonstrates the application and evaluation of the approach on the *iTransport* case study.

4.5.1 Experimental Setting

The major business goals of *iTransport* is improving its performance (i.e. response time and network usage) and energy consumption. This is translated into a set of constraints, which address those quality attributes. The quality attributes of interest are: *response time* of an application (in milliseconds *ms*) that is measured through the application's end to end delay; *energy consumption* (in mega joules *MJ*) that is the total energy consumption by all the devices in the application; *network usage* (mega bytes *MB*). The sources of uncertainty in *iTransport* application come from the hyper-connectivity (nodes leave and join) and varying network delays due to network congestion. The constraints employed for evaluation in this chapter are summarised in Table 4.2. Historical performance and experts' judgement can inform the adjustment of the prior constraints [193, 35]. The cost is a composite of operating cost, and switching cost that is added once we switch. In the context of *iTransport*, the average cost encompasses thing's connectivity (includes switching), execution in the cloud and/or fog, and other costs mentioned in Chapter 3. After that we elicit the operating and switching costs from stakeholders in dollars ($), as depicted in Table 4.3. Risk-free interest rate is hard to estimate or apply on the case. Consequently, for simplicity, we assumed that the risk-free interest rate *r* is 1.00% throughout the evaluation.

4.5.2 Evaluation

The next series of experiments aim to show how the proposed design-time method could be applied on *iTransport* through two case scenarios.

4.5.2.1 Case 1: Energy Consumption as a Concern

Motivation: In this experiment, we aim to show how the real options theory can complement the architect's decision related to the optimal architecture option and whether the selected option is "suitable" or not, by focusing on a single quality concern. It will also illustrate whether a more diversified option embeds further options (i.e. has better added value).

Table 4.2 Quality Attribute Responses

Case	Quality Attribute	Constraint
1	Response Time	Application should handle the request in $\leq$ *4000 ms*
	Network Usage	Average load on the network $\leq$ *1Mbps*
	Energy Consumption	Average energy consumption in the whole architecture option $\leq$ *110 MJ*
2	Response Time	Application should handle the request in $\leq$ *350 ms*
	Network Usage	Average load on the network $\leq$ *500Kbps*
	Energy Consumption	Average energy consumption in the whole architecture option $\leq$ *120 MJ*

Table 4.3 Costs of diversified architecture options

dao_i	Operating Cost	Switching Cost
1	$1800	$600
2	$1000	$500
3	$1200	$1000
4	$1500	$700
5	$900	$300
6	$1100	$900
7	$1600	$600
8	$1200	$400
9	$1500	$700

Experimental Setup: The first case scenario for evaluation is *Case 1*, shown in Table 4.2. The architect chooses the most suitable *dao* for implementation aiming to improve the long-term energy consumption with respect to the first constraint scenario in Table 4.2. We assume that, currently, the response time and network usage are not critical attributes and maintaining the energy consumption is the concern. Consequently, we did not consider additional utility gained from the response time or network usage. The architect has suggested dao_1, which includes a set of fixed IoT devices and processes the data through the cloud, as the best option for deployment. This choice is based on his/her knowledge and experience. Therefore, in this experiment, we aim to show whether this choice is "suitable" for the concerns.

The long-term value of the current option is evaluated to determine its benefit, in comparison with the other diversified architecture options. The utility is significantly affected by the outcome of the uncertain quality attribute *energy consumption* (measured in *MJ*). The expected favourable and unfavourable outcomes at each time period are depicted using utility trees in Figure 4.5. Note that we assume that every single time period is equivalent to 40 timesteps. For instance, at node **D** in Figure 4.5a, a utility of \$950 is gained from investing in energy consumption after 80 timesteps ($V^{EC}_{dao_1}(t = 80)$). All the utility values are elicited from the architects. Since we only have one quality constraint (i.e. $w_{EC} = 1$), then $B_{dao_i}(t) = V^{EC}_{dao_i}(t)$ when quantifying all system values $S_{dao_i}(t)$. Table 4.4 summarises the option and net value of all the *DAO* for energy consumption constraint. We have demonstrated in this chapter some examples for utility trees and binomial trees for illustration purposes, whereas the rest are shown in Appendix B.

Analysis: Next, we will demonstrate the necessary steps for valuation of options using binomial option pricing model. We assume that the initial system value S_{ini} for all the *DAO* is \$1000, throughout the evaluation process. Generally, for the binomial trees in Figure 4.6-4.7, each node in the tree consists of two cells: upper and lower. The upper *orange* cells denote the system value (equation 4.5), whereas the lower *green* cells indicate the option value (equation 4.6 and 4.9). For illustration purposes, we will show how the option value of dao_1 is calculated, whereas the other *DAO* follow the same procedures.

Consider **node D** from the tree for the evaluation of 80-timestep as presented in Figure 4.6b, where the upper orange cell value is computed as follows:

$$S_{dao_1}(t = 80) = S_{ini} + B_{dao_1}(t = 80) = 1,000 + 950 = \$1,950$$

where $B_{dao_1}(t = 80)$ is the benefit with respect to utility gain when lowering the energy consumption. The lower green cell value is computed as follows:

$$O_{dao_1}(t = 80) = max(0, S_{dao_1}(t = 80) - c_{dao_1}) = max(0, 1,950 - 1,800) = \$150$$

Consider also **node B** from the tree for evaluating 80-timestep as presented in Figure 4.6b, the upper orange cell value is computed as follows:

$$S_{dao_1}(t = 40) = S_{ini} + B_{dao_1}(t = 40) = 1,000 + 900 = \$1,900$$

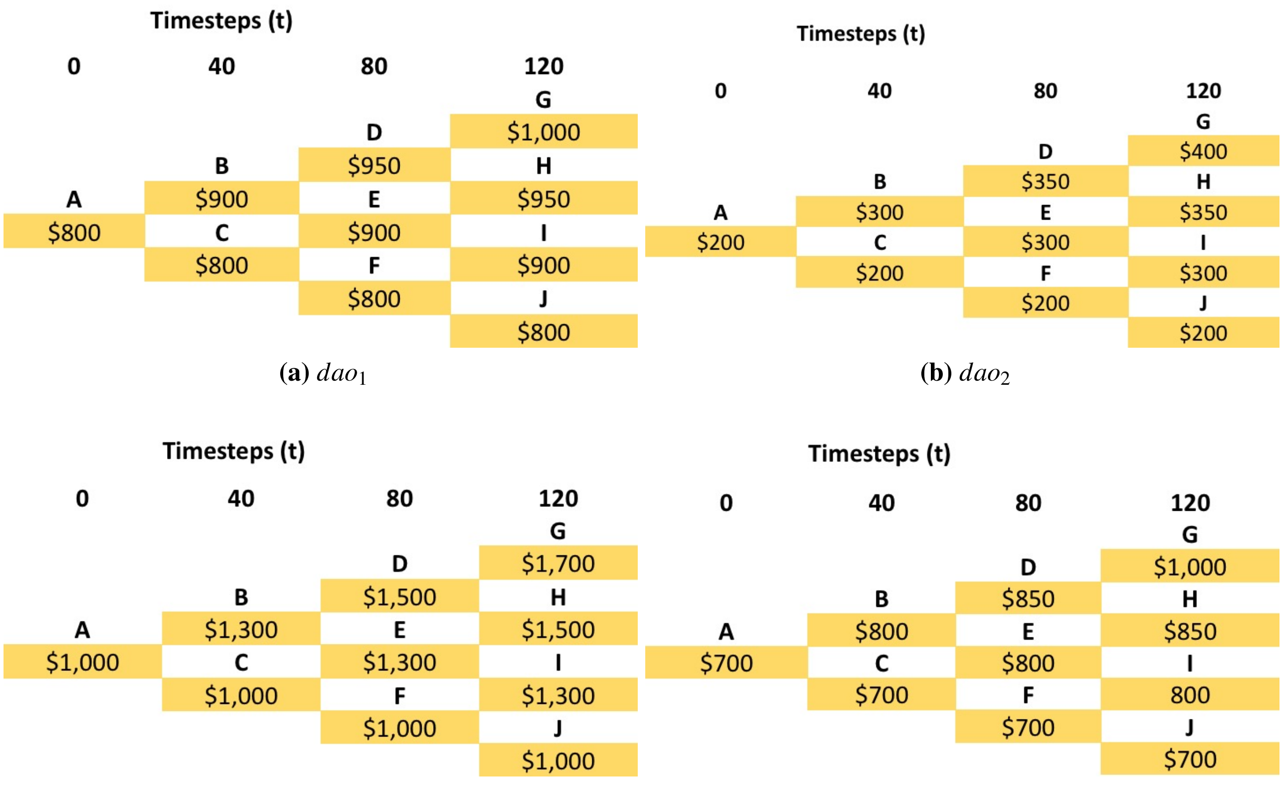
After that we compute the u and d values as follows:

$$u = \frac{S_{dao_1}(t=80)}{S_{dao_1}(t=40)} = \frac{1,950}{1,900} = 1.03$$

where $S_{dao_1}(t=80)$ is taken from the upper cell in node **D**, and $S_{dao_1}(t=40)$ is used from the upper cell in node **B**.

$$d = \frac{S_{dao_1}(t=80)}{S_{dao_1}(t=40)} = \frac{1,900}{1,900} = 1.00$$

where $S_{dao_1}(t=80)$ is taken from the upper cell in node **E**, and $S_{dao_1}(t=40)$ is used from the upper cell in node **B**.



(a) dao_1 **(b)** dao_2

(c) dao_3 **(d)** dao_4

Figure 4.5 Utility Tree of all options for Case 1.

The risk-adjusted probability p is computed as follows:

$$p = \frac{1+r-d}{u-d} = \frac{1+0.001-1.00}{1.03-1.00} = 0.334$$

Note that u, d, and p are calculated in the rest of analysis using the same formulation as above. After that, the lower green cell value is computed as follows:

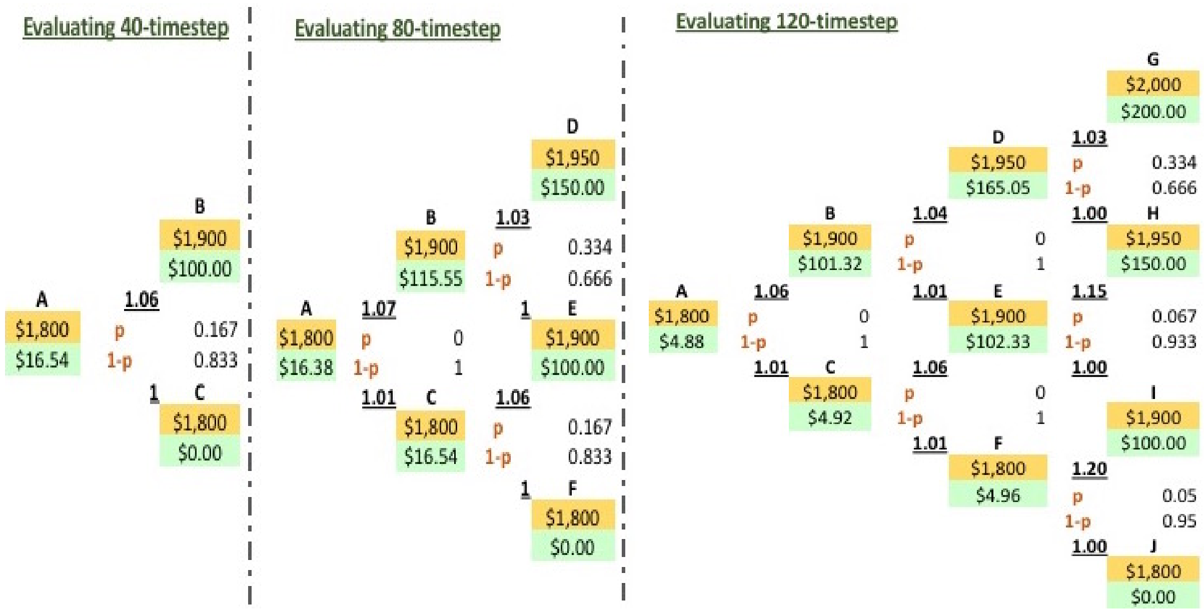
$$O_{dao_1}(t=40) = \frac{p*O_u(t=80)+(1-p)*O_d(t=80)}{1-r} = \frac{0.334*1,950+0.666*1,900}{1-0.01} = \$115.55$$

Consider, then, **node A** from the tree for evaluating 80-timestep, the upper orange cell value is calculated as follows:

$$S_{dao_1}(t=0) = S_{ini} + B_{dao_1}(t=0) = 1,000+800 = \$1,800$$

The lower green cell value is computed as follows:

$$O_{dao_1}(t=0) = \frac{p*O_u(t=40)+(1-p)*O_d(t=40)}{1-r} = \frac{0*115.55+1*16.54}{1-0.01} = \$16.38$$

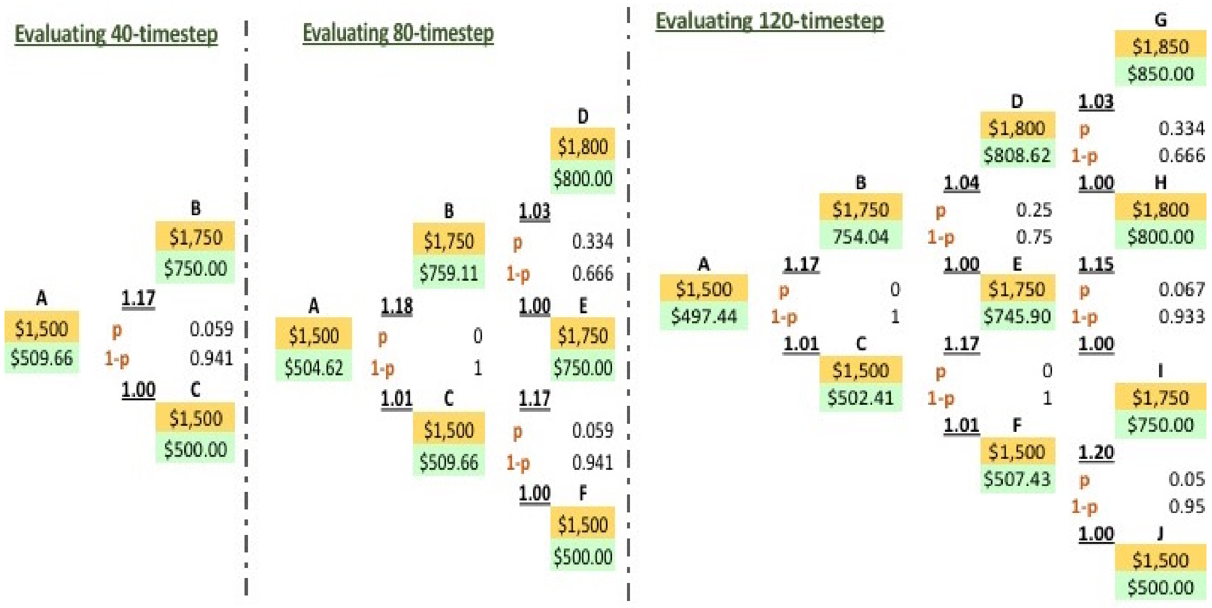


(a) T=40-timestep **(b)** T=80-timestep **(c)** T=120-timestep

Figure 4.6 Binomial Tree of dao_1 for Case 1.

Therefore, option value formula for 80-timestep $O_{dao_1}^{80\text{-}timestep}$ is equal to:

$$O_{dao_1}^{80\text{-}timestep} = O_{dao_1}(t=0) = \$16.38$$



(a) T=40-timestep **(b)** T=80-timestep **(c)** T=120-timestep

Figure 4.7 Binomial Tree of dao_1 for Case 1.

The same formulation is used to compute option value for 40-timestep $O_{dao_1}^{40\text{-}timestep}$ (Figure 4.6a) and for 120-timestep $O_{dao_1}^{120\text{-}timestep}$ (Figure 4.6c). Finally, the option value formula O_{dao_1} of dao_1 (Figure 4.6) is:

$$O_{dao_1} = O_{dao_1}^{40\text{-}timestep} + O_{dao_1}^{80\text{-}timestep} + O_{dao_1}^{120\text{-}timestep} = 16.54 + 16.38 + 4.88 = \$37.80$$

The same formulation is applied to other diversified options. The corresponding operating costs to each *dao* is taken from Table 4.3. The option value of dao_2, as computed from Figure 4.7 is:

$$O_{dao_2} = O_{dao_2}^{40\text{-}timestep} + O_{dao_2}^{80\text{-}timestep} + O_{dao_2}^{120\text{-}timestep} = 209.11 + 207.04 + 199.00 = \$615.15$$

For the other options as seen in Table 4.4, O_{dao_3}=\$2426.41, O_{dao_4} =\$949.96, O_{dao_5} =\$1,221.42, O_{dao_6} =\$1,821.59, O_{dao_7} =\$33.06, O_{dao_8} =\$174.15, and O_{dao_9} =\$339.73. Note that if dao_1 will continue to be deployed, then there is no switching cost (i.e. \$0), because it is already selected for deployment by the team of architects. From the above valuation, it is clear that the currently selected option for deployment dao_1 has low option value (\$37.8) as compared with dao_2 to dao_6. However, it is still better than dao_7, dao_8, and dao_9. More specifically, when energy consumption becomes important, the current *dao* is no longer

Table 4.4 Option and Net values of diversified architecture options for Energy Consumption Concern (i.e. Case 1). *Note that the dao_i with the highest net value is highlighted in gray.*

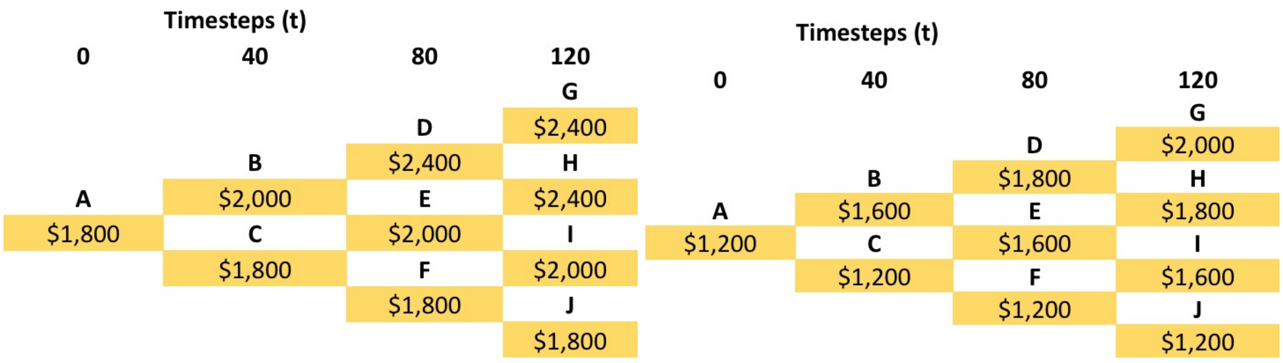
dao_i	Option Value O_{dao_i}	Net Value NV_{dao_i} (i.e. after deducting switching cost)
1	\$37.80	\$37.80
2	\$615.15	\$115.15
3	\$2,426.41	**\$1,426.41**
4	\$949.96	\$249.96
5	\$1,221.42	\$921.42
6	\$1,821.59	\$921.59
7	\$33.06	\$-566.94
8	\$174.15	\$-225.85
9	\$339.73	\$-360.27

applicable. So by switching to other *dao*, the architecture could be able to cope with the uncertainties. Further, dao_3 provides the highest value of \$2,426.41 as compared to other options. Even though the switching cost is quite high (\$1,000), yet dao_3 had the best net value of $\$2,426.41 - \$1,000 = \$1,426.41$ as compared to dao_1 (which provides a value of (\$37.80 – \$0 = \$37.80)) and the other options. Therefore, it is more favourable to switch to dao_3, which creates the best value. This may be because dao_3 is more diverse than dao_1, due to the presence of different types of things (fixed/mobile). So dao_3 is more flexible than dao_1 in handling and improving the energy consumption concerns, because the mobile things may consume less power due to their hyper-connectivity (nodes leave/join). We can see from Table 4.4 that dao_7, dao_8, and dao_9 provided a negative net option value. This indicates that these *DAO* will not be able to handle the run-time uncertainties. Therefore, it is advantageous to eliminate them from the list of candidate options for run-time evaluation.

4.5.2.2 Case 2: Response Time and Network Usage as Concerns

Motivation: In this experiment, we aim to determine the options' impact on two quality attributes: *response time* and *network usage*. The evaluation is performed with respect to *Case 2* in Table 4.2. We will also show whether the architect's choice is complementary to the option's value and whether diversity is always effective or not.

Experimental Setup: As a starting point, we elicit the utility outcomes from the uncertain variable in *response time* (measured in seconds) and *network usage* (measured in Kbps). This is translated into utility trees for response time and network usage of all the diversified architecture options. Consider dao_1 as an example (Figure 4.8), there are few possible utility outcomes for this option, as shown in Figure B.9a and B.9b. Investing more in network usage (Figure B.9b) could result in a noticeable improvement, i.e. more value ($2000 at node G), which could enhance the lifetime of the application. However, in some occasions, investing more in another quality attribute, such as response time, may not deliver a remarkable value. This is clearly seen in Figure B.9a, after 80 timesteps the system is not creating any noticeable added value. For Case 2, the architect's initial choice was dao_4 due to the presence of fog-cloud as a computation location, so here we will illustrate whether it will continue to deliver value. Now, we will examine the option value of the architect's choice dao_4 and other diversified options, with respect to *response time* and *network usage*. We exploit the same rationale used in Case 1. In Case 2, the stakeholders have set the following ranking weights: $w_{RT} = 0.5$ and $w_{NU} = 0.5$. Therefore, $B_{dao_i}(t) = 0.5 * V^{RT}_{dao_i}(t) + 0.5 * V^{NU}_{dao_i}(t)$ when quantifying all system values $S_{dao_i}(t)$. The corresponding operating and switching costs to each option is taken from Table 4.3. In this case, we have developed two scenarios: *best* and *worst*, which will explained thoroughly in the analysis.



(a) Utility Tree for Response time of dao_4. **(b)** Utility Tree for Network Usage of dao_4.

Figure 4.8 Utility Tree of dao_4 for Case 2 (Best Scenario).

Analysis (Best Case Scenario): Let us consider the case where the architects are confident about their perceptions with respect to *DAO* utilities In particular, the architects have very good knowledge about the *DAO* utilities. Next, we will demonstrate how the approach will compute the option value for more than one quality attribute (i.e. response time and energy consumption) where dao_4 is used as an example. Consider **node D** for the evaluation of 80-timestep as presented in Figure 4.9. The system value, in the upper orange cell value, is calculated as follows:

$$S_{dao_4}(t = 80) = S_{ini} + B_{dao_1}(t = 80) = S_{ini} + \sum_{q \in Q} w_q V^q_{dao_i}(t = 80)$$

$$S_{dao_4}(t=80) = S_{ini} + 0.5 * V^{RT}_{dao_i}(t=80) + 0.5 * V^{NU}_{dao_i}(t=80)$$

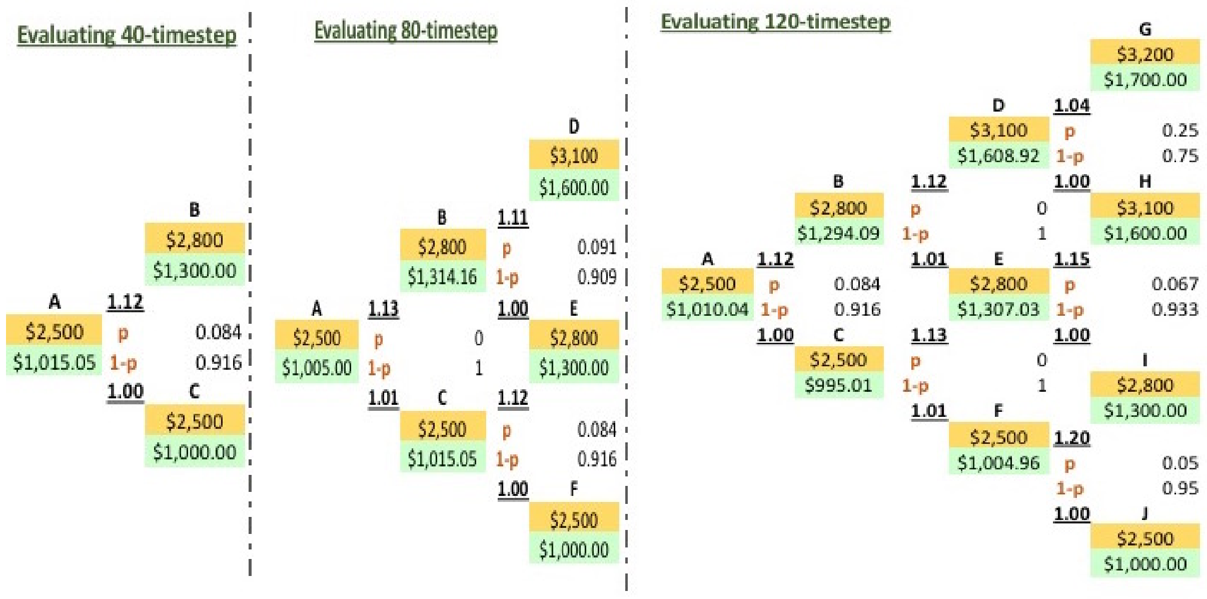
$$= 1,000 + 0.5 * 2,400 + 0.5 * 1,800 = \$3,100$$

The lower green cell value is computed as follows:

$$O_{dao_4}(t=80) = max(0, 3,100 - 1,500) = \$1,600$$

The other system values and options are calculated using the same formulation as in Case 1. Finally, the option value formula O_{dao_4} of dao_4 is:

$$O_{dao_4} = O^{40\text{-}timestep}_{dao_4} + O^{80\text{-}timestep}_{dao_4} + O^{120\text{-}timestep}_{dao_4} = 1,015.05 + 1,005.00 + 1,010.04 = \$3030.09$$



(a) T=40-timestep **(b)** T=80-timestep **(c)** T=120-timestep

Figure 4.9 Binomial Tree of dao_4 for Case 2 (Best Scenario).

From Figure 4.9, we found that the current architect's option would be unable to deal with emerging uncertainties (i.e. lacks future options). It has low option value of \$3,030.09 with an expensive operating cost of \$1500, as compared to other options. Except for dao_7, dao_8, and dao_9, which produced the lowest option and net values (i.e. they will be excluded from selection). Consequently, it is better to abandon dao_4 and switch to a more diversified option. Particularly, it is better to invest in dao_3, dao_5, and dao_6 (because they have the best option values as depicted in Table 4.5), rather than dao_4. By considering the switching

Table 4.5 Option and Net values of diversified architecture options for Response Time and Network Usage concerns (i.e. Case 2 for Best case scenario). *Note that the dao_i with the highest net value is highlighted in gray.*

dao_i	Option Value O_{dao_i}	Net Value NV_{dao_i} (i.e. after deducting switching cost)
1	\$3,633.29	\$3,033.29
2	\$3,746.98	\$3,246.98
3	\$4,952.04	\$3,952.04
4	\$3,030.09	\$3,030.09
5	\$4,494.36	\$4,194.36
6	\$6,583.76	**\$5,683.76**
7	\$481.56	\$-118.44
8	\$34.57	\$-365.43
9	\$183.92	\$-516.08

costs in Table 4.3, the results remain the same, paying \$0 for dao_4 provides a value of (3,030.09-0= \$3,030.09) less than dao_3, dao_5 and dao_6. Even spending \$1,000 for dao_3 (4,952.04-1,000=\$3,952.04), \$1,200 for dao_5 (4,494.36-300=\$4,194.36) and \$900 for dao_6 (6,583.76-900=\$5,683.76) creates more value than with dao_4. Although dao_3 was the winner in the previous scenario, yet it provides lower value than dao_6. This is due to having much lower option value and higher switching cost. This is a case where we did not benefit from diversity of having fixed and mobile devices, due to having lower net value. Conversely, a more diversified option is necessary which may improve the response time and network usage concerns, due to the option of having both cloud and fog for computations and processing. This depicts that diversifying is based on the context and quality attribute to be achieved. However, further evaluation is necessary to validate the significance of real options to diversity in other case studies. As depicted in Table 4.5, dao_7, dao_8, and dao_9 also provided a negative net option value, similar to Case 1. In this context, we recommend excluding these *DAO* from the shortlisted options for run-time evaluation, as their deployment would incur extra unnecessary costs.

Analysis (Worst Case Scenario): Let us consider the case where the architects are not confident enough about their estimations with respect to the *DAO* utilities. This may be due to their little knowledge about the software architectures in that domain and/or the high uncertainty in QoS of such environments (e.g. IoT). In this context, the evaluation would produce uncertain valuation of options. In Table 4.6, we illustrate the option value and net value (i.e. after deducting the switching cost) for the *DAO*, whereas the utility and binomial trees are shown in Appendix B. As depicted in Table 4.6, dao_7, dao_8, and dao_9 will excluded

Table 4.6 Option and Net values of diversified architecture options for Response Time and Network Usage Concerns (i.e. Case 2 for Worst case scenario). *Note that the dao_i with the highest net value is highlighted in gray.*

dao_i	Option Value O_{dao_i}	Net Value NV_{dao_i} (i.e. after deducting switching cost)
1	\$3917.13	\$3317.13
2	\$4193.36	**\$3693.36**
3	\$4346.37	\$3346.37
4	\$1222.00	\$1222.00
5	\$3746.76	\$3446.76
6	\$4493.42	\$3593.42
7	\$33.06	\$-566.94
8	\$174.15	\$-225.85
9	\$339.73	\$-360.27

from implementation, because of their negative net value (i.e. similar to previous cases). Further, dao_4 showed the second lowest net value of \$1222.00. In this context, the architect would better abandon this option and switch to another one. For that, we found that dao_2 provides the best net option value (\$3693.36) and thus will be assigned for deployment at run-time. However, we cannot guarantee that dao_2 will be the best option and can "survive" with dynamic environmental conditions at run-time. This is due to the following: (i) the low confidence of the architects (i.e. hard for them to estimate the utilities); (ii) there is a small difference between the net value of dao_1, dao_2, dao_3, dao_5, and dao_6. To this regard, this calls for run-time evaluation of the concerned *DAO* (i.e. dao_1-dao_6).

4.5.3 Discussion

In this chapter, we aim to answer the following:
RQ2: To what extent design-time architecture evaluation methods that adopt economics-driven principles, such as real options analysis, can systematically evaluate complex design decisions, such as diversification, in the face of operational uncertainties in dynamic environments? To answer this question, we have proposed an approach that makes a novel extension to the CBAM. The method supports reasoning about complex design decisions using real options. In particular, the architect could use the approach to incept the building blocks of the architecture through pre-deployment valuation of decisions in the presence of uncertainty (at design-time). Our approach can work in situations, where the architect has some confidence about the value of the decisions and the likely dynamic

scenarios that can confront the architecture (e.g. Case 1 and Case 2). However, in the absence of data, expertise and/or knowledge of a dynamic environment, which is the case in IoT, the evaluation of architecture decisions tends to be limited. For instance, there are some cases (e.g. Worst Scenario for Case 2) where the architects had low confidence with respect to the candidate architecture options and the way it can benefit from diversification. This could be attributed to the fact that there is no documented guidelines, artefacts, and benchmarks for evaluating diversity in design.

In the experimental evaluation, we have shown how our method can be used to determine how and when diversification can contribute to long-term value (and when it may cease to add value) in the presence of uncertainty. In particular, we emphasised the need for real options to value flexibility in design and how the compound options strategy could illustrate the added value of a diversified architecture option. For example, when the current diversified option cannot accommodate the context changes (i.e. uncertainties), it is advantageous to search for a diversified choice that embeds more options. Our computations also demonstrate that an early exercise of an option may seem to deliver value as in Figure 4.6 and 4.9. However, it is necessary to wait to determine the long-term value, which may differ by time. Moreover, the approach could help the architect to determine the suitable *dao* for a particular context. For instance, when the energy consumption was the concern in Case 1, dao_3 was recommended by the approach (Table 4.4). Whereas when low response time and network usage were priorities in Case 2, the approach showed that the previously recommended dao_3 (less diverse) and other options are not capable for that. Instead dao_6 provided the best option net value (Table 4.5). In this context, we have shown how the method can provide systematic assessment to evaluate diversity in design using value-based reasoning, in the presence of uncertainty. Another potential use for the design-time evaluation approach is shortlisting the candidate architecture options. For example, from Case 1 and Case 2, it is clear that dao_7, dao_8, and dao_9 are not suitable candidates for adoption in both contexts. This was due to their very low net values (i.e. less than zero), which means that they should be eliminated from the list of candidates. In the Worst Scenario for Case 2 (Table 4.6), it was hard for the architect to confirm which one of the *DAO* to deploy at run-time. This is due to the minor difference between *DAO* net value, which happened in accordance to the little expertise knowledge and high fluctuation in QoS in dynamic environments.

To apply this method in practice, there are some questions that need to be answered:

- **Which stakeholders need to participate and in what ways?** Our main objective is providing a long-term economics-driven approach for architecture design and evolution that can be used by software project stakeholders (i.e. architect, software development team, project management team, *etc*). IoT is an emerging paradigm that is currently on its architecting phase and could benefit from the approach. IoT architects backed by fog, cloud computing and sensing knowledge, IoT companies and vendors, governmental and research institutions such as IoT-A consortium [194] need to participate to develop these types of applications. The stakeholders share their knowledge to elicit the application information and the impacts of each architecture decision on quality attributes of interest. For instance, the utility values in Figure 4.5-4.8 manifest on how the stakeholders/market would "feel" about the system, if a particular quality attribute is improved.

- **Which tools must be available?** Software tools to construct the binomial trees are necessary. If the architect wants to implement the *iTransport* IoT application, rather than the general design-time evaluation method, some tools are required. In particular, fixed and mobile things such as smart phones, physical and virtual sensors and other online monitoring tools (e.g. Amazon Cloud Watch [195]) are necessary to stress test the *iTransport* application, which can then aid the architect in estimating the utility trees.
- **Are there specific applications for which this will work best and are there other contexts in which it would be possible or feasible?** This method is generic, which in turn could be applied in architecting and evaluating applications working on non-stationary and unpredictable environments, such as IoT, cloud, grid deployments, microservices and service-oriented architectures.

4.6 Conclusion

In this chapter, we have described an approach, which makes a novel extension of the CBAM. The approach reasons about diversification in software architecture design decisions in dealing with operational uncertainties in IoT using real options. The assumption here is that a diversified architecture has better potential in meeting the requirements through diversifying the architecture design decisions within the architecture. Architects and decision makers can benefit from visualisation of value using binomial real options analysis to assess the extent to which diversification of architecture design decisions can continue to deliver value. The method can be used to inform the architect whether an architecture decision needs to be diversified and what the trade-offs between cost and long term value resulting from diversification are. Real Options analysis could also aid the architect in shortlisting the candidate architecture options for run-time evaluation and deployment. In particular, Case 1 and 2 highlighted the suitable options (e.g. dao_1-dao_6) and others (e.g. dao_7-dao_9 with negative option values) that need to be eliminated. However, real options analysis is a static method for economic modelling and forecasting of architecture decisions. This type of design-time evaluation relies on human experts, who can only *partially* anticipate the fitness and the extent to which an (diversified) architecture can cope with operational uncertainties, unanticipated usage scenarios, and emergent behaviours. It is based on assumptions (i.e. experts knowledge) which may (not) be valid in dynamic environments, such as IoT. For instance, there are some cases (e.g. Worst Case scenario for Case 2) where the architects had low confidence with respect to the candidate architecture options. This calls for a systematic continuous evaluation method which intertwines design-time and run-time evaluation to provide more informed assessment of design options and capture deviations from the design-time evaluation for technical and value potentials (addressed in Chapter 5).

Chapter 5

Reactive Run-time evaluation

Context. Run-time properties of modern software system environments, such as IoT, are a challenge for existing software architecture evaluation methods. Such systems are largely data-driven, characterised by their dynamism, unpredictability in operation, hyper-connectivity, and scale. Properties, such as performance, delayed delivery, and scalability, are acknowledged to pose great risk and are difficult to evaluate at design-time only. Economics-driven design-time evaluation is necessary as an initial step in a continuous evaluation framework to determine the long-term value of architecture options and hence shortlist the candidate options for next step evaluation. Run-time evaluation could potentially be used to complement design-time evaluation, enabling significant deviations from the expected performance values to be captured.
Objective. This chapter proposes a *novel reactive run-time architecture evaluation method* suited for systems that exhibit uncertainty and dynamism in their operation.
Method. Our method uses machine learning and cost-benefit analysis at run-time to continuously profile the architecture decisions made, to assess their added value. We demonstrate the applicability and effectiveness of this approach in the context of an IoT system architecture, where some architecture design decisions were diversified to meet QoS requirements.
Results. Our method showed its efficiency in automatically evaluating the cost-benefit trade-offs of architecture options, detecting significant deviations in QoS, and selecting the architecture option with most balanced cost-benefit trade-off. It also provides the architect with the flexibility of tuning the approach's parameter (e.g. the relative importance of data, stability, monitoring intervals, *etc*) to best benefit from the evaluation. Finally, our experiments show that it outperformed the baseline and state-of-the-art approaches, in terms of selection of most balanced architecture options.
Conclusion. Our approach could potentially assist architects in evaluating design-time de-

cisions at run-time, which could then inform deployment, refinement, and/or phasing-out decisions.

5.1 Introduction

Intertwining and interleaving run-time evaluation with design-time has the potential to change ad hoc and "trial and error" practices for architecting complex, scalable, and dynamic software systems. Consider, for example the case where architects embed *design diversity* [15] into their solutions in an attempt to meet quality requirements under uncertainty and to mitigate risks and Service Level Agreement violations. Diversification [15] encompasses design decisions and architecture tactics that can be used to adapt the system to unforeseen changes [16]. For a given concern, architecture diversity "spices" the architecture with a variety of design decisions and strategies (e.g. the choice of data collection and processing strategies and tools), which can better cope with uncertainties at run-time. Diversified design decisions, whether planned or accidental [17], can be expensive; their behaviour and value can be best evaluated at run-time. However, there are no systematic continuous software architecture evaluation methods that intertwine design-time and run-time evaluation.

This chapter addresses this gap by proposing a *novel run-time architecture evaluation method* suited for systems that exhibit uncertainty and dynamism in their operation. In this context, this chapter aims to answer the third research question introduced in Chapter 1: **RQ3: How can run-time architecture evaluation using cost-benefit analysis and machine learning techniques complement the design-time one to handle uncertainty?**

Our method uses machine learning and cost-benefit analysis at run-time to continuously profile architecture decisions for their added value, identify significant deviations from previously expected benefits, and automatically determine which architecture options have the optimal cost-benefit trade-off. As such, it can inform deployment, refinement and/or phasing out architectural decisions.

Our method makes the following contributions to the research literature:

- a run-time method for tracking the benefits of architecture design decisions using machine learning;
- a method inspired by multi-objective optimisation to evaluate the cost-benefit trade-offs of architecture decisions at run-time.

The remainder of this chapter is structured as follows: Section 5.2 illustrates the necessary background to understand the approach. Section 5.3 then discusses the research questions which the approach aims to answer. Section 5.4 presents the proposed approach. Section 5.5 explains the experimental settings and the use of the iFogSim tool. Section 5.6 reports on evaluation of the research questions, Section 5.7 presents further analysis of the proposed approach. Section 5.8 discusses the potential of the approach in providing extra insights to the architects and stakeholders, as well as the limitations.

5.2 Background

In this section, we will provide the background necessary to understand the run-time evaluation approach.

5.2.1 Reinforcement Learning

Reinforcement learning is an agent-oriented approach that learns using observed rewards, by mapping situations to actions as an attempt to maximise a scalar reward signal [196]. Generally, it is a sequential decision-making process, where in every step, the agent chooses an action and the reward is computed. It aims at optimising the rewards to get the best possible outcome. Further, there is no supervisor, hence the reward signal is the only metric for the determination of the right action to take. Moreover, reinforcement learning is a trial and error learning process, where agents discover appropriate policies with acceptable reward from the experiences of its environment. The major challenge of this type of learning is the trade-off between exploration and exploitation [196]. The former indicates the examination of non-optimal actions, which may provide better reward in the future. The latter denotes implementing the optimal known decision to maximise the reward. For further elaboration, the exploitation process may result in missing an optimal decision, which may yield better cumulative future reward rather than the present best one. Whereas the cost of exploration may exceed its benefits in some cases.

5.2.2 Time-Decayed Function

In [27], an exponential smoothing function was used as a time-decayed function to handle the challenges of online class imbalance learning. The proffered function is similar to reinforcement learning, which tracks the rewards resulting from actions via the occurrence probability (percentage) of examples belonging to a particular class. It is different from the traditional way of considering all observed examples equally, instead they are updated incrementally by using a time decay (forgetting) factor to emphasise the current status of data and to weaken the effect of old data. In particular, the adoption of a time-decay function is advantageous, due to the following [197, 198]: (i) it weakens the effect of old data; (ii) it is very east to compute; and (iii) minimum data is necessary. In this context, our approach leverages a time-decayed function to provide the architect with flexibility of tuning the relative importance of past versus recent observations, when valuing the benefit of diversified architecture options (further information on how time-decay function is used in our approach is illustrated in Section 5.4.4).

5.2.3 Change Detection Approaches

Generally, in machine learning, the notion of concept drift denotes a change in the underlying distribution of the data, which the approach is learning over time [5]. Examples of application include making inferences based on financial data, energy demand and climate data analysis, web usage or sensor network monitoring, and so forth [30]. Non-stationary and uncertain environments are challenged by their rapid changes (i.e. drifts). Therefore, change detection approaches are necessary to discover these concept drifts and hence perform the corresponding actions. In [30], the change detection approaches are classified into the following families: Hypothesis Tests (HT), Change-Point Methods (CPM), Sequential Hypothesis Tests (SHT), and Change Detection Tests (CDT). The latter approaches determine variations in the underlying data through statistical methods (e.g. sample mean, variance, *etc*), but they differ on how data is processed [30].

HT and CPM operate on fixed-length observations, whereas SHT and CDT sequentially investigate the incoming observations. Therefore, HT and CPM are not suitable for applications operating in an online manner, due to the high complexity with respect to analysing all observations at once. Though SHT is partially suitable for online observations, it has the following drawback: SHT examines the incoming observations until it has enough statistical confidence for decision-making (i.e. a "drift" or "no drift" is detected), but after the decision is made it stops processing the observations. This is a common pitfall in this type of sequential analysis, since the main objective is to continually collect information to ensure true change detection. The CDT attempted to overcome the prior limitations, since they are continuously tracking the observations (i.e. fully sequential manner), which in turn provide reduced computational complexity. To this end, our approach adopts the change detection test proposed by [5], but for a different purpose than that of [5].

In [5], the authors were interested in a supervised learning approach able to make accurate predictions in non-stationary environments. In this case, their approach first attempts to

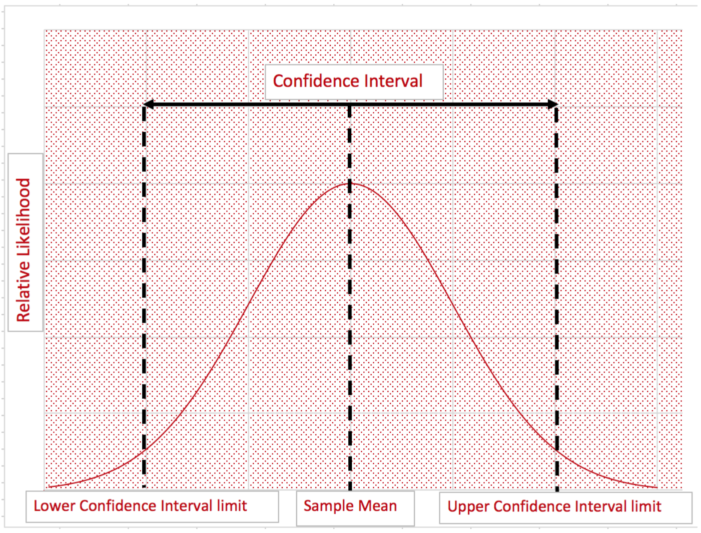

Confidence Interval
Relative Likelihood
Lower Confidence Interval limit
Sample Mean
Upper Confidence Interval limit

Figure 5.1 A representation for the confidence interval statistical technique used by [5] as the change detection test.

make a prediction, and then gains access to the true outcome from a supervisor. The average classification error of the predictions is then computed and monitored over time to detect changes in the probability distribution of the data. Changes are detected based on confidence intervals of the classification error average, as depicted in Figure 5.1. In particular, if the classification error is within the boundaries of the confidence interval (i.e. the upper and lower limits of the interval) then we are confident that there is no change, otherwise a change is detected. The upper and lower limits of interval are adjusted based on the level of confidence required (further explanation is found in Section 5.4.5). Our approach, on the other hand, can be understood in the framework of reinforcement learning, where there is no supervisor. We use the CDT to detect changes in the benefit of software architectures over time, rather than changes in the classification error as done in [5].

5.2.4 Multi-Objective optimisation

When evaluating software architectures, some of the evaluation approaches appeal to multi-objective optimisation (MOP) [135]. MOP is not trivial as the process involves optimising multiple conflicting objectives. For this purpose, multi-objective evolutionary algorithms have been widely used, such as Non-dominated Sorting Genetic Algorithm-II (NSGA-II) [29]. NSGA-II relies on the concept of non-dominance to decide which solutions are better. A non-dominated solution is a solution that is similar or better on all objectives, and strictly better in at least one objective [199]. Non-dominated solutions can thus be seen as better

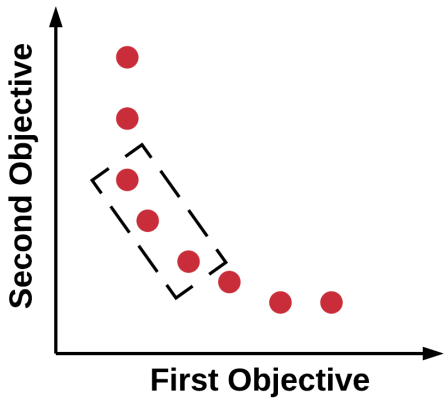


Figure 5.2 An illustrative example of Pareto front and application of the knee point strategy. In the example, the knee points inside the dotted box have better chances to win (i.e. provide a more balanced trade-off between the two objectives).

solutions than dominated ones. NSGA-II thus searches for the set of solutions that are non-dominated by any other solution, which can be referred to as a Pareto front. Figure 5.2 shows an illustrative example of a Pareto front, where both the first and second objectives are to be maximised. NSGA-II then relies on humans to decide which of the solutions from the Pareto front to adopt for a given problem. However, this choice may not be easy.

Some MOP scenarios use a *knee point* strategy on their Pareto fronts (Figure 5.2) to help choosing a solution. A knee point is "almost always the most preferred solution, since it requires an unfavourably large sacrifice in one objective to gain a small amount in the other objective" [200]. In particular, as in Figure 5.2, moving in any direction out of the dotted box may generate a small improvement in one objective, but with a large deterioration in the other objective. Therefore, the knee point strategy promises to find the most balanced decision. Our approach adopted a knee point strategy inspired by NSGA-II for MOP. Further details on how this strategy is used in our approach are given in Section 5.4.6.

5.3 Research Questions

Deciding on which diversified architecture options to implement is not straightforward due to the uncertainties related to the dynamicity and the unbounded scalability of the things in composition, as exemplified in Chapter 3. Our approach thus leverages design-time and run-time knowledge to embrace uncertainties. Based on design-time knowledge, the architects can decide on the options to be implemented. We use CBAM to evaluate different architecture options potentials at design-time [14] (Chapter 4), where the architecture options providing high option value are considered for diversification of the IoT Case study (Chapter 3). However, as the environment is dynamic, value potentials can fluctuate at run-time. This leads to the following questions, which our chapter aims to answer:

RQ3.1 ***How to evaluate the benefit of each*** *dao* ***over time?*** A key requirement for determining the added value of a *dao* is through tracking its benefit over time based on its quality attributes. In non-dynamic environments, tracking the benefit based on a simple average of its value at each time *t* could be sufficient. However, in dynamic environments, simple average may take a long time to reflect changes [27] in benefit. A method to enable tracking the current benefit over time is necessary. It is essential to note that we have found that a noticeable reduction in the price volatility of resources (e.g. storage and processing) has begun from December 2017 [201]. In this context, when the price volatility is so low, the use of exponential decay factor for cost is not necessary. Therefore, we have provided a continuous measure only for the benefit of a *dao* over time.

RQ3.2 ***How can run-time evaluation determine changes in*** *dao's* ***value over time and inform subsequent decisions?*** To properly support decision-making, a run-time architecture evaluation approach should be able to identify when changes in the benefit of a *dao* are truly significant. A decision to change the software architecture based on an insignificant change in benefit would lead to an unstable system. Moreover, when a significant change is detected, run-time evaluation should be able to identify which *dao* provides a better trade-off between benefit and cost. Therefore, a method to detect both significant changes and balanced trade-offs is desired.

To answer these questions, we propose a run-time evaluation approach inspired by self-adaptive systems. The approach is able to profile situations where options can be more effective and provide continuous updates on their value potentials. Specifically, to answer ***RQ3.1***, our approach adopts an *exponential time-decay function* inspired by reinforcement learning [27]. This function enables us to track the current benefit of a *dao* by weakening the effect of old data (i.e. emphasise on the recent versus past observations). To answer ***RQ3.2***, our proposed approach adopts *change detection tests* to check whether the benefit of the *dao* currently being used is getting significantly worse [5]. If it is significantly worse, a method inspired by the multi-objective optimisation literature [28, 29] is adopted to identify the *dao* with the optimal trade-off between cost and benefit. Based on the profiling of options over time, it is also possible to determine which ones are not fit for the purpose, and hence could be phased out.

The steps of the approach are summarised in Figure 5.3. Here, the design-time evaluation is the one proposed in Chapter 4, which uses options theory (i.e. an economics-driven approach) [176] to evaluate the diversified architectural options. The run-time evaluation is our proposed approach in this chapter, which is discussed in Section 5.4.4, 5.4.5, and 5.4.6.

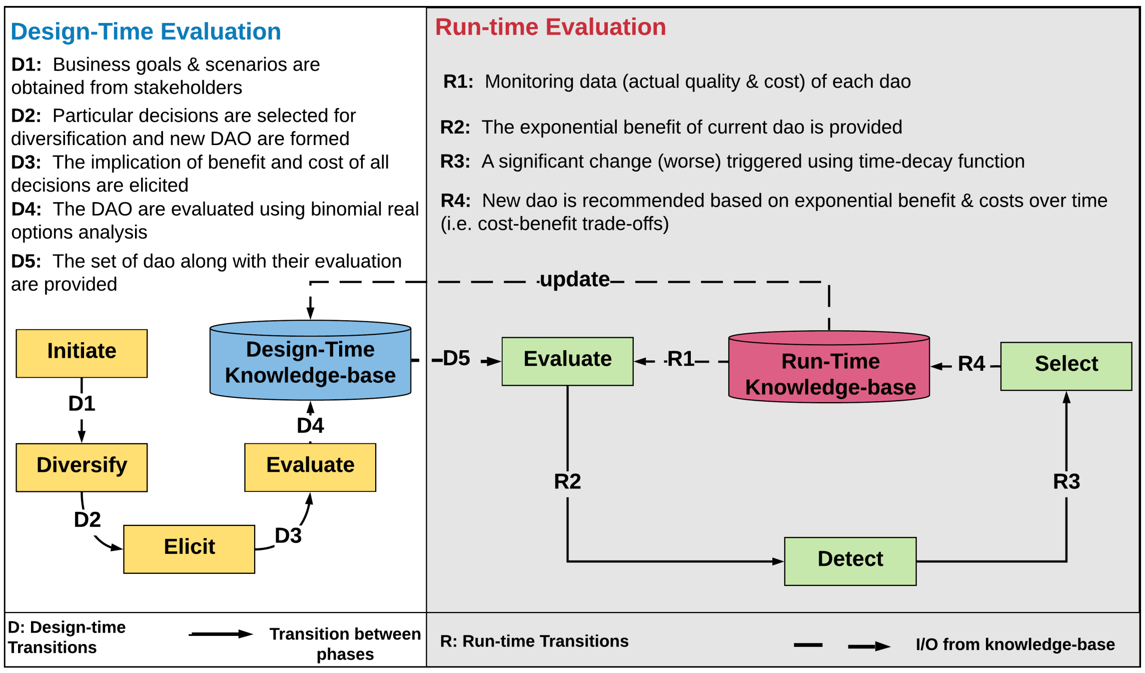

Design-Time Evaluation
D1: Business goals & scenarios are obtained from stakeholders
D2: Particular decisions are selected for diversification and new DAO are formed
D3: The implication of benefit and cost of all decisions are elicited
D4: The DAO are evaluated using binomial real options analysis
D5: The set of dao along with their evaluation are provided
Initiate
D1
Diversify
D2
Elicit
D3
Evaluate
D4
Design-Time Knowledge-base
update
D5
Run-time Evaluation
R1: Monitoring data (actual quality & cost) of each dao
R2: The exponential benefit of current dao is provided
R3: A significant change (worse) triggered using time-decay function
R4: New dao is recommended based on exponential benefit & costs over time (i.e. cost-benefit trade-offs)
Evaluate
R1
Run-Time Knowledge-base
R4
Select
R2
R3
Detect
D: Design-time Transitions
Transition between phases
R: Run-time Transitions
I/O from knowledge-base

Figure 5.3 Steps of the approach, where the design-time evaluation (Chapter 4) forms the initial design decisions and run-time evaluation complements it.

5.4 Proposed Approach

This section explains our proposed approach. Sections 5.4.4 explains how our approach addresses RQ3.1. Sections 5.4.5 and 5.4.6 explain how our approach addresses RQ3.2.

5.4.1 Diversified Architecture Options (DAO)

Design diversity [15, 33, 167, 16, 14] is used to design for dependability under uncertainty: the greater the uncertainty, the more diversity the architects may need to apply to improve QoS (e.g. performance, energy consumption, *etc*). We denote a software architecture which

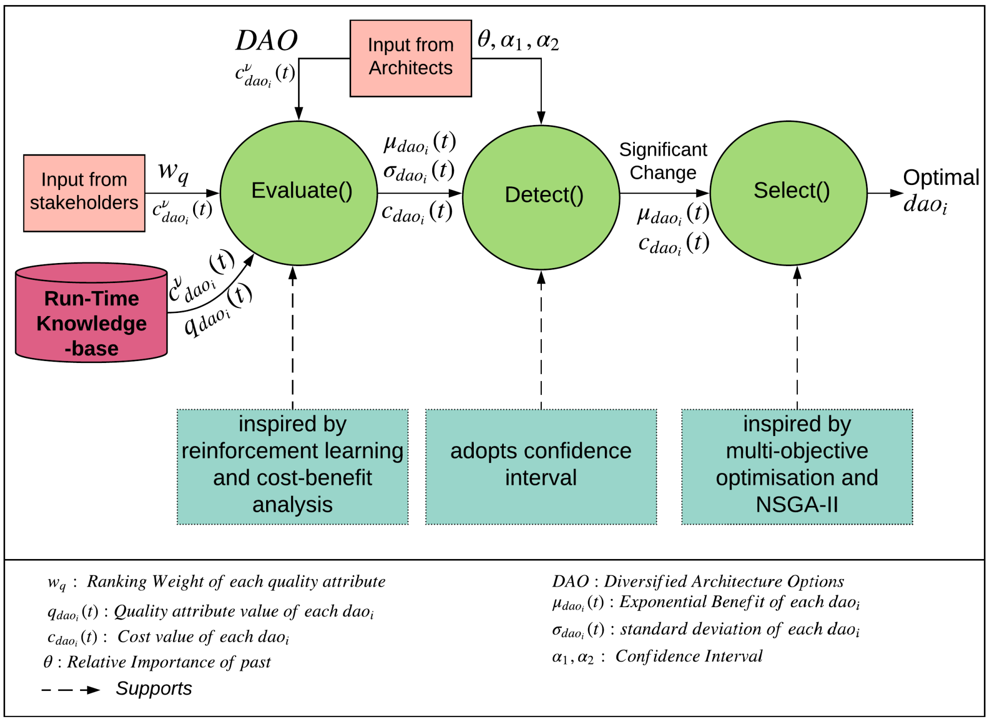

DAO
$c^{\nu}_{dao_i}(t)$
Input from Architects
$\theta, \alpha_1, \alpha_2$
Input from stakeholders
w_q
$c^{\nu}_{dao_i}(t)$
Evaluate()
$\mu_{dao_i}(t)$
$\sigma_{dao_i}(t)$
$c_{dao_i}(t)$
Detect()
Significant Change
$\mu_{dao_i}(t)$
$c_{dao_i}(t)$
Select()
Optimal dao_i
Run-Time Knowledge -base
$c^{\nu}_{dao_i}(t)$
$q_{dao_i}(t)$
inspired by reinforcement learning and cost-benefit analysis
adopts confidence interval
inspired by multi-objective optimisation and NSGA-II
w_q : Ranking Weight of each quality attribute
$q_{dao_i}(t)$: Quality attribute value of each dao_i
$c_{dao_i}(t)$: Cost value of each dao_i
θ : Relative Importance of past
Supports
DAO : Diversified Architecture Options
$\mu_{dao_i}(t)$: Exponential Benefit of each dao_i
$\sigma_{dao_i}(t)$: standard deviation of each dao_i
α_1, α_2 : Confidence Interval

Figure 5.4 An overview of the operational procedures of run-time evaluation approach.

embeds diversity as diversified architecture option *dao* and the set of *dao* as *DAO*. A *dao* implements a set of diversified decisions to meet some quality goals and trade-offs. Consider a set of architecture decisions D, where a decision $d_{ka} \in D$; k denotes a particular capability, including connectivity, data collection, data management, *etc*; and a indicates the software architecture component and connection that implements this capability k. For example, in an IoT system, architecture decisions for the capability of data collection d_1 could be performed either through fixed d_{11}, mobile d_{12}, or fixed+mobile sensors d_{13}. Another example is data processing could be performed either in cloud d_{21}, or fog+cloud d_{22}. Therefore, dao_i could collect data from fixed and mobile sensors (d_{13}) and processes it in cloud-fog (d_{23}). Other examples of *DAO* are depicted in Chapter 3 through Table 3.1. In the IoT system case, the diversity in each *dao* can refer to using fixed and/or mobile fog devices for data collection capability; using different cloud providers and heterogeneous fog devices for data processing capability.

5.4.2 The Proposed Approach In The Context Reinforcement Learning

Given a set of possible diversified architecture options (i.e. states), the current state of the system denotes the use of current *dao*. The proposed approach is learning how to recognise the *dao* with the most balanced cost-benefit trade-off at a given moment in time. This could be translated as an "action" to take. In particular, the approach chooses the action (i.e. most balanced *dao*) with respect to feedback (i.e. rewards) from the environment. This feedback is captured from the observed quality attributes over time (i.e. benefit) and cost, through a time-decayed function that comes from reinforcement learning [196]. In this sense, our approach can be understood in the framework of reinforcement learning approaches.

However, from the context of software architecture evaluation, we need to do the following: (1) perform reinforcement learning in an online way where bad actions chosen during, e.g., early stages of the learning process, can have serious inappropriate consequences to the architecture evaluation; and (2) consider multiple goals, rather than a single one. Therefore, we cannot adopt existing reinforcement learning approaches out of the box.

To deal with (1), we adopt simulators and/or monitor the diversified architecture options in parallel. We also then employ a pre-defined rule to decide which action to take (i.e. which *dao* to choose) based on the modelled rewards. This rule was designed to minimise the chances of making poor architecture recommendations. To handle (2), we get inspiration from the MOP literature [28]. In our case, the suggestion of which *dao* to adopt is performed in an innovative way, based on non-dominated solutions (a key concept adopted by many MOP algorithms), knee point strategy, and the marginal loss (Section 5.4.6).

In a nut shell, our approach performs *optimisation guided by machine learning strategies (i.e. strategies derived from reinforcement learning)*, where a decay function is adopted that continually learns and updates the aggregated benefit of monitored quality values forming the exponential benefit. It then selects the balanced (knee) diversified architecture options when needed based on the learned exponential benefits.

5.4.3 Stakeholder Involvement versus Automated Management

As seen in Figure 5.5, the stakeholders first specify the business goals, scenarios and diversified architecture options along with their anticipated utilities (i.e. manual evaluation). After that these options are evaluated using binomial real options analysis (i.e. semi-automated evaluation). The run-time evaluation complement the design-time decisions by assessing the

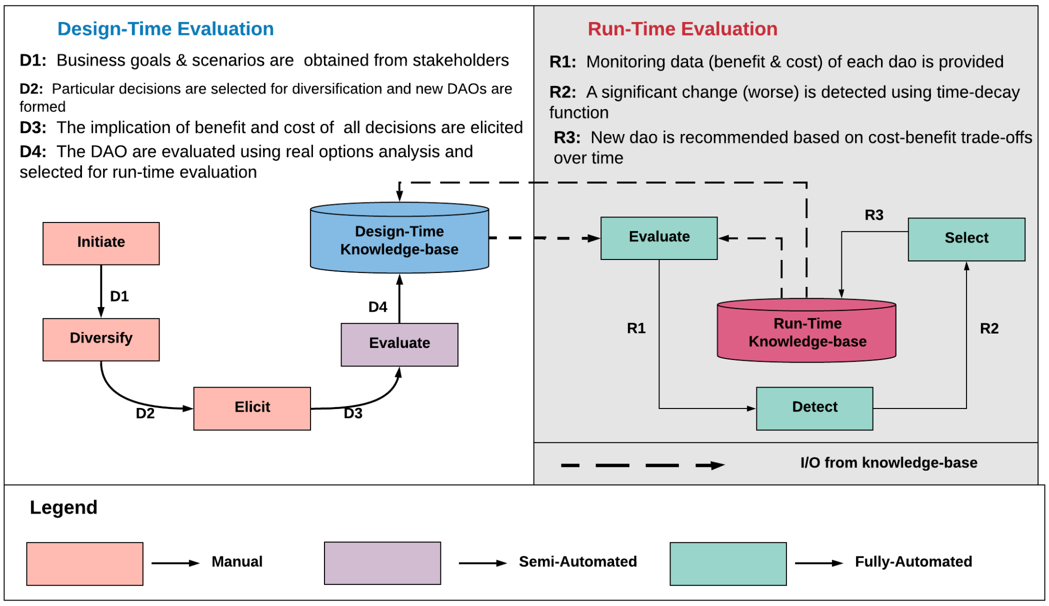

Design-Time Evaluation
D1: Business goals & scenarios are obtained from stakeholders
D2: Particular decisions are selected for diversification and new DAOs are formed
D3: The implication of benefit and cost of all decisions are elicited
D4: The DAO are evaluated using real options analysis and selected for run-time evaluation
Run-Time Evaluation
R1: Monitoring data (benefit & cost) of each dao is provided
R2: A significant change (worse) is detected using time-decay function
R3: New dao is recommended based on cost-benefit trade-offs over time
Initiate
D1
Diversify
D2
Elicit
D3
Evaluate
D4
Design-Time Knowledge-base
Evaluate
R1
Detect
R2
Select
R3
Run-Time Knowledge-base
I/O from knowledge-base
Legend
Manual
Semi-Automated
Fully-Automated

Figure 5.5 An overview of the level of stakeholder involvement and automated management in the design-time and run-time evaluation approach.

behaviour of architecture options over time, checking if it requires replacement and finally recommends the most balanced *dao* instead (i.e. fully-automated evaluation).

5.4.4 Evaluating The Diversified Architecture Options

The goal of the evaluation is to *determine what are the diversified architecture options, the quality attributes of interest and how they will react over time*. This step is divided into further sub-steps as explained below and summarised in Algorithm 2.

1. **Identifying the diversified architecture options and quality attributes of interest:** This step is inspired by the formulation of CBAM [9, 23], except for the fact that the benefit and cost of options vary over time. It also explicitly considers that diversified architecture options are composed of architecture decisions, which are defined by the architects (Figure 5.4). In particular, the following must be specified:

   - The set K of capabilities selected for diversification. Examples of capabilities are data collection, connectivity, processing, routing topology, and data management.
   - The set D of architecture decisions. An architecture decision $d_{ka} \in D$ specifies an architecture component a to implement a given capability $k \in K$.

- Set of diversified architecture options DAO, where $|DAO|$ represents the number of DAO and each $dao_i \in DAO$ is composed of a set of architecture decisions.
- The set Q of qualities of interest. Response time and energy consumption are examples of quality attributes of interest.
- Each d_{ka} has a cost and quality which may vary over time. The cost of each architecture decision is $c_{ka}(t)$ and quality is $q_{ka}(t)$, where ka identifies a decision $d_{ka} \in D$, c is a measure of cost, $q \in Q$ is a measure of quality and t is a time stamp. In particular, each architecture decision d_{ka} will be associated to one measure of cost and $|Q|$ measures of quality at each time stamp.

2. **Assessing the weight of each quality of interest**: A ranking weight (w_q) must be chosen by the stakeholders to reflect the relative importance of each quality attribute to the run-time benefit of the system as depicted in Figure 5.4, which should satisfy equation 5.1:
$$\sum w_q = 1; \forall q : w_q \geqslant 0 \tag{5.1}$$

3. **Quantifying the benefit of** DAO **over time:** The benefit $B_{dao_i}(t)$ of dao_i at timestamp t is a measure of the contribution of each quality attribute to value creation, i.e., added value. In other words, each $B_{dao_i}(t)$ is a function of $q_{dao_i}(t)$, $\forall q \in Q$, whereas each $q_{dao_i}(t)$ is a composite of the quality $q_{ka}(t)$ of each of its component architecture decisions $d_{ka} \in dao_i$. To compute $q_{dao_i}(t)$, the qualities of the architecture decisions need to be aggregated based on how they are connected to each other (line 3). Table 5.1 depicts some aggregate functions for QoS. Note that, the approach could work on any quality attribute, such as security, response time, throughput, *etc*, based on the application context.

   We monitor the benefits of *dao* as a whole. The modelling for qualities of *dao* follows linear aggregation and is consistent with online QoS modelling approaches that have been widely adopted in the service computing community (e.g., [202, 203]). Though the use of linear aggregation for qualities is the widely adopted practice in service community, it is acknowledged to be limited when capturing dependencies of decisions affecting qualities. Additional online aggregation functions, for capturing dependencies of decisions affecting qualities, is a non-trivial problem; its solution will constitute a significant contribution to both the software architecture and services community, which is worth separate reporting due to the complexity of its treatment.

Each $q_{dao_i}(t)$ has one constraint, which follows one of the following possible formats: $q_{dao_i}(t) \leqslant q^{max}$, $q_{dao_i}(t) \geqslant q^{min}$, $q^{min} \leqslant q_{dao_i}(t) \leqslant q^{max}$.

To place all quality attributes in the same scale (line 4-9), equation 5.2 is for scaling of negative quality values (i.e. the lower the better, e.g. response time), whereas equation 5.3 could be used for scaling positive quality values (i.e. the higher the better, e.g. throughput). $q'_{dao_i}(t)$ denotes the normalised value of a given q of a particular dao_i at time t.

$$q'_{dao_i}(t) = \begin{cases} \frac{q^{max} - q_{dao_i}(t)}{q^{max} - q^{min}} & if q^{max} - q^{min} \neq 0 \\ 1 & if q^{max} - q^{min} = 0 \end{cases} \tag{5.2}$$

$$q'_{dao_i}(t) = \begin{cases} \frac{q_{dao_i}(t) - q^{min}}{q^{max} - q^{min}} & if q^{max} - q^{min} \neq 0 \\ 1 & if q^{max} - q^{min} = 0 \end{cases} \tag{5.3}$$

Table 5.1 Aggregate Functions. The Max, Min and $\sum$ operations are over all architecture decisions that are connected to each other in the specified way (parallel or sequence).

QoS Attribute	Parallel	Sequence
Response Time	$Max(q_{ka})$	$\sum q_{ka}$
Energy Consumption	$\sum q_{ka}$	$\sum q_{ka}$
Cost	$\sum c_{ka}$	$\sum c_{ka}$

Therefore, the benefit of a diversified architecture option can be computed using equation 5.4, if none of its quality attributes violates any constraint (line 12). If a dao at time t, violates any constraint (line 10), its benefit is set to zero (line 11).

$$B_{dao_i}(t) = \sum_{q \in Q} w_q * q'_{dao_i}(t) \tag{5.4}$$

This step uses the run-time knowledge (i.e. observed QoS) to compute the benefit of each dao, which is then used as input to step 5 (Figure 5.4).

4. **Quantifying the cost of *DAO*:** CBAM [23] extends ATAM (Architecture Trade-off Analysis method) [2] with explicit focus on the costs and benefits of the architecture decisions in meeting scenarios related to quality attributes. Our consideration for the cost is similar to what is mentioned previously in Chapter 4. For example, the cost can

relate to one or more dimensions of interest. This can include the cost of configuration, deployment, testing, leasing, execution, *etc*. We estimate the costs using parametric models, experts' judgement (i.e. architects and other stakeholders) [110, 188], as well as run-time knowledge (i.e. monitoring tools). Unlike CBAM, our approach considers the switching costs between options, which could include the configuration, license cost, *etc*. This is in addition to the operating costs, such as costs of deploying and maintaining the *dao*. The cost associated with an architecture option at time t is denoted by $c_{dao_i}(t)$ (line 13), which is computed using equation 5.5 (Figure 5.4).

$$c_{dao_i}(t) = \sum c^{v}_{dao_i}(t) \tag{5.5}$$

where v: considers a variety of costs (e.g. deployment cost, leasing cost, *etc*). Further, the approach receives monetary values for cost, which could then be normalised in the same way as negative quality values in equation 5.2.

5. **Determine the exponential benefit of** ***DAO*** **over time:** The benefit of a given option is monitored using a time-decay function inspired by reinforcement learning [27]. This function enables the architect to continuously learn the current benefit of a given *dao* over time. At a given timestep t, the exponential benefit $\mu_{dao_i}(t)$ of a *dao* is the time-decayed average of its benefit, incrementally computed based on all timesteps up to t (line 14), as seen in equation 5.6. This average allows us to tune the importance given to the present and past observations of the benefit and to learn more about the behaviour of *DAO* over time. How much emphasis is given to the present/past is controlled by a pre-defined parameter θ. In particular, the relative importance of the present is denoted by $1-\theta$, whereas of the past by θ, where $0 \leq \theta < 1$. The θ parameter affects the system's stability. More specifically, a high θ (e.g., larger than 0.95) corresponds to a greater emphasis on the historical observations of benefit, leading to more robustness to noise, making the quantification of benefit more stable, but slower at adapting to changes. Conversely, smaller values give more emphasis to the more recent observations of benefit and swifter adaptation to changes. However, too low values can lead to unstable quantification of benefit (due to higher sensitivity to noise). To this regard, the architect has the freedom to adjust the θ parameter, with respect to the required system stability (Figure 5.4).

$$\mu_{dao_i}(t) = \theta \mu_{dao_i}(t-1) + (1-\theta) B_{dao_i}(t) \tag{5.6}$$

ALGORITHM 2: Evaluate()

Input: diversified architecture options DAO, number of DAO ($|DAO|$), quality $q_{dao_i}(t)$, number of quality attributes $|Q|$, weight of each quality attribute w_q, relative importance of the past θ

Output: exponential benefit $\mu_{dao_i}(t)$, standard deviation $\sigma_{dao_i}(t)$, cost $c_{dao_i}(t)$

```
1  for i = 1 to |DAO| do
2      for $q = 1 : |Q|$ do
3          Compute quality $q_{dao_i}(t)$ based on Table 5.1
           {For negative quality attribute (i.e. the lower the better, e.g. Response Time):}
4          if $q_{dao_i}(t)$ is a negative quality & $q^{max} - q^{min} \neq 0$ then
5              $q'_{dao_i}(t) = \frac{q^{max} - q_{dao_i}(t)}{q^{max} - q^{min}}$
           {For positive quality attribute (i.e. the higher the better, e.g. Throughput):}
6          else if $q_{dao_i}(t)$ is a positive quality & $q^{max} - q^{min} \neq 0$ then
7              $q'_{dao_i}(t) = \frac{q_{dao_i}(t) - q^{min}}{q^{max} - q^{min}}$
8          else
9              $q'_{dao_i}(t) = 1$
       end
       {Check constraints' violation:}
10     if any $q'_{dao_i}(t)$ violates constraints then
11         $B_{dao_i}(t) = 0$
       else
12         Compute benefit $B_{dao_i}(t) = \sum_{q \in Q} w_q * q'_{dao_i}(t)$
       end
13     Quantify cost $c_{dao_i}(t) = \sum c^{v}_{dao_i}(t)$
14     Compute exponential benefit $\mu_{dao_i}(t) = \theta\mu_{dao_i}(t-1) + (1-\theta)B_{dao_i}(t)$
15     Determine variance $\sigma^2_{dao_i}(t) = \theta\sigma^2_{dao_i}(t-1) + (1-\theta) * (B_{dao_i}(t) - \mu_{dao_i}(t))^2$
16     Determine standard deviation $\sigma_{dao_i}(t) = \sqrt{\sigma^2_{dao_i}(t)}$
   end
```

6. **Determine the variance and standard deviation of *dao* over time:** The time-decayed variance $\sigma^2_{dao_i}(t)$ of the benefit of this option is computed using equation 5.7 (line 15). After that, the standard deviation $\sigma_{dao_i}(t)$ is computed as the square root of variance as depicted in equation 5.8 (line 16).

$$\sigma^2_{dao_i}(t) = \theta\sigma^2_{dao_i}(t-1) + (1-\theta) * (B_{dao_i}(t) - \mu_{dao_i}(t))^2 \tag{5.7}$$

$$\sigma_{dao_i}(t) = \sqrt{\sigma^2_{dao_i}(t)} \tag{5.8}$$

5.4.5 Detecting Significant Changes Over Time

At every timestep t, the detect module triggers an alert if there is a significant change for the worse on the benefit $B_{dao_i}(t)$. If such a detrimental change is detected, then it may be necessary to switch from the current *dao* to another better *dao* for the current circumstances of the software system. The steps for detecting significant detrimental changes are shown in Algorithm 3 and are explained below.

To detect changes, we use the confidence interval of the maximum exponential benefit seen so far and its corresponding exponential standard deviation, as shown in equation 5.9. In particular, two variables computed based on the evaluate phase are used: $\mu^{max}_{dao_i}$ is the maximum exponential benefit seen so far, and $\sigma^{max}_{dao_i}$ is its corresponding standard deviation. Whenever a new reading arrives at time t, those values are updated if $\mu_{dao_i}(t) > \mu^{max}_{dao_i}$ (line 2-4). The parameter α is a parameter that affects the confidence level [5]. In particular, confidence levels 95% and 99% correspond to α = 1.96 and 2.58, respectively.

$$[\mu^{max}_{dao_i} - \alpha\sigma^{max}_{dao_i}, \mu^{max}_{dao_i} + \alpha\sigma^{max}_{dao_i}] \tag{5.9}$$

If the current exponential benefit $\mu_{dao_i}(t)$ is outside the left boundary of the confidence interval (line 5), a significantly detrimental change is detected (line 6). This leads to the insight that the current $dao's$ benefit is getting worse and may need to be replaced.

Replacements may affect the system's stability. The parameter α enables us to tune the sensitivity of the approach to changes, and therefore how stable/unstable it will be over time. For instance, consider that the architect has chosen the confidence level of 95%. Then, if the current exponential benefit is outside the left boundary of the 95% confidence interval ($\mu_{dao_i}(t) - \sigma_{dao_i}(t) \leq \mu^{max}_{dao_i} - 1.96 * \sigma^{max}_{dao_i}$), a significantly detrimental change is informed to the software architects. However, if a 99% confidence interval has been chosen, a significantly detrimental change would only be informed when $\mu_{dao_i}(t) - \sigma_{dao_i}(t) \leq \mu^{max}_{dao_i} - 2.58 * \sigma^{max}_{dao_i}$.

In this context, the detect step uses $\mu_{dao_i}(t)$ and $\sigma_{dao_i}(t)$ along with α parameters (set by the architect) to detect changes, as shown in Figure 5.4.

When a significantly detrimental change is detected, $\mu_{dao_i}^{max}$ and $\sigma_{dao_i}^{max}$ are reset and re-computed from scratch using the values of the exponential benefit accumulated since a warning was triggered (line 7). A warning is triggered based on a more relaxed confidence interval (line 8-9). For example, if using 99% as the confidence interval to detect significant detrimental changes, a confidence level of 95% could be used to issue a warning. Otherwise, no change/warning is detected (line 10).

ALGORITHM 3: Detect()

```
Input: confidence intervals ($\alpha_1$, $\alpha_2$), exponential benefit $\mu_{dao_i}(t)$, standard deviation $\sigma_{dao_i}(t)$
Output: change/warning/no change is detected

1  $\mu_{dao_i}^{max} = \mu_{dao_i}(0)$, $\sigma_{dao_i}^{max} = \sigma_{dao_i}(0)$
   {Update the maximum exponential benefit and its corresponding exponential standard deviation}
2  if $\mu_{dao_i}(t) > \mu_{dao_i}^{max}$ then
3      $\mu_{dao_i}^{max} = \mu_{dao_i}(t)$
4      $\sigma_{dao_i}^{max} = \sigma_{dao_i}(t)$
   end
   {Check if a change is detected}
5  if $\mu_{dao_i}(t) - \sigma_{dao_i}(t) \leq \mu_{dao_i}^{max} - \alpha_2 * \sigma_{dao_i}^{max}$ then
6      a change is confirmed
7      reset $\mu_{dao_i}^{max}$ and $\sigma_{dao_i}^{max}$
   {Check if a warning is triggered}
8  else if $\mu_{dao_i}(t) - \sigma_{dao_i}(t) \leq \mu_{dao_i}^{max} - \alpha_1 * \sigma_{dao_i}^{max}$ then
9      a warning is triggered
   else
10     no change/warning is detected
```

To summarise, a change detection method is necessary to check whether the current *dao* is getting worse based on its accumulated exponential benefit $\mu_{dao_i}(t)$. In this context, the change detection test used could be adjusted to detect only significant changes and hence improves the stability of the system (i.e. avoid alerting the software architects of insignificant triggers).

5.4.6 Selecting The Architecture Option With The Optimal Trade-offs

If a significant detrimental change is detected in the *dao* currently being used (dao_{curr}), this means that it may be beneficial to replace this *dao* by another one from *DAO* elicited in step 1 of Section 5.4.4. In such a situation, it is desirable to know which *dao* among *DAO* has recently been providing the optimal trade-off between cost and benefit (Figure 5.4). Such a *dao* is assumed to be the best one to use now and hence the best one to switch to.

To determine the *dao* with the optimal trade-offs, we were inspired by the literature on MOP, NSGA-II, and a knee selection method [28, 29]. Our problem has two objectives: maximise benefit (Equation 5.6) and minimise cost (Equation 5.5). It is therefore a multi-objective problem with two objectives. As such, we calculate the *expected* marginal loss in order to identify the *dao* with the likely most balanced trade-off between benefit and cost. The process of determining the *dao* with the optimal trade-offs between cost and benefit at time *t* is divided into three sub-processes as explained below and summarised in Algorithm 4.

1. **Determine the Non-Dominated DAO:** This includes determining which diversified architecture options are non-dominated by any other *dao* [28, 29] (line 2-9). A given dao_i dominates dao_j iff:($c_{dao_i}(t) \leq c_{dao_j}(t)$ and $\mu_{dao_i}(t) \geq \mu_{dao_j}(t)$) and ($c_{dao_i}(t) < c_{dao_j}(t)$ or $\mu_{dao_i}(t) > \mu_{dao_j}(t)$). According to the definition above, non-dominated architecture options can be considered as better than dominated ones.

2. **Calculate the marginal loss of the Non-Dominated DAO over time:** The marginal loss is used for the purpose of computing the *expected* marginal loss (Step 3). In particular, we use a loss function that aggregates cost and benefit into a single value as follows (line 12):

$$L_\lambda(dao_i, t) = \lambda c_{dao_i}(t) - (1-\lambda)\mu_{dao_i}(t) \tag{5.10}$$

   where $\lambda \in [0,1]$ is a pre-defined parameter that controls the relative importance between cost and exponential benefit. The loss describes how (un)desirable a certain *dao* is.

   The marginal loss $L'_\lambda(dao_i, t)$ represents how much worse the loss would be if dao_i was not available and we had to use the one with the second optimal trade-off, given a certain λ (line 13-15). It can be calculated based on the following equation [28]:

$$L'_{\lambda}(dao_i,t) = \begin{cases} min_{(i \neq ii)} L_{\lambda}(dao_{ii},t) - L_{\lambda}(dao_i,t) \\ \qquad : if \quad i = argmin_i L_{\lambda}(dao_{i'},t) \\ 0 \quad : otherwise \end{cases} \tag{5.11}$$

where *argmin* function is used to check if dao_i has the minimum loss. If this is true, then we will iterate over all *DAO* to find the minimum difference between losses of a particular dao_{ii} and dao_i. This is set as the marginal loss. If dao_i is not the one with the minimum loss, we set $L'_{\lambda}(dao_i,t)$ to 0.

3. **Determine the expected marginal loss of Diversified Architecture Options over time:** The *dao* with the optimal trade-off between cost and benefit at time t is the *dao* with the maximum expected marginal loss at time t. This option can be suggested to the software architect as the optimal *dao* to be adopted at time t. This *dao* may or may not be the same as current option dao_{curr}.

   As in [28], we use an approximation of the expected marginal loss rather than the true marginal loss. The expected marginal loss has been proposed and used in the MOP literature [28] to facilitate the choice of which solution from the Pareto front to adopt in practice. In contrast, a single-objective problem would use a single fixed value for λ. However, we need to compute the marginal loss with a sample of different λ values. In our work, we used equally spaced values λ: $\{0.1, 0.2, \cdots, 0.9\}$, where $|\lambda| = 9$ is the number of λ values used. The expected marginal loss ($approxM[L'_{\lambda}(dao_i,t)]$) can be approximated by taking the average of the marginal losses computed using different sampled values for λ [28] (line 16-17), as shown in equation 5.12. The optimal dao_i is the one with maximum expected marginal loss (line 18).

$$approxM[L'_{\lambda}(dao_i,t)] = \frac{\sum_{\lambda \in \{0.1,0.2,\cdots,0.9\}} L'_{\lambda}(dao_i,t)}{|\lambda|} \tag{5.12}$$

As explained above, to determine the *dao* with the optimal trade-off between cost and benefit at time t, we need to know the exponential benefit and standard deviation of the *DAO* at this timestep, including the *DAO* that are not currently in-use by the software system. This information can be obtained in a number of different ways:

- If a given dao_i uses the same D as the current dao_{curr}, but connects them in a different way, the exponential benefit and standard deviation can be tracked over time even if

dao_i is not currently being used. This is because the qualities of interest of the D are being monitored via dao_{curr} and just need to be aggregated using a different function to compute dao_i's exponential benefit and standard deviation.

- A given $dao_i \neq dao_{curr}$ can be activated for use in parallel with dao_{curr} at pre-defined time intervals T' to track their qualities of interest. In this case, dao_j's exponential benefit and standard deviation are updated at every $T' > 1$ units of time, saving the overhead of having to use more than one *dao* at every timestep. The accuracy of dao_j's exponential benefit and standard deviation will depend on how large T' is.

- A simulation based on what-if test scenarios can be used to estimate the exponential benefit and standard deviation of $dao_i \neq dao_{curr}$. Simulators are typically used during the architecture prototyping, analysis, and refinement stages to evaluate the response and sensitivity of the architecture for these tests. In this case, dao_i's exponential benefit and standard deviation could be updated frequently, possibly at every unit of time. However, their accuracy depends on the simulator. Due to the exponential time-decay factor used for calculating exponential benefit, inaccuracies on exponential benefit of architecture options not in-use at time t can be quickly found if and when we switch from a *dao* to another. If the newly adopted one is found to have significantly poorer benefit than previously estimated, the change detection mechanism will detect this and trigger the procedure to determine whether we should switch to another one. For instance, if some of the IoT devices forming the selected *dao* become unavailable after selecting it due to the highly dynamic application environment, making it not possible to execute the selected *dao*, this will lead to a decrease in exponential benefit and hence cause a change to an alternative architecture.

In summary, our proposed approach monitors the extent to which the concerned architecture design decisions satisfy the quality attributes over time; it provides feedback on their performance against the said qualities. The feedback is used to adapt the architecture as seen fit. In particular, the feedback determines the benefit of diversified architecture options and hence can aid the architect in demonstrating the situations where the *dao* would work and the others which are not suitable for that context. As an example, if a particular *dao* does not perform well in any of the provided scenarios (e.g. when the energy consumption is a priority) over a prolonged period of time, this is a strong indicator of a wrong choice of the design decisions and choices. The approach incorporates human expertise in the decision of whether or not to change or keep this architecture as an option, as the modelling of diversified architecture options is the fundamental domain knowledge from the engineers, as for every software system.

```
ALGORITHM 4: Select()
Input: change detected in $dao_{curr}$, a pre-defined parameter $\lambda$, number of $\lambda$ ($|\lambda|$), number of $DAO$ ($|DAO|$),
       exponential benefit $\mu_{dao_i}(t)$, cost $c_{dao_i}(t)$
Output: optimal $dao_i$
---------------------------------------------------------------
{A change is detected in $dao_{curr}$, then do:}
{Initialisation:}
1  $\lambda : \{0.1, 0.2, \cdots, 0.9\}$, $|\lambda| = 9$
{Determine the Non-Dominated $DAO$:}
2  for $i$ = 1 to $|DAO|$ do
3      $dominant = 0$
4      for $j$ = 1 to $|DAO|$ do
5          if $(c_{dao_i}(t) \leq c_{dao_j}(t) \& \mu_{dao_i}(t) \geq \mu_{dao_j}(t)) \& (c_{dao_i}(t) < c_{dao_j}(t) | \mu_{dao_i}(t) > \mu_{dao_j}(t))$ then
6              $dao_i$ dominates $dao_j$
7              $dominant = 1$
           end
       end
8      if $dominant = 0$ then
9          Add $dao_i$ to list of non-dominant $DAO$
       end
   end
{Calculate marginal loss for the non-dominant $DAO$:}
10 for $i$ = 1 to Length (list of non-dominant $|DAO|$) do
11     foreach $\lambda \in \{0.1, 0.2, \cdots, 0.9\}$ do
12         $L_\lambda(dao_i, t) = \lambda c_{dao_i}(t) - (1 - \lambda)\mu_{dao_i}(t)$
13         if $i = argmin_i L_\lambda(dao_{i'}, t)$ then
14             $L'_\lambda(dao_i, t) = min_{(i \neq ii)} L_\lambda(dao_{ii}, t) - L_\lambda(dao_i, t)$
           else
15             $L'_\lambda(dao_i, t) = 0$
           end
       end
   end
{Determine the expected marginal loss for the non-dominant $DAO$:}
16 for $i$=1 to Length (list of non-dominant $|DAO|$) do
17     $approxM[L'_\lambda(dao_i, t)] = \frac{\sum_{\lambda \in \{0.1, 0.2, \cdots, 0.9\}} L'_\lambda(dao_i, t)}{|\lambda|}$
   end
18 Return optimal $dao_i$ = $dao_i$ with $\max(approxM[L'_\lambda(dao_i, t)])$
```

5.5 Data Synthesis And Analysis

Our experiments focus on architecting the sensing-actuating functionality of *iTransport* to show how run-time evaluation can complement design-time evaluation. At design-time, the architect has followed the procedure of Chapter 3 and Chapter 4 to preliminary decide on the *DAO* (and their composing *D*) for implementing this functionality. The experiments were executed on Intel Core i7 processor machine with 16GB of RAM. The data synthesis process is performed using iFogSim [35], whereas Matlab is exploited for data analysis. To simulate the qualities of interest of each architecture decision over time, we adopt the *iFogSim* [35] tool. This tool builds on Cloudsim [47]; it provides the architect with the freedom of hierarchically composing the fog devices, clouds, and data streams. In iFogSim, we have hierarchically composed the application as shown in Figure 3.1. The candidate *DAO* used in this study are shown in Table 3.1. In particular, each *dao* is composed of different types of data collection (type of sensors) and connectivity (computation locations) as architecture decisions, meaning that the processing performed by each *dao* is executed differently. Connectivity was simulated by either executing the *object detector* and *tracker* modules (shown in Figure 3.1) in the cloud and/or fog. For data collection, two gateways were used, where each is connected to an average of 50 smart cameras (Figure 5.7): fixed, mobile, as well as fixed and mobile, having a total of 100 fog devices. The fog devices and cloud are configured based on [204, 205, 35].
The goal is to continually optimise the following two conflicting requirements:

- **Benefit:** It needs to be maximised and is based on three quality attributes of interest:

  1. *Response Time: iTransport* is time-critical application, so it should respond as early as possible. In *iTransport*, the response time (RT) of an application is the application's end to end delay (in milliseconds *ms*) and measured using iFogSim.
  2. *Network Usage:* High network usage would cause network congestion, so it should be as low as possible. The network usage (NU in mega bytes *MB*) is also measured using iFogSim.
  3. *Energy Consumption:* IoT will lead to unlimited energy consumption if not controlled [206]. Therefore, energy consumption needs to be minimised. In this context, when designing IoT architectures, architects have to consider energy consumption as a major concern [207]. In *iTransport*, the energy consumption (EC in mega joules *MJ*) is the total energy consumption by all the devices in the application, which is also determined through iFogSim.

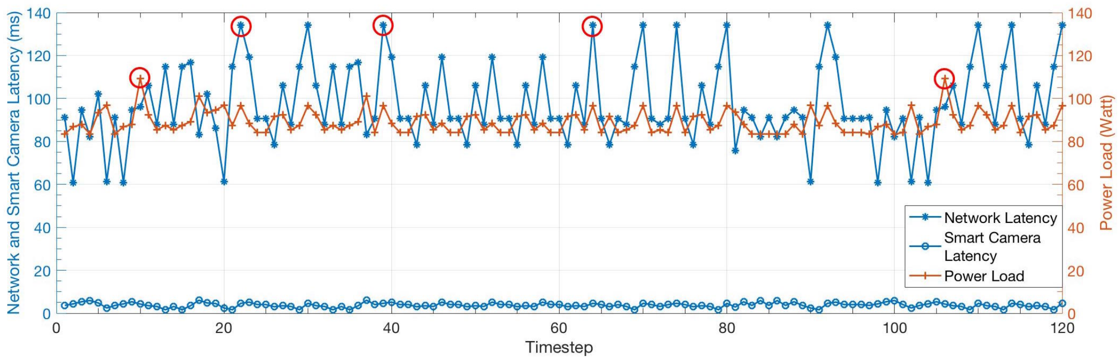


Figure 5.6 Sample of changing environmental conditions facing *iTransport* (i.e. input for one of the *DAO*). *Note that the red circles denote some examples of accidental spikes.*

- **Cost:** It needs to be minimised. It is a composite of operating cost, and switching cost that is added once we switch. In the context of *iTransport*, the average cost encompasses a thing's connectivity (includes switching), execution in the cloud and/or fog (i.e. leasing cost of processing services), and other costs mentioned in Section 3.4. The mean overall costs have been collected from iFogSim.

The iFogSim tool takes as input: the network latency, power load, smart camera's latency (i.e. fog devices), number of cameras, number of gateways, tuple configurations, cloud and fog devices configurations. The sources of uncertainty in *iTransport* come from the varying network delays due to network congestion. There are other factors, which impact the environmental conditions, such as uncertainty in QoS of cloud service providers (e.g. Amazon, Google Cloud, *etc*) and software-defined capabilities of fog service providers in terms of their processing power, as well as the hyperconnectivity of the nodes. In this context, we have generated data corresponding to each *dao* including typical as well as worst case scenarios. For instance, we varied the power load as 80-110 watt with a fluctuation of 5-10%, following the typical power load values from [205]. The latency was varied in the range of low to high, i.e. 1-6ms with a fluctuation of 20-30% on the smart camera. We also varied the network latency with an average of 100*ms* and fluctuation of 20-25%, as exemplified in Figure 5.6. The choice of this fluctuation was based on [208], as it normally provides acceptable throughput across various networking protocols, but also causes accidental spikes that represent worst case scenarios.

We also simulated changes by using diverse smart cameras' configuration [205, 35], as depicted in Table 5.2. Based on that, it outputs the energy consumption of devices, application's response time, and network usage. This setting was intentionally designed as a worst case that goes beyond a stable setting. Further, the iFogSim takes the pricing configurations for IoT devices (i.e. fog devices) and cloud to generate the mean costs of each *dao* (i.e. application architecture). All the pricing configurations are used with respect to AWS IoT services [209]. We have run the simulations for each *dao* for 120 timesteps.

Table 5.2 Simulation Parameters for *iTransport*.

Parameter	Value
Device Configurations	**CPU (GHz), RAM (GB), Cost ($/day)**
Cloud Datacenter	3GHz CPU, 40GB RAM, $0.1-0.3/day
Wifi and ISP Gateways	3GHz CPU, 4GB RAM, $0.0053-0.0056 /day
Smart Camera	[1.6, 1.867, and 2.113] GHz CPU, 4GB RAM,$0.0053-0.0056 /day
Number of Smart Cameras	80-120
Network Configurations (From-To)	**Avg Latency**
From ISP Gateway to Cloud Datacenter	80-120 ms
From Wifi Gateway to ISP Gateway	1-6 ms
From Smart Camera to Wifi Gateway	1-6 ms
Tuple Configurations (Message size):	**CPU Length (MIPS), Network Length (Bytes)**
Raw Video Stream:	1000, 20000
Motion Video Stream:	2000, 2000
Object Location:	500, 2000
Warning:	1000, 100
Tracking parameters	28,100
Average QoS:	
Energy Consumption of devices	80-120MJ
Applications Response time	300-4000 ms
Network Usage	500 Kbps- 2 Mbps
Evaluation Settings:	
θ	$\{0.7, 0.9, 0.99\}$
α_1, α_2	$\{80, 92\%\}$, $\{90, 95\%\}$, $\{99, 99.9\%\}$
Normal(Constraint[RT, EC, NU]; Weights[w_{RT}, w_{EC}, w_{NU}])	[400ms, 130MJ, 1Mbps]; [0.4,0.3,0.3]
Strict(Constraint[RT, EC, NU]; Weights[w_{RT}, w_{EC}, w_{NU}])	[350ms, 130MJ, 500Kbps]; [0.4,0.2,0.4]
Normalised {Operating;Switching} Cost	$\{0.3-0.5; 0.2-0.4\}$

5.6 Experimental Evaluation

In the next series of experiments, we aim to show the usefulness of the approach by evaluating how well it addresses the research questions introduced in Section 5.3.

The system that uses the *dao* evaluated by our approach as having the optimal trade-offs is called *Informed-Selection System*. We compare this system against the following baseline systems:

1. **Static-Selection System:** This is a typical type of system used in practice [210, 14], where the expert implements a single *dao* based on its assessed value at design-time. For our work, the value is determined using the Binomial option pricing model

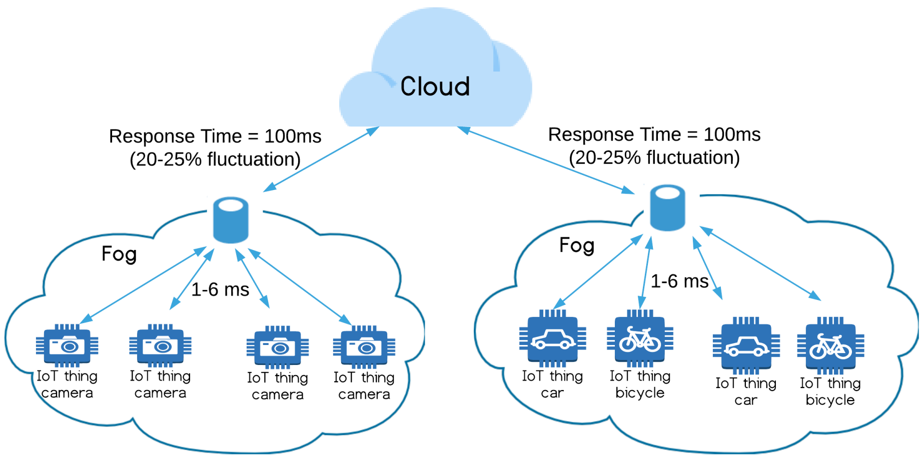


Figure 5.7 The Initial Experiment Configuration for iFogsim.

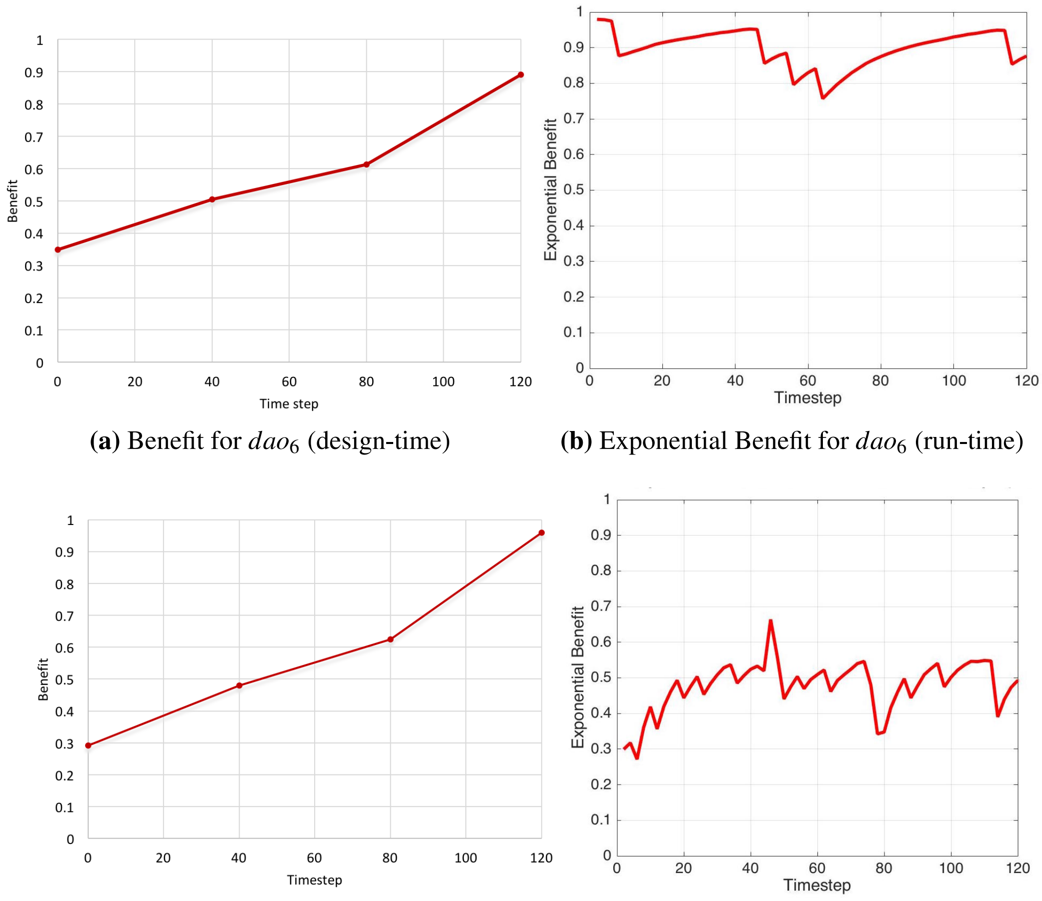
[4, 42, 14], which estimates the future benefits and costs of options based on a binomial decision tree. This approach is introduced in Chapter 4.

2. **Predefined-Selection System:** This is inspired by [211, 48], where the architect will choose the *dao* that is likely to perform the best for a given context. This selection is typically based on experience, backed up by back-of-the-envelope calculations for the cost, benefits, and technical potential. However, the selection may fail to predict potential fluctuations in value, quality potentials, and costs. As an example, our case used dao_6 during week days (because of peak hours) and dao_3 during weekends (because of less demand).

3. **Random-Selection System:** Our design for the baseline system follows the argument of [7, 212] but for the context of services. When a significant change is detected, it selects a *dao* randomly independent of its QoS over time. This is because the *DAO* are deemed to be functionally equivalent but deployed in different environments and geographical location (i.e., distributed Fogs/clouds). All the results related to the Random-selection approach are based on the average of 30 runs (the choice of 30 runs is recommended by [213]).

RQ3.1: How to evaluate the benefit of each dao over time?

Motivation: This experiment aims to show the usefulness of run-time evaluation over design-time evaluation. More specifically, it conveys that the run-time evaluation visualises scenarios and dynamics, which can hardly be captured at design-time. It is also important to confirm the design-time choices. For that, we compare the run-time approach against the design-time architecture evaluation approach proposed in Chapter 4. The latter uses *options theory* [176] to evaluate and justify the employment of architecturally diversified decisions and their augmentation to long-term value creation under uncertainty. The architect has the freedom to estimate the increases and decreases in the value potentials for the candidate architecture options over time, backed up by their experience.



(a) Benefit for *dao*$_6$ (design-time) **(b)** Exponential Benefit for *dao*$_6$ (run-time)

(c) Benefit for for *dao*$_1$ (design-time) **(d)** Exponential Benefit for *dao*$_1$ (run-time)

Figure 5.8 The results of assessing design-time and run-time evaluation.

Experimental setup: The design-time architecture evaluation approach uses experts' (e.g. architects and other stakeholders) assumptions on the likely utilities of a *dao* over a pre-defined period of time. In our experiments, the pre-defined period of time corresponded to 120 timesteps. The expert's opinion is depicted by a utility tree that is provided at design-time, without making use of any run-time information. We investigate how the design-time architecture evaluation approach (Chapter 4) and the proposed run-time approach in this chapter evaluate two *DAO*: the *dao* with the best benefit (*dao*$_6$) over time and the *dao* with the worst benefit (dao_1). The utility tree created for the design-time approach used in the experiments is shown in Figure B.9a and B.9b for dao_1 and Figure B.10c and B.10d for dao_6. The design-time approach (Chapter 4) then uses this utility tree to compute the likely benefit of a *dao* over the pre-defined period of time. This benefit is the one depicted in Figures 5.8a and 5.8c. Therefore, even though this is a design-time evaluation approach, it provides information on the expected run-time benefit of the *DAO*, being a meaningful design-time approach to compare against.

Figure 5.8a depicts the *dao* with the best benefit (i.e. dao_6) over time computed using the design-time approach (Chapter 4), whereas Figure 5.8b shows the exponential benefit quantified using our run-time architecture evaluation approach. Further, the benefit of dao_1 (i.e. worst) is shown in Figure 5.8c, whereas its exponential benefit is plotted in Figure 5.8d. Note that we have normalised the design-time benefit to ensure fair comparison with the run-time benefit, as the design-time evaluation approach provides a monetary value for the benefit of *DAO*. We assume that the cost of both options is constant over time, whereas the benefit is varying over time.

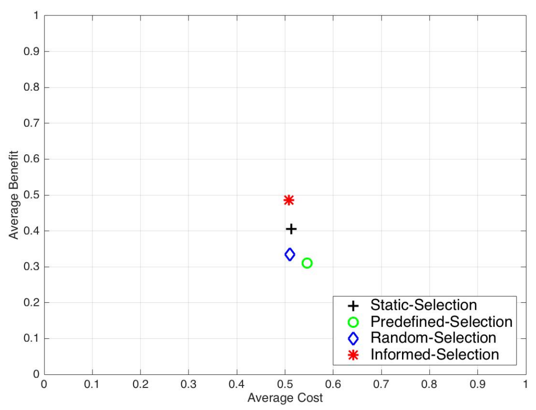
Analysis: As we can see, the design-time approach was conservative and estimated that dao_6 had initially low benefit and then improved over time. Our run-time approach, on the other hand, is able to show to the software architect that dao_6 has high benefit from the beginning. Figure 5.8c shows the benefit value of dao_1 over time computed using the design-time approach (Chapter 4). In this scenario, the design-time approach incorrectly estimated that the value of dao_1 was going to increase over time. Our approach was able to discern at run-time that this was not really the case, as shown in Figure 5.8d. Even though the exponential benefit ascended from 0.3 to 0.7 until timestep 25, this was followed by a deterioration to an average of 0.4 (Figure 5.8d).

RQ3.2: How can run-time evaluation determine changes in dao's value over time and inform subsequent decisions?

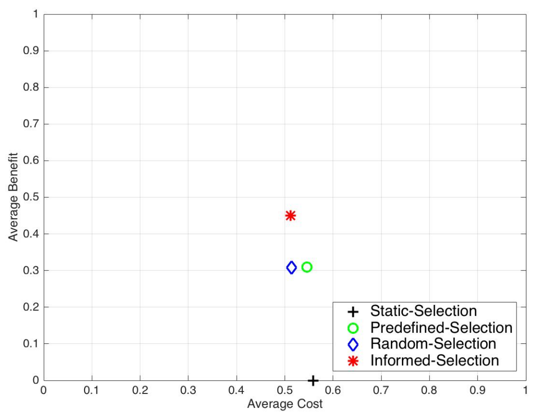
Motivation: This experiment aids the architect to evaluate the *DAO* and suggest the ones with most balanced cost-benefit trade-offs using the informed-selection approach as compared with other approaches introduced earlier in Section 5.6. For that, we consider that when our approach detects a significant detrimental change, the software architect decides to implement the *dao* that our approach recommends as the one with the optimal trade-off between cost and benefit, based on *strict* quality constraints. This experiment will also aid the architect in refining the selection of design-time *DAO* by checking their cost-benefit at run-time. It can also indicate which *DAO* were not performing well, and hence require phasing-out. Back to IoT context, the use of fog computing and cloud computing in mobile crowd sensing applications is highly affected by the QoS requirements. Consider a scenario where the stakeholders require the *iTransport* application to quickly track the accident in the city centre, especially in rush hours. In this context, low response time and network usage is of higher concern rather than energy consumption. Therefore, the ranking score (i.e. weights) for response time and network usage qualities are higher than energy consumption. Also the use of fog-cloud computing is advisable over cloud computing. So object detector and tracker modules will be executed in the fog. We are uncertain about the network latency (due to dynamic traffic and variable load) and mobility of devices (nodes join/leave the network). This will cause instability, which may require a switch to another architecture option.

Experimental setup: The environmental conditions that keep changing at run-time are network latency, camera latency and power load. These potentially affect the aggregated benefit, which is composed of application response time, energy consumption in devices and cloud, and network usage. Therefore, we might not get a linear effect from input to output – this is due to the use of aggregated exponential benefit, where the change detection and selection is based on it. Next, we will demonstrate the strict quality constraint scenario to show whether the changes detected by the approach are aligned with the input environmental conditions. Consider the case where the architect can adjust the quality weights to reflect priorities from stakeholders. For example, when the application response time and network usage become a priority, energy consumption priorities can be downgraded. In our case study, we assume a w_{RT} of 0.4 for response time, w_{EC} of 0.2 for energy consumption, and w_{NU} for network usage of 0.4. We also consider that the architect has constrained the application to handle the request in less than *350 ms* and network usage does not exceed *500KB*; whereas, the energy consumption should not exceed *130 MJ*. Historical performance and experts' judgement can inform the adjustment of the prior constraints [193, 35]. In this experiment, the focus is on the application's response time and network usage concerns. Nevertheless, the same experiment could be applied on other stakeholders' concerns, such as the ones discussed in Section 3.4. Figure 5.9 shows the average benefit and cost of the system as a whole across 120 timesteps, i.e., calculated based on the *DAO* currently being used for two scenarios: *best case* and *worst case*. We have also plotted the environmental changing conditions (the power load, network latency, and camera latency) for best case scenario, in Figure 5.10a and 5.10b, along with their impact on response time, network usage, and energy consumption.

Analysis (Best case scenario): The design-time evaluation approach (Chapter 4) suggests the systems start operation using the *dao* that are believed to provide the most balanced



(a) Average Benefit and Cost for the best case scenario across 120 timesteps.



(b) Average Benefit and Cost for the worst case scenario across 120 timesteps.

Figure 5.9 The evaluation of the four approaches' decision-making process under strict quality constraints.

cost-benefit trade-off at run-time (dao_6). Different changes were observed after 5 timesteps (Figure 5.9a). The mobility of devices has caused a decrease in the overall benefit, due to highly changing response time causing a violation in the response time constraint. The random-selection approach selects a random *dao*, which is not always the best. In addition, throughout the experiment, this approach suffered from too many switches (9 switches), causing instability and lowering the benefit to about 0.35. The pre-defined selection performs a bit worse than the random one; this is because the constraint was too strict for options recommended by that approach, causing various violations. The static-selection is the second best one; this is because the selection is geared towards selecting *dao* with better potentials for the selected scenarios. This strategy considers design-time knowledge, which can be challenged or confirmed by future runs of the system. The informed-selection approach provides the most appealing results compared with other approaches: this is because the informed-selection approach can continually recommend the *dao* with the most balanced cost-benefit trade-offs. For example, we observed that when the first change was triggered, the informed-selection figured out that the initially selected *dao* was still the optimal choice to keep. Yet, after 22 timesteps when another change was detected, dao_4 was recommended due to its large improvements for response time. However, this requires cost of leasing these services and maintaining their devices (switching cost).

By mapping the environmental conditions to the evaluation of *DAO* based on the informed-selection approach (Figure 5.10a and 5.10b), the approach has detected significant deviations which are consistent with input environmental conditions. In particular, the change could be triggered either from high fluctuation and/or a deterioration in one/all of QoS. For instance, from timestep 1 to 5, we see an increase in the values of application response time and network usage, caused by a (smaller) increase in network and camera latency (Figure 5.10a). These result in a decrease in the exponential benefit, which is considered as significant when reaching timestep 5 (Figure 5.10c).

Since the approach is still building knowledge about the current *dao* and having highly dynamic changing conditions during the initial observation period, this caused a change detection at almost every consecutive 5 timesteps until timestep 27 (Figure 5.10). However, these only led to switches when another better *dao* was available (at timestep 22). This led to improvements in the exponential benefit of the proposed informed-selection approach over the following timesteps (Figure 5.10c).

Further, at $t = 46$ (Figure 5.10a), an increase in network camera latency has caused a noticeable rise in the application's response time and network usage, which accordingly resulted in the detection of a significant change in environmental conditions. However, the approach found that the current *dao* still provided the most balanced cost-benefit trade-off and hence the approach kept using this *dao* for the next 15 timesteps (though other change detection were triggered). After that, at $t = 61$, the approach detected a significant change (i.e. a high power load, network and camera latency) and has then selected another *dao* which provided the most balanced cost-benefit trade-off. After $t = 61$, the exponential benefit was just oscillating resulting in no significant changes (i.e. no changes were detected), as reflected by the exponential benefit (Figure 5.10c).

Analysis (Worst case scenario): The design-time evaluation approach suggests the systems to start operation using the *dao* that turns out to provide the worst balanced cost-benefit trade-off at run-time (i.e. dao_2). In this respect, a replacement is advocated by our approach. Figure 5.9b shows that the most balanced trade-off between benefit and cost is achieved by the informed-selection approach, followed by the random- and predefined-selection approaches. Here, informed-selection achieved much higher benefit than other approaches. Since the response time constraint is violated from the beginning, the static-selection approach experiences a zero benefit. In this context, high application response time is not recommended, because this is a safety-critical application. From this experiment, dao_1 has never been recommended by the approach because of its low response time and high energy consumption, which may inform the architect to phase-out. For the best and worst case scenarios, the architect can use the run-time evaluation approach to visualise the cost-benefit trade-offs of the suggested *DAO* over time (Figure 5.9) for informed-selection approach against other approaches. It can also show how the adoption of optimal *DAO* provided an improved benefit over time (i.e. added values).

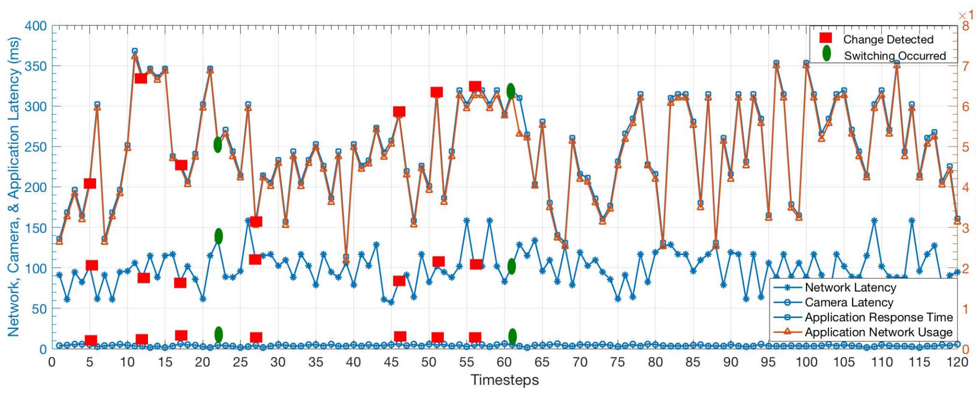
5.7 Further Analysis Of The Proposed Approach

This set of the experiments aims at evaluating the robustness and scalability of the proposed approach. It also aims at providing a better understanding of the influence of the stability parameters (θ, α) on the proposed approach. This better understanding can guide software architects in the decision of which values to use for these parameters.

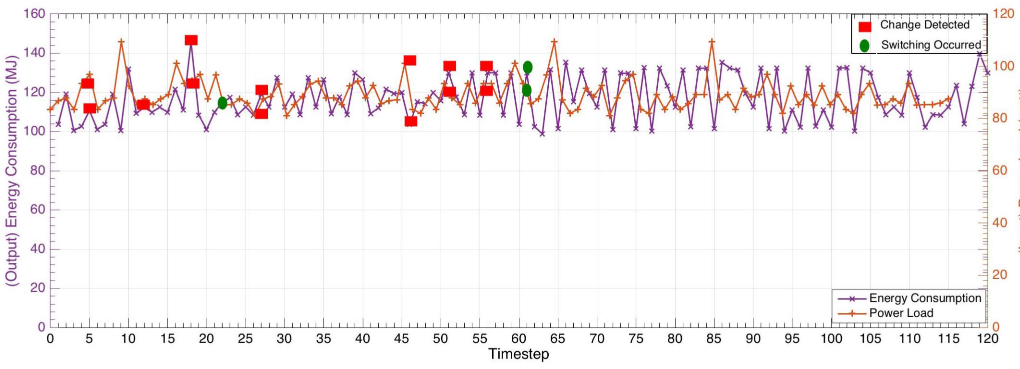
5.7.1 Impact Of Frequency Of Monitoring

Motivation: The proposed approach depends on the possibility of either monitoring the *DAO* that are not currently in-use, or using a simulator to get an idea of their likely behaviour. If we opt for monitoring the *DAO* that are not currently being used, this will lead to an overhead in terms of time taken for the informed-selection approach to do the analysis. Therefore, one may opt for not monitoring them very often. Practically, its hard to monitor the architectural options (e.g. virtual sensors, physical sensors) every timestep, due to their availability. There is a trade-off between the execution time and % error, which we aim to measure in this experiment. To this regard, this could guide the architect on deciding the monitoring intervals. In this context, this experiment aims to provide answers for the following question: *What is the impact of the frequency of monitoring on the accuracy of the approach?*.

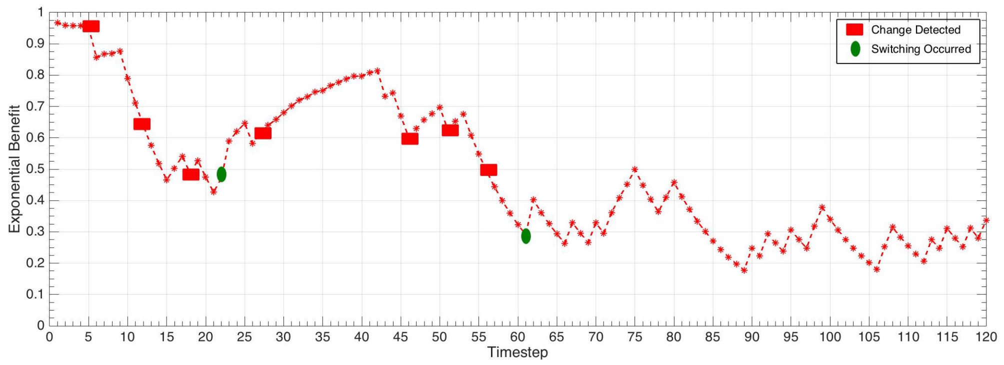
Experimental setup: For that, we use an error metric *% Relative Error* to measure the error in decision-making between monitoring every T' intervals and every timestep. In this context, we will measure the error *for each dao separately* and *mean exponential benefit for the whole process*. For the former, the $\%RelativeError = \left|\frac{Actual\mu_{dao_i}(t) - Monitored\mu_{dao_i}(t)}{Actual\mu_{dao_i}(t)}\right| * 100$. The actual exponential benefit is the one monitored every timestep till T', where $T' = \{5, 10, 20, 30, 40, 50, 60, 70, 80\}$. The monitored exponential benefit is the one monitored every T' intervals. For example, if T'=5, then from t=1 to 4, the exponential benefit will be the same and then at t=5, it is updated. Figure 5.11 depicts the % relative error versus T' intervals for each *dao*. As for the *mean exponential benefit for the whole process*, the $\%RelativeError = \left|\frac{ActualMean\mu - MonitoredMean\mu}{ActualMean\mu}\right| * 100$. The actual mean exponential benefit is the average benefit if monitoring occurs at every timestep, so it is constant for all. The monitored mean exponential benefit is the average benefit if monitoring occurs at every T' intervals. We consider the error in the whole process to evaluate how much worse the mean



(a) The impact of network and camera latency on application's response time and network usage.



(b) The impact of power load on application's energy consumption.



(c) Exponential Benefit of informed-selection with change detection and switches.

Figure 5.10 An illustration of the input environmental conditions for the 120 timesteps including network latency, smart camera's latency, power load on the devices, change detection, and switching occurrence, as well as the output exponential benefit of informed-selection approach for strict quality constraints and prioritized ranking score for the best case scenario as an example. *The red squares represent change detections that did not lead to switching DAO, and green circles represent change detection followed by dao switching.*

benefit (based on informed-selection approach) would be if monitoring happened every T' intervals rather than every timestep. This includes the monitoring, change detection, and decision-making processes. We have developed two scenarios: *best case* (Case 1 and 3), where the optimal *dao* is initially chosen; *worst case* (Case 2 and 4), where the worst *dao* is initially selected. This is illustrated in Figure 5.12, where each point in the plot has 3 coordinates (X: execution time; Y: monitoring interval; Z: % Error)
Analysis (for each *dao*): In Figure 5.11, the relative error for dao_2 and dao_3 ranges from 0-30%. This is due to the low fluctuation in these options. This in turn lowers their impact on the accuracy. However, dao_1 and dao_5 violated the constraint at $t = 1$, which resulted in an error of about 70%. This is because dao_1 and dao_5 had good exponential benefit after the first timestep (i.e. not violating constraint), but the monitoring was based on the first timestep only.
Analysis (for whole process): Case 3 presents the smallest error (Figure 5.12c), which was about 0.27%. The error was small because the initially selected *dao* was optimal and fluctuating less than the other *DAO*. As a result, the informed-selection approach managed to keep using it for different monitoring intervals (i.e. no switching). In this context, the architect could use higher monitoring intervals to save execution time. This is due to the decrease in execution time from 0.45 seconds (T' = 5) to 0.22 seconds (T'= 50).

Further, Case 1 experienced 10% increase in error (constant for all intervals), as shown in Figure 5.12a. This rise is due to the fact that when monitoring happened at every timestep, three switches were recommended by our approach. However, when the monitoring interval increased, the approach had quite outdated information about the current *dao*. Therefore, it did not advocate any switches, which resulted in slight degradation in benefit. To this regard, if time is the concern, the architect could choose higher monitoring intervals to benefit from lower overhead in terms of time and cost.

On the contrary, the selection of the worst *dao* in cases 2 and 4 caused a significant rise of 65% in relative error (Figure 5.12b, 5.12d), because the approach had outdated knowledge about the initial *dao*. To exemplify in case 2 at $T' = 50$, the approach updates the current benefit with respect to the first timestep. Therefore, for the first 50 timesteps the benefit was very low and a switch was recommended after $T' = 50$. On the other hand, if monitoring occurred at every timestep, then the approach will advocate switching after 5 timesteps to an optimal *dao*. This explains the rise in the relative error to 40%. For the latter scenarios, it is recommended for the architect to use lower monitoring intervals with higher overhead, rather than experiencing a reduction in benefit. To this end, our approach could aid the architect in tuning the monitoring interval with respect to execution time overhead and relative error.

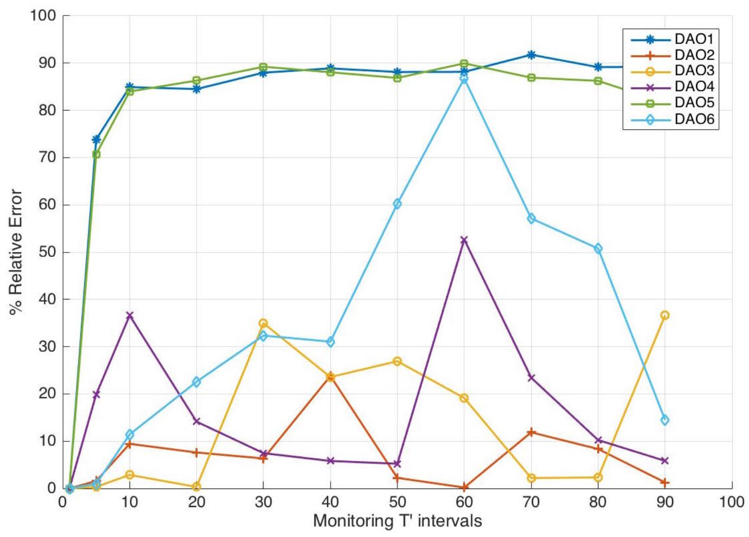
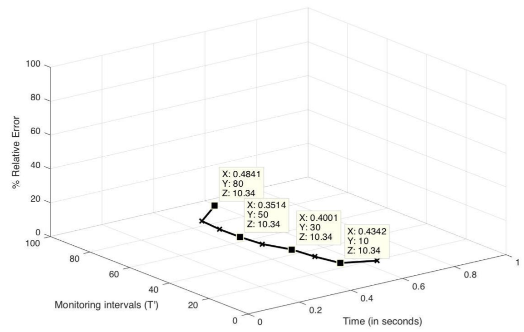
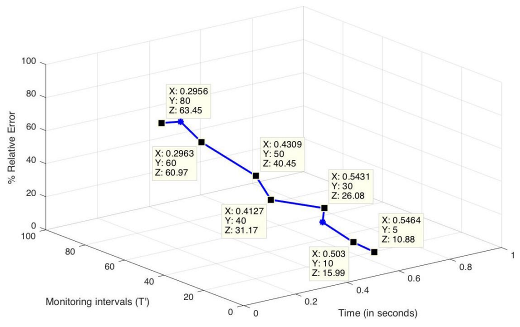


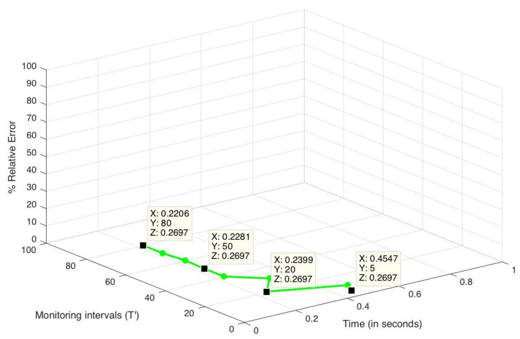
Figure 5.11 The % Relative Error of monitoring every T’ intervals for each *dao*.



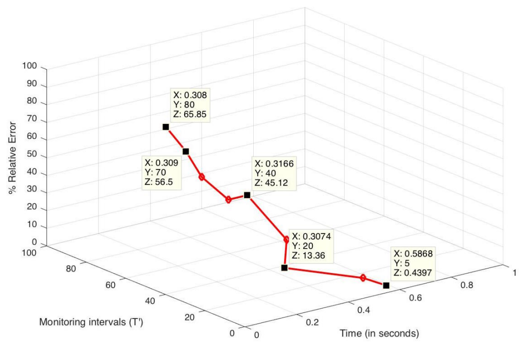
(**a**) Case 1 (Best)



(**b**) Case 2 (Worst)



(**c**) Case 3 (Best)



(**d**) Case 4 (Worst)

Figure 5.12 The % relative error and execution time of monitoring every T’ intervals for four scenarios.

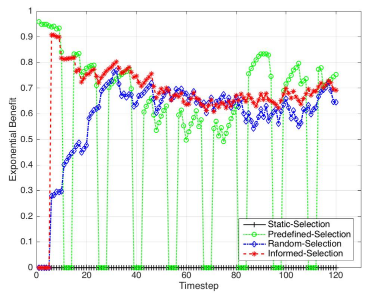
5.7.2 Robustness To Noise

Motivation: An alternative to monitoring each *dao* at regular T' intervals would be to use a simulator to monitor the likely benefit of the *DAO* that are not currently in-use. However, simulators are likely to produce noisy quality indicators, which differ from the actual qualities that these *DAO* would have if they were currently being used. In order to evaluate the impact of the noise potentially produced by simulators, we investigate how the proposed approach reacts to different levels of noise i.e. *To what extent the informed-selection system can deal with noise data as compared with other systems?*.

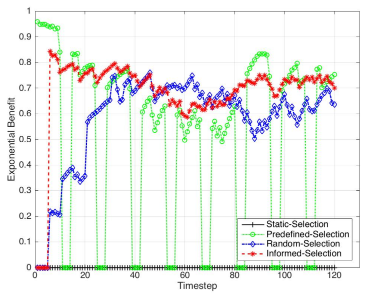
Experimental setup: In this experiment, we have generated *Gaussian Noise* [214] on the QoS data. A given quality attribute $q'_{dao_i}(t)$ is replaced by a value drawn from a Gaussian distribution $\mathscr{N}(q'_{dao_i}(t), s)$, where $q'_{dao_i}(t)$ is the mean and $s = \{0.05(low), 0.1(mid), 0.5(high)\}$ is the standard deviation. The smaller/larger s represent the cases where there is less/more noise. This reflects the cases in which simulators are more/less reliable. It enables us to check how robust the approach is to wrong measurements provided by a simulator. In particular, we expect the proposed approach to be affected by such erroneous quality information, but to quickly react and recover from it if a poor switch occurs. This is because, once the switch occurs, the true benefit of the *dao* can be determined. If this benefit is worse than expected, a change will be detected, leading to a switch to another potentially better *dao*.

The results are based on the average of 30 runs, due to the randomness of noise (the choice of 30 runs is recommended by [213]). In this experiment, we have compared the proposed approach against the state-of-the-art and baseline approaches introduced in Section 5.6. For this experiment, the design-time evaluation approach suggests the systems to start operation using the *dao* that are believed to provide the worst cost-benefit trade-off at run-time (dao_2). This choice is advocated to show how our approach can deal with different noise levels, in even worst case scenarios. Figure 5.13a, 5.13b, and 5.13c show the exponential benefit of four approaches for the selected *DAO* across 120 timesteps. Whereas, Figure 5.13d, 5.13e, and 5.13f depict the mean exponential benefit and cost of four approaches for the selected *DAO* across 120 timesteps.

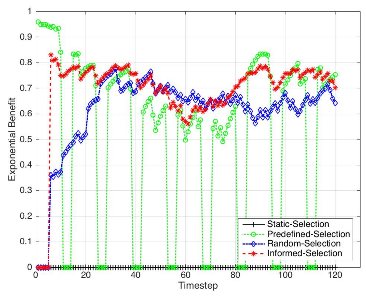
Analysis: As clearly illustrated in Figure 5.13, the informed-selection approach has managed to select optimal options, although by introducing higher noise levels, the choice could have become much worse. Followed by the random-selection, which benefited from the change detection test. However, it selected *DAO*, which were not always the best. The static-selection and predefined-selection approaches are based on design-time knowledge, thus they are not affected by noise. However, the static-selection produced zero benefit across 120 timesteps, due constraints' violation. The predefined-selection was highly fluctuating because of constraints' violation in some timesteps. Therefore, the informed-selection approach produced the best results overall (Figure 5.13d, 5.13e, and 5.13f). Further, the informed-selection has quickly recovered from wrong choices due to noise. For instance in Figure 5.13c, dao_2 was initially selected for deployment (in one of the runs), which turned out to be the worst *dao*. After 5 timesteps, the approach detected deterioration in exponential benefit, and hence recommended dao_4 to switch to. Ten timesteps later, another change is triggered and the informed-selection chose dao_5 to replace the current *dao*. We have found that dao_5 was not the optimal one and dao_6 seemed to be better (i.e. this choice was affected by noise). Though, the informed-selection approach has quickly recovered from the poor switch and suggested dao_6 after 5 timesteps. To this extent, our approach managed to self-repair from incorrect choices (due to varying noise levels).



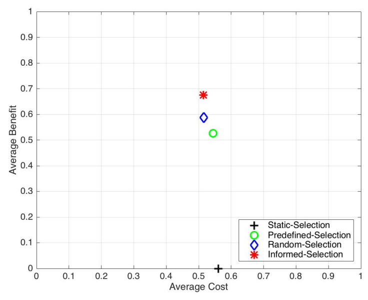
(a) Exponential Benefit (low)



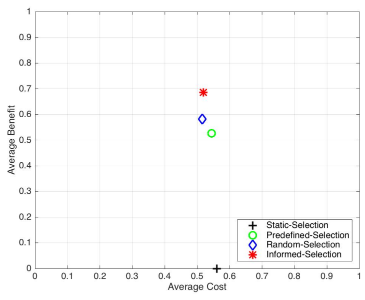
(b) Exponential Benefit (mid)



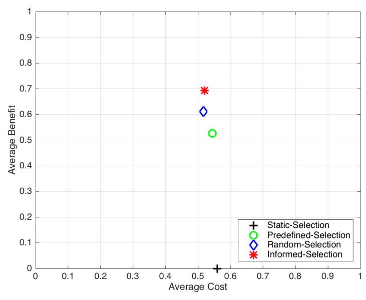
(c) Exponential Benefit (high)



(d) Overall behaviour (low)



(e) Overall behaviour (mid)



(f) Overall behaviour (high)

Figure 5.13 The behaviour of all the approaches after the application of varying noise levels on the data.

5.7.3 Scalability Of The Proposed Approach

Motivation: This experiment will answer the following: *How scalable is the proposed approach to larger numbers of dao?*. The scalability of the proposed approach (informed-selection) is an important indicator to show to what extent our run-time evaluation can be applied in practice. There is a trade-off between the candidate options and the execution time of the algorithm.

Experimental setup: In this experiment, the mean execution of the informed-selection algorithm is computed over 120 timesteps for varying number of *dao*, as illustrated in Figure 5.14.

Experimental Analysis: The informed-selection approach performs very well until reaching 500 *DAO*, it takes less than 3 seconds. This is applicable for safety-critical IoT applications, which in case a change is detected, the approach will be able to search for the optimal solution in a few seconds. However, after 500 architecture options, the execution time starts to increase reaching 17 seconds for 1000 options. Therefore, for an application, which requires more than 1000 options, our approach will take further time for decision-making.

Time Complexity Analysis: We also analyse the time complexity of the run-time evaluation approach through Algorithm 2, 3, and 4, respectively. First, in Algorithm 2, there are three main steps: (i) Iterate over all the *DAO* ($O(|DAO|)$), line 1; (ii) Iterate over all the quality attributes for each dao_i ($O(|Q|)$), line 2; (iii) Compute the quality $q_{dao_i}(t)$ based on Table 5.1 ($O(\text{Compute } q_{dao_i}(t))$), line 3; and (iv) Quantify the exponential benefit, cost, variance and standard deviation at time t for each dao_i ($O(|Q|+|\nu|)$), line 12-16. Therefore, the overall computational complexity for Algorithm 2 is:

$$O(|DAO| * |Q| * (\text{Compute}q_{dao_i}(t)) + |Q| + |\nu|))$$

The time complexity for the cost is almost negligible if number of variety of costs $|\nu| < |Q|$. If $|Q| < \text{Compute}q_{dao_i}(t)$ then the overall complexity would be:

$$O(|DAO| * |Q| * (\text{Compute}q_{dao_i}(t)))$$

$\text{Compute}q_{dao_i}(t)$ depends on the number of capabilities, the number of components implement that these capabilities and how they are connected. In our approach, we use Table 5.1 for the computation of specific quality attributes based on how the *DAO* are connected. However, our approach is flexible to include other types of quality attributes and different connections between them.

Second, Algorithm 3 executes the change detection for only one *dao* (line 1-10), so this would lead to a complexity of:

$$O(1)$$

Finally, Algorithm 4 has three main procedures which constitute to the time complexity: (i) Determine the list of non-dominated DAO ($O(|DAO|^2)$), line 2 and 4; (ii) Calculate marginal loss for the non-dominant DAO ($O((\text{\#of non-dominated}DAO)^2 * |\lambda|)$), line 10 and 11, and 13; (iii) Determine the expected marginal loss for the non-dominant DAO ($O(\text{\#of non-dominated } DAO * |\lambda|)$, line 16 and 17 ; and (iv) Determine the optimal dao_i from list of non-dominated DAO ($O(\text{\#of non-dominated } DAO)$), line 18.
In this context, the total time complexity for the run-time approach equals to:

$$O(|DAO|^2 + (\text{\#of non-dominated DAO})^2 * |\lambda|)$$

If $(\text{\#of non-dominated}DAO)^2 * |\lambda| < |DAO|^2$, then the time complexity would be approximately:

$$O(|DAO|^2)$$

From our empirical results, we can see that the run-time approach's scalability is consistent with its time complexity analysis.

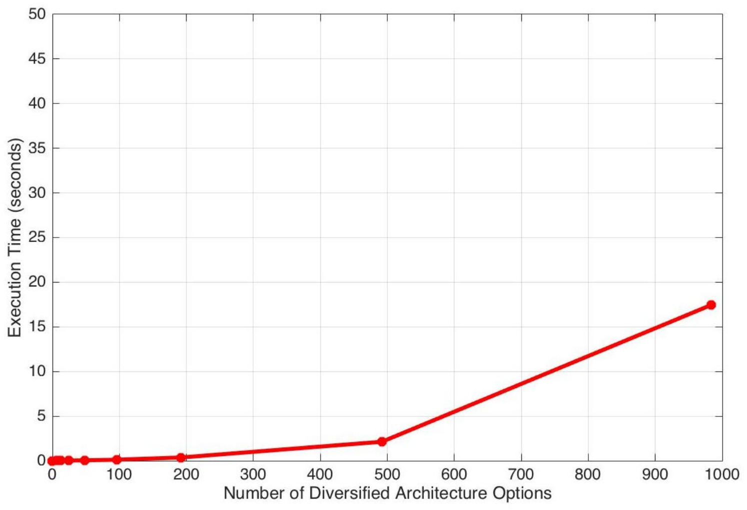


Figure 5.14 The execution time for large number of DAO for every timestep.

5.7.4 Impact Of The Parameter θ On The System's Stability

Motivation: A low θ value corresponds to a greater emphasis on the most recent observations of benefit and swifter adaptation to changes in the benefit. However, very low values can lead to unstable quantification of benefit, causing too much switching over time. A high value (e.g., larger than 0.95) places more importance to the past observations of benefit, leading to more stability, but potentially failing to track changes in benefit. This experiment will answer the following: *What is the impact of the parameter θ on the system's stability?*. In this context, this experiment is performed to indicate the most applicable value for the relative importance under typical environment settings.

Experimental setup: This experiment considers the values of $\theta \in \{0.7, 0.9, 0.99\}$. In this experiment, we started with dao_6, which is advocated by the architect to be the optimal *dao* for the normal quality constraints as depicted in Table 5.2. If a change is detected, the approach recommends another *dao* to be implemented. In this context, the exponential benefit of the selected *DAO* by the approach over 120 timesteps is computed and plotted in Figure 5.15.

Analysis: As seen in Figure 5.15, $\theta = 0.7$ leads to highly fluctuating benefit over time, which may cause the system to recommend a lot of switches throughout time, as compared with $\theta = 0.9$ (the moderate fluctuation) and $\theta = 0.99$ (too stable, which may not be realistic). For instance, $\theta = \{0.7, 0.9, 0.99\}$ has recommended the following number of switches {13, 4, 1}, respectively. Therefore, based on this experiment, we recommend for the architect to evaluate the four approaches for the next experiments, with respect to $\theta = 0.9$, as it indicates the most realistic forgetting factor.

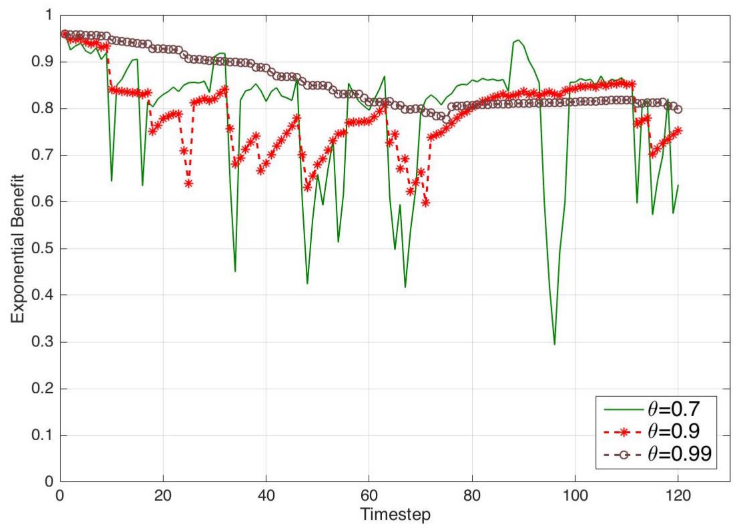


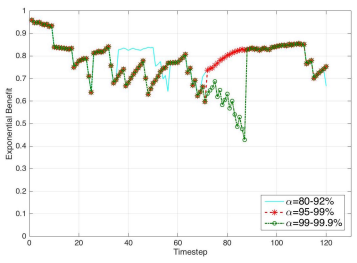
Figure 5.15 Exponential Benefit of the informed-selection approach using different values for the stability parameter θ.

5.7.5 Impact Of The Parameter α On The System's Stability

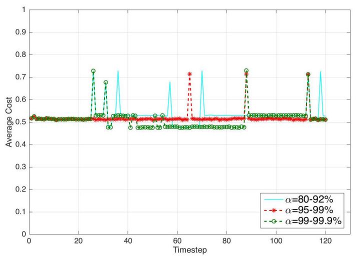
Motivation: The confidence interval is an interval with two boundaries (α_1 and α_2), where the architect is not confident about the exponential benefit values outside this interval. In other words, if the current exponential benefit $\mu_{dao_i}(t)$ is outside the left boundary of the confidence interval, this is an indicator that the current option is getting worse and requires replacement. For instance, If a system is highly sensitive to changes, then we can enlarge the α values used ($\alpha \geq 95\%$), whereas for the opposite case ($\alpha \leq 92\%$) is more applicable and so forth. Based on that, the architect can learn what α values to use and for which scenarios. In order to adjust the α values, there is a trade-off between the number of switches and overall benefit (i.e. stability versus improved benefit), which will be measured in this experiment. To this end, the experiment aims at answering the following: *What is the impact of the parameter α on the system's stability?*.
Experimental setup: We have developed two cases for analysis: *case 1* where the best *dao* changed over time, which caused several switches; *case 2:* the best *dao* performed well over time, resulting in no switches. This experiment considers the values of $\alpha_1, \alpha_2 \in \{80, 92\%\}; \{90, 95\%\}; \{99, 99.9\%\}$. Figure 5.16 shows the impact of varying α values on the approach's stability, whereas Table 5.3 summarises the corresponding number of change detection and switches. Figure 5.16a, 5.16d show the exponential benefit of *DAO* recommended by the informed-selection approach, whereas Figure 5.16b, 5.16e depict their corresponding cost. The mean exponential benefit and cost of 120 timesteps for varying α values are depicted in Figure 5.16c, 5.16f to show the overall behaviour of four approaches.

Analysis (Case 1): The use of low confidence interval {80,92%} caused higher change detection than other confidence intervals. This is due to the detection of unnecessary changes. So in a case where the *DAO* are highly changing, this caused higher number of switches (7), as depicted in Table 5.3 and Figure 5.16a. A high confidence interval {99,99.9%} can lead to neglecting significant changes, because the system is confident enough about the data. There is a trade-off between the number of switches and improvement in benefit. For case 1, the informed-selection recommended 3 additional switches (when $\alpha = \{80, 92\%\}$ over $\alpha = \{99, 99.9\%\}$), with 5% increase in exponential benefit and 0.5% increase in cost over {99,99.9%}. Although {95,99%} and {99,99.9%} advocated the same number of switches, yet {95,99%} produced a 3.6% increase in exponential benefit and 1% increase in cost over {99,99.9%}. This is because {99,99.9%} neglected significant changes, as stated before.

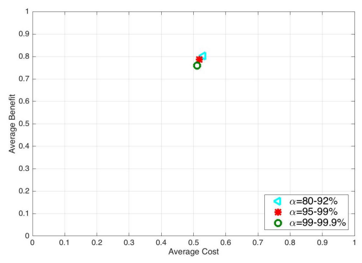
Analysis (Case 2): Though for {80,92%} the informed-selection detected 9 changes (Table 5.3), yet it managed to continue with the optimal option without any recommendation for switching. This is because it has found that there is no other *dao* better than the current one. Therefore, the informed-selection approach has a safety mechanism, which recommends a switch only if there is another better *dao*, otherwise it will keep using the same *dao*. Further, no change was detected for {95-99%} and {99-99.9%}, this explains why all the confidence intervals produced the same overall behaviour as seen in Figure 5.16f. Besides, the low confidence interval also caused higher overhead in terms of searching for another optimal option (informed-selection). In all cases, the informed-selection approach managed to recommend the optimal *dao* in terms of balancing between cost and benefit over time. To this regard, a confidence interval of {95-99%} is more applicable in most of the cases, because it detects significant changes and neglect unnecessary ones. We recommend the architect to use {95-99%} in the evaluation of the four approaches for the next experiments.



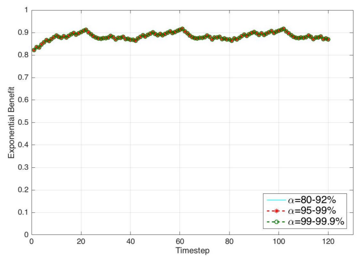
(a) Exponential Benefit (Case 1)



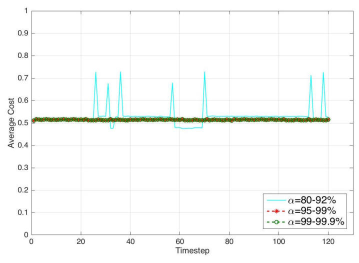
(b) Average Cost (Case 1)



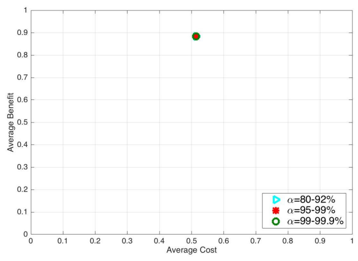
(c) Overall behaviour (Case 1)



(d) Exponential Benefit (Case 2)



(e) Average Cost (Case 2)



(f) Overall behaviour (Case 2)

Figure 5.16 The impact of varying α values on the informed-selection approach.

Table 5.3 The number of change detection and switching for informed-selection approach for varying α values.

Parameter/Case	1	2
# of switches($\alpha_1 = 80\%, \alpha_2 = 92\%$)	7	0
# of detection ($\alpha_1 = 80\%, \alpha_2 = 92\%$)	13	9
# of switches($\alpha_1 = 95\%, \alpha_2 = 99\%$)	4	0
# of detection ($\alpha_1 = 95\%, \alpha_2 = 99\%$)	10	0
# of switches($\alpha_1 = 99\%, \alpha_2 = 99.9\%$)	4	0
# of detection ($\alpha_1 = 99\%, \alpha_2 = 99.9\%$)	8	0

5.8 Illustration Of Some Insights The Approach Can Provide

This section provides extra benefits the approach could provide. For instance, it discusses and demonstrates the usefulness of diversification and whether it can always add value to the decision-making based on what-if scenarios.

Motivation: Design diversification has the potential to mitigate risks and improve the dependability in design in situation exhibiting uncertainty in operation and usage [167, 16]. Design diversity has raised the potential awareness to apply diversification in the decision-making process, which in turn may cause a noticeable improvement in the way we design dependable and evolvable software. However, do diversification continuously deliver value over time? In this experiment, we aim to evaluate the added value of diversification to the decision-making process.

Experimental setup: In this experiment, we are trying to show whether the inclusion of new *DAO* will benefit the decision-making. To demonstrate that, we have tested our approach with respect to two cases: (1) Add new six *DAO* where the approach benefits from them and a noticeable improvement in the overall behaviour occurred (Case 1); (2) Add new six *DAO* where the approach does not benefit from them (Case 2). We used the original six *DAO* recommended by the approach (Table 3.1) and created new six *DAO* adhering to the same topology of original six *DAO*, but with different QoS fluctuations to generate Cases 1 and 2.

For Case 1, we have randomly generated their QoS over time using the mean of original *DAO* and fluctuation of $5\% - 25\%$ for response time, $0\% - 15\%$ for energy consumption, and $20\% - 29\%$ for network usage. As for Case 2, the QoS fluctuation is $30\% - 40\%$ for response time, $25 - 40\%$ energy consumption, and $30\% - 40\%$ for network usage. These volatility values were chosen with respect to [215, 216]. In Case 1, the low fluctuation has resulted in additional *DAO* with improved aggregated benefit, whereas the newly created

DAO in Case 2 has generated *DAO* with highly fluctuating benefit (i.e. seems good at the beginning but then turns out to be worse after being selected).

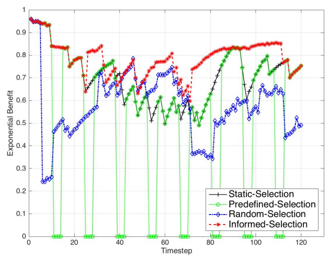
Analysis: In *Case 1*, we have added 6 *DAO*, which seemed to add value to the decision-making process. The overall average benefit and cost for the informed-selection approach was better than the other approaches (Figure 5.17g, 5.17h), with noticeable rise in exponential benefit over time (Figure 5.17a, 5.17b), and slight decrease in cost (Figure 5.17d, 5.17e). This is because the informed-selection approach has benefited from the inclusion of new *DAO*, by providing more stable behaviour in terms of benefit. For instance, after 10 timesteps the approach has detected a significant change in current option (i.e. dao_6). It has then selected dao_{12} instead, which seemed to provide more stable benefit (Figure 5.17b) with almost similar cost (Figure 5.17e) to the one chosen in default case (i.e. 6 *DAO* instead of 12 *DAO*).

In *Case 2*, the additional 6 *DAO* were highly fluctuating over time, which explains the slight degradation in benefit (Figure 5.17c) as compared to the original 6 *DAO* only (Figure 5.17a). However, the addition of new options has introduced further instability to the random-selection approach by selecting the worse *DAO* (4 more switches than the original case), as depicted in Table 5.4. For example, after 5 timesteps, the approach has detected a significant deviation in current option (i.e. dao_6). After that, it chose dao_{11}, which provided the most balanced trade-off between benefit and cost. The exponential benefit of dao_{11} was increasing, which explains why the approach did not detect any change. However, this has caused the application to not benefit that much from diversification. At $t = 71$, the approach detected a change and till $t = 120$, it has chosen similar *DAO* to the default case.

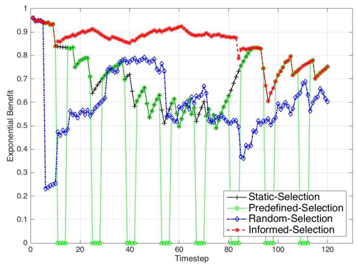
To this regard, the proposed approach successfully showed that diversification was helpful in Case 1, and was not helpful in Case 2. This concludes that diversification will not always add value and run-time evaluation could aid in assessing the worthiness of this exercise.

Table 5.4 The evaluation of embedding diversification to the architecture with respect to the number of change detection and switches.

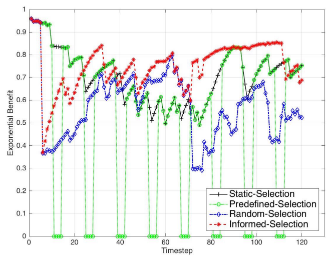
Parameter/Approach	Static -selection	Predefined -selection	Random -selection	Informed -selection
# of switches (6 *DAO*)	0	8	8	4
# of detection (6 *DAO*)	0	0	8	10
# of switches (12 *DAO*) [Case 1]	0	8	10	3
# of detection (12 *DAO*)[Case 1]	0	0	10	4
# of switches (12 *DAO*) [Case 2]	0	8	12	4
# of detection (12 *DAO*)[Case 2]	0	0	12	5



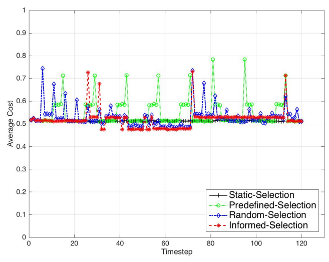
(a) Exponential Benefit (Default Case)



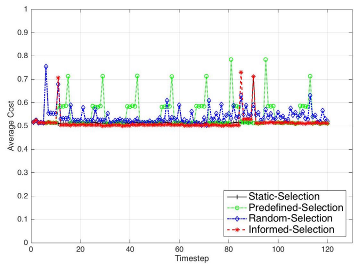
(b) Exponential Benefit (Case 1)



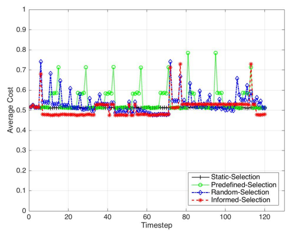
(c) Exponential Benefit (Case 2)



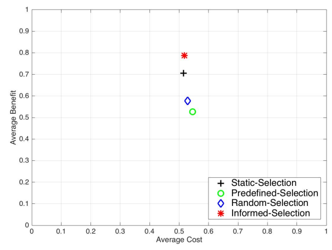
(d) Average Cost (Default Case)



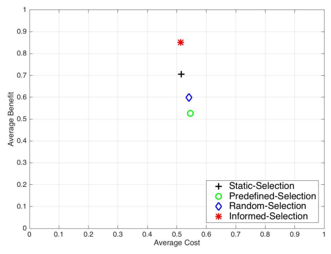
(e) Average Cost (Case 1)



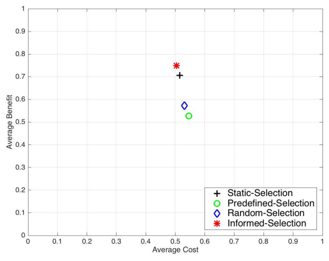
(f) Average Cost (Case 2)



(g) Overall behaviour (Default Case)



(h) Overall behaviour (Case 1)



(i) Overall behaviour (Case 2)

Figure 5.17 The evaluation of embedding diversification to the architecture on the decision-making (positive impact (Case 1) and negative impact (Case 2)) as compared to the default case (6 original *DAO* in Table 3.1).

5.9 Discussion

In this chapter, we aimed to answer the thesis's third research question **RQ3: How can run-time architecture evaluation using cost-benefit analysis and machine learning techniques complement the design-time one to handle uncertainty ?** Our reactive run-time evaluation approach allows reasoning about value added under uncertainty, facilitated by the use of reinforcement learning to profile the fitness values of options rather than the traditional (static) predictions of design-time decisions. The exponential decay factor provides architects with a visual demonstration of the benefit of options over time and aids them in complementing design-time decisions (Figure 5.8). Our approach also alerts architects of significant detrimental changes in the benefit of the option being employed, and highlights which of the candidate options provides the optimal cost-benefit trade-off when changes occur for normal and strict constraints (Figure 5.9). This could assist architects in the process of eliminating options with poor balance between benefits and costs over time. For instance, in the best and worst case scenarios in the experiment addressing **RQ3.2**, we found that one out of six options could potentially be eliminated because it was always worse than the others (i.e. the cloud-based option). There is a trade-off between monitoring the architecture options at every timestep and every T' intervals; our approach was able to show this trade-off to the architect (Figure 5.11, 5.12). Our approach is also robust when dealing with noise (Figure 5.13) and scalable to be applied in practical settings (Figure 5.14). Further, the exponential decay factor that we use, weights recent values more heavily, which yields more practical results (Figure 5.15). The approach uses the confidence interval to automatically tune the sensitivity of the approach to changes, which improves the system's stability (Figure 5.16). In summary, our approach allows the architect to determine the added value of embedding diversification in the architecture (Figure 5.17).

In a nut shell, the proposed run-time evaluation approach assesses the architecture design decisions through the following: (i) evaluating, complementing, and refining design-time decisions with run-time ones; (ii) tuning the relative importance of present/past data for knowledge accumulation about an architecture which can aid in learning the behaviour of architecture decisions over time; (iii) efficiently assessing the value of architecture decisions at different monitoring intervals; (iv) allowing the architect to tune the sensitivity of the approach to changes, and therefore how stable/unstable it will be over time.

5.10 Threats to Validity

In answering RQ3 (Chapter 5), we have run a number of experiments in a controlled environment. We discuss the threats related to internal and external validity in the context of these experiments.

5.10.1 Threats to Internal Validity

These are concerned with the impact of evaluation parameters on the proposed approach. For that, we have analysed our approach with varying input parameters (e.g. the relative importance of present/past, the confidence interval, *etc*) to ensure acceptable accuracy and stability. On the contrary, the values of these parameters might vary depending on some characteristics (e.g., the environmental conditions and more complex dependencies between architecture decisions), which could be investigated for future research. Further, one of the threats to internal validity is that the real environment may differ from the simulated data due to uncertainties at run-time. Therefore, we have tested the sensitivity of the approach to noise, to check how well our proposed approach can handle this issue. We have also demonstrated the applicability of our approach to monitor the environment at every T' intervals if the real-time evaluation was expensive and the architects did not wish to use simulation.

5.10.2 Threats to External Validity

These are linked to the run-time data synthesis and analysis used in the experiments. In particular, the notion of run-time evaluation is meant to contextualise dynamic and behavioural evaluation; such evaluation needs to be conducted by monitoring a running system; analysing historic data from a running system or using a simulated environment that mimics the behaviour of the running system to perform stress and what-if analysis for the performance of the architecture design decisions under changing environment and/or extreme scenarios. The results of such evaluations are time-dependent in non-stationary environments as it is the norm for systems such as IoT. Evaluation that is based on simulation can still be considered as design-time if the evaluation is performed at design-time and before deployment. However, we also see potential for the same simulated approach to work in parallel with the running system, with symbiotic feedback between the simulator and the running system to perform anticipatory evaluation of key design decisions and their possible variants based on the run-time context, which may be difficult without the aid of simulation.

Additionally, there will always be a trade-off between using the simulators and physical IoT devices in experimentation and data generation. This is due to the high cost of the actual deployment of IoT devices as compared to simulators. However, some companies, such as Amazon, IBM, and Intel, are motivating the need for having IoT simulation instrumenting what-if test scenarios, typically used during the architecture analysis and refinement stages to evaluate the response and sensitivity of the architecture to these tests.

To generalise, our approach can steer the evaluation process using simulated data; partially simulated data and/or using monitored run-time data. The simulation can be used to simulate what-if an option would be deployed. This is particularly useful if the architect would like to solicit an early assessment on the option, where the cost of deployment would be expensive and the outcome can not be verified with high confidence. Simulation can also be used to simulate the performance of some design decisions under worst and stress scenarios. Henceforth, simulation can be a cost-effective strategy to first assess the performance and then adapt, if the option deems to be sensible. Further, transfer learning methodology [217] has been recently adopted in [218], where the QoS measurements are taken from a simulator and only a few samples are taken from the real system leading to much lower cost and faster learning. In this context, the approach could learn from both simulated and run-time data, which is a subject for future work.

A major challenge in mobile crowdsensing is the resource constraint of the mobile devices. This can constrain the processing which can be done on the devices forming the *dao* in our approach. On the other hand, mobile devices that are not constrained in resources can have a high energy consumption to be able to perform on-device computations. High energy consumption is therefore another challenge to our approach. However, the new mobile devices are designed to handle processing tasks with acceptable energy consumption. The IoT environment is highly dynamic and characterised by hyper-connectivity, due to high refresh rates and continuous upgrades. More specifically, it is expected that nodes can leave and join; nodes can be subject to upgrades and replacements; some nodes could be replaced by inferior ones; some new nodes can share common characteristics with its predecessors, offering enhancements for some qualities. Though it would be difficult for the approach to predict all possible types of IoT devices and versioning that can be encountered in real settings, our approach assumes the evaluation of reference architectures for this setting, where commonalities and variabilities are analysed as part of the diversification procedure for incepting and evaluating the diversified architecture options.

5.11 Conclusion

In this chapter, we proposed a novel run-time architecture evaluation method suited for systems that exhibit uncertainty and dynamism in their operation, such as IoT. This method provides continuous assessment of design-time decisions. It can inform deployment, refinement and/or phasing out decisions. Specifically, we used strategies derived from machine learning and cost-benefit analysis at run-time to continuously profile and evaluate the architecture decisions for their added value. We demonstrated the use and significance of our approach by applying it to the case of designing diversification in an urban traffic monitoring IoT application to cater for the uncertainty in meeting quality requirements. Moreover, we evaluated our run-time approach with respect to baseline design-time approaches. Based on our experimental evaluation, our method could assist architects in evaluating design-time decisions at run-time, which could improve their decision-making process.

Additionally, the proposed reactive approach accumulates the run-time knowledge of the benefit of the architecture decisions. It then assumes that the best *dao* to switch to is the one which has recently been obtaining the most balanced cost-benefit. This is effective in some contexts, e.g., when the deployed architecture option is not violating the QoS constraints that much, or when its performance level is not significantly changing over time. However, in other contexts, this type of learning may suggest wrong decisions and recommend unnecessary switches due to the lack of knowledge about the future benefit of candidate architecture decisions. Taking into account the future potential of a diversified architecture option may provide a more informative evaluation. This calls for extending the following run-time approach with additional machine learning algorithms to anticipate the benefit of architecture decisions over time.

Chapter 6

Proactive Run-Time Architecture Evaluation

Context. Based on the results of Chapter 4 and 5, design-time evaluation and reactive run-time evaluation are not enough to evaluate complex and dynamic design decisions. They lack the ability to forecast the future benefits at run-time. Therefore, they could trigger unnecessary switches, which affect the architecture stability and result in increased costs. A proactive approach is necessary as a complementary component in a continuous architecture evaluation framework.
Objective. This chapter proposes the first *proactive* approach to continuous architecture evaluation.
Method. The approach evaluates software architectures by not only tracking their performance over time, but also forecasting their likely future performance based on machine learning. This enables architects to make informed decisions on potential changes to the architecture, so that its performance remains good over time, at the same time as unnecessary switches to different architecture decisions are avoided. We apply the proposed approach on *iTransport* to show how machine learning can fundamentally guide the continuous evaluation of software architectures and influence the outcome of architecture decisions. A series of experiments is conducted to demonstrate the applicability and effectiveness of the approach.
Results. Our experimental results show that the proactive approach was more advantageous than the design-time and reactive approaches, because it learns and forecasts, and hence makes smarter decisions. In some contexts, the proactive approach has contributed to a significant improvement in benefit: about 20-60% as compared to design-time approaches. In other contexts, it also produced the best results (i.e. 40-70% better benefit than reactive, state-of-the-art and baseline approaches).

Conclusion. Proactive approaches leveraging the time series forecasting analytics can fundamentally guide the continuous evaluation of software architectures and influence the outcome of the decisions made.

6.1 Introduction

In Chapter 4, we have used CBAM [9] along with options theory [26, 42] to evaluate the potential of different architecture options at design-time, where the architecture options providing high option value are then considered for deployment. However, as the environment is dynamic, value potentials can fluctuate at run-time. This led to our subsequent work in Chapter 5, where we have proposed a reactive approach for evaluating software architectures at run-time, using techniques inspired by reinforcement learning [27]. This approach monitors architecture design decisions with respect to some quality attributes of interest. If the adequacy of a given decision starts to deteriorate, the approach then reacts by suggesting possible refinements or changes of the decisions.

Meanwhile, modern developments practices have given rise to harvesting operational data (e.g. QoS data), learning and analysing data, to continually feedback to the development to refine, tune, and enhance architecture design decisions. The availability of this data has provided new opportunities for continuous evaluation, whether they are real-time or simulated ones. It could potentially: (i) aid the architect in continuously learning about architecture decisions; (ii) complement design-time decisions; (iii) help in forecasting how well they will behave in the future; (iv) proactively deal with variability scenarios.

Though the approach in Chapter 5 continuously helps the architect in understanding the past behaviour of the architecture decisions in question, such a reactive approach is limited. Even though it can monitor past observed performance of architecture decisions, it cannot proactively reason about their future potentials. Meanwhile, a decision that has worked well in the past may not necessarily work well in the future, given the potential changes and uncertainty underlying the environment where the system is embedded. Reactive approaches may cause the current decision to seem poor, even though it could become attractive in the near future. This could trigger unnecessary architecture adaptations that would cause the system to be unstable. Conversely, a decision that may not seem attractive based on the past could have better future potentials. Discarding that decision could lead to poor future performance. In summary, reactive approaches can lead to partially justified decisions, unnecessary adaptation, and increase development cost, while slowing down the operations. Existing work has only exploited data to provide benefit (i) mentioned in the previous paragraph, and partially provide benefit (ii). The full power of data has yet to be harnessed to provide benefits (i)-(iv), which are essential to better inform software architecture decisions at run-time.

This chapter thus, provides the first investigation of proactivity in continuous software architecture evaluation. It exploits machine learning in the form of time series forecasting analytics to produce a more powerful continuous architecture evaluation approach, able to provide benefits (i)-(iv), which we contribute to the research literature.

Overall, this chapter answers the thesis's fourth research question: **RQ4: How can the use of forecasting analytics improve the state of run-time architecture evaluation?** This research question is further divided into following:

- **RQ4.1: How can proactivity in continuous evaluation be realised and conducted? For instance, can continuous time series forecasting analytics enable us to predict the behaviour of software architectures over time and, if so, how well?**

- **RQ4.2: How can proactive approaches complement reactive ones to provide a well-rounded and more effective continuous architecture evaluation? What benefits can they bring to continuous software architecture evaluation and decision-making at run-time?**

To answer these RQs, this chapter proposes a novel approach to continuous software architecture evaluation. It uses continuous time series forecasting [30–32] as a built-in mechanism to complement reactive run-time evaluation (Chapter 5) with proactive run-time evaluation. In this way, the proposed approach not only takes into account the past performance of architecture decisions, but also their future potentials, better informing run-time architecture decisions.

A series of experiments were conducted to demonstrate the applicability and effectiveness of this approach using the case of architecting for IoT (i.e. *iTransport*). The suitability of various *time series forecasting* algorithms [30–32] to realise proactivity is investigated under this case study (RQ4.1). The impact of proactivity on run-time decision-making is then evaluated through a comparative study against design-time and reactive continuous evaluation approaches (RQ4.2). Simulated run-time and operations data [35] were used to test the potentials of the approach on a wider spectrum of scenarios and cases (similar to the data used in Chapter 5). These experiments also provide software architects with systematic guidance on how the approach can be realised in practice given a wide set of scenarios and data capabilities for forecasting (i.e. how to select forecasting algorithm, how to best benefit from forecasts, *etc*).

Our results show that the proactive approach was more advantageous than the design-time and reactive approaches in some important scenarios. In particular, we have set two scenarios with different QoS constraints (i.e. the target QoS should not exceed a particular threshold) based on the context and the stakeholders requirements. For normal constraint scenarios, the proactive approach provided similar overall behaviour to the reactive approaches, but with improved system stability (i.e. smaller number of switches). It also contributed to a significant improvement in benefit: about 20-60% as compared to design-time approaches. For most of the scenarios where architects have set strict constraints, it also produced the best results (i.e. 40-70% better than reactive and baseline approaches).

Our novel contributions are:

- The first investigation of whether and how time series forecasting can be used to successfully forecast the future benefit of architecture decisions.
- The first proactive approach to continuous software architecture evaluation. The approach uses the power of forecasting analytics to complement design-time decisions by taking into account not only the past, but also the future potentials of architecture decisions.
- A detailed analysis of when and why the proposed approach can help to better inform architecture decisions.
- Experimental guidelines aiding architects on how to tune the proposed approach.

The remainder of this chapter is structured as follows: Section 6.2 discusses the necessary background to understand the proposed approach. Section 6.3 then presents the proposed proactive approach. Section 6.4 provides a comprehensive and comparative evaluation. Section 6.5 provides a further discussion on the method, whereas Section 6.6 discusses threats to validity. Section 6.7 concludes the work.

6.2 Background

A time series is a sequence of observations (e.g. response time, energy consumption, *etc*) measured/received sequentially over time [219]. Time series forecasting consists in forecasting the value of a future observation (output attribute), based on the values of a number of previous observations (input attributes) in the sequence. In this work, we will consider that, once the true value of the future observation becomes known, it can be used to create a training example (i.e., an example where both the input and output attributes are already known) for updating the current forecasting model. After that, this training example is discarded. The learning procedure where forecasting models are updated over time whenever a new observation arrives is called online learning, and is adequate when data arrive at high speed, as is the case with IoT.

In this section, we will briefly explain two types of online learning algorithms, for stationary and non-stationary environments. More detailed explanation on how the forecasting model is built and trained is given in Section 6.3.1.

6.2.1 Online learning models for stationary environments

Learning models for stationary environments assume that the true function underlying the relationship between input attributes and forecast values does not change over time. The online learning models for stationary environments investigated in this work are: k-Nearest Neighbours (kNN), Perceptron, and Stochastic Gradient Descent (SGD) Multiclass.

6.2.1.1 k-Nearest Neighbours (kNN)

It is a type of lazy learning [220], where all the training and learning of examples is performed when the forecasts are required. The algorithm uses a function that computes the output attribute (i.e. forecast values) using the mean of all the output values of its k nearest neighbours. For instance, if k=3, then the output attribute equals the average of the output values of its 3 nearest neighbours. The nearest neighbours are usually determined based on the Euclidean distance between the input attributes of the example to be predicted and a training example.

6.2.1.2 Perceptron

This is a type of neural network approach [221], which trains a single neuron to output a forecast. The neuron has weights associated with each input attribute. It provides a forecast by computing a function that receives as input the weighted average of the input attribute values. The weights start with random values, and are then iteratively updated based on incoming training examples, so as to minimise the forecasting error. The size of the updates is controlled by a parameter called learning rate.

6.2.1.3 Stochastic Gradient Descent (SGD) Multiclass

SGD is an online learning algorithm that has shown its success for large-scale learning and forecasting [222–224]. The aim of SGD is to minimise the forecasting errors when training large numbers of examples. To this regard, it employs a differential error function for optimisation. However, in other cases, it also uses non-differential functions to adjust the error within a certain margin (i.e. particular threshold). It is different from classical gradient descent which selects all the training examples as a batch, instead it randomly nominates an example for training (i.e. stochastic) [225]. In particular, it iteratively accepts a training example and then amends the forecasting model through gradient descent over the corresponding instantaneous objective function [226]. The SGD MultiClass is an extension of SGD, which is mainly used for solving multiclass classification and regression problems [32]. It has the ability to perform online training of various models [226], such as support vector machine (SVM), logistic regression, and linear regression. It also uses a learning rate for the same purpose as the Perceptron. In our case, we have used SVM as a learning model. Because it adopts an efficient and special type of loss function named hinge, which aids in providing better forecasts than other models (i.e. output is more accurate) [227]. The hinge loss is a non-differential loss function, which is known to be lazy [32]. This implies that the forecast model parameters are only updated if the training example violates the margin constraint rather than when every example which arrives. Therefore, this improves the efficiency of the SVM [227]. Further details related to SGD Multiclass is found in [32, 227].

6.2.2 Online learning models for non-stationary environments

Learning models for non-stationary environments assume that the true function underlying the relationship between input attributes and forecast values may change over time, requiring specific mechanisms for models to update to such changes. The algorithms for non-stationary environments investigated in this work are: Fast Incremental Model Trees with Drift Detection (FIMTDD), Fading Target Mean (FTM) and Additive Experts (AddExp) [32].

6.2.2.1 Fast Incremental Model Trees with Drift Detection (FIMTDD)

This is an online machine learning algorithm that uses model trees for forecasting [228]. In particular, it incrementally splits the training examples across the leaf nodes based on the input attribute values with respect to some statistical analysis. The algorithm uses change detection tests, which are updated with every example. These change detection tests are used to determine whether there is evidence that the true function underlying the relationship between input attributes and forecast values is changing, based on the forecasting errors obtained by the tree. If a change is detected, the tree structure is modified with respect to an alternate tree adaptation strategy [228]. This strategy determines the node that is likely to be involved in the change, and labels it as requiring re-growing. A new alternate tree is constructed at this node in parallel with old one. Whenever, a new example arrives at a labelled node, the algorithm will employ it to grow both the alternate and old trees. The new alternate tree will replace the old tree once it becomes more accurate. FIMTDD has three main parameters that affect the forecast error: (1) the type of tree adopted, where a regression tree is found to provide better results than model tree; (2) the tie threshold, which is a threshold below which a split will be forced to break ties; and (3) learning rate, which is used for training Perceptrons in leaves (i.e. explained in Section 6.2.1.2). More information related to FIMTDD is found in [228, 32].

6.2.2.2 Fading Target Mean (FTM)

This learning algorithm returns a forecast which is the mean of the target of the training instances [32]. It also uses a fading (forgetting) factor [229] that gives more emphasis to more recent target mean. This is based on the simple assumption that the importance of examples reduces with their age, and enables the approach to adapt to changes. The fading factor can be tuned based on the need to emphasise the most recent versus the past target mean. Further details related to FTM is found in [229, 32].

6.2.2.3 Additive Ensemble (AddExp)

All the previous learning models are considered *single learners*, whereas there are other algorithms which leverage a group of learners. The latter are named *ensemble learners* [30]. AddExp [31] is an ensemble learner. Each learner is assigned a weight and produces a forecast for the current example. At every timestep, the AddExp algorithm gets the models' forecasts and provides a forecast which is computed as the weighted average of all models' forecasts. The weight of each expert is then updated based on its forecasting error. Errors on more recent examples are emphasised over old examples. The error of the ensemble forecast is also computed over time and, if it is larger than a certain threshold, a new model is added to the ensemble. This, together with the weight associated to each model, enables the ensemble to adapt to changes in the underlying function of the problem. Finally each expert is trained over time with respect to the available training data.

6.3 Proactive Architecture Evaluation Approach

The proposed continuous evaluation approach intertwines run-time evaluation with design-time evaluation for more informed decision-making. It builds on the reactive run-time approach by adopting forecasting analytics as depicted in the forecast and learn module from Figure 6.1. The use of forecasts makes the approach proactive, because it does not only rely on reacting to potential changes in the benefit of architecture decisions over time, but attempts to forecast them. In this way, architecture decisions could be made before negative changes in benefit become detrimental.

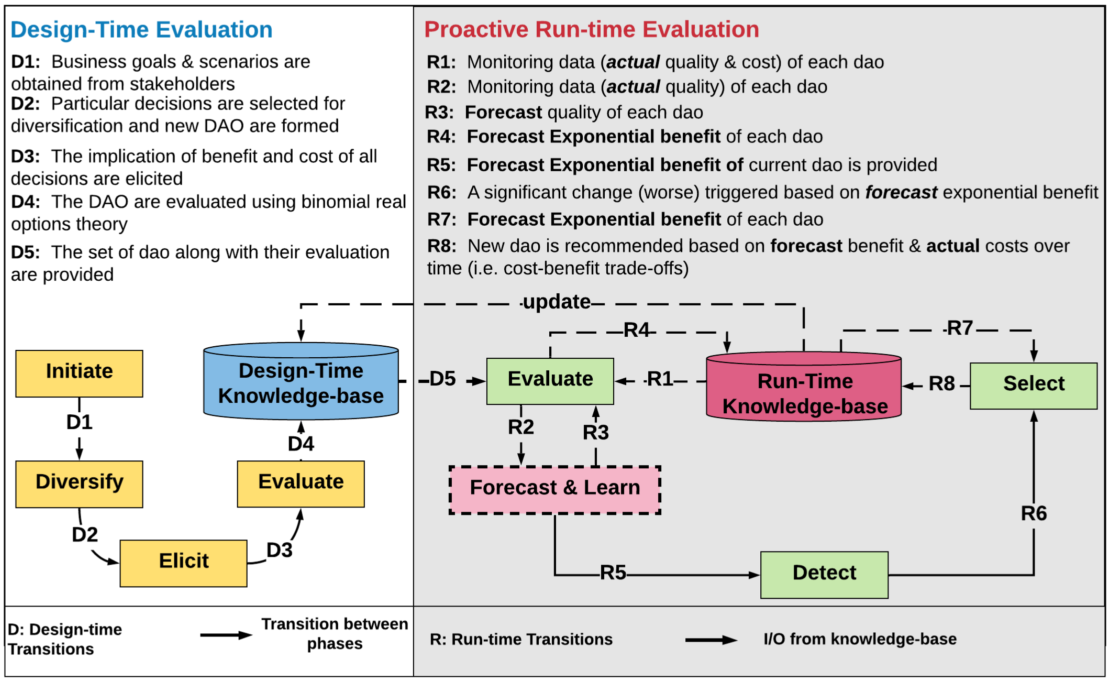

Design-Time Evaluation
D1: Business goals & scenarios are obtained from stakeholders
D2: Particular decisions are selected for diversification and new DAO are formed
D3: The implication of benefit and cost of all decisions are elicited
D4: The DAO are evaluated using binomial real options theory
D5: The set of dao along with their evaluation are provided
Proactive Run-time Evaluation
R1: Monitoring data (actual quality & cost) of each dao
R2: Monitoring data (actual quality) of each dao
R3: Forecast quality of each dao
R4: Forecast Exponential benefit of each dao
R5: Forecast Exponential benefit of current dao is provided
R6: A significant change (worse) triggered based on forecast exponential benefit
R7: Forecast Exponential benefit of each dao
R8: New dao is recommended based on forecast benefit & actual costs over time (i.e. cost-benefit trade-offs)
update
R4
R7
Initiate
D1
Diversify
D2
Elicit
D3
Evaluate
D4
Design-Time Knowledge-base
D5
Evaluate
R1
Run-Time Knowledge-base
R8
Select
R2
R3
Forecast & Learn
R5
Detect
R6
D: Design-time Transitions
Transition between phases
R: Run-time Transitions
I/O from knowledge-base

Figure 6.1 Steps of the continuous evaluation approach, where the design-time evaluation (Chapter 4) forms the initial design decisions and proactive run-time evaluation complements it by adopting time series forecasting.

The approach tracks and evaluates the actual benefit and cost using exponential decay factors (Section 5.4.4). The benefit is then modelled based on quality attribute forecasting models (Section 6.3.1). Then, instead of detecting significant deviations based only on previous benefit values, the proactive approach considers their *future* potentials (Section 6.3.2). The same applies to the selection of architecture options having the optimal cost-benefit trade-off, which is also based on their forecast benefits (Section 6.3.3). This is done to avoid critical problems, such as reactively recommending an architecture design option which does not introduce added value over time, triggering unnecessary adaptations and hence leading to an unnecessary increase in costs. The future potentials are assessed based on *time-series forecasting approaches* [30, 32], which forecast the future benefits of diversified architecture options based on previous quality attribute values observed. The *Forecast and Learn* step is explained in Section 6.3.1. Figure 6.2 illustrates the operational procedures performed by the proactive run-time evaluation approach along with stakeholders and architects involvement.

6.3.1 Forecast and Learn

The forecasting model learns a function that receives as input previous quality attribute values monitored by the Evaluate step (Section 5.4.4), and outputs a forecast of the future value of the quality attribute. One forecasting model is produced for each quality attribute. We refer to a point in time when a new quality attribute value has been observed as a timestep. At each timestep t, the following two main operations are run, for each quality attribute:

1. **Learn:** a new training example is produced if enough quality attribute values have been observed to compose it. This training example is used to train (update) the forecasting model corresponding to this quality attribute. A training example is composed of ζ input attributes, where ζ is a pre-defined parameter, and one output attribute. The input attributes are past quality values $(q'_{dao_i}(t'-h-\zeta-1), q'_{dao_i}(t'-h-\zeta), \cdots, q'_{dao_i}(t'-h-1), q'_{dao_i}(t'-h))$, where h represents the number of timesteps in the future for which we wish a prediction to be made. The output attribute is the quality value that we want to learn how to predict $(q'_{dao_i}(t'))$. Therefore, a training example can be seen as a tuple $<q'_{dao_i}(t'-h-\zeta-1), q'_{dao_i}(t'-h-\zeta), \cdots, q'_{dao_i}(t'-h-1), q'_{dao_i}(t'-h), q'_{dao_i}(t')>$

2. **Forecast:** a forecast $\hat{q}_{dao_i}(t+h)$ is provided based on the forecasting model corresponding to the quality attribute. The forecast is made by feeding the past ζ observed quality attributes $(q'_{dao_i}(t-\zeta-1), q'_{dao_i}(t-\zeta), \cdots,, q'_{dao_i}(t-1), q'_{dao_i}(t))$ as inputs to the forecasting model.

Table 6.1 illustrates the training examples and forecasts produced at each timestep for a given quality attribute. In this illustrative example, the quality value observed in a given timestep t has the same numeric value as the timestep itself, so that the example is easier to follow. However, in real scenarios, the quality value received at a given timestep t does not necessary have the value t.

Suppose that the number of input attributes ζ used for forecasting is 4 and that we wish to forecast $h = 3$ timesteps in the future. Therefore, the algorithm will require 4 past quality values ($q'_{dao_i}(t-3)$, $q'_{dao_i}(t-2)$, $q'_{dao_i}(t-1)$, $q'_{dao_i}(t)$) as input attributes to provide a forecast $\hat{q}_{dao_i}(t+3)$.

A given training example $< q'_{dao_i}(t-h-\zeta-1), q'_{dao_i}(t-h-\zeta), \cdots, q'_{dao_i}(t-h-1), q'_{dao_i}(t-h), q'_{dao_i}(t) >$ can only be produced at a given timestep t if all its input and output attribute values have already been observed. Therefore, the first training example used to train the forecasting model can only be produced at timestep $t = 7$. This also means that the first forecast by the model can only be provided at timestep $t = 7$. Before that, even though new quality values were being observed and stored to compose training examples, no complete training example had been produced yet. So, the forecasting model could not be built. From timestep $t = 7$ onwards, the number of training examples produced by the approach increases by 1 at each timestep. Each new training example produced by the approach can be used to train (update) the forecasting model, so that it will produce improved forecasts over time. Even though it is possible to provide forecasts from timestep $t = 7$ on wards, it may be desirable to wait to start providing forecasts only after a pre-defined number of training examples η_{min} has been produced. This is because a forecasting model trained on too few examples may not perform well, and we would not have enough training examples to estimate its forecasting ability either.

Each forecasting model is built and updated based on a given machine learning algorithm. This algorithm may or may not be the same for different forecasting models. The choice of which machine learning algorithm to use for each quality attribute depends on which algorithm is able to provide better forecasts for that quality attribute. Our work investigates the use of two types of machine learning algorithms: (i) online learning models for stationary environments; and (ii) online learning models for non-stationary environments [30]). Online learning means that the models are updated over time based on new training examples produced over time. Learning algorithms for stationary environments assume that the true function underlying the relationship between input attributes and forecast values does not change over time. Learning algorithms for non-stationary environments assume that this function may change, therefore adopting specific mechanisms to swiftly update forecasting models to such changes. More details on how that is achieved can be found in Section 6.2. We investigate the suitability of different learning algorithms to forecast quality attributes in Section 6.4.

A more detailed explanation of the steps followed by the proposed approach is provided below with respect to Algorithm 5. This algorithm is run for each quality attribute q.

1. **Provide the input parameters:** The architect will enter the following parameters to initialise the forecasting procedures: q (quality value q for which the forecasting model

Table 6.1 An Illustration of the Process of Creating Training Examples to Train the Forecasting Models. The table illustrates what training example is produced at each timestep, and what forecast can be provided at each timestep, when $\eta_{min} = 1$. The value of "-" in a cell represents the absence of a certain input attribute or forecast at a given timestep. A training example can only be created when the true value of all of of its input and output attributes has already been observed.

Current Timestep (t)	# Training Examples	Training Example Generated					Timestep Being Forecast
		Input attributes				Output Attribute	
		$q'_{dao_i}(t-6)$	$q'_{dao_i}(t-5)$	$q'_{dao_i}(t-4)$	$q'_{dao_i}(t-3)$	$q'_{dao_i}(t)$	$t+3$
1	0	-	-	-	-	1	-
2	0	-	-	-	-	2	-
3	0	-	-	-	-	3	-
4	0	-	-	-	1	4	-
5	0	-	-	1	2	5	-
6	0	-	1	2	3	6	-
7	1	1	2	3	4	7	10
8	2	2	3	4	5	8	11
9	3	3	4	5	6	9	12
10	4	4	5	6	7	10	13

is being used), $q'_{dao_i}(t)$ (normalised quality values for each dao over time), ζ (number of input attributes required for forecasting), h (number of further ahead timesteps), η_{min} (minimum number of training examples to start using the forecasting models), and τ_q (error threshold for using the model). The two latter parameters are used to prevent using poor forecasting models. In particular, if the number of training examples used to build the forecasting models is too small, these models are likely to perform poor forecasts, and the estimation of their error will not be reliable either. Therefore, none of the forecasting models is used before being trained on η_{min} examples. In this case, the approach will behave reactively, as in Chapter 5. Once η_{min} training examples have become available, if the estimated error of a given forecasting model for quality attribute q is above τ_q, this model is not used at the current timestep and the approach behaves reactively for this quality attribute. The architect has the flexibility to adjust the prior parameters. The impact of such parameters will be investigated in Section 6.4.

2. **Create training example:** If enough past quality attributes have been stored in the queue (line 6), a new training example will be created (line 7).

3. **Evaluate the forecasting model:** The current training example is used to update the estimate of the predictive performance of the forecasting model. Therefore, the forecasting model is first requested to provide a forecast for the current value of the quality attribute (line 10). As the current value of the quality attribute is known, we

can check how large the error of the forecast is by computing $e = \hat{q}_{dao_i}(t) - q'_{dao_i}(t)$ (line 11). Therefore, we can use this error to update the error average incurred by the forecasting model, based on some error metric (line 12). This is going to be used later on to decide whether to adopt the forecasts provided by this model.

The error metric used in this work is the Root Mean Squared Error, shown in equation 6.1.

$$e_q(t) = \sqrt{\frac{\sum_{t'=1}^{t} (\hat{q}_{dao_i}(t') - q'_{dao_i}(t'))^2}{\eta}} \tag{6.1}$$

where η is the number of training examples received so far. The error on the current training example (line 11) can be used to update $e_q(t)$ incrementally if desired. In this way, there is no need for storing all previous errors.

4. **Train the forecasting model:** The new training example $\{q'_{dao_i}(t-h-\zeta-1), q'_{dao_i}(t-h-\zeta), \cdots, q'_{dao_i}(t-h-1), q'_{dao_i}(t-h), q'_{dao_i}(t)\}$ is used to train its corresponding forecasting model (line 13).

5. **Provide a forecast:** If the error average incurred by the forecasting model is below the threshold τ_q and the number of training examples used to train the forecasting model is larger than η_{min} (line 14), the forecasting model is used to produce a forecast $\hat{q}_{dao_i}(t+h)$ (line 15-16). Otherwise, the algorithm returns *null* (line 17).

   When a forecast is produced, it is used to update the estimated benefit ($\hat{B}_{dao_i}(t)$), which is computed as in equation 6.2. In this context, $q^*_{dao_i}(t)$ is equal to $\hat{q}_{dao_i}(t+h)$. Otherwise, if a forecast is not produced, the approach behaves reactively and uses the actual normalised quality value instead (i.e. $q^*_{dao_i}(t) = q'_{dao_i}(t)$).

$$\hat{B}_{dao_i}(t) = \sum_{q \in Q} w_q * q^*_{dao_i}(t) \tag{6.2}$$

   The approach also allows the architect to use the average forecasts of $t+h$ instead of using a single value of h. For instance, if $h \in \{1,5,8,15\}$, then $q^*_{dao_i}(t)$ will be equal to the mean of $\hat{q}_{dao_i}(t+h)$ for varying h values. The use of average forecasts may provide more realistic results than single h, as it considers a window of time in the future, rather than a specific point in the future. This potential advantage of using the average of the forecasts will be investigated in Section 6.4.3.

ALGORITHM 5: ForecastAndLearn()

Input: quality attribute q for which to forecasting model is being trained, normalised quality value $q'_{dao_i}(t)$ observed at time t, number of input attributes ζ, number of further ahead timesteps h, minimum number of training examples for enabling forecasts η_{min}, error threshold per quality attribute τ_q

Output: forecast quality $\hat{q}_{dao_i}(t)$

```
1  if this is the first call to ForecastAndLearn then
       {queue stores past observations of quality values and $\eta$ keeps track of the number
        of examples produced by the algorithm. Their values persist over future calls of
        ForecastAndLearn. }
2      Initialise $Model[q].queue$ to empty.
3      Initialise $Model[q].\eta = 0$.
4      Initialise $Model[q].error = 0$
   end
5  Add $q'_{dao_i}(t)$ to $Model[q].queue$.
6  if $Size(Model[q].queue) == \zeta$ then
7      Create new training example
        $< q'_{dao_i}(t-h-\zeta-1), q'_{dao_i}(t-h-\zeta), \cdots, q'_{dao_i}(t-h-1), q'_{dao_i}(t-h), q'_{dao_i}(t) >$.

8      $Model[q].\eta = Model[q].\eta + 1$
9      Delete first element of $Model[q].queue$.
       {Evaluate the forecasting model by forecasting the output of the training example }
10     $\hat{q}_{dao_i}(t) = Model[q].Forecast(h, \{q'_{dao_i}(t-h-\zeta-1), q'_{dao_i}(t-h-$
        $\zeta), \cdots, q'_{dao_i}(t-h-1), q'_{dao_i}(t-h)\})$
11     $e = \hat{q}_{dao_i}(t) - q'_{dao_i}(t)$
12     Use $e$ to update the root mean squared error $Model[q].error$ (Equation 6.1).
       {Train the forecasting model }
13     $Model[q].Train(<$
        $q'_{dao_i}(t-h-\zeta-1), q'_{dao_i}(t-h-\zeta), \cdots, q'_{dao_i}(t-h-1), q'_{dao_i}(t-h), q'_{dao_i}(t) >)$
   end
   {Provide a forecast }
14 if ($Model[q].error \leq \tau_q$ & $Model[q].\eta \geqslant \eta_{min}$ ) then
15     $\hat{q}_{dao_i}(t+h) = Model[q].Forecast(h, \{q'_{dao_i}(t-\zeta), q'_{dao_i}(t-\zeta+1), \cdots, q'_{dao_i}(t-$
        $2), q'_{dao_i}(t-1)\})$
16     Return $\hat{q}_{dao_i}(t+h)$
   else
17     Return $null$.
   end
```

Further, the approach provides the ability to check quality constraints' violation. If $q^*_{dao_i}(t)$ violates the constraint, equation 6.2 will be used to compute estimated benefit $\hat{B}_{dao_i}(t)$, otherwise $\hat{B}_{dao_i}(t)$ is set to zero.

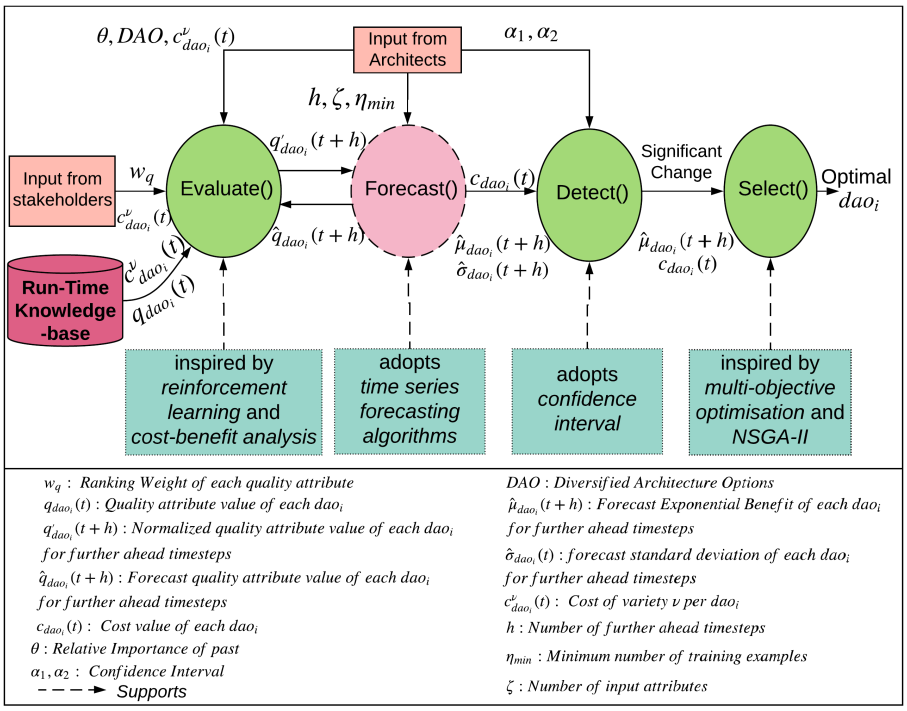

$\theta, DAO, c^{v}_{dao_i}(t)$
Input from Architects
α_1, α_2
h, ζ, η_{min}
$q'_{dao_i}(t+h)$
Input from stakeholders
w_q
Evaluate()
Forecast()
$c_{dao_i}(t)$
Detect()
Significant Change
Select()
Optimal dao_i
$c^{v}_{dao_i}(t)$
$\hat{q}_{dao_i}(t+h)$
$\hat{\mu}_{dao_i}(t+h)$
$\hat{\sigma}_{dao_i}(t+h)$
$\hat{\mu}_{dao_i}(t+h)$
$c_{dao_i}(t)$
Run-Time Knowledge -base
$c^{v}_{dao_i}(t)$
$q_{dao_i}(t)$
inspired by reinforcement learning and cost-benefit analysis
adopts time series forecasting algorithms
adopts confidence interval
inspired by multi-objective optimisation and NSGA-II
w_q : Ranking Weight of each quality attribute
$q_{dao_i}(t)$: Quality attribute value of each dao_i
$q'_{dao_i}(t+h)$: Normalized quality attribute value of each dao_i for further ahead timesteps
$\hat{q}_{dao_i}(t+h)$: Forecast quality attribute value of each dao_i for further ahead timesteps
$c_{dao_i}(t)$: Cost value of each dao_i
θ : Relative Importance of past
α_1, α_2 : Confidence Interval
Supports
DAO : Diversified Architecture Options
$\hat{\mu}_{dao_i}(t+h)$: Forecast Exponential Benefit of each dao_i for further ahead timesteps
$\hat{\sigma}_{dao_i}(t)$: forecast standard deviation of each dao_i for further ahead timesteps
$c^{v}_{dao_i}(t)$: Cost of variety v per dao_i
h : Number of further ahead timesteps
η_{min} : Minimum number of training examples
ζ : Number of input attributes

Figure 6.2 An overview of the operational procedures of proactive run-time evaluation approach.

6.3.2 Detect Based on Forecast and Learn

The same procedure as in Section 5.4.5 is applied here, except that the change is detected with respect to the forecast benefits rather than the actual ones. The forecast benefit ($\hat{B}_{dao_i}$), forecast exponential benefit ($\hat{\mu}_{dao_i}$), variance ($\hat{\sigma}^2_{dao_i}(t)$), and standard deviation ($\hat{\sigma}_{dao_i}(t)$) are calculated in the same way as in Section 5.4.4. However, we use the forecast quality values $\hat{q}_{dao_i}(t)$ rather than actual $q'_{dao_i}(t)$ here.
Therefore, they are computed as follows:

$$\hat{\mu}_{dao_i}(t) = \theta\hat{\mu}_{dao_i}(t-1) + (1-\theta)\hat{B}_{dao_i}(t) \tag{6.3}$$

$$\hat{\sigma}^2_{dao_i}(t) = \theta\hat{\sigma}^2_{dao_i}(t-1) + (1-\theta) * (\hat{B}_{dao_i}(t) - \hat{\mu}_{dao_i}(t))^2 \tag{6.4}$$

$$\hat{\sigma}_{dao_i}(t) = \sqrt{\hat{\sigma}^2_{dao_i}(t)} \tag{6.5}$$

The architect also has the flexibility to adjust θ, α_1 and α_2 (Figure 6.2) with respect to required accuracy and stability similar to the reactive approach, as mentioned earlier in Section 5.4.4. Algorithm 6 depicts the steps of change detection with respect to forecast values. At each timestep, the approach updates the maximum forecast exponential benefit $\hat{\mu}^{max}_{dao_i}$ (line 2-3) and its corresponding exponential standard deviation $\hat{\sigma}^{max}_{dao_i}$ (line 4). After that, the approach checks if a change/warning is detected. If the approach detects a significant change (line 5-6), the maximum forecast exponential benefit and its corresponding standard deviation are reset (line 7). In this context, the select function will then search for an optimal *dao* based on the forecast exponential benefit (Section 6.3.3). The approach also has the ability to just trigger a warning to the architect based on the α value chosen (line 8-9). Otherwise, the approach indicates that neither a change nor a warning were detected (line 10).

ALGORITHM 6: DetectBasedOnForecastAndLearn()

Input: confidence intervals (α_1, α_2), forecast exponential benefit $\hat{\mu}_{dao_i}(t)$, standard deviation $\hat{\sigma}_{dao_i}(t)$ based on forecast benefit
Output: change/warning/no change is detected
The red parameters denote the differences with respect to the reactive approach

```
   {Initialise the maximum forecast exponential benefit and its corresponding standard
    deviation}
1  $\hat{\mu}^{max}_{dao_i} = \hat{\mu}_{dao_i}(0)$, $\hat{\sigma}^{max}_{dao_i} = \hat{\sigma}_{dao_i}(0)$
   {Update the maximum forecast exponential benefit and its corresponding forecast
    exponential standard deviation}
2  if $\hat{\mu}_{dao_i}(t) > \hat{\mu}^{max}_{dao_i}$ then
3      $\hat{\mu}^{max}_{dao_i} = \hat{\mu}_{dao_i}(t)$
4      $\hat{\sigma}^{max}_{dao_i} = \hat{\sigma}_{dao_i}(t)$
   end
   {Check if a change is detected}
5  if $\hat{\mu}_{dao_i}(t) - \hat{\sigma}_{dao_i}(t) \leq \hat{\mu}^{max}_{dao_i} - \alpha_2 * \hat{\sigma}^{max}_{dao_i}$ then
6      a change is confirmed
7      reset $\hat{\mu}^{max}_{dao_i}$ and $\hat{\sigma}^{max}_{dao_i}$
   {Check if a warning is triggered}
8  else if $\hat{\mu}_{dao_i}(t) - \hat{\sigma}_{dao_i}(t) \leq \hat{\mu}^{max}_{dao_i} - \alpha_1 * \hat{\sigma}^{max}_{dao_i}$ then
9      a warning is triggered
   else
10     no change/warning is detected
```

```
ALGORITHM 7: SelectBasedOnForecastAndLearn()
Input: change detected in $dao_{curr}$, a pre-defined parameter $\lambda$, number of $\lambda$ ($|\lambda|$),
       number of $DAO$ ($|DAO|$), forecast exponential benefit $\hat{\mu}_{dao_i}(t)$, cost $c_{dao_i}(t)$
Output: optimal $dao_i$
The red parameters denote the differences with respect to the reactive approach

   {A change is detected in $dao_{curr}$, then do:}
1  {Initialisation:}
2  $\lambda : \{0.1, 0.2, \cdots, 0.9\}$, $|\lambda| = 9$
3  {Determine the Non-Dominated $DAO$:}
4  for $i$ = 1 to $|DAO|$ do
5      $dominant = 0$
6      for $j$ = 1 to $|DAO|$ do
7          if $(c_{dao_i}(t) \leq c_{dao_j}(t) \& \hat{\mu}_{dao_i}(t) \geq \hat{\mu}_{dao_j}(t)) \& (c_{dao_i}(t) < c_{dao_j}(t) | \hat{\mu}_{dao_i}(t) >$
             $\hat{\mu}_{dao_j}(t))$ then
8              $dao_i$ dominates $dao_j$
9              $dominant = 1$
           end
       end
10     if $dominant = 0$ then
11         Add $dao_i$ to list of non-dominant $DAO$
       end
   end
12 {Calculate marginal loss for the non-dominant $DAO$:}
13 for $i$ = 1 to Length (list of non-dominant $|DAO|$) do
14     foreach $\lambda \in \{0.1, 0.2, \cdots, 0.9\}$ do
           $\hat{L}_{\lambda}(dao_i, t) = \lambda c_{dao_i}(t) - (1 - \lambda)\hat{\mu}_{dao_i}(t)$
15         if $i = argmin_i \hat{L}_{\lambda}(dao_{i'}, t)$ then
16             $\hat{L'}_{\lambda}(dao_i, t) = min_{(i \neq ii)} \hat{L}_{\lambda}(dao_{ii}, t) - \hat{L}_{\lambda}(dao_i, t)$
           else
17             $\hat{L'}_{\lambda}(dao_i, t) = 0$
           end
       end
   end
18 {Determine the expected marginal loss for the non-dominant $DAO$:}
19 for $i$=1 to Length (list of non-dominant $|DAO|$) do
20     $approxM[\hat{L'}_{\lambda}(dao_i, t)] = \frac{\sum_{\lambda \in \{0.1, 0.2, \cdots, 0.9\}} \hat{L'}_{\lambda}(dao_i, t)}{|\lambda|}$
   end
21 Return optimal $dao_i$ = $dao_i$ with $\max(approxM[\hat{L'}_{\lambda}(dao_i, t)])$
```

6.3.3 Select based on Forecast and Learn

If a significant change is detected (based on its forecast exponential benefit), then an optimal *dao* is selected based on its forecast quality values (Algorithm 7). In this context, it follows the same procedures as in Section 5.4.6, but using the forecast quality values rather than the observed ones. Algorithm 7 summarises the steps and their differences with respect to Algorithm 4.

6.4 Experimental Evaluation

We performed experiments with the aim of evaluating the proposed proactive approach and its worthiness through architecting the urban traffic monitoring system *iTransport* described in Chapter 3. The experiments are divided into two parts. The first part (Section 6.4.2) focuses on evaluating the forecasting ability of the forecast models used by the proposed proactive approach and understanding the conditions under which they do or do not perform well. Together with the proposed approach as described in Section 6.3.1, it provides the answer to RQ4.1, outlined in Section 6.1. The second part (Section 6.4.3) investigates whether and when it is worth adopting the proposed proactive approach for architecture evaluation. Together with the proposed approach as described in Sections 6.3.2 and 6.3.3, it provides the answer to RQ4.2 outlined in Section 6.1. Section 6.4.1 explains the experimental setup.

6.4.1 Data Synthesis and Experimental Environment

The data synthesis process is performed using iFogSim (explained in Section 5.5), whereas the Massive Online Analysis framework (MOA) [32] and Matlab are exploited for data analysis.

To test and evaluate the forecast models, we have used MOA [32]; an open source framework for data stream mining that comprises a group of online methods. This framework receives the input attributes (i.e. list of data quality observations), and then it outputs the forecasts with respect to the selected forecasting models. We also used Matlab to implement the proposed approach. All experiments were executed on Intel Core i7 processor machine with 16GB of RAM. Table 6.2 depicts the experimental parameters used in this section. Further details on the experimental design used for each part of the experiments are provided in Sections 6.4.2 and 6.4.3.

6.4.2 Forecasting Ability of the Proposed Proactive Approach

In this section, we aim to provide answers for the following: RQ4.1: *How can proactivity in continuous evaluation be realised and conducted? For instance, can continuous time series forecasting analytics enable us to predict the behaviour of software architectures over time and, if so, how well?*. This question is further divided into the sub-questions below.

Table 6.2 Simulation Parameters for *iTransport*

Parameter	**Value**
Device Configurations	**CPU (GHz), RAM (GB), Cost ($/day)**
Cloud Datacenter	3, 40, 0.1-0.3
Wifi and ISP Gateways	3, 4,0.0053-0.0056
Smart Camera	[1.6, 1.867, & 2.113], 4, 0.0053-0.0056
Number of Smart Cameras	80-120 cameras
Network Configurations (From-To)	**Avg Latency**
ISP Gateway-Cloud Datacenter	80-120 ms
Wifi Gateway-ISP Gateway	1-6 ms
Smart Camera-Wifi Gateway	1-6 ms
Tuple Configurations (Msg. size):	**CPU (MIPS), Network (Bytes)**
Raw Video Stream:	1000, 20000
Motion Video Stream:	2000, 2000
Object Location:	500, 2000
Warning:	1000, 100
Tracking parameters	28,100
Average QoS:	
Applications Response Time (RT)	300-4000 ms
Energy Consumption of devices (EC)	80-120MJ
Network Usage (NU)	500 KBytes- 2 MBytes
Evaluation Settings:	
θ	0.9
α_1, α_2	$\{95, 99\%\}$
Detection time step	5
Normal(Constraint[RT, EC, NU]; Weights[w_{RT}, w_{EC}, w_{NU}])	[400ms, 130MJ, 1Mbps]; [0.4,0.3,0.3]
StrictCase1(Constraint[RT, EC, NU]; Weights[w_{RT}, w_{EC}, w_{NU}])	[4000ms, 110MJ, 2Mbps]; [0.2,0.6,0.2]
StrictCase2(Constraint[RT, EC, NU]; Weights[w_{RT}, w_{EC}, w_{NU}])	[350ms, 130MJ, 500Kbps]; [0.4,0.2,0.4]
StrictCase3(Constraint[RT, EC, NU]; Weights[w_{RT}, w_{EC}, w_{NU}])	[380ms,120MJ, 800Kbps]; [0.4,0.3,0.3]
Normalised Operating Cost	$\{0.3-0.5\}$
Normalised Switching Cost	$\{0.2-0.4\}$
Error Threshold τ_q	[0.15(RT),0.25 (EC),0.20 (NU)]

RQ4.1.1: How many input attributes are needed to provide acceptable forecast ?

Motivation: We need to understand what are good values for the number of input attributes ζ, i.e., what ζ values yield the smallest forecasting errors.

Experimental Design: We use the *kNN learner* against different numbers of input attributes: 2, 6, and 14. These choices aimed to show how low, mid, and high values could affect the forecasting error. Forecasting error is measured based on the mean absolute error ($MAE = \frac{\sum|\hat{q}_{dao_i}(t)-q'_{dao_i}(t)|}{\eta}$) and root mean square error ($RMSE = \sqrt{\frac{\sum(\hat{q}_{dao_i}(t)-q'_{dao_i}(t))^2}{\eta}}$), where η is the number of training examples, $q'_{dao_i}(t)$ is the actual normalised quality for a *dao* as seen in equation 5.3 and 5.2, and $\hat{q}_{dao_i}(t)$ is the quality attribute forecast.

The simulations are performed for 120 timesteps. Therefore, if $\zeta = 2$, then $\eta = 118$, if $\zeta = 6$, then $\eta = 114$ and if $\zeta = 14$, then $\eta = 106$. The comparison of the different numbers of input attributes will be supported by non-parametric Friedman tests [230]. This test is a rank-based test widely used for comparisons across multiple groups. In our case, each group is a given number of input attributes for a given quality attribute, and each observation within a group corresponds to the forecasting error obtained for each of the six *DAO*. The null hypothesis is that all groups have similar forecasting error. The alternative hypothesis is that at least one of the groups has different forecasting error. The same procedure was repeated for MAE and RMSE as forecasting error metrics, and for each quality attribute. The level of significance of the test was set to 0.05.

Table 6.3 reports the forecasting errors for the group of input attributes, whereas Table 6.4 shows their average ranking and *p*-value. If the null hypothesis is rejected, we will perform the post-hoc tests, Bonferroni-Dunn [230], between each group and the top ranked one to determine which groups are significantly different from the best ranked one.

Analysis: Based on Table 6.3, the *p*-value of MAE is 1 (response time), 0.2466 (energy consumption), and 0.3679 (network usage). So the null hypothesis of MAE is not rejected, for all quality attributes. This implies that, for all the quality attributes, there is no significant statistical difference between the number of input attributes. The same applies for RMSE with *p*-value of: 0.3679 (response time), 0.2466 (energy consumption), and 0.8187 (network usage). To decide the number of input attributes to be used in the remaining experiments, we have exploited their Friedman rankings. Based on Table 6.4, the rankings of varying input attributes are very similar. However, $\zeta = 6$ provided lower error for energy consumption quality than the rest. Whereas, $\zeta = 14$ produced more favourable outcome for response time than the others, but it requires 14 timesteps before the initial forecasts can start being made. Therefore, we decided to adopt $\zeta = 6$ in the remaining of the experiments. We recommend a similar procedure to be followed to decide what ζ to adopt in practice.

Overall Observations: *The forecasting models seem to be quite robust to different values of ζ, given that the rankings were all very similar. So, architects may not need to fine tune it so much.*

Table 6.3 Evaluation for kNN learner for 6 *DAO* (2, 6, and 14 input attributes).

# of Input Attributes	**Error Metric in *Response Time* for 6 DAO**											
	dao$_1$		*dao*$_2$		*dao*$_3$		*dao*$_4$		*dao*$_5$		*dao*$_6$	
	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE
2	0.01	0.01	0.15	0.18	0.01	0.01	0.05	0.17	0.06	0.18	0.06	0.18
6	0.01	0.01	0.15	0.18	0.01	0.01	0.05	0.17	0.06	0.18	0.06	0.18
14	0.01	0.01	0.01	0.01	0.01	0.01	0.06	0.17	0.06	0.17	0.06	0.18
# of Input Attributes	**Error Metric in *Energy Consumption* for 6 DAO**											
	dao$_1$		*dao*$_2$		*dao*$_3$		*dao*$_4$		*dao*$_5$		*dao*$_6$	
	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE
2	0.09	0.16	0.13	0.21	0.10	0.18	0.17	0.20	0.23	0.29	0.25	0.34
6	0.08	0.14	0.13	0.21	0.10	0.18	0.17	0.20	0.23	0.29	0.25	0.34
14	0.09	0.16	0.15	0.21	0.10	0.19	0.21	0.26	0.28	0.34	0.18	0.23
# of Input Attributes	**Error Metric in *Network Usage* for 6 DAO**											
	dao$_1$		*dao*$_2$		*dao*$_3$		*dao*$_4$		*dao*$_5$		*dao*$_6$	
	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE
2	0.15	0.18	0.14	0.19	0.17	0.20	0.08	0.15	0.11	0.19	0.11	0.19
6	0.15	0.18	0.14	0.19	0.17	0.20	0.08	0.15	0.11	0.19	0.11	0.19
14	0.16	0.21	0.14	0.18	0.17	0.20	0.08	0.16	0.11	0.17	0.11	0.18

The input attribute with minimum (not necessarily significantly better) MAE is highlighted in yellow, whereas the minimum RMSE (not necessarily significantly better) is highlighted in orange.

Table 6.4 Average ranking and p-value for determining the statistical difference between the use of 2, 6, and 14 input attributes.

# of Input Attributes	Average Ranking of input attributes					
	Response Time		Energy Consumption		Network Usage	
	MAE	RMSE	MAE	RMSE	MAE	RMSE
2	1.17	1.50	1.17	1.50	1.50	1.67
6	1.17	1.50	1.00	1.17	1.50	1.67
14	1.17	1.00	1.67	1.67	1.17	1.33
p-value	**1**	**0.3679**	**0.2466**	**0.2466**	**0.3679**	**0.8187**

The input attribute with minimum (not necessarily significantly better) MAE is highlighted in yellow, whereas the minimum RMSE (not necessarily significantly better) is highlighted in orange.

RQ4.1.2: How many training examples are required for the forecasting error to become acceptable?

Motivation: In this experiment, we aim to determine the minimum number of training examples η_{min} (i.e. forecast quality values) after which the method could use the forecasts $\hat{q}_{dao_i}(t)$ made for each quality attribute to compute the forecasting benefit for selection of a *dao*. This reveals how many timesteps are typically required for the forecasts to be considered reliable and their error estimates to stabilise. Before that, the small number of examples could lead to poor forecasts and unreliable error estimates. In this context, this experiment will show to the architect how to choose the suitable number of training examples after which forecasts can start being used.

Experimental Design: In this experiment, the MAE and RMSE metric will be used to measure the error between the actual quality attribute values and their forecasts for various h values, where $h \in \{1,5,8,15\}$. We will plot the average MAE (Figure 6.3) and average RMSE (Figure 6.4) over time for kNN learner for t+1. Whereas other plots related to $t+h$ are shown in Appendix C.1 for reference. The results for dao_3 and dao_6 are reported here, whereas the other *DAO* showed similar results.

Analysis: As depicted in Figure 6.3 and 6.4, the forecasting error sharply reduces over the beginning of the learning period, reaching a more stable value after 13-15 training examples, for all *DAO* for $t+1$. Same applies for other $t+h$, where $h \in \{5,8,15\}$, as seen in Figure

C.1, C.2, and C.3. Therefore, η_{min} does not need to be a large value. In our remaining experiments, we will adopt $\eta_{min} = 15$.

Despite the sharp decrease in the error during the initial stage of the learning, how low the error can get depends on the *dao* and quality attribute being forecast. In practice, once the error estimates are deemed more reliable (i.e., once η_{min} is reached), the parameter τ_q can be used to prevent the adoption of forecasting models whose forecasting error is deemed high.

Overall Observations: *The error sharply reduces over the initial learning period. Therefore, η_{min} does not need to be a large value. In these experiments, $\eta_{min} = 15$ was a reasonable value.*

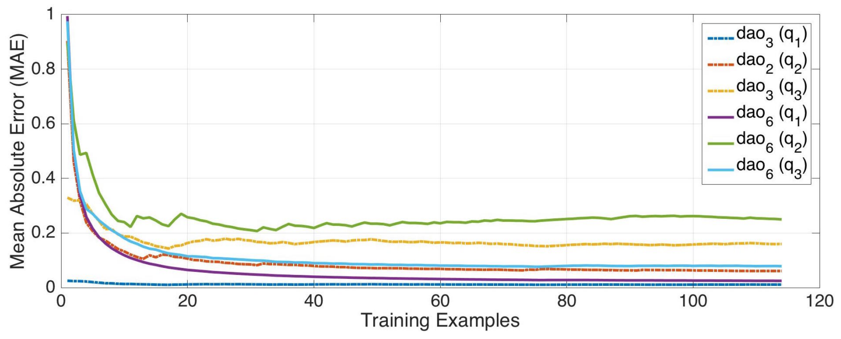


Figure 6.3 The MAE over time for 3 quality attributes to determine the number of training examples for kNN Learner.

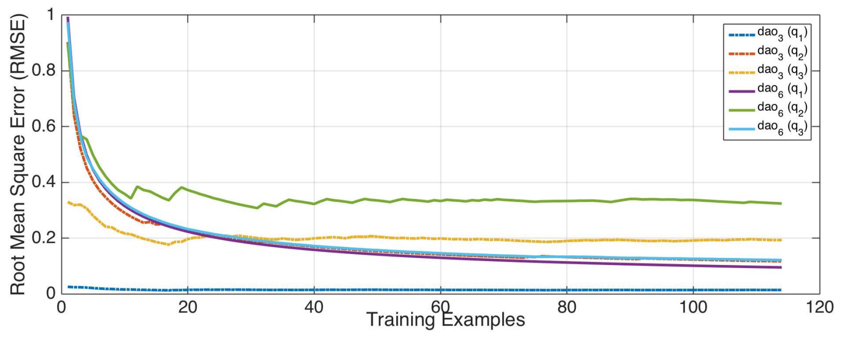


Figure 6.4 The RMSE over time for 3 quality attributes to determine the number of training examples for kNN Learner.

RQ4.1.3: Is it necessary to adopt learning algorithms for non-stationary environments?

Motivation: This experiment aims to show which type of learning algorithms (from the ones introduced in Section 6.2) could be adopted for non-stationary environments, such as IoT.

The architect can benefit from this experiment to select the most suitable learner for each quality attribute.

Experimental Design: In this experiment, we will evaluate the performance of the sole use of single learners for stationary (kNN, Perceptron, and SGD Multiclass) and non-stationary environments (FIMTDD, and FTM). We will also analyze the ensemble learner (AddExp) for non-stationary environments by adopting each of the prior single learners as experts to AddExp. MAE and RMSE are utilised as error metrics to evaluate the learners' performance. Six attributes (i.e. ζ) are used as input for the forecasting method based on Experiment RQ4.1.1. We have also investigated the case of parameter tuning for each learner. However, these set of trials did not lead to any significant improvement for the forecast error. Therefore, we adopted the default values for AddExp and single learners except for FIMTDD. The final parameters used in this experiment are depicted in Table C.1.

Quality Attribute	p-value	
	MAE	RMSE
Response Time	5.95×10^{-7}	1.60×10^{-7}
Energy Consumption	9.00×10^{-8}	2.81×10^{-7}
Network Usage	2.37×10^{-8}	1.43×10^{-7}

Table 6.5 The statistical difference between the learners measured using p-value with respect to each quality attribute.

Analysis: Based on Table 6.6, the kNN, FIMTDD, and FTM (used with/without AddExp) provided the minimum errors for all the *DAO*. The Friedman test was conducted to check whether there is a significant statistical difference between groups. According to it (Table 6.5), a MAE p-value of 5.95×10^{-7} (response time), 9.00×10^{-8}, (energy consumption) and 2.37×10^{-8} (network usage); a RMSE p-value of 1.60×10^{-7} (response time), 2.81×10^{-7}, (energy consumption) and 1.43×10^{-7} (network usage) indicated: *the null hypothesis is rejected (i.e., the learners are significantly different in terms of MAE and RMSE*).

The kNN learner was the one that obtained the top Friedman rankings. By employing the Bonferroni-Dunn test [230] between each learner and kNN, we found that in most of the quality attributes for both RMSE and MAE, there is a significant difference between kNN (top ranked) and two learners (SGD Multiclass and AddExp with SGD Multiclass). For RMSE, there was also a significant difference between kNN and two learners (Perceptron and AddExp with Perceptron). To this regard, kNN provides better forecasting error than the following learners: SGD Multiclass, AddExp with SGD Multiclass, Perceptron and AddExp with Perceptron.

Therefore, we recommend for the architect to exploit kNN learner for forecast, and also adopt it for the remaining of the experiments in this chapter.

Interestingly, kNN is an algorithm for stationary environments, whereas the algorithms for non-stationary environments did not perform best. This suggests that, in this domain, even though the input attributes themselves suffer changes over time, such changes do not affect the relationship between input attributes and the output value being forecast. Therefore, machine learning algorithms able to cope with changes in the relationship between input attributes and the output value were not necessary. When such algorithms are not necessary, they can be detrimental, because they can confuse noise with changes in the relationship between input attributes and output.

It is worth noting that dao_1, dao_2, and dao_3 had less error than other DAO. This is due to the use of cloud rather than fog-cloud, which resulted in lower fluctuation in QoS than fog-based ones. This aided the experts to forecast quality values close to the actual ones. However, the energy consumption for dao_5 and dao_6 was highly fluctuating, which affected the performance of forecast models.

Overall Observations: *Even though IoT is a non-stationary environment that will cause the quality attributes to significantly vary over time, our experiments indicate that the relationship between input attributes and output does not vary, not requiring forecasting algorithms for non-stationary environments.*

Table 6.6 Evaluation for Learners for 6 *DAO*.

Quality Attribute for dao_1	Single Learners										AddExp with									
	SGD MultiClass		kNN		Perceptron		FIMTDD		FTM		SGD MultiClass		kNN		Perceptron		FIMTDD		FTM	
	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE
Response Time	0.03	0.03	0.01	0.01	0.05	0.16	0.01	0.01	0.01	0.01	0.03	0.03	0.01	0.01	0.06	0.16	0.01	0.01	0.01	0.01
Energy Consumption	0.78	0.78	0.08	0.14	0.15	0.24	0.08	0.14	0.08	0.14	0.79	0.79	0.08	0.14	0.18	0.30	0.09	0.16	0.08	0.14
Network Usage	0.38	0.42	0.15	0.18	0.29	0.43	0.15	0.18	0.15	0.18	0.38	0.42	0.15	0.19	0.35	0.49	0.15	0.19	0.15	0.19

Quality Attribute for dao_2	Single Learners										AddExp with									
	SGD MultiClass		kNN		Perceptron		FIMTDD		FTM		SGD MultiClass		kNN		Perceptron		FIMTDD		FTM	
	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE
Response Time	0.38	0.42	0.15	0.18	0.29	0.43	0.15	0.18	0.15	0.18	0.38	0.42	0.15	0.19	0.30	0.44	0.15	0.19	0.15	0.19
Energy Consumption	0.84	0.85	0.13	0.20	0.20	0.30	0.13	0.20	0.13	0.20	0.85	0.86	0.13	0.20	0.25	0.35	0.13	0.21	0.13	0.20
Network Usage	0.41	0.43	0.14	0.18	0.25	0.33	0.14	0.18	0.14	0.18	0.41	0.43	0.14	0.18	0.29	0.37	0.14	0.18	0.14	0.18

Quality Attribute for dao_3	Single Learners										AddExp with									
	SGD MultiClass		kNN		Perceptron		FIMTDD		FTM		SGD MultiClass		kNN		Perceptron		FIMTDD		FTM	
	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE
Response Time	0.02	0.03	0.01	0.01	0.05	0.15	0.01	0.01	0.01	0.01	0.02	0.03	0.01	0.01	0.05	0.15	0.01	0.01	0.01	0.01
Energy Consumption	0.86	0.87	0.09	0.18	0.16	0.26	0.10	0.19	0.09	0.18	0.87	0.87	0.10	0.18	0.17	0.27	0.09	0.18	0.09	0.18
Network Usage	0.35	0.40	0.17	0.20	0.26	0.33	0.17	0.20	0.17	0.20	0.36	0.40	0.17	0.21	0.36	0.48	0.17	0.21	0.17	0.21

Quality Attribute for dao_4	Single Learners										AddExp with									
	SGD MultiClass		kNN		Perceptron		FIMTDD		FTM		SGD MultiClass		kNN		Perceptron		FIMTDD		FTM	
	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE
Response Time	0.95	0.95	0.05	0.17	0.10	0.23	0.05	0.17	0.05	0.17	0.96	0.96	0.05	0.17	0.10	0.23	0.05	0.17	0.05	0.17
Energy Consumption	0.60	0.62	0.17	0.20	0.21	0.27	0.17	0.20	0.17	0.20	0.61	0.63	0.17	0.20	0.23	0.29	0.17	0.20	0.17	0.20
Network Usage	0.85	0.85	0.08	0.15	0.16	0.25	0.08	0.15	0.08	0.15	0.86	0.86	0.08	0.15	0.17	0.25	0.08	0.15	0.08	0.15

Quality Attribute for dao_5	Single Learners										AddExp with									
	SGD MultiClass		kNN		Perceptron		FIMTDD		FTM		SGD MultiClass		kNN		Perceptron		FIMTDD		FTM	
	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE
Response Time	0.95	0.95	0.06	0.18	0.11	0.24	0.06	0.18	0.06	0.18	0.96	0.96	0.06	0.18	0.11	0.24	0.06	0.18	0.06	0.18
Energy Consumption	0.71	0.75	0.24	0.29	0.37	0.46	0.24	0.29	0.24	0.29	0.71	0.75	0.24	0.28	0.51	0.88	0.24	0.28	0.24	0.28
Network Usage	0.85	0.85	0.11	0.19	0.19	0.28	0.11	0.19	0.11	0.19	0.86	0.86	0.11	0.19	0.21	0.30	0.11	0.19	0.11	0.19

Quality Attribute for dao_6	Single Learners										AddExp with									
	SGD MultiClass		kNN		Perceptron		FIMTDD		FTM		SGD MultiClass		kNN		Perceptron		FIMTDD		FTM	
	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE
Response Time	0.94	0.94	0.06	0.18	0.10	0.23	0.06	0.18	0.06	0.18	0.96	0.96	0.06	0.18	0.10	0.23	0.06	0.18	0.06	0.18
Energy Consumption	0.59	0.66	0.25	0.34	0.49	0.71	0.26	0.35	0.26	0.35	0.59	0.66	0.25	0.34	0.50	0.71	0.26	0.35	0.26	0.35
Network Usage	0.82	0.82	0.11	0.19	0.16	0.24	0.11	0.19	0.11	0.19	0.83	0.83	0.11	0.19	0.17	0.24	0.11	0.19	0.11	0.19

Friedman Test	Single Learners										AddExp with									
	SGD MultiClass		kNN		Perceptron		FIMTDD		FTM		SGD MultiClass		kNN		Perceptron		FIMTDD		FTM	
	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE
Average Rank	11.17	10.50	3.00	3.20	7.17	8.17	3.33	3.50	3.17	3.33	13.0	12.17	3.17	3.50	9.17	10.17	4.50	4.00	3.17	3.67

The learner with minimum (not necessarily significantly better) MAE is highlighted in yellow, whereas the minimum RMSE (not necessarily significantly better) is highlighted in orange.

RQ4.1.4: How far ahead can we predict the future benefit based on the quality attribute forecasts?

Motivation: This experiment could help the architect in improving the forecasts by tuning *h* parameter and showing to what extent the forecasts could be beneficial.

Experimental Design: In this experiment, the forecasting method is evaluated with respect to anticipations made for timesteps further ahead, such as t + 1, t + 5, t + 8, and t + 15. The MAE and RMSE metric are also exploited to measure the difference between the forecast and actual value, and how the latter metric varies over time. In this context, a bar plot is used to compare between four categories of forecast (Figure 6.5 and 6.6). Further, the Friedman test is also conducted to investigate the statistical significance of the differences between different choices of *h* values.

Analysis: As depicted in Figure 6.5 and 6.6, the MAE and RMSE are slightly increasing when *h* increases. For instance, for the response time quality, MAE is almost 0.035 (t+1), 0.035 (t+5), 0.036 (t+8), and 0.038 (t+15). The Friedman test confirms that there is no statistical difference between different *h* values (*p*-value = 0.0602). This indicates that the forecasting model is actually able to provide competitive forecasts for timesteps further ahead in the future.

Overall Observations: *The errors obtained by the forecasting models varied depending on the quality attribute being forecast and the dao being monitored. The MAE was smaller than 0.1 in several cases. Since most of the quality attributes when normalised, ranged from approximately 0.2 to 0.9. Therefore, a MAE of 0.1 is considerably low, indicating that forecasts are possible and thus could potentially be used to enable proactivity.*

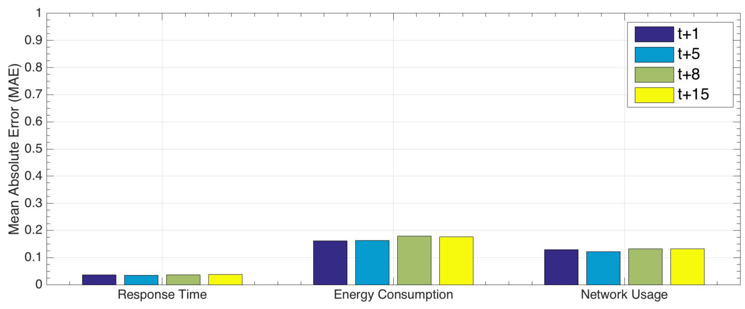


Figure 6.5 Average MAE of *DAO* for 3 quality attributes for varying forecast timesteps.

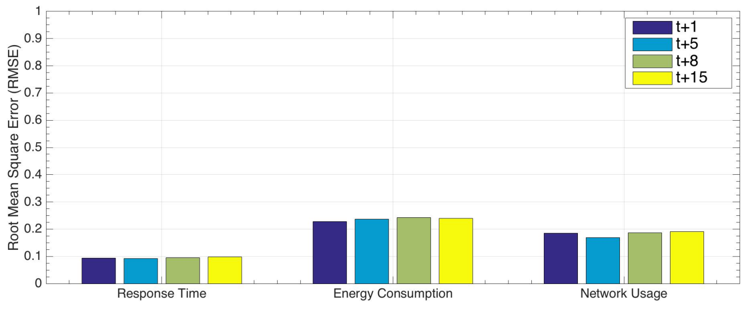


Figure 6.6 Average RMSE of *DAO* for 3 quality attributes for varying forecast timesteps.

6.4.3 Usability and Efficiency of the Proactive Approach

In this section, we aim to show how the forecasting method could be applied in *iTransport* and evaluate its performance against other approaches. This series of experiments aim to provide answers for the following: **RQ4.2:** *How can proactive approaches complement reactive ones to provide a well-rounded and more effective continuous architecture evaluation? What benefits can they bring to continuous software architecture evaluation and decision-making at run-time?* These questions are further elaborated through several sub-questions explained next.

RQ4.2.1: What is the performance of the proactive approach in comparison to the reactive approach when using short-term forecasts (i.e., $h = 1$)?

Motivation: This experiment examines the proactive approach in comparison to the reactive approach in the scenario where the proactive approach is the closest to the reactive one, i.e., when the forecast is a very short-term forecast ($h = 1$). It reveals to what extent forecasts, even if short-term, can be beneficial to run-time architecture evaluation. In particular, it shows whether the *dao* suggestions by the proactive approach (even if short-term forecasts) in comparison to the reactive approach could lead to improvement in mean exponential benefit, mean cost, and application's stability.
Experimental Design: For the *Proactive Informed-selection* approach (Section 6.3), the change detection and the selection of optimal *dao* to switch to is according to the *forecast* values. The *Reactive Informed-Selection* approach (Chapter 5) evaluates, detects the change, and selects the optimal *dao* based on test *actual* quality values.

We examine two scenarios for evaluation: (Best Case) The design-time evaluation approach suggests the systems to start operation using the *dao* that are believed to provide the optimal cost-benefit trade-offs at run-time (i.e. dao_6); (Worst Case) The design-time evaluation approach suggests the systems to start operation using the *dao* that turns out to provide the worst balanced cost-benefit trade-offs at run-time (i.e. dao_2). This was for all the cases, except for Case 1, where the design-time knowledge has recommended dao_3 which turned out to be the best, whereas dao_2 turned out to be the worst. We have developed these scenarios to demonstrate how the reactive and proactive approaches can deal with extreme scenarios. In particular, we assume that the stakeholders have a very good knowledge about the *dao* and its dynamicity for best case and vice versa. In addition, we have generated the design-time evaluation based on the gathered run-time information. For instance, we found that dao_6 almost provides the best QoS over time in most of the cases and so forth. Chapter 4 shows how we performed the design-time evaluation for the best case scenario for Case 2.

For this experiment, we have used the following parameters based on experiments in Section 6.4.2. More specifically, the number of input parameters ζ is 6 (Experiment RQ4.1.1), the minimum number of training examples η_{min} is 15 (Experiment RQ4.1.2), and the adopted learning algorithm is kNN (Experiment RQ4.1.3). The choice of the prior parameters is based on the minimum acceptable forecasting error. We have also set the error threshold τ_q to [0.15 (RT),0.25 (EC),0.20 (NU)] based on the average forecasting error for t+1 (Experiment RQ4.1.4). For this experiment, we only tested the two approaches for number of further ahead timesteps $h = 1$, whereas other values for h will be evaluated in the next experiment.

Regarding the IoT context, the use of fog computing and cloud computing in mobile crowd sensing applications is highly affected by the QoS requirements. Consider a scenario where the stakeholders require the *iTransport* application to quickly track the accident in the city centre, especially in rush hours. In this context, low response time and network usage is of higher concern rather than energy consumption. Therefore, the ranking score (i.e. weights) for response time and network usage qualities are higher than energy consumption. Also the use of fog-cloud computing is advisable over cloud computing. So object detector and tracker modules will be executed in the fog. We are uncertain about the network latency (due to dynamic traffic and variable load) and mobility of devices (nodes join/leave the network). This will cause instability, which may require a switch to another architecture option.

We constructed four different scenarios for evaluation: *normal* and *strict* (3 cases) constraints, which are introduced in Table 5.2. For instance, we consider that the architect has constrained the application to handle the request in less than *400 ms* and network usage does not exceed *1Mbps*; whereas, the energy consumption should not exceed *130 MJ* (normal case) and so forth. These cases are also used in the next experiment. Historical performance and experts' judgment can inform the adjustment of the prior constraints [193, 35]. The architect can adjust the quality weights to reflect priorities from stakeholders. For example, for the normal case, we assume a w_{RT} of 0.4 for response time, w_{EC} of 0.3 for energy consumption, and w_{NU} for network usage of 0.3. We have used the typical experiment settings for the method presented in Table 5.2.

In order to check how good the proactive approach is compared to the reactive approach, we compare their exponential benefit, cost, and number of *DAO* switches. An approach with higher exponential benefit and lower cost is a better approach. An approach with less *DAO* switches is more stable. Stability is desirable, so long as it does not hurt the exponential benefit and cost.

Table 6.7 The number of switches, mean exponential benefit and mean cost for Reactive and Procative informed-selection approach for short-term forecasts (i.e. $t+1$). The best cases are highlighted in green.

Cases \ Approach		Reactive			Proactive		
		# of switches	Mean Exponential Benefit	Mean Cost	# of switches	Mean Exponential Benefit	Mean Cost
Normal	Best	4	0.7871	0.5184	2	0.7876	0.5140
	Worst	5	0.7475	0.5218	3	0.7481	0.5174
Case 1	Best	2	0.4367	0.5084	2	0.4367	0.5084
	Worst	0	0.3801	0.5585	1	0.3511	0.5265
Case 2	Best	2	0.4862	0.5084	2	0.4862	0.5082
	Worst	3	0.4502	0.5133	3	0.4502	0.5133
Case 3	Best	3	0.6274	0.5036	4	0.7173	0.5212
	Worst	4	0.5879	0.5070	5	0.6778	0.5246

Analysis: Table 6.7 summarises the mean exponential benefit, mean cost and number of switches for all the scenarios. In this experiment, the forecast is performed for the next timestep (i.e. $t+1$). Our proactive approach showed promising results for some of the cases, as follows (Table 6.7):

- For the Normal scenario, the average exponential benefit and cost by the proactive approach is similar/slightly better than the reactive approach, but with less switches (e.g. for best case two switches instead of four switches), which enhances the iTransport application's stability.

- In the Case 1 and Case 2, the reactive and proactive approaches provided the same outcome. This may be due to the following: (i) The energy consumption constraint is too strict in Case 1, which was violated most of the times and hence the forecasts did not provide extra benefits; (ii) For Case 2, knowledge for further ahead timesteps is necessary to provide better results.
- For the Case 3, it provides high benefit and high cost, with a slight increase in number of switches, as compared with reactive approach (e.g. for best case provided four switches instead of three). This may be due to the fact that the approach has slightly favored the benefit over the cost.

To this regard, using the proactive approach is recommended over the reactive approach, as it provides better/similar outcome (i.e. based on the context).

Overall Observations: *The proactive approach provided similar or better results than the reactive one. When it was better, it either improved the system stability while maintaining similar exponential benefit and cost, or considerably improved the exponential benefit through a higher number of switches.*

RQ4.2.2: What is the performance of the proactive approach against state-of-the-art architecture evaluation approaches?

Motivation: The previous section only compared the proactive approach against the reactive one, and when using $h = 1$. RQ4.1.4 also showed that the error of forecasts using $t+15$ was similar to that obtained using $t+1$. However, this does not mean that the proactive approach itself would benefit more from using $t+15$. RQ4.2.2 is an extension to these experiments, where we evaluate how the proactive approach works under varying forecast timesteps ($t+1$, $t+5$, $t+8$, and $t+15$), as compared with state-of-the-art architecture evaluation techniques. We also aim to evaluate how averaging the forecast values for $t+h$, where $h \in 1,5,8,15$ will affect the selection of *dao*. This experiment can also illustrate how to choose a suitable h for use with the proposed approach. Architects willing to use the approach could investigate it with different values of h during an initial period of time, and then decide which h value to carry on using, based on experiments and analyses similar to those performed in this section.
Experimental Design: In this experiment, we compare the proactive system against the reactive approach and the following baseline architecture evaluation systems:
(1) Static-Selection System: a typical type of system used in practice [210, 14], where the expert implements a single *dao* based on its assessed value at design-time. For our work, the value is determined using the Binomial option pricing model [4, 42, 14], which estimates the

benefits and costs of options based on a binomial decision tree. This approach is introduced in Chapter 4.

(2) Predefined-Selection System: This is inspired by [211, 48], where the architect will choose the *dao* that is likely to perform the best for a given context. This selection is typically based on experience, backed up by back-of-the-envelope calculations for the cost, benefits, and technical potential. However, the selection may fail to predict potential fluctuations in value, quality potentials, and costs. Our case used dao_6 during week days (because of peak hours) and dao_3 during weekends (because of less demand).

(3) Random-Selection System: Our design for the baseline system follows the argument of [7, 212] but for the context of services. When a significant change is detected, it selects a *dao* randomly independent of its QoS over time. This is because the *DAO* are deemed to be functionally equivalent but deployed in different environments and geographical location (i.e., distributed Fogs/clouds). All the results related to the Random-selection approach are based on the average of 30 runs (the choice of 30 runs is recommended by [213]).

(4) Reactive Informed-Selection system: It evaluates and detects the change as the proactive system, however, the selection for optimal *dao* is based on actual rather than the forecast quality values. This approach is introduced in Chapter 5.

In this experiment, we aim to demonstrate when the proactive approach will/not work well and how do the number of further ahead h forecasts affect the selection. The same applies to the average forecasts, where we will compute the mean of forecasts along with their mean error. We will then evaluate the impact of average forecasts on the decision-making.

We have constructed two cases: *best* and *worst* cases, similar to RQ4.2.1. For all cases, we have experimented using the error threshold shown in Table 5.2. We have summarised the results of four cases in Table 6.9 and 6.10. We have also conducted the Friedman test [230] to check whether there is a significant statistical difference between all the approaches. We have implemented Friedman tests by including the exponential benefit for all the cases (i.e. Normal, Case 1, *etc*, as well as best and worst cases). If there is a significant statistical difference between groups (i.e. p-value < 0.05), we will run Bonferroni-Dunn test [230] between each approach and the top-ranked approach. In addition, to provide a detailed understanding of the behaviour of the proposed approach, we use representative examples for the evaluation as follows: (i) Figure 6.7 to show *DAO* deployment over time, especially for Case 2 ($t+1$, average forecasts where $h \in \{1,5,8,15\}$) for best case; (ii) Figure 6.8 to illustrate the differences between the five approaches for Case 2 for best case. Figures showing other cases are shown in Appendix C.2. The behaviour of the proposed approach in those cases can be explained in a similar way to the cases illustrated in Figures 6.7 and 6.8.

Approach	Average Rank with respect to exponential benefit
Static-Selection	2.250
Predefined-Selection	1.750
Random-Selection	2.000
Reactive Informed-Selection	3.875
Proactive Informed-Selection for $t+1$	4.375
Proactive Informed-Selection for $t+5$	5.875
Proactive Informed-Selection for $t+8$	5.000
Proactive Informed-Selection for $t+15$	5.250
Proactive Informed-Selection for average forecasts	5.125
p-value	$\mathbf{1.49 \times 10^{-8}}$

Table 6.8 The average rank of approaches and statistical difference between them measured using p-value with respect to the exponential benefit. *The top ranked approach is highlighted in orange.*

Table 6.9 The number of switches, mean exponential benefit and mean cost for Static-, Predefined-, and Random-Selection approaches.

Cases \ Approach		**Static-Selection**			**Predefined-Selection**			**Random-Selection**		
		# of switches	Mean Exponential Benefit	Mean Cost	# of switches	Mean Exponential Benefit	Mean Cost	# of switches	Mean Exponential Benefit	Mean Cost
Normal	Best	0	0.7057	0.5131	8	0.5268	0.5439	7	0.5966	0.5222
	Worst	0	0	0.5585	8	0.5268	0.5439	7	0.5491	0.5231
Case 1	Best	0	0.4055	0.5131	8	0.3098	0.5439	4	0.3434	0.5111
	Worst	0	0	0.5841	8	0.3094	0.5458	4	0.2914	0.5120
Case 2	Best	0	0.4907	0.5841	8	0.4161	0.5439	1	0.4291	0.5510
	Worst	0	0.3801	0.5585	8	0.4161	0.5439	0	0.3801	0.5585
Case 3	Best	0	0.6924	0.5148	8	0.5168	0.5439	7	0.5447	0.5240
	Worst	0	0	0.5585	8	0.5168	0.5439	7	0.4924	0.5296

Table 6.10 The number of switches, mean exponential benefit and mean cost for Reactive and Procative informed-selection approach for $t+h$. The best cases are highlighted in green.

Cases \ Approach		# of switches	Mean Exponential Benefit	Mean Cost	# of switches	Mean Exponential Benefit	Mean Cost	# of switches	Mean Exponential Benefit	Mean Cost	# of switches	Mean Exponential Benefit	Mean Cost	# of switches	Mean Exponential Benefit	Mean Cost	# of switches	Mean Exponential Benefit	Mean Cost
		Reactive			**Proactive**														
					t+1			**t+5**			**t+8**			**t+15**			**Average Forecasts**		
Normal	Best	4	0.7871	0.5184	2	0.7876	0.5140	3	0.7885	0.5170	3	0.7885	0.5170	3	0.7885	0.5170	3	0.7885	0.5178
	Worst	5	0.7475	0.5218	3	0.7481	0.5174	4	0.7490	0.5212	4	0.7490	0.5212	4	0.7490	0.5212	4	0.7490	0.5212
Case 1	Best	2	0.4367	0.5087	2	0.4367	0.5087	2	0.4367	0.5087	2	0.4367	0.5087	2	0.4367	0.5087	2	0.4367	0.5087
	Worst	0	0.3801	0.5585	1	0.3511	0.5265	2	0.4015	0.5652	1	0.3531	0.5251	3	0.3925	0.5679	0	0.3801	0.5585
Case 2	Best	2	0.4862	0.5084	2	0.4862	0.5082	1	0.6762	0.5279	1	0.6762	0.5279	1	0.6762	0.5279	1	0.6762	0.5279
	Worst	3	0.4495	0.5119	3	0.4495	0.5119	2	0.6395	0.5316	2	0.6395	0.5316	2	0.6395	0.5316	2	0.6395	0.5316
Case 3	Best	3	0.6274	0.5036	4	0.7173	0.5212	3	0.7192	0.5166	4	0.7163	0.5218	4	0.7163	0.5218	2	0.7086	0.5195
	Worst	4	0.5879	0.5070	5	0.6778	0.5246	4	0.6796	0.5200	5	0.6767	0.5252	5	0.6767	0.5252	3	0.6691	0.5229

Analysis – Overall Results Across Cases:

We first report the results of the Friedman test (Table 6.8). We have found that there is a significant statistical difference between the approaches in terms of exponential benefit (i.e. p-value = $1.49 \times 10^{-8} < 0.05$). The proactive for $t+5$ was the one that obtained the top Friedman rankings. By employing the Bonferroni-Dunn test [230] between each approach and proactive for $t+5$, we found that there is a significant difference between proactive for $t+5$ (top ranked) and three approaches (static-selection, predefined-selection, and random-selection). However, there is no significant difference between the proactive for $t+5$ (top ranked) and reactive, proactive for $t+1$, $t+8$, $t+15$ and average forecasts.

When conducting statistical tests *across cases*, a lack of significant difference between two approaches can happen due to three possible reasons:

1. The two approaches achieved similar results for all cases.

2. One approach performed better for some of the cases, and the other approach performed better for the remaining cases, leading to no significant difference across cases.

3. The two approaches performed similar in several cases, and one approach performed better for the remaining cases. As a result, the test is unable to conclude that the latter approach performed in general better than the former approach across cases.

When analysing Table 6.10, we can see that the reason for the lack of significant difference between proactive for $t+5$ and reactive is reason #3. For instance, the reactive and proactive for $t+5$ produced similar exponential benefit for normal and Case 1. On the contrary, the proactive for $t+5$ obtained considerable improvements in mean exponential benefit than the reactive approach in Cases 2 and 3.

It is thus clear that the proactive approach with $t+5$ obtains in general better exponential benefit than the static-selection, predefined-selection, and random-selection approaches, and similar or better exponential benefit than the reactive approach. Therefore, the proactive approach with $t+5$ is recommended over these other approaches.

As for the case of also having no statistical difference between top-ranked proactive approach and other proactive approaches across cases, this may have occurred because these approaches provided similar exponential benefit across the cases (reason #1 above). Specifically, we can see from Table 6.10 that the average exponential benefit for all the proactive approaches was very similar. However, there is a difference between the proactive approaches in terms of the number of switches. In particular, average forecasts provided 1–2 switches less than other proactive approaches in Case 3 (best and worst), and 1-3 switches less for Case 1 (worst). Therefore, the proactive approach with average forecasts is recommended over the other proactive approaches.

Over the rest of this section, the behaviour of the proposed approach compared to other approaches is discussed in more detail. A discussion on the specific behaviour of the static, predefined and random approaches can be found in Appendix C.2.1, and further illustrates why these approaches do not perform well.

Analysis – Summary of Results With Respect to Each Case Separately:

- *Normal Case (Best/Worst Scenario)*: The results for reactive and all proactive approaches were similar in terms of exponential benefit and cost. In particular, Table 6.10 shows that the reactive and proactive approaches provided mean exponential benefit of around 0.79 and cost of around 0.51 or 0.52 for the best case, and mean exponential benefit of around 0.75 and cost of around 0.52 for the worst case. However, the proactive approaches led to less switches than the reactive approach. Further, the proactive approach produced 30-50% improvement in mean exponential benefit, 5% smaller mean cost, with improved stability (i.e. 3-4 switches less) compared to predefined and random approaches on the best case (Tables 6.9 and 6.10). It also provided a 10% increase in mean benefit and similar cost, but with an increase in the number of switches (i.e. 2-3 switches) compared to the static-selection, which adopts a fixed *dao* over time. Similar results were obtained for the worst case scenario, except when comparing against the static-selection approach. This approach obtained a very poor exponential benefit of zero (due to its violations to the constraints), and a 7% higher cost than the proactive approaches.

- *Case 1 (Best/Worst Scenario)*: The reactive and proactive approaches obtained the same results in the best scenario (same number of switches, mean exponential benefit and cost, as shown in Table 6.10). The proactive approach showed better mean exponential benefit (i.e. 10-30%) and cost (i.e. 3-7%) than baseline and state-of-the-art approaches with better stability, except that the static-selection did not perform any switches (Table 6.9 and 6.10). In the worst scenario, the reactive and proactive for average forecasts obtained similar results, whereas the other proactive approaches led to varied results (either slightly worse exponential benefit and better cost or slightly better exponential benefit and worst cost) with larger number of switches. This may be due to the very strict quality constraint, which has been violated in most of the times. This caused a high fluctuation in the exponential benefit, which ended up triggering additional switches and producing varying behaviour between different $t+h$ values. Therefore, among the proactive approaches, we recommend the one with average forecasts as it considers the forecasts for varying h values. We have seen similar behaviour in the worst case scenario, except that static-selection showed the worst benefit and cost (Table 6.9 and 6.10). In particular, the proactive approach for average forecasts provided better benefit and cost with same number of switches as static-selection.

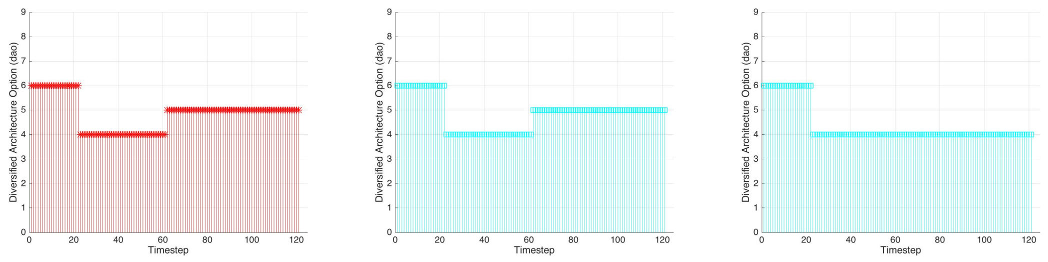
- *Case 2 (Best/Worst Scenario)*: The proactive approaches for long-term forecasts (i.e. $h \in \{5, 8, 15\}$) and average forecasts produced the best results in the best case, in comparison to the reactive and state-of-the-art approaches. In particular, they obtained an increase of about 40% in the average exponential benefit, a reduction of about 5% in cost, and less number of switches (i.e. one instead of two), compared to the reactive approach (Table 6.10). They also obtained a rise of 40-60% in the mean exponential benefit with 5-10% reduction in cost and less number of switches, in comparison to state-of-the-art and baseline approaches (Table 6.9 and 6.10). Similar behaviour occurred for the worst case.

- *Case 3 (Best/Worst Scenario)*: The proactive approaches achieved considerably better exponential benefit at a slightly higher cost than the reactive approach and state-of-the-art approaches (approximately 5-40% increase in benefit) for the best scenario (Table 6.9 and 6.10). The proactive approaches using specific h values led to less switches than the predefined and random approaches, but did not lead to an advantage in terms of number of switches compared to the reactive approach. However, the proactive approach with average forecasts led to the best behaviour in terms of number of switches, except in comparison to the static approach, which uses a fixed *dao* throughout time. Therefore, the proactive approach using average forecasts depicted the best overall results for the best case. A similar behaviour was obtained for the worst scenario.

Detailed Analysis of the Behaviour of the Approaches:

Next, we will demonstrate key representative examples for the *DAO* evaluation by all approaches and its corresponding behaviour for Case 2. The underlying behaviour of the approaches for other cases is similar, and therefore were omitted.

The different exponential benefits and costs obtained by different approaches results from the different choices of *dao* performed by these approaches over time. For Case 2, the proactive approach with $t+1$ always led to the same *dao* choices as the reactive approach, whereas the proactive approach with $t+5$, $t+8$ and $t+15$ led to the same *dao* choices as the proactive approach with average forecasts. An example for Case 2 (Best) is shown in Figure 6.7. Therefore, we only plotted the exponential benefit and cost for $t+1$ and average forecasts in Figure 6.8.

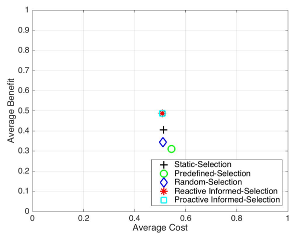


(a) The selected *DAO* by reactive approach **(b)** The selected *DAO* by proactive approach for $t+1$ **(c)** The selected *DAO* by proactive approach for average forecasts

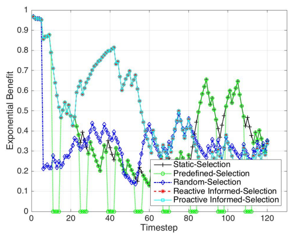
Figure 6.7 The selected *DAO* by reactive/proactive ($t+1$ and average forecasts) approach for **Case 2** for the *best* scenario.

From Figure 6.8, we can see that all approaches have initially observed a gradual decrease in the exponential benefit in **Case 2** (best) over the first 5 timesteps. This means that the current option (dao_6) was not working well. This may be due to internal problems in devices (e.g. ageing affects) and/or high computation in fixed devices. The approaches then behaved as follows in response to that:

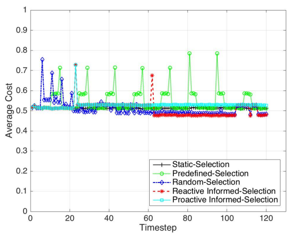
- The static-selection approach never changes the *dao*. Therefore, it kept using the same option (i.e. dao_6) regardless of exponential benefit changes at run-time. This led to better mean benefit (i.e. 0.4907) and mean cost (i.e. 0.5841) than random-selection and predefined-selection approaches (Figure 6.8a and 6.8d), but much worse than reactive and proactive approaches. This is because, despite this decrease in exponential benefit,



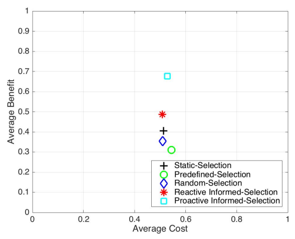
(a) Overall behaviour ($t+1$, Best)



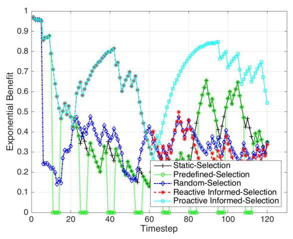
(b) Exponential Benefit ($t+1$, Best)



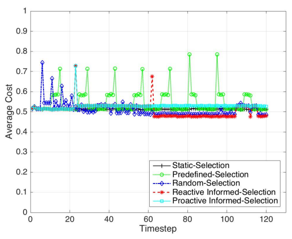
(c) Average Cost ($t+1$, Best)



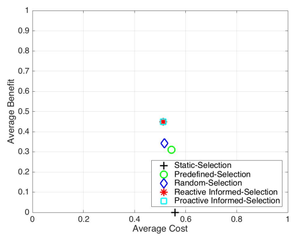
(d) Overall behaviour (Avg Forecasts, Best)



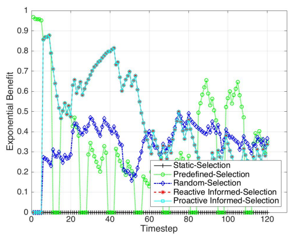
(e) Exponential Benefit (Avg Forecasts, Best)



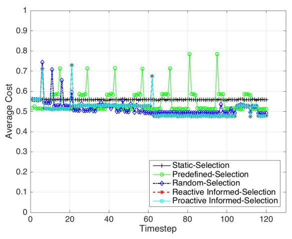
(f) Average Cost (Avg Forecasts, Best)



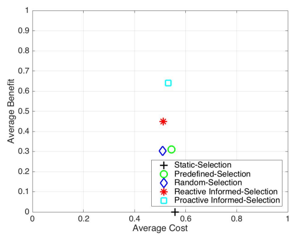
(g) Overall behaviour ($t+1$, Worst)



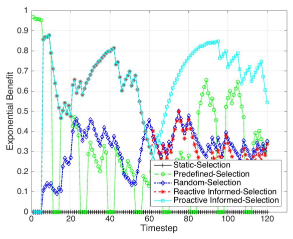
(h) Exponential Benefit ($t+1$, Worst)



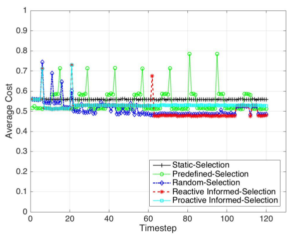
(i) Average Cost ($t+1$, Worst)



(j) Overall behaviour (Avg Forecasts, Worst)



(k) Exponential Benefit (Avg Forecasts, Worst)



(l) Average Cost (Avg Forecasts, Worst)

Figure 6.8 The evaluation of the five approaches' decision-making process for **Case 2** ($t+1$, Average Forecasts) for *best/worst* scenarios.

dao_6 still provided relatively good exponential benefit for some time intervals (e.g. from 85^{th} to 95^{th} and 100^{th} to 110^{th} timesteps), as seen in Figure 6.8b.

- The predefined-selection also ignores run-time changes, but it deploys predefined *DAO* from design-time evaluation based on the contextual requirements (dao_3 in weekends and dao_6 in weekdays). The predefined-selection produced very high exponential benefit up to 0.8-0.9 in some contexts, where dao_6 was deployed (e.g. Figure 6.8b). However, due to goal violation in other contexts (i.e. dao_3), it provided zero exponential benefit. This describes the high fluctuation leading to low overall exponential benefit (e.g. Figure 6.8a). This poor result is a consequence of the design choices not matching the architects' expectations at run-time.

  In particular, the behaviour of predefined-selection was due to the following: (i) The deployed *DAO* did not work as expected. In particular, at run-time, it turned out that dao_3 (i.e. one of the predefined *DAO* for run-time deployment) had poor exponential benefit, which resulted in low mean exponential benefit; (ii) Even though dao_3 was not working well, the predefined-selection did not consider the run-time changes (i.e. no change detection test was adopted). Therefore, the predefined approach did not benefit from the design-time knowledge. Instead it applied unnecessary predefined switches between *DAO* (i.e. had negative impact on the architecture stability) without noticeable improvement in benefit.

- Random-selection switches to a random *dao*, which ends up producing a drastic drop in exponential benefit (from 0.95 to 0.3 as shown in Figure 6.8b) until t=25, instead of improving the benefit or at least keeping it the same. This has been followed by adhoc switches, which resulted in a significant degradation of benefit over time (an average of 0.4291), and a high mean cost of 0.5510 with very high number of switches (7 switches), as seen in Figure 6.8a, 6.8b, and 6.8c.

- The proactive and reactive approaches have continued adopting dao_6 till $t = 25$ (Figure 6.7). Though there was significant degradation in exponential benefit from 0.95 to 0.42 (Figure 6.8b and 6.8e), other *DAO* did not obtained better balanced cost-benefit than dao_6 until $t = 25$. These approaches were able to detect that and keep using dao_6 until $t = 25$, leading to competitive cost and benefit (Figures 6.8b, 6.8c, 6.8e,and 6.8c). Afterwards, both approaches detected a change at $t = 27$ and hence suggested a replacement to dao_4, which resulted in a remarkable improvement in benefit from 0.42 to 0.82 (Figure 6.8b and 6.8e).

- The reactive and proactive approaches provided similar behaviour (i.e. performed same selection of *dao*) until $t = 61$ (Figure 6.7). This is because the proactive approach will have to wait for 21 timesteps to start using the forecasts ($\eta_{min} + \zeta = 15 + 6 = 21$), as stated in RQ1.2. In this context, the proactive approach used the actual values of quality attributes as the reactive approach. After that, both approaches switched to dao_4 at $t = 27$. Since dao_4 showed a gradual increase in exponential benefit until the 55^{th} timestep (Figure 6.8b and 6.8e), no significant changes were detected by both approaches until this timestep.

The reactive and proactive for short-term and long-term forecasts approaches have observed a significant drop in exponential benefit at $t = 61$. We will discuss the approaches' behaviour in response to that, as follows (Figure 6.8a-6.8f):

- Due to the lack of future benefits, the reactive and proactive (i.e. $t + 1$) approaches' evaluation to *DAO* was based on either actual/short-term forecasts, respectively. Therefore, this resulted in several detection and replacements to inefficient *dao* (e.g. dao_5), which caused a gradual rise in exponential benefit from 0.3 to 0.5 (Figure 6.8b) with a decrease in cost from 0.5 to 0.45 (Figure 6.8c) at $t = 71$. This has been followed by a drop in exponential benefit from 0.5 to 0.2 (Figure 6.8b) then fluctuation in exponential benefit over time. This in turn has affected the mean exponential benefit (i.e. 0.5), as seen in Figure 6.8a.

- The proactive approach using average and long-term forecasts has overcome the previous unnecessary switches. They kept evaluating dao_4 over time (Figure 6.7c) with respect to its long-term future benefit. This has led to a reduction in number of replacements (i.e. more stable system) because the change detection is based on long-term future value rather than actual/short-term one.

- Finally, a remarkable rise in exponential benefit by almost 175% has occurred at $t = 61$ for proactive approach using average and long-term forecasts, as compared with short-term forecasts and reactive approach (Figure 6.8b and 6.8e). This has caused an increase of about 40% for the average exponential benefit and less switches (Figure 6.7) with 4% increase in mean cost, in reference to reactive approach (Figure 6.8d).

The behaviour of the approaches for the worst case scenario was similar to that obtained for the best case scenario, in terms of their *DAO* evaluation and selection. For example, the predefined-selection used the same predefined design-time decisions, whereas the random-selection detected significant changes in benefit and selected adhoc *DAO* (Figure 6.8h and 6.8k). The proactive approach and reactive approach initially suffered from low benefit in the first 5 timesteps (Figure 6.8h and 6.8k), but then they quickly handled that and switched to the same *DAO* as in the best case scenario over time (Figure 6.8b,6.8e, 6.8h, and 6.8k). The static-selection adopted a single *dao* over time regardless of run-time changes. In particular, it has recommended an architecture option (dao_2), which seemed to violate the constraints all the time. This explains why it produces zero benefit over 120 timesteps (e.g. Figure 6.8h and 6.8g).

6.5 Discussion

In this chapter, we aimed to answer the thesis's fourth research question:
RQ4: How can the use of forecasting analytics improve the state of run-time architecture evaluation? To answer this question, we proposed a proactive approach (Section 6.3.1) and then tested the approach on iTransport application (Section 6.4). Particularly, in Section 6.4, we provided the architect with guidelines on: (i) how to determine the input attributes required for forecasting (Table 6.4 and 6.3) and the number of training examples to potentially benefit from forecasting (Figure 6.3 and 6.4); (ii) how to select the learners (Table 6.5 and 6.6); (iii) how to compare between the further ahead timesteps (Figure 6.5 and 6.6). We also provided a fair comparison of the proactive approach against reactive, state-of-the-art, and baseline approaches (Table 6.9 and 6.10, and Figure 6.8). Our proactive approach showed better results (Table 6.7 and 6.10) than reactive approach in terms of stability (i.e. lower number of switches) and improved exponential benefit. Further, it outperformed the baseline, state-of-the-art, and reactive approaches (Table 6.9 and 6.10).

From the results in Section 6.4, we can see that the proactive approach outperformed the state-of-the-art approaches in most of the cases in terms of exponential benefit and cost with less switches than the random and predefined approaches. It also provided similar/better benefit than the reactive approach. Even though we cannot generalise the results of each case (normal, case 1, 2 and 3) to other cases in this or other domains, there are some observations on the behaviour of the approach that can explain the situations in which one approach may be expected to perform similarly or better than the other approaches.

For instance, when there is a lot of variability in exponential benefit over time, the proactive approach with average forecasts may recommend less switches than the reactive approach or the proactive approaches with single values for h. This is because this approach may be able to forecast that, despite a given *dao* being potentially poor at present, it will become better again in the near future, meaning that no switch is required.

Furthermore, when the constraint for one of the quality attributes (e.g. energy consumption) is too strict, the proactive approach provided similar behaviour to the reactive approach. This is because the constraint was violated at some time intervals and the variability in the quality was low, and so we did not benefit from the forecasts.

Additionally, when there are trends in the changes in exponential benefit, the proactive approach may achieve better exponential benefit and cost trade-offs than the reactive and state-of-the-art approaches, since it has the ability to detect the changes that will happen in the future and hence may recommend better architecture options.

6.6 Threats to Validity

In this section, we discuss the threats related to internal, construct and external validity in the context of these experiments.

6.6.1 Threats to Internal Validity

These are concerned with the impact of experimental parameters when evaluating the proposed framework. We managed the threats related to internal validity in Chapter 5 through the following: (i) analysing the run-time approach by varying the evaluation parameters (e.g. the relative importance of present/past, the confidence interval, *etc*) and hence selecting the ones which provide acceptable accuracy and stability; (ii) testing the sensitivity of run-time approach to noise, to check how well the run-time approach can handle this issue; and (iii) demonstrating the applicability of the run-time approach to monitor the environment at every T' intervals if the real-time evaluation was expensive and the architects did not wish to use simulation. Similar treatment to the experimental parameters applies to Chapter 6.

6.6.2 Threats to Construct Validity

These are related to used metrics, which reflect what we intend to measure [63]. We managed the Threats to Construct Validity in relation to Chapter 6. For instance, we have adopted the MAE and RMSE as forecasting performance measures for the proactive run-time approach (Chapter 6). These measures are unbiased towards under or over estimations, which make them adequate for determining the appropriate number of input attributes; selecting the suitable number of training examples to start forecasting; comparing between the forecasting algorithms; and choosing the suitable number of future ahead timestep. Friedman, post-hoc tests, and Bonferroni-Dunn rank-based tests [230] were adopted to demonstrate the significance of the statistical differences between groups (i.e. forecasting algorithms and further ahead timesteps) in MAE and RMSE.

6.6.3 Threats to External Validity

These are linked to the generalisation of the experiments [63]. Though we have tested our reactive run-time approach (the second component of the proposed framework) in Chapter 5 across state-of-the-art for architecture evaluation and on a dynamic environment, as well as the proactive run-time approach (the final component of the proposed framework) in Chapter 6 against state-of-the-art and reactive approaches. We cannot claim final generality. Instead further experimentation on a greater number of independent domain-specific cases could confirm the applicability of the proposed proactive approach for other domains (e.g. volunteer computing, microservice architectures, *etc*).

Another external threat to validity is the computational overhead. In particular, the computational overhead of the approach needs to be tested before it can be adopted in industry. We can overcome this overhead by implementing the approach in a decentralised manner. As for the scalability of our continuous evaluation approach, it has been evaluated for the reactive part, whereas the proactive approaches employed are already used in practice for big data [32]. Finally, we have discussed in Chapter 5, the threats related to the use of simulation to aid the run-time evaluation in analysing the potentials of candidate architecture decisions. Similar treatment applies to the proactive approach.

6.7 Conclusion

In this chapter, we contributed a novel approach for continuous software architecture evaluation, by providing continuous learning as a built-in mechanism for proactive evaluation that complements reactive approaches. We have shown how proactive approaches leveraging the time series forecasting analytics can fundamentally guide the continuous evaluation of software architectures and influence the outcome of the decisions made. This approach employs a forecast of the value of architecture decisions to reason about diversification at run-time. We applied our method on an IoT application and compared it to other evaluation methods.

We also explored how various forecasting learning techniques can be "orchestrated" at run-time to better support evaluation and to demonstrate the impact of forecasting analytics on continuous evaluation and decision-making. In most of the cases, our *proactive* run-time approach produces promising results, because it learns and forecasts, and hence makes smarter decisions. This work has paved the way for conducting further studies of continuous software architecture evaluation. In the next chapter, we will discuss the possible opportunities for future investigations.

Chapter 7

Reflections, Future Directions, and Conclusion Remarks

7.1 Reflections

In this section, we will discuss first how the research questions have been addressed, followed by the proposed framework with respect to its applicability and beneficiaries.

7.1.1 How the research questions have been addressed

As aforementioned in Chapter 1, the major aim of this thesis is to address the *lack of systematic continuous architecture evaluation framework that intertwines design-time and run-time evaluation to handle complex architectures and improve the decision-making, in dynamic and uncertain environments, such as IoT*. In this context, we have proposed a novel continuous evaluation architecture framework, which we address through the thesis's main research questions, introduced in Chapter 1. In this section, we summarise the answers to the research questions.

- **RQ1: What is the state-of-the-art in software architecture evaluation under uncertainty and to what extent continuous architecture evaluation is used?** We have provided a systematic literature review, in Chapter 2, covering the existing architecture evaluation methods which provide implicit or explicit management of uncertainty. We have found that continuous evaluation has been mentioned under different terms, such as dynamic, run-time, *etc*. Therefore, we consolidated these efforts, provided a classification framework to categorise them, and discussed their strengthens and weaknesses. We automatically and manually searched the most well-known venues for software architecture and engineering, other related systematic reviews and mapping studies, and significant bibliographical data sources, as well as applied snowballing process to collect our primary studies. This has potentially aided us in determining the gaps and the necessary elements for developing a continuous architecture evaluation framework. The results of our investigation are the following: (a) design-time architecture evaluation approaches garnered more attention than run-time ones, though the latter are increasingly important to handle the dynamism and increasing complexity in software systems; (b) there is a lack of examples on demonstrating how continuous evaluation approaches can be realised and conducted; (c) few methods focus on managing trade-offs between benefits and costs at run-time; (d) there is less focus on adopting machine learning techniques for the evaluation; (e) most of the run-time approaches tend to be reactive (and may recommend unnecessary switches and hence increase deployment costs). Therefore, this calls for a systematic continuous architecture evaluation approach that intertwines design-time and run-time evaluation and extends the existing architecture evaluation methods to improve the decision-making.

- **RQ2: To what extent design-time architecture evaluation methods that adopt economics-driven principles, such as real options analysis, can systematically evaluate complex design decisions, such as diversification, in the face of operational uncertainties in dynamic environments?**
  The review in Chapter 2 motivated the need for further examples of economics-driven design-time evaluation for analysing cost-benefit trade-offs in the context of dynamic and uncertain environments, such as IoT. Further, they have not been used before to evaluate complex design decisions, such as diversification, under uncertainty. Therefore, we have extended the CBAM and binomial real options analysis in Chapter 4 to show how economics-driven principles can benefit the design-time evaluation. We have demonstrated the use of binomial real options analysis to quantify and visualise the long-term value of engineering diversification into the architecture to handle IoT challenges with respect to the *iTransport* application (introduced in Chapter 3). In this context, we have illustrated how it can be used to evaluate complex design decisions such as diversification in the face of operational uncertainties in dynamic environments. In particular, the use of binomial real options analysis allows the architect to value the flexibility of architecture options under uncertainty. The experimental evaluation selected the optimal architecture option for deployment with respect to multiple stakeholders QoS concerns. It also shortlisted the candidate decisions for deployment. Finally, it highlighted the need for run-time evaluation to complement

design-time decisions. This is because real options analysis is a static method for economic modelling and forecasting of architecture decisions. This type of design-time evaluation relies on human experts, who can only *partially* forecast the fitness and the extent to which an (diversified) architecture can cope with operational uncertainties, unanticipated usage scenarios, and emergent behaviours. It is based on assumptions (i.e. experts knowledge) which may (not) be valid in unpredictable environments, such as IoT. For instance, in the experimental evaluation, we have illustrated some cases where the architects had low confidence with respect to the candidate architecture options and hence run-time evaluation was necessary to evaluate these options.

- **RQ3: How can run-time architecture evaluation using cost-benefit analysis and machine learning techniques complement the design-time one to handle uncertainty?**
  The review also in Chapter 2 showed that the current run-time architecture evaluation methods are based on assumptions which may not be true indefinitely. For instance, some approaches face scalability and complexity concerns, while others assume that the non-functional data about architecture decisions is available at every timestep, which may not be true in non-stationary environments such as IoT. They also lack the capability for checking whether the current architecture decision is getting worse. Therefore, we have proposed a novel run-time architecture evaluation approach, in Chapter 5, to tackle the prior problems and complement design-time decisions (suggested by design-time approach introduced in Chapter 4). In particular, the approach uses a time-decay function inspired by reinforcement learning to quantify the benefit of architecture decisions over time by emphasising on the recent observations rather than the past ones. This could enable the architect to learn more about the behaviour of architecture decisions over time. It also adopts change detection tests to check significant deviations from the currently deployed architecture option. If it is significantly worse, a method inspired by the multi-objective optimisation literature is adopted to identify the architecture option with the most balanced trade-off between cost and benefit. A set of controlled experiments were conducted, derived by the *iTransport* application, to guide the architect in tuning the parameters of the approach (i.e. the relative importance of past/present) to best learn the behaviour of architecture options. In this context, the approach has demonstrated its ability to explicitly deal with uncertain environments through the use of time-decay function. The architect can also experiment with the input parameters with respect to the trade-off between architecture stability and learning accuracy. It also provides the architect with flexibility to set

a monitoring interval to profile the benefit of the architecture options. Furthermore, this approach outperformed the baseline and state-of-the-art approaches, in terms of providing better overall performance (i.e. high benefit with acceptable cost). The experiments have demonstrated the approach's scalability and robustness to varying Gaussian noise levels. Finally, threats to validity related to these experiments were discussed in Section 5.10.

In some contexts, the reactive learning may suggest wrong decisions and recommend unnecessary switches due to the lack of knowledge about the future benefit of candidate architecture decisions. Taking into account the future potential of diversified architecture options is important to provide a more informative evaluation. This calls for a proactive approach, which aims to harvest the power of forecasting analytics to anticipate the benefit of architecture decisions over time.

- **RQ4: How can the use of forecasting analytics improve the state of run-time architecture evaluation?**
  The review in Chapter 2 demonstrated the need for a novel proactive evaluation approach as a part of the continuous evaluation framework. This is because, it is clear that most of the current run-time approaches (e.g. [138, 155, 122], *etc*,) tend to be reactive when simplistic learning, partial or incomplete knowledge is used: may suggest wrong decisions due to the unexpected future environment changes; recommend unnecessary switches due to the lack of future knowledge about the candidate architecture decisions. This in turn may affect the architecture's stability and overall behaviour. In this context, in Chapter 6, we adopted continuous time series forecasting as built-in mechanisms to complement reactive run-time evaluation with proactive run-time evaluation. In this way, the approach does not take into account only the past likely performance of architecture decisions, but also their future potentials, better informing run-time architecture decisions. Moreover, our work has investigated the use of online machine learning models for stationary and non-stationary environments through a series of experiments. In particular, the experiments provided the architect with recommendations on how to best benefit from the proactive approach through choice of learners, input parameters, *etc*, grounded on experimentation and evidence. For instance, we have adopted the kNN algorithm [220] due to its efficiency as compared with other online machine learning models. Our experiments also demonstrated the applicability and effectiveness of the approach using the case of architecting for IoT through the *iTransport* application. We provided a comparative study between the proposed

proactive approach, reactive, baseline and state-of-the-art approaches. The proactive approach showed its effectiveness in terms of the selection of optimal architecture options, significant improvement in benefit over time (e.g. about 40-70% better than reactive, baseline, and state-of-the-art approaches in some scenarios), and enhanced architecture stability. Finally, threats to validity related to these experiments were discussed in Section 6.6.

7.1.2 Applicability and beneficiaries of the framework

We demonstrate the applicability and beneficiaries of the framework, through the following set of questions:

- *What are the types of decisions supported by the proposed framework?* Architecture design decisions could be *static* or *dynamic* in nature. Several structural design decisions are static in nature; this implies that these decisions can be expensive to change and cannot be altered very frequently at run-time. Henceforth, the architect should evaluate them cautiously at design-time. Example of these decisions include network-related decisions such as the physical connectivity between devices (e.g. how data bits are moving in/out of the IoT device), logical connectivity (e.g. what protocols the software uses to transport these bits, such as MQTT), and also the network topology. These decisions are affected by the expected incoming data volumes, cost, memory requirements, *etc*. Therefore, they are quite difficult to change. The architect can leverage the contribution of Chapter 4 to assess the value of architecture design decisions when the analyst/architect has some confidence about the value of the decisions and understanding of the likely dynamic scenarios that can confront the architecture.

  However, there are other decisions, which are dynamic in nature and could be customised at run-time (e.g., predefined decisions that could be tailored to fit the run-time context; strategies and tactics to address behavioural requirements). For instance, different deployment strategies, such as the use of cloud, fog-cloud, *etc*, are an example of a decision that can be best evaluated dynamically. When deemed to be necessary, diversification was also employed to provide "malleability" to alter the structure through inclusion of a limited number of tactics that can better meet the behavioural requirements. In this context, our work is particularly interested in investigating and evaluating dynamic design decisions.

- *How often should the architecture decisions be monitored?* Our framework has the flexibility to monitor the architecture decisions every T' intervals. We have demonstrated to what extent our framework can be robust to monitoring of the actual diversified architecture options at larger intervals T', to investigate scenarios where a simulator is not available and the application cannot afford monitoring the actual diversified architecture options very often.

- *How often should changes be impacted on the architecture decisions under concern?* Our framework only recommends changes to the architecture when necessary, i.e., when the benefit is becoming worse and there is another better architecture option. The proactive approach further considers the future likely benefit to make a recommendation.

- *Should the evaluation of architecture decisions rely on historic data or forecast data?* In many real-world scenarios, the assumption that IoT applications are operating in a stationary environment (e.g., fixed workload, fixed hardware/devices, guaranteed network connectivity, known probability distribution of QoS) is simply not true [6, 7]. Therefore, the reliance on historical data only will not always provide the true value of architecture options. In this thesis, our framework complements the previous properties with time series forecasting (i.e. online learning). The method allows the architect to do the following: tune the number of timesteps further ahead for anticipations; select the forecasting learning model; assess the quality of the forecasting learning model (judged by some error metrics). Based on that, the architect can determine when (not) to use the forecasts to compute the benefit of architecture options. This could potentially maximise the benefit from forecasts.

- *Who can potentially benefit from the proposed framework?* Architects from several domains could benefit potentially from our continuous evaluation framework. For instance, architects using DevOps are currently deploying some industrial monitoring tools (e.g. AppDynamics [231] and New Relic [232]). Further, an application built using AWS IoT and Greengrass suites [209] could benefit from our framework in the context of evaluating the benefit of each option at run-time using the CloudWatch monitoring tool [195]. Our framework could be integrated to one of these tools, to aid the architect in evaluating, refining and/or phasing-out of architecture design decisions. The forecasting ability could reduce the number of unnecessary adaptations causing a decrease in development and implementation costs.

- *What domains can potentially benefit from the proposed framework?* Other applications operating in dynamic environments, such as volunteer computing, cloud-based architectures, and service-oriented architectures, could also potentially employ the approach. For instance, Cloud-based architectures, which are essentially based on service-oriented architectures, can use our framework to justify the choice of abstract architecture model and its possible concrete instantiations over different releases. Inputs from the evaluation can help architects in refining the abstract model; adjust, limit or rethink modes for dynamic composition of concrete ones prior to future deployments.

  The evaluation of self-adaptive systems introduces new challenges that have to deal with evaluating uncertainties and dynamism, making design-time evaluation approaches limited or unfit. Classical approaches for designing adaptive systems tend to take design-time reasoning and evaluation, when deciding on encoding switches, policies, adaptive decisions, *etc*. Continuous architecture evaluation framework can benefit self-adaptive evaluation: (i) assisting in the systematic inception, elaboration, refinement and evaluation of the adaptive design decisions and/or models (i.e. configurations) that can influence QoS and justifying their need before the system is deployed in the next release cycle; (ii) valuing, profiling, and forecasting the likely performance of these decisions and/or models in meeting qualities in relation to alternatives over time as the software evolves; (iii) continuously evaluating their cost-effectiveness and what to be encoded from these decisions and/or models; and (iv) determining the frequency of adaptation, which have a significant impact on the architecture's stability and hence decide on which options could better improve the architecture's stability for run-time deployment.

7.2 Future Directions

There are several areas of development to this thesis which could be interesting for future research. In this section, we outline some of these directions.

- *Dependencies between architecture decisions:* As shown in Chapter 5, the modelling for diversified architecture options qualities follows linear aggregation and is consistent with online QoS modelling approaches that have been widely adopted in the service computing community (e.g., [202, 203]). Though the use of linear aggregation for qualities is the widely adopted practice in service community, it is acknowledged to be limited when capturing dependencies of decisions affecting qualities. Additional online aggregation functions, for capturing dependencies of decisions affecting qualities, is a non-trivial problem; its solution will constitute a significant contribution to both the software architecture and services community. In this context, a future potential research area is modelling of quality aggregation functions to consider dependencies between architecture decisions.

- *Cost Fluctuation over time:* We have found that the pricing in most of IoT service providers (e.g. amazon) is based on the following [209]: (1) Fixed charge to reserve the required resources for a particular time; (2) Pay-per-use, where the consumer pays for CPU per hour or per GB/TB of data; and (3) Auction-based, where a consumer could book resources if s/he pays the highest price for these resources. For these types of pricing methodologies, a noticeable reduction in the price volatility of resources has begun from December 2017 [201]. In this context, when the price volatility is so low, the use of the exponential decay factor for cost is not necessary. Therefore, we focused on providing a continuous measure for the benefit of each *dao* (i.e. exponential benefit). However, the exploration of the impact of high price volatility (i.e. using exponential decay for the cost) on the decision-making could be investigated as a future work.

- *Potentials of using the framework in self-adaptive systems:* As aforementioned in Section 7.1.2, our framework could benefit the self-adaptive systems in enhancing the evaluation of architecture decisions, especially in dynamic and uncertain environments. Therefore, one potential direction is to incorporate the framework into self-adaptive systems, to see how beneficial it can be to this domain.

- *Potentials of complementing the framework with users feedback:* Continuous architecture evaluation has provided mechanisms which balance stakeholders' (mainly architects) involvement and automated behavioural evaluation for qualities (e.g. automatic management of trade-offs). The work can further benefit from involving end users in the evaluation through getting their feedback [233, 234]. The level of abstraction that is more suitable to users would be the requirements level. We also recognise that requirements models can be equally informative for other stakeholders, such as the strategic management and system analysts of the system, where their feedback is based on the actual operation of the system. Getting users feedback requires engineering of that feedback acquisition system [235]. For example, adaptivity to the users' archetypes and their profiles when providing feedback will maximise the response rate and the quality of the feedback provided [236]. Further investigation and analysis of the use of users feedback on real case studies is necessary to confirm applicability and efficiency.

- *Potentials of using the framework on architecture's sustainability:* We have explored the potential of applying our framework in order to evaluate software architecture for sustainability. During the lifetime of software, there are plenty of uncertainties and dynamics which software experiences. Our framework provides explicit management of uncertainty through automatic management of cost-benefit trade-offs, forecasting the impact of candidate architecture options on quality attributes of interest, proactive evaluation, *etc*. Therefore, our framework could potentially be used to analyse and forecast the implications of uncertainties and dynamics on architecture sustainability over time.

  Architecture sustainability is an interpretation of the capability of enduring and retaining the system functional activities over the long-term [237]. "A long-living software system is sustainable if it can be cost-efficiently maintained and evolved over its entire lifecycle" [93]. There have been a number efforts to better understand sustainability, through the illustration of the long-term need for sustainability [238], possible metrics for sustainability [239], and the necessary milestones to achieve it [240].

  Architectural changes could have a negative impact on sustainability. Typically, these changes could occur due to the emergence of new requirements, technologies and markets, as well as, changes in the old requirements and current business strategies or goals. Quick design decisions that meet short-term objectives and constraints, but sacrifice long-term value can affect the sustainability of the architecture. Further, late detection of errors may result in the execution of a string of unnecessary complex design decisions [238]. The complexity of revisiting the architecture design decisions can motivate the need for phasing-out the architecture and looking for an alternative replacement. Avgeriou et al. [238] address three approaches to achieve architecture sustainability by handling changes systematically: refactoring, renovating, and rearchitecting. However, the effectiveness of these approaches has been looked at from a technical point of view, without clear connection to their long-term value under uncertainty.

  The Karlstone manifesto [241] has categorised software architecture sustainability into five dimensions: individual, social, technical, economical, and environmental [242, 243]. Individual sustainability addresses the well-being of humans by maintaining their capital, in terms of health, education, skills, *etc*. Securing the societal communities is the main concern of social sustainability, which includes social equity, justice, employment, democracy, *etc*. Technical sustainability refers to "longevity of information, systems, and infrastructure and their adequate evolution with changing surrounding conditions" [241]. The long-term impact of human acts on the natural surroundings is included in environmental sustainability. One of the fundamental dimensions for software sustainability is economics. It is the continuous delivery of added value and capital maintenance [241].

Despite that the sustainability dimensions are interleaving, finding a common measure is troublesome [241]. To this regard, we recommend evaluating and extending the current continuous evaluation framework to include a single dimension *economic sustainability*. We believe that the continuous evaluation framework could aid the architect in quantifying and assessing the economic sustainability, due to the management, evaluation and forecasting of cost-benefit trade-offs of architecture decisions in presence of uncertainty. This requires further treatment, validations, and experimental evaluation.

- *Potentials of applying transfer learning on the continuous evaluation framework:* We aim to demonstrate the usefulness of the transfer learning methodology (introduced in Section 5.10) on the run-time architecture evaluation approach. In this context, the use of transfer methodology could learn from both simulated and run-time data and hence potentially confirm the accuracy of the forecasting models.

- *Potentials of exploring other change detection tests for the continuous evaluation framework:* Though our change detection test produced satisfactory results, we plan to extend our approach to explore other change detection tests, introduced in Section 5.2.3.

- *Experiments and Tool Support:* Another interesting future direction is testing the framework on a real application. This could provide the architect with extra insights on the provision of an application's challenges in a real setting. For instance, as mentioned in Section 7.1.2, the framework could be adopted to evaluate an application built using AWS IoT and Greengrass suites [209] by profiling the QoS data from the CloudWatch monitoring tool [195]. Another potential direction is the further use of a simulator along with the running system. This could show the extra insights the approach can provide in a run-time context. Further, as mentioned in Section 6.6.3, we intend to implement the framework in a decentralised manner, which may potentially improve the computational overhead. However, further investigations are necessary to confirm the efficiency of decentralisation mode. This thesis has studied a representative web-operated IoT; nevertheless, the application of the framework to other dynamic and variant-intensive systems can benefit from the framework. In this context, testing it on other domains, such as volunteer computing and microservice architectures, would be worth exploring. Finally, implementing our framework as a tool support can accelerate the adoption of our contributions in an industrial setting.

7.3 Closing Remarks

In this section, we aim to summarise the outcome of the research carried out during the different stages of this thesis. The major conclusions that we can draw from this research are, as follows:

1. There is a pressing need for a continuous architecture evaluation framework that systematically evaluate the cost-benefit trade-offs of architecture decisions, operating in uncertain environments, such as IoT.
2. The design diversity methodology could be used to handle the heterogeneity, scale, and high dynamism challenges when architecting for IoT.
3. A potential use for binomial real options analysis at design-time is quantifying the long-term value of the flexibility of (diversified) architecture decisions under uncertainty.
4. Run-time evaluation can be useful to complement the design-time decisions.
5. The use of reinforcement learning could aid the architect in learning the behaviour of architecture decisions over time being useful for the run-time architecture evaluation part of the framework.
6. When evaluating the cost-benefit trade-offs of architecture decisions at run-time, the trade-off between architecture's stability and learning accuracy should be taken into consideration as well, to best benefit from approach.
7. The proactive method is able to forecast the future potentials of architecture decisions for further timesteps ahead.
8. The proactive run-time method leveraging time series forecasting analytics to assess diversity in design performs better than baseline, state-of-the-art, and reactive architecture evaluation approaches.

Appendix A

List of Included Studies in Systematic Literature Review

In this appendix, we first tabulate the studies with respect to domain maturity level in Table A.1 and then provide a list of included studies in the systematic literature review in Table A.2, A.3, and A.4.

Table A.1 Studies with respect to Research Area maturation level.

Research Area Maturation Level	**Studies**	**# of Studies**
Basic Research	[73, 18, 102]	3
Concept Formulation	[84, 95, 87, 85, 96, 103, 83, 42, 101, 41] [117, 104, 120, 137, 119, 100, 21, 88, 86] [138, 81, 121–124, 45, 116, 139, 126]	29
Development and Extension	[82, 106]	2
Internal Enhancement	[97, 44]	2
External Enhancement	[10, 105]	2
Popularization	[80, 9]	2
Total		40

Table A.2 Studies included in the review.

Study	Ref	Author(s)	Year	Title
S1	[73]	R. Kazman, L. Bass, G. Abowd, & M. Webb	1994	SAAM: A method for analyzing the properties of software architectures
S2	[82]	P. Bengtsson & J. Bosch	1998	Scenario-based software architecture reengineering
S3	[9]	R. Kazman, J. Asundi, & P. Clements	2001	Quantifying the costs and benefits of architectural decisions
S4	[80]	R. Kazman, M. Klein, P. Clements & others	2003	Evaluating software architectures
S5	[10]	P. Bengtsson, N. Lassing, J. Bosch, & H. Vliet	2004	Architecture-level modifiability analysis (ALMA)
S6	[41]	R. Bahsoon & W. Emmerich	2004	Evaluating architectural stability with real options theory
S7	[84]	S. Cheng	2004	Rainbow: cost-effective software architecture-based self-adaptation
S8	[95]	M. Ionita, P. America, D. Hammer, H. Obbink & J. Trienekens	2004	A Scenario-Driven Approach for Value, Risk, and Cost Analysis in System Architecting for Innovation
S9	[87]	L. Zhu, A. Aurum, I. Gorton, & R. Jeffery	2005	Tradeoff and sensitivity analysis in software architecture evaluation using analytic hierarchy process
S10	[85]	T. Al-Naeem, I. Gorton, M. Babar, F. Rabhi & B. Benatallah	2005	A quality-driven systematic approach for architecting distributed software applications
S11	[96]	R. Kazman, L. Bass & M. Klein	2006	The essential components of software architecture design and analysis
S12	[103]	L. Grunske	2006	Identifying good architectural design alternatives with multi-objective optimization strategies
S13	[83]	G. Tesauro	2007	Reinforcement learning in autonomic computing: A manifesto and case studies
S14	[42]	I. Ozkaya, R. Kazman & M. Klein	2007	Quality-attribute based economic valuation of architectural patterns
S15	[101]	C. Kim, D. Lee, I. Ko & J. Baik	2007	A Lightweight Value-based Software Architecture Evaluation

Table A.3 Studies included in the review (Continued).

Study#	Ref	Author(s)	Year	Title
S16	[117]	R. Bahsoon & W. Emmerich	2008	An economics-driven approach for valuing scalability in distributed architectures
S17	[104]	Y. Liu, M. Babar & I. Gorton	2008	Middleware Architecture Evaluation for Dependable Self-managing Systems
S18	[120]	W. Heaven, D. Sykes, J. Magee & J. Kramer	2009	A case study in goal-driven architectural adaptation
S19	[137]	D. Kim & S. Park	2009	Reinforcement learning-based dynamic adaptation planning method for architecture-based self-managed software
S20	[119]	J. Yang, G. Huang, W. Zhu, X. Cui & H. Mei	2009	Quality attribute tradeoff through adaptive architectures at runtime
S21	[100]	J. Lee, S. Kang & C. Kim	2009	Software architecture evaluation methods based on cost benefit analysis and quantitative decision making
S22	[21]	I. Epifani, C. Ghezzi, R. Mirandola & G. Tamburrelli	2009	Model evolution by run-time parameter adaptation
S23	[88]	R. Calinescu & M. Kwiatkowska	2009	Using quantitative analysis to implement autonomic IT systems
S24	[105]	He. Christensen, K. Hansen & B. Lindstrøm	2011	Lightweight and continuous architectural software quality assurance using the asqa technique
S25	[18]	R. Pooley & A. Abdullatif	2010	Cpasa: continuous performance assessment of software architecture
S26	[97]	F. Faniyi, R. Bahsoon, A. Evans & R. Kazman	2011	Evaluating security properties of architectures in unpredictable environments: A case for cloud
S27	[86]	N. Esfahani, E. Kouroshfar & S. Malek	2011	Taming uncertainty in self-adaptive software
S28	[138]	R. Calinescu, K. Johnson & Y. Rafiq	2011	Using observation ageing to improve Markovian model learning in QoS engineering
S29	[81]	R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola & G. Tamburrelli	2011	Dynamic QoS management and optimization in service-based systems
S30	[102]	M. Osterlind, P. Johnson, K. Karnati, R. Lagerstrom & M. Valja	2013	Enterprise architecture evaluation using utility theory

Table A.4 Studies included in the review (Continued).

Study#	Ref	Author(s)	Year	Title
S31	[44]	N. Esfahani, S. Malek & K. Razavi	2013	GuideArch: guiding the exploration of architectural solution space under uncertainty
S32	[121]	D. Cooray, E. Kouroshfar, S. Malek & R. Roshandel	2013	Proactive self-adaptation for improving embedded, the reliability of mission-critical, and mobile software
S33	[122]	N. Esfahani, A. Elkhodary & S. Malek	2013	A learning-based framework for engineering feature-oriented self-adaptive software systems
S34	[123]	C. Ghezzi & A. Sharifloo	2013	Dealing with non-functional requirements for adaptive systems via dynamic software product-lines
S35	[124]	E. Fredericks, B. DeVries & B. Cheng	2014	Towards run-time adaptation of test cases for self-adaptive systems in the face of uncertainty
S36	[45]	E. Letier, D. Stefan & E. Barr	2014	Uncertainty, risk, and information value in software requirements and architecture
S37	[106]	V. Eloranta, U. Heesch, P. Avgeriou, N. Harrison & K. Koskimies	2015	Lightweight Evaluation of Software Architecture Decisions
S38	[116]	B. Ojameruaye, R. Bahsoon & L. Duboc	2016	Sustainability debt: a portfolio-based approach for evaluating sustainability requirements in architectures
S39	[139]	G. Moreno, J. Camara, D. Garlan & B. Schmerl	2016	Efficient decision-making under uncertainty for proactive self-adaptation
S40	[126]	V. Donckt, M. Jeroen, D. Weyns, M. Iftikhar & R. Singh	2018	Cost-Benefit Analysis at Runtime for Self-Adaptive Systems Applied to an Internet of Things Application

Appendix B

Utility and Binomial trees

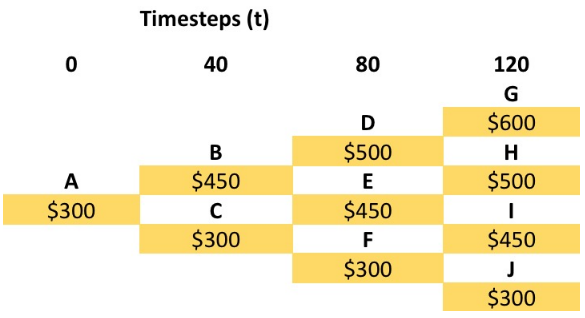
In this appendix, we present the utility trees, which represent the perception of stakeholders with respect to different quality priorities and constraints. We then illustrate the binomial trees generated by the economics-driven design-time evaluation approach introduced in Chapter 4. These trees along with the approach are used to analyse the trade-offs between the long-term benefit and costs of *DAO*. This could aid the architect in selecting the suitable *dao* with respect to the context.

B.1 Case 1: Energy Consumption as a Concern

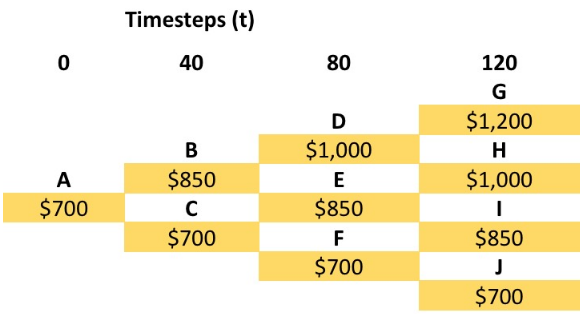
In this section, we plot the rest of utility and binomial trees used for *DAO* evaluation in Case 1 (Chapter 4).

B.2 Case 2: Response Time and Network Usage as Concerns

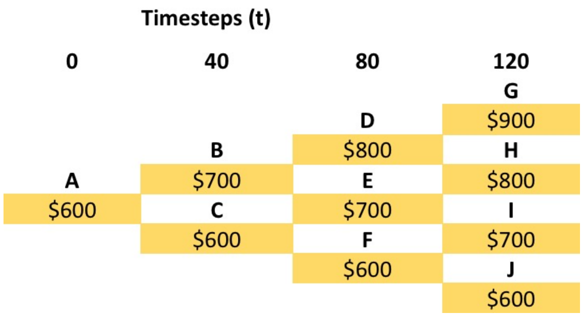
In this section, we demonstrate the rest of utility and binomial trees used for *DAO* evaluation in Case 2 (Chapter 4). Here, we have two cases: *Best* Case and *Worst* Case. For the best case, we assume that the stakeholders have good knowledge about the IoT domain, and hence their perception are almost valid for *DAO*. These are shown in Figure B.9-B.11 and B.12-B.19. In the worst case scenario, the stakeholders are not confident about their intuition with respect to *DAO* utilities. We present the utility and binomial trees related to this case in Figure B.20-B.31.



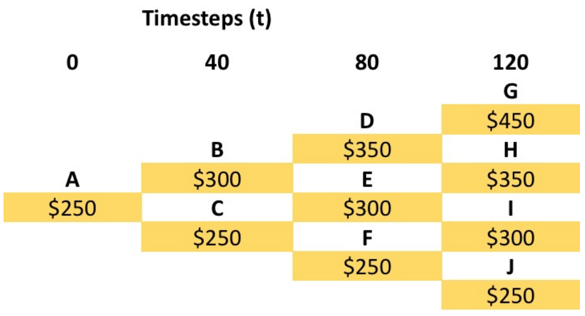
(a) *dao*$_5$



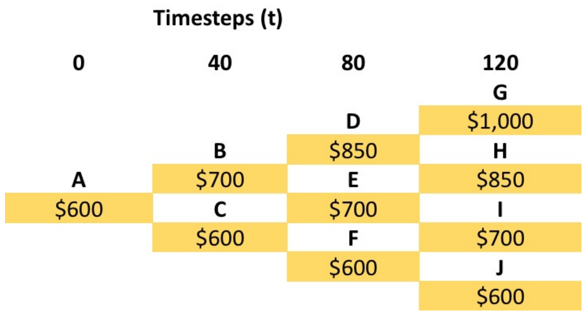
(b) *dao*$_6$



(c) *dao*$_7$



(d) *dao*$_8$



(e) *dao*$_9$

Figure B.1 Utility Tree of all options for Case 1.

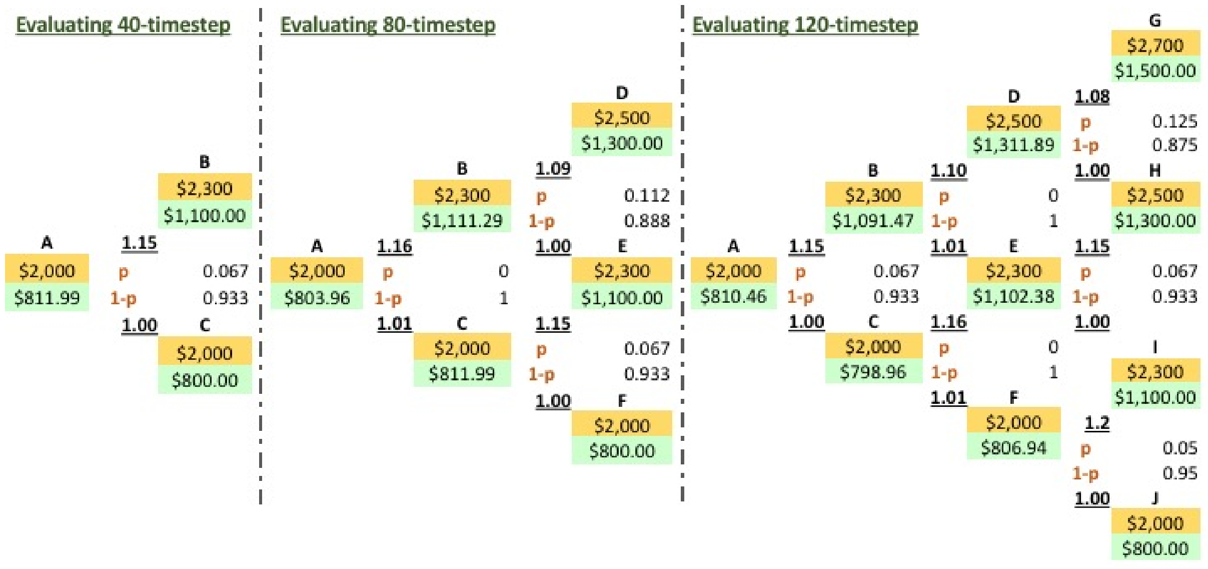


Figure B.2 Binomial Tree for dao_3 (Energy Consumption Concern, i.e. Case 1).

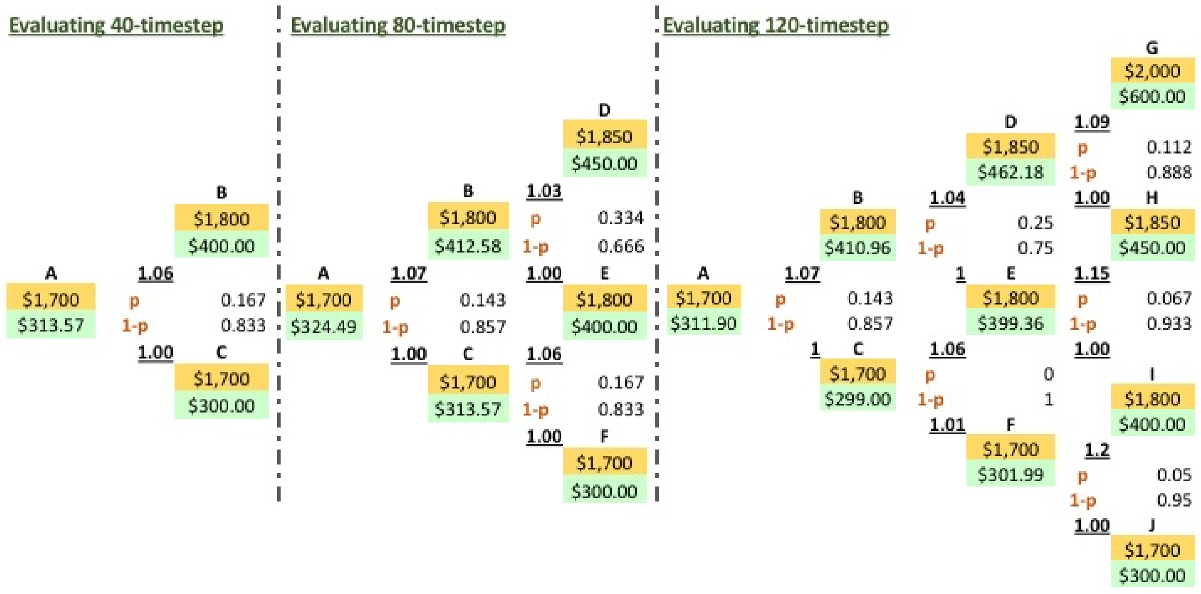


Figure B.3 Binomial Tree of dao_4 for Case 1.

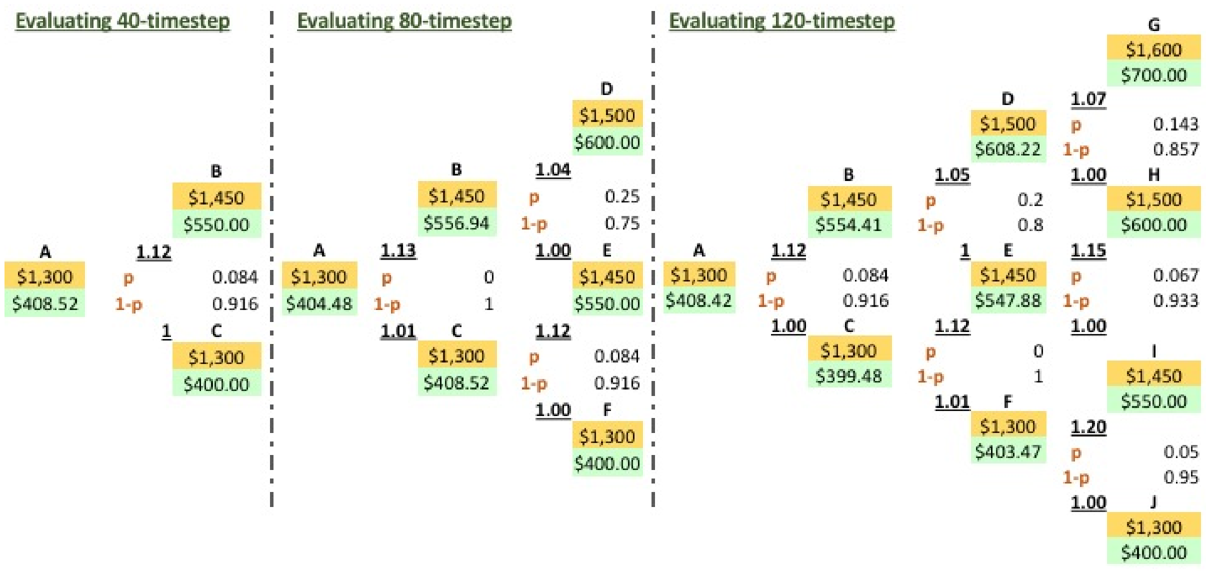


Figure B.4 Binomial Tree of dao_5 for Case 1.

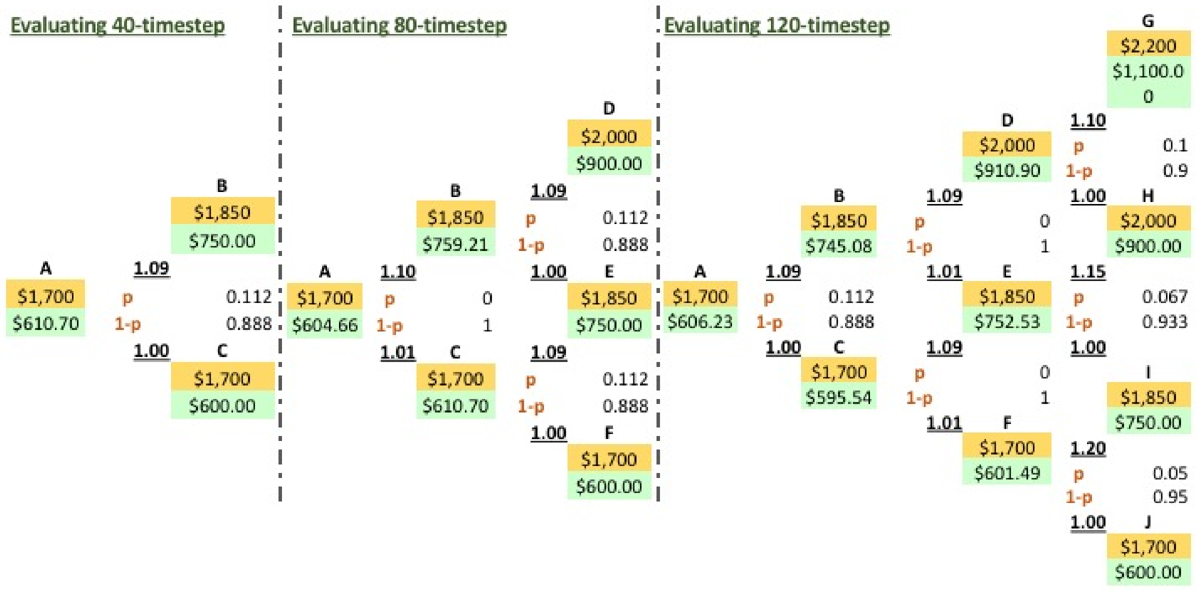


Figure B.5 Binomial Tree of dao_6 for Case 1.

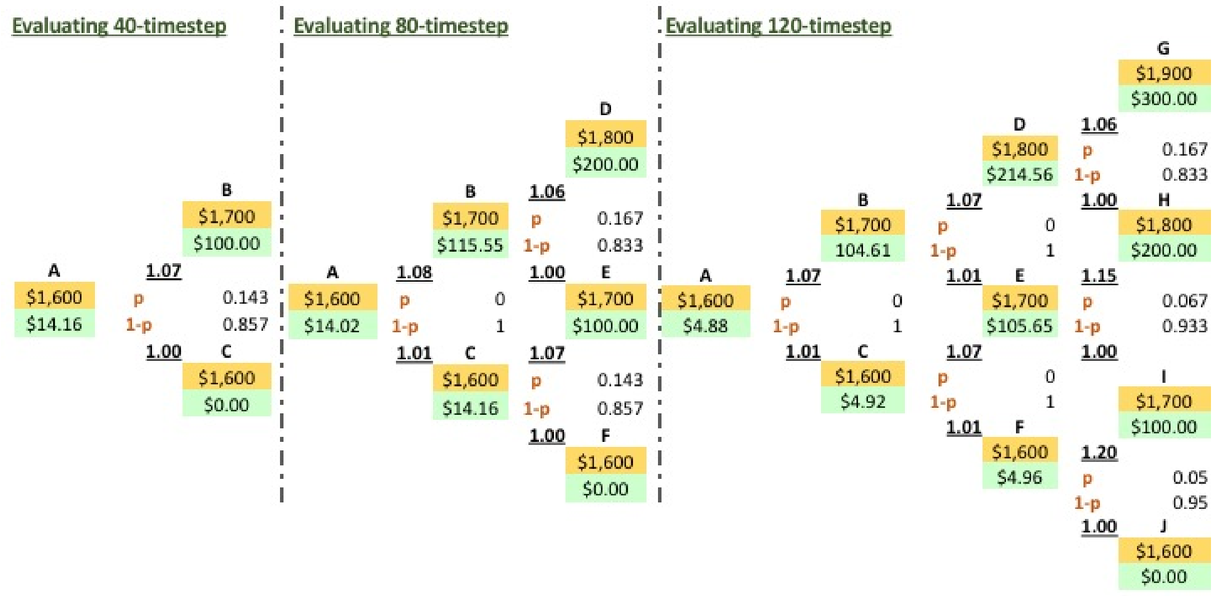


Figure B.6 Binomial Tree of dao_7 for Case 1.

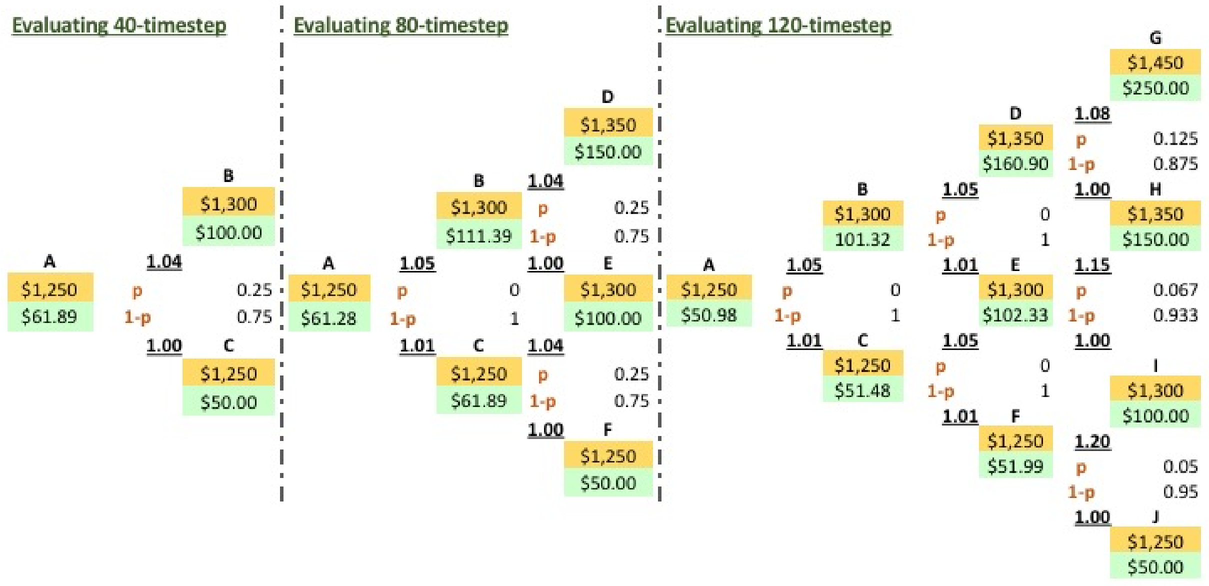


Figure B.7 Binomial Tree of dao_8 for Case 1.

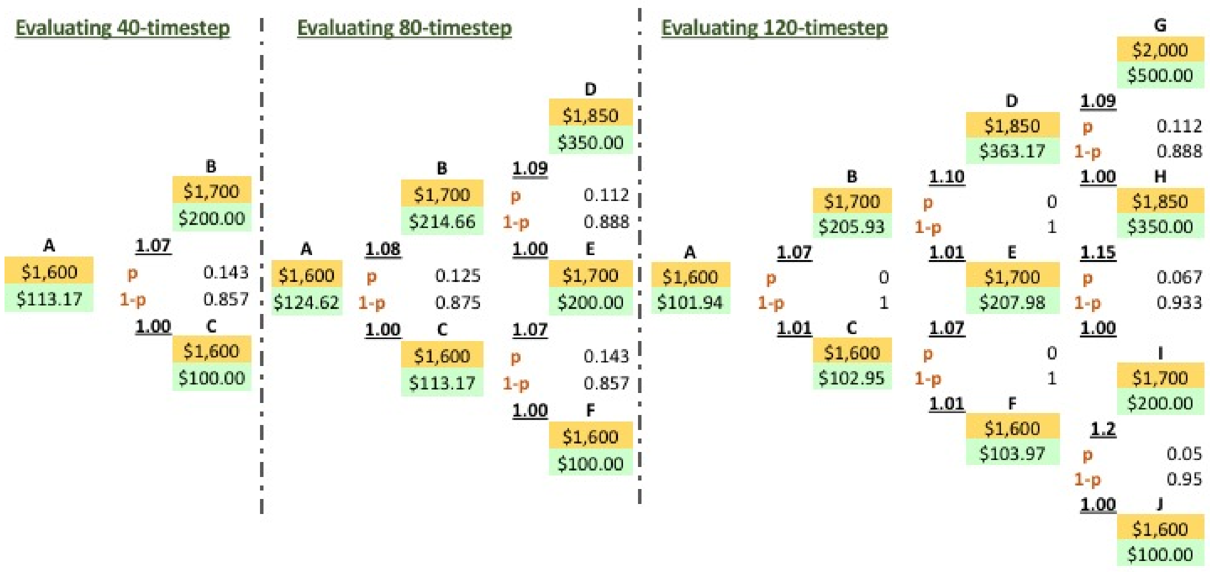
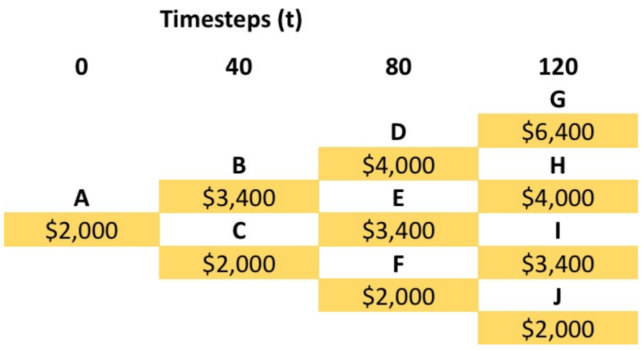
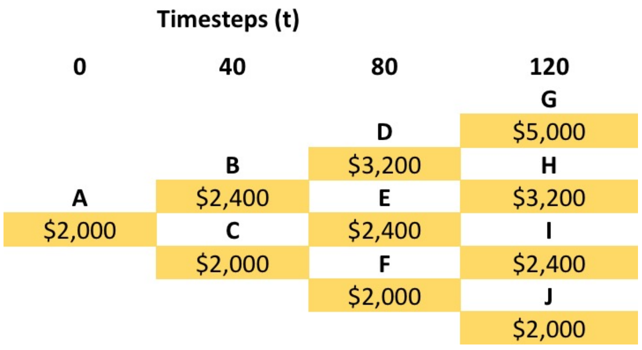


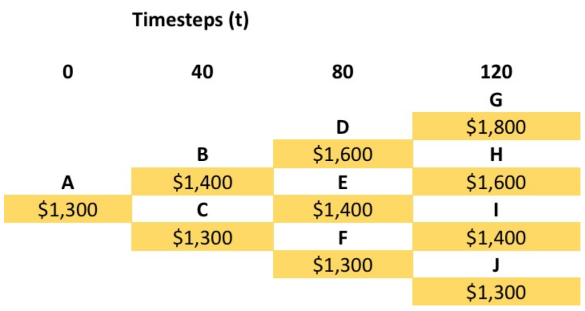
Figure B.8 Binomial Tree of dao_9 for Case 1.



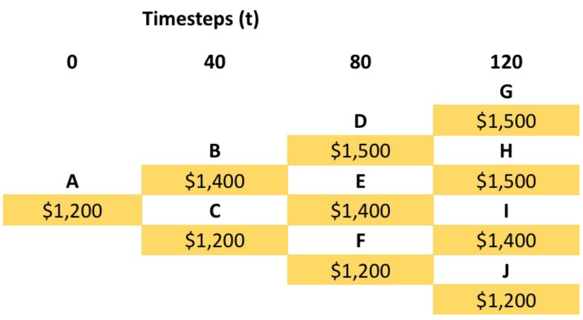
(**a**) Utility Tree for Response time of dao_1.



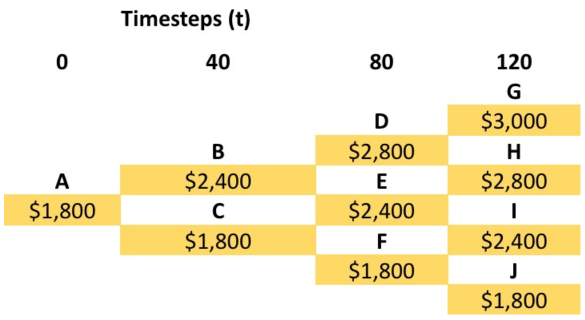
(**b**) Utility Tree for Network Usage of dao_1.



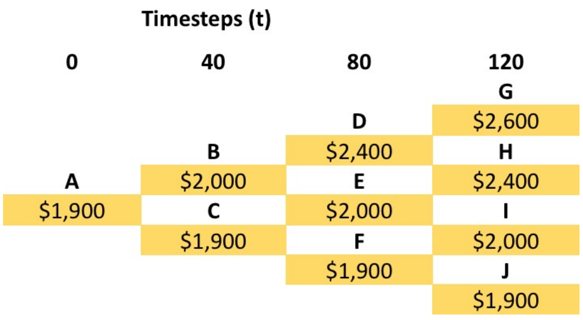
(**c**) Utility Tree for Response time of dao_2.



(**d**) Utility Tree for Network Usage of dao_2.

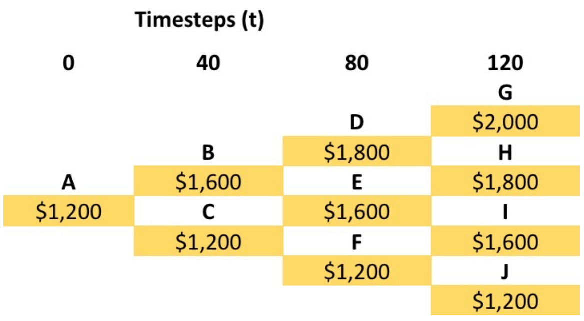


(**e**) Utility Tree for Response time of dao_3.

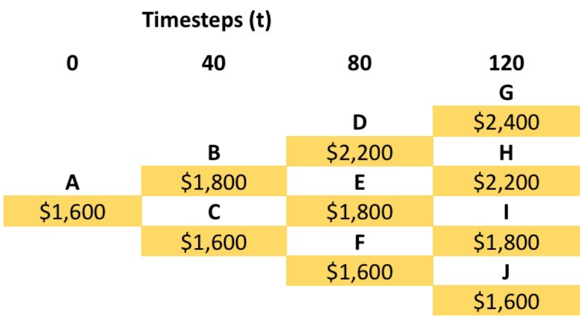


(**f**) Utility Tree for Network Usage of dao_3.

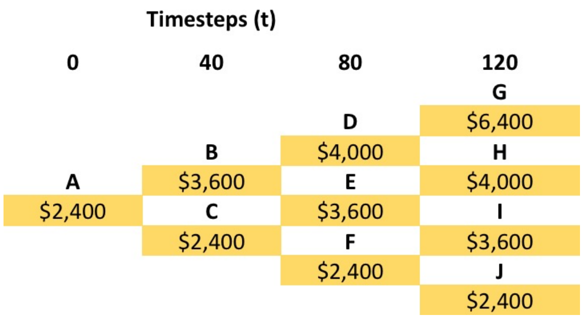
Figure B.9 Utility Tree of options dao_1, dao_2 and dao_3 for Case 2 (Best Scenario).



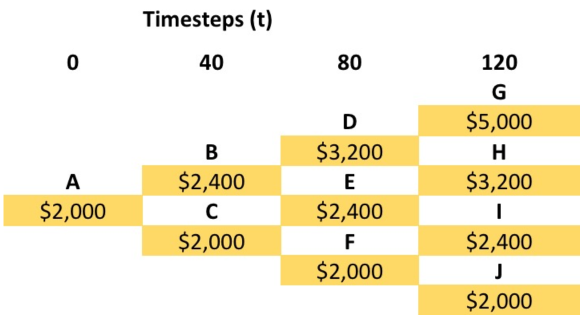
(a) Utility Tree for Response time of dao_5.



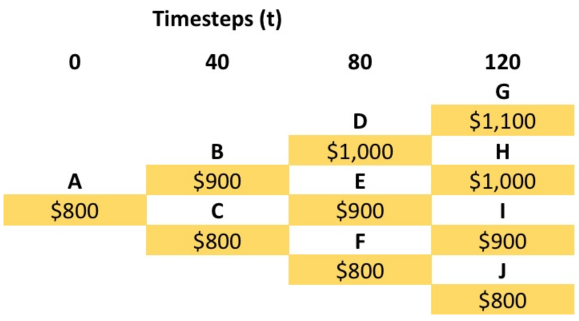
(b) Utility Tree for Network Usage of dao_5.



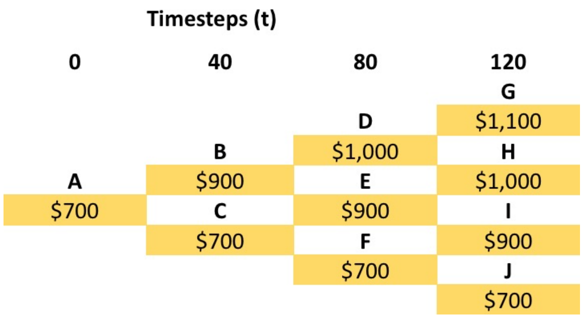
(c) Utility Tree for Response time of dao_6.



(d) Utility Tree for Network Usage of dao_6.

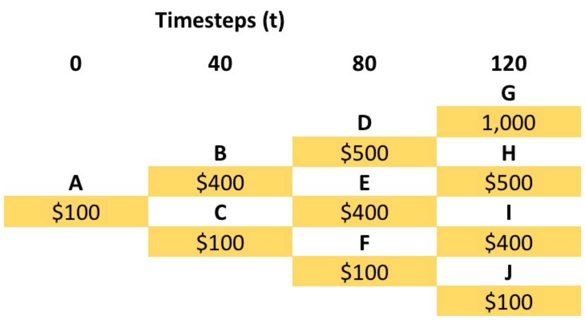


(e) Utility Tree for Response time of dao_7.

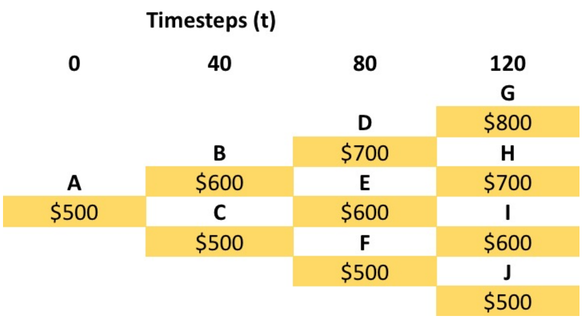


(f) Utility Tree for Network Usage of dao_7.

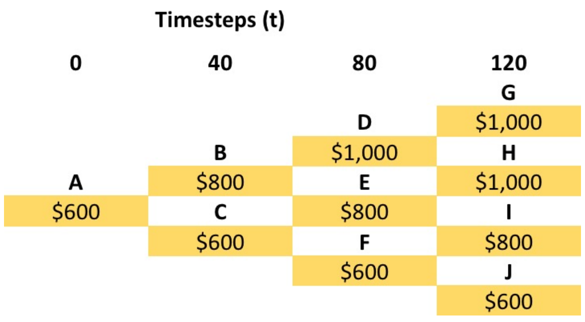
Figure B.10 Utility Tree of options dao_5 to dao_7 for Case 2 (Best Scenario).



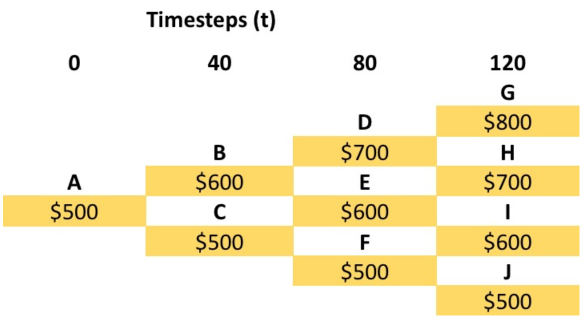
(**a**) Utility Tree for Response time of dao_8.



(**b**) Utility Tree for Network Usage of dao_8.



(**c**) Utility Tree for Response time of dao_9.



(**d**) Utility Tree for Network Usage of dao_9.

Figure B.11 Utility Tree of options dao_8 and dao_9 for Case 2 (Best Scenario).

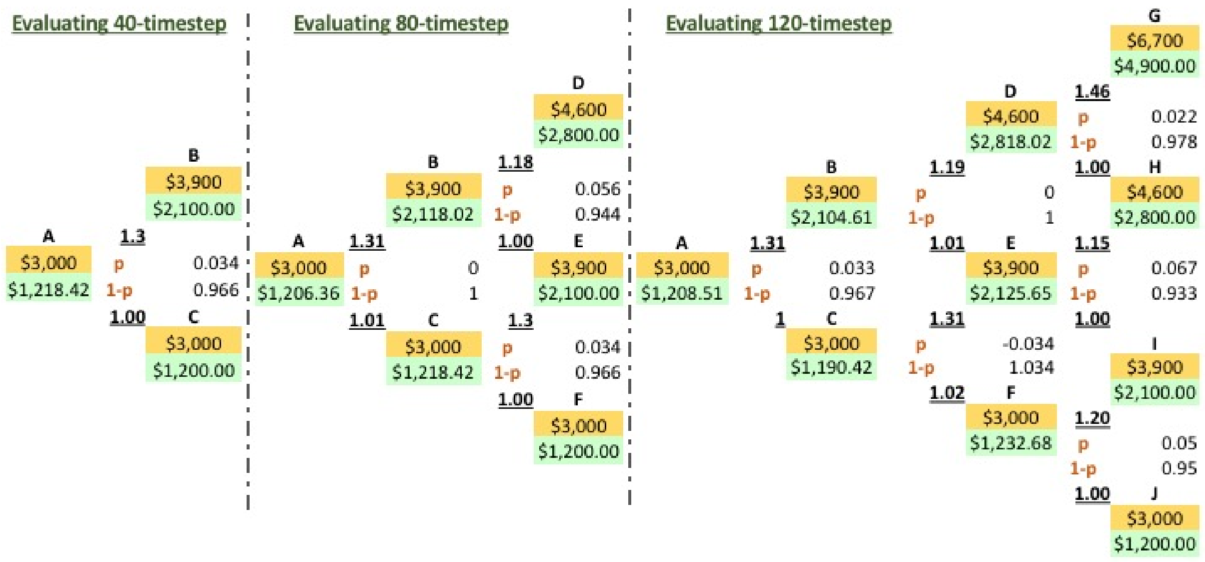


Figure B.12 Binomial Tree of dao_1 for Case 2 (Best Scenario).

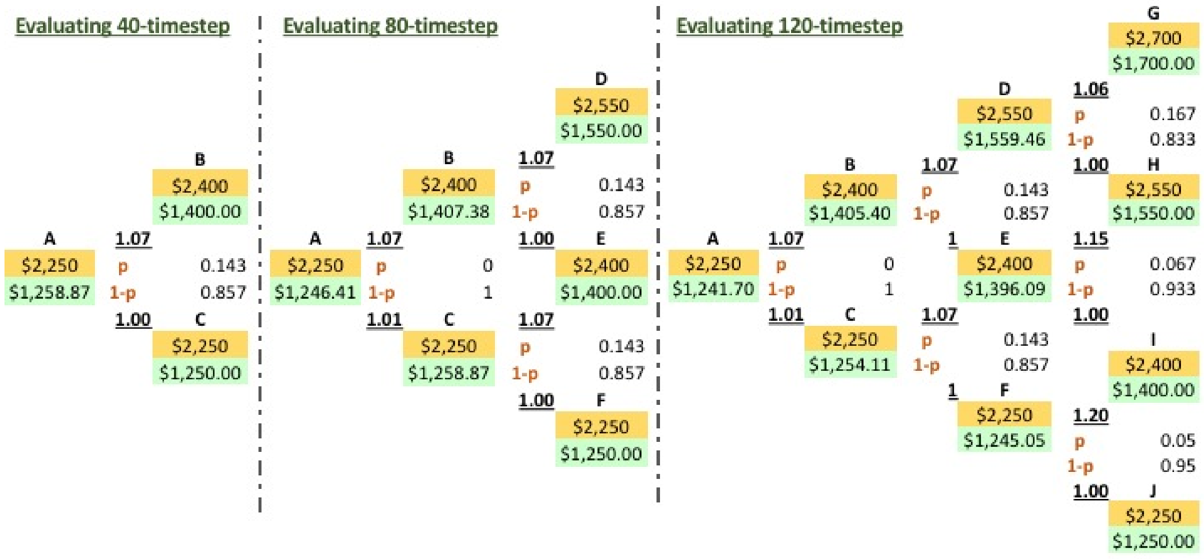


Figure B.13 Binomial Tree of dao_2 for Case 2 (Best Scenario).

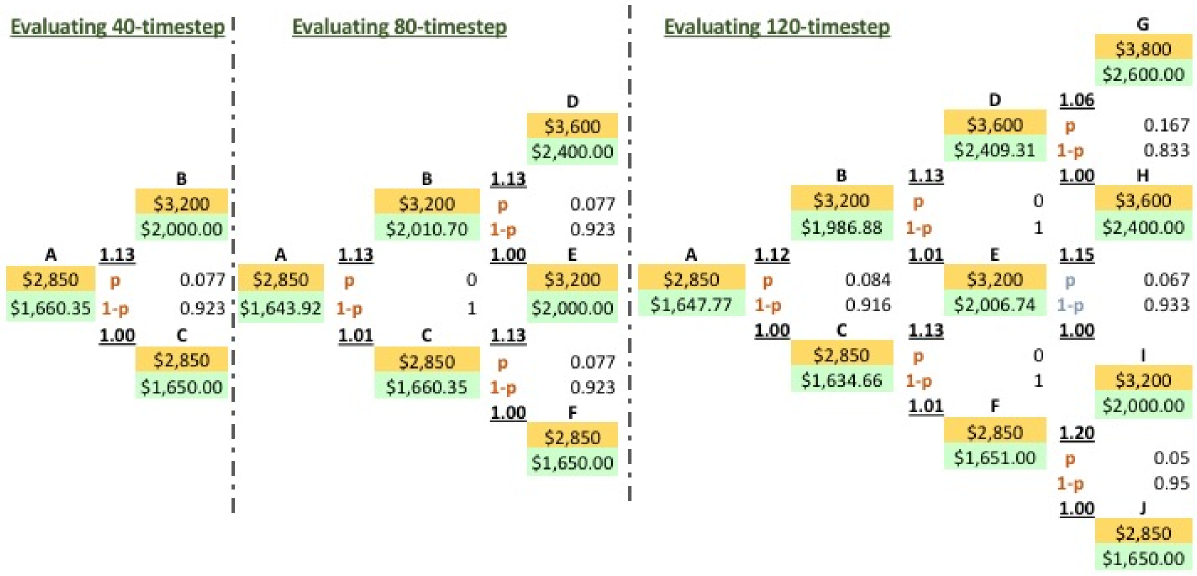


Figure B.14 Binomial Tree of dao_3 for Case 2 (Best Scenario).

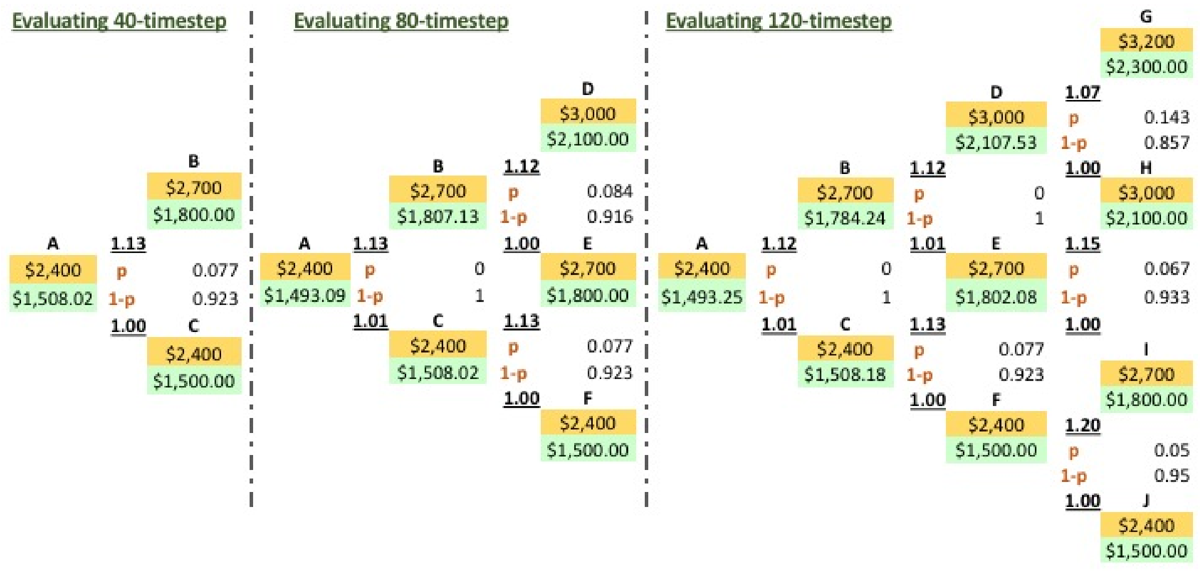


Figure B.15 Binomial Tree of dao_5 for Case 2 (Best Scenario).

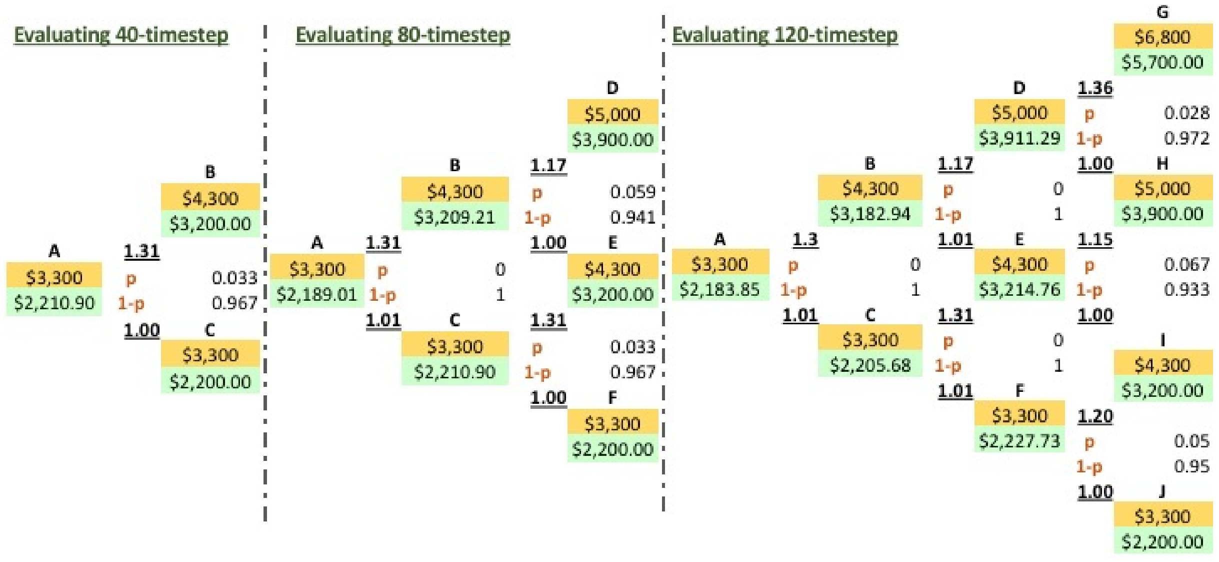


Figure B.16 Binomial Tree of dao_6 for Case 2 (Best Scenario).

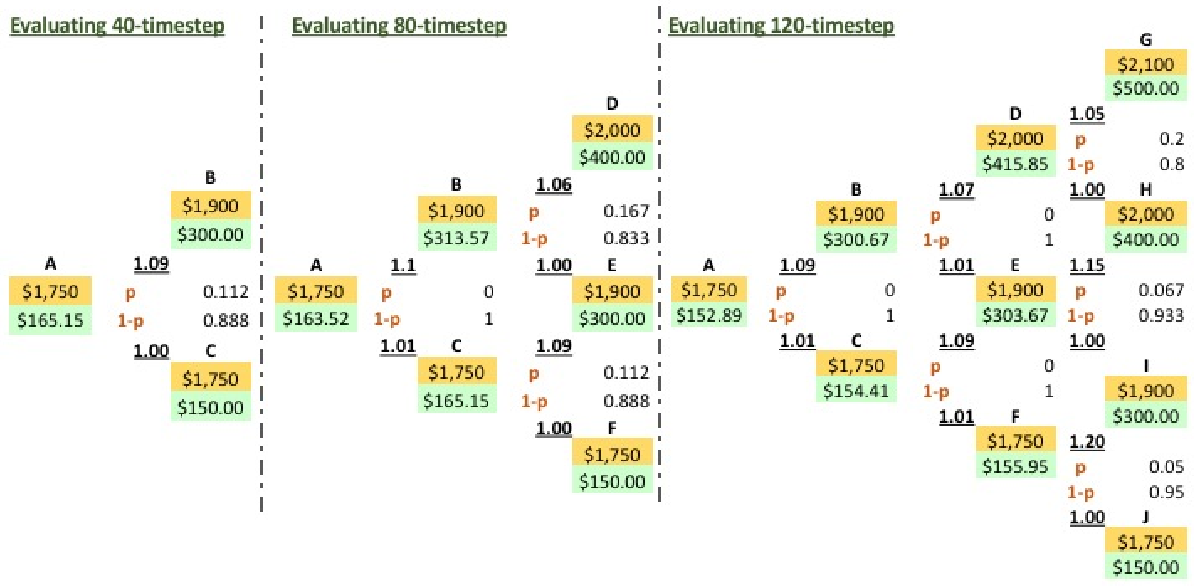


Figure B.17 Binomial Tree of dao_7 for Case 2 (Best Scenario).

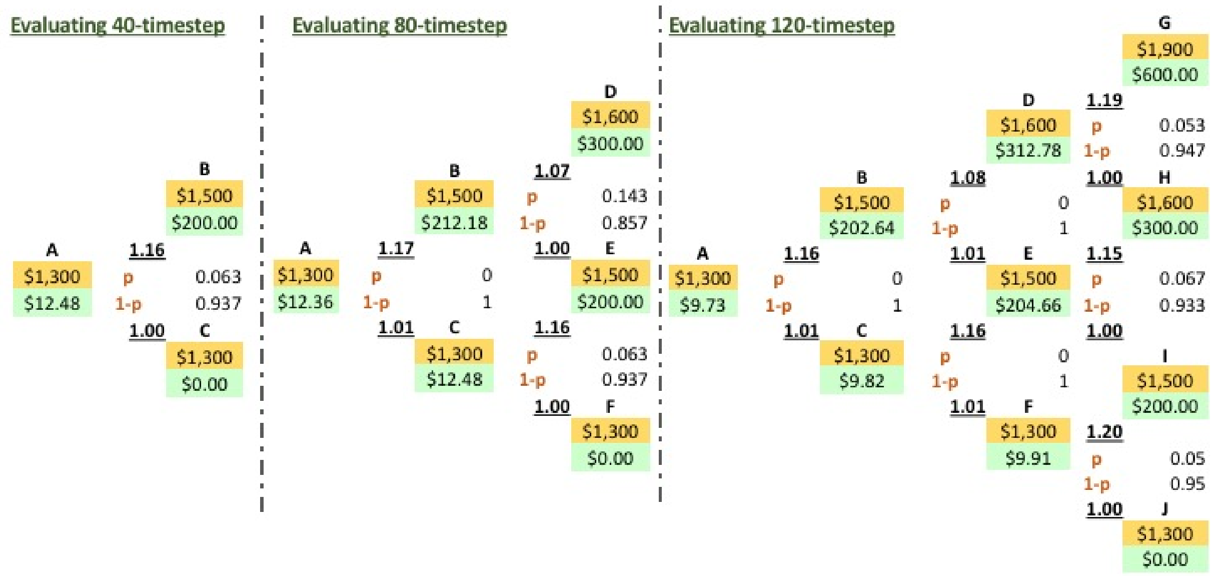


Figure B.18 Binomial Tree of dao_8 for Case 2 (Best Scenario).

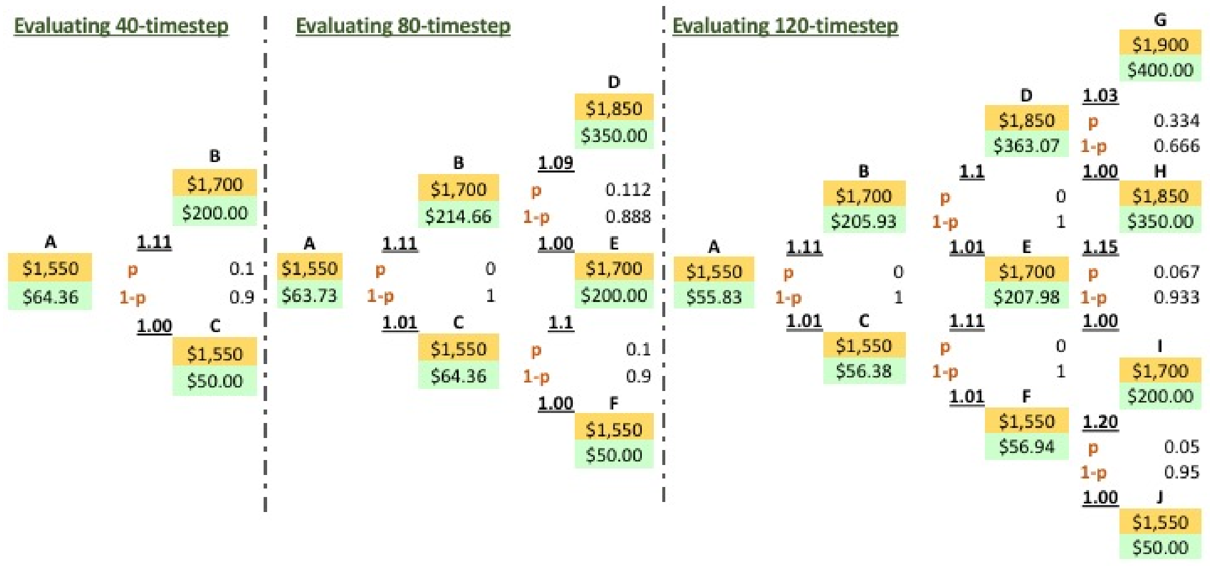
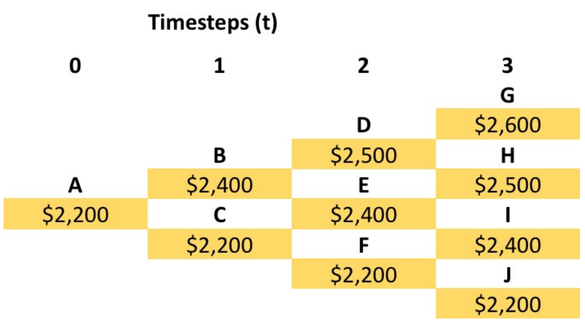
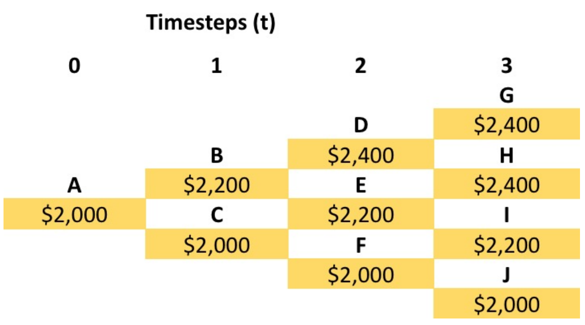


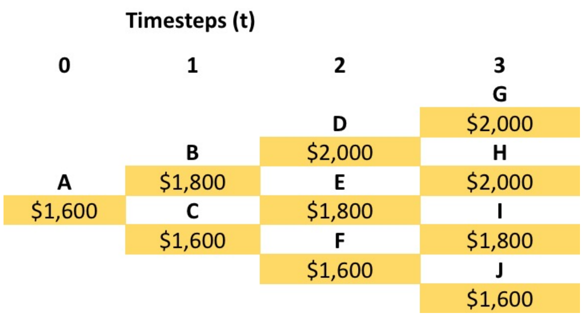
Figure B.19 Binomial Tree of dao_9 for Case 2 (Best Scenario).



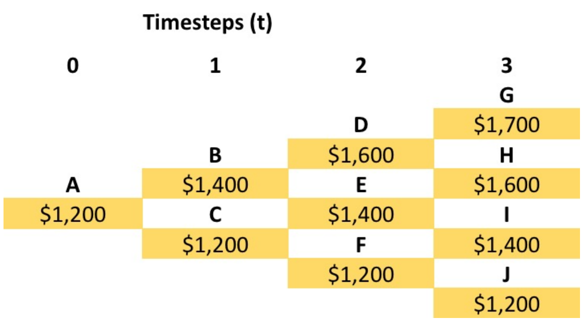
(**a**) Utility Tree for Response time of dao_1.



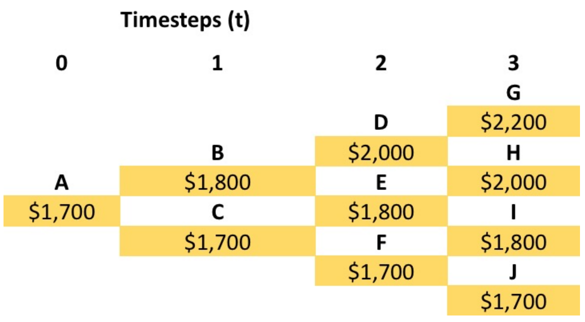
(**b**) Utility Tree for Network Usage of dao_1.



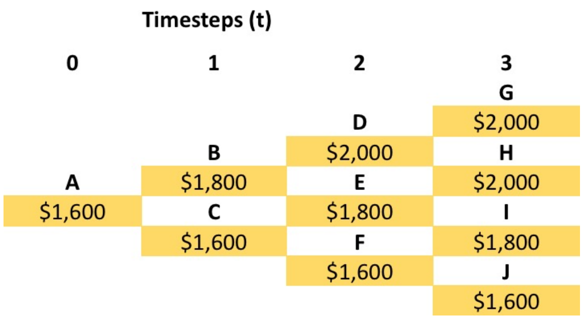
(**c**) Utility Tree for Response time of dao_2.



(**d**) Utility Tree for Network Usage of dao_2.



(**e**) Utility Tree for Response time of dao_3.



(**f**) Utility Tree for Network Usage of dao_3.

Figure B.20 Utility Tree of dao_1 to dao_3 for Case 2 (Worst Scenario).

Timesteps (t)

0	1	2	3
			G
		D	\$1,400
	B	\$1,200	H
A	\$1,000	E	\$1,200
\$800	C	\$1,000	I
	\$800	F	\$1,000
		\$800	J
			\$800

(a) Utility Tree for Response time of dao_4.

Timesteps (t)

0	1	2	3
			G
		D	\$1,500
	B	\$1,400	H
A	\$1,200	E	\$1,400
\$1,000	C	\$1,200	I
	\$1,000	F	\$1,200
		\$1,000	J
			\$1,000

(b) Utility Tree for Network Usage of dao_4.

Timesteps (t)

0	1	2	3
			G
		D	\$1,800
	B	\$1,600	H
A	\$1,500	E	\$1,600
\$1,200	C	\$1,500	I
	\$1,200	F	\$1,500
		\$1,200	J
			\$1,200

(c) Utility Tree for Response time of dao_5.

Timesteps (t)

0	1	2	3
			G
		D	\$1,400
	B	\$1,300	H
A	\$1,200	E	\$1,300
\$1,100	C	\$1,200	I
	\$1,100	F	\$1,200
		\$1,100	J
			\$1,100

(d) Utility Tree for Network Usage of dao_5.

Timesteps (t)

0	1	2	3
			G
		D	\$2,000
	B	\$1,900	H
A	\$1,700	E	\$1,900
\$1,600	C	\$1,700	I
	\$1,600	F	\$1,700
		\$1,600	J
			\$1,600

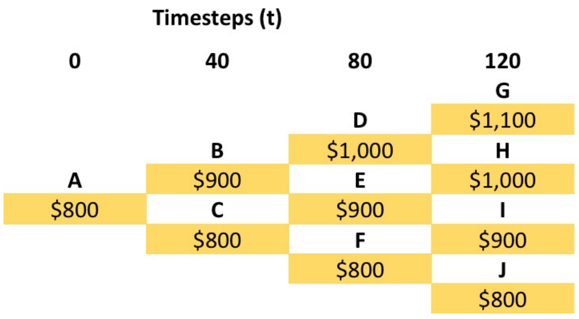
(e) Utility Tree for Response time of dao_6.

Timesteps (t)

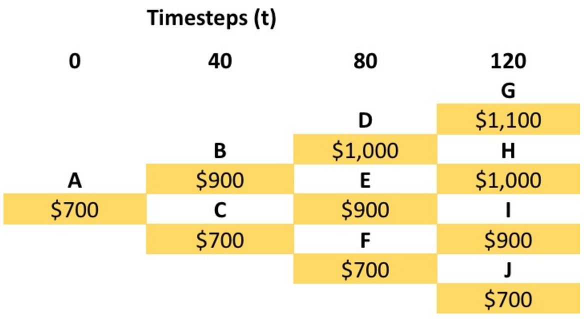
0	1	2	3
			G
		D	\$2,400
	B	\$2,000	H
A	\$1,800	E	\$2,000
\$1,600	C	\$1,800	I
	\$1,600	F	\$1,800
		\$1,600	J
			\$1,600

(f) Utility Tree for Network Usage of dao_6.

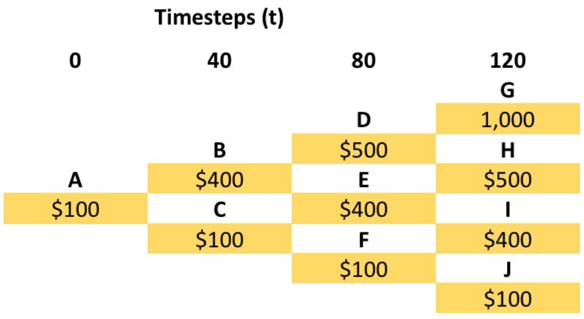
Figure B.21 Utility Tree of dao_4 to dao_6 for Case 2 (Worst Scenario).



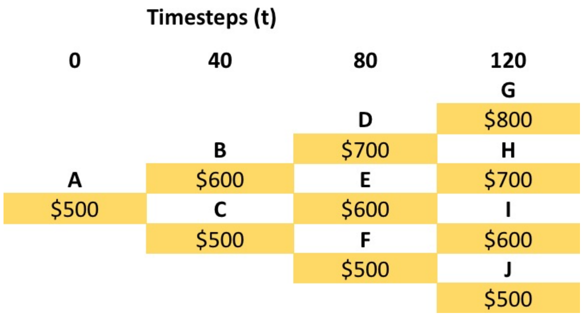
(a) Utility Tree for Response time of dao_7.



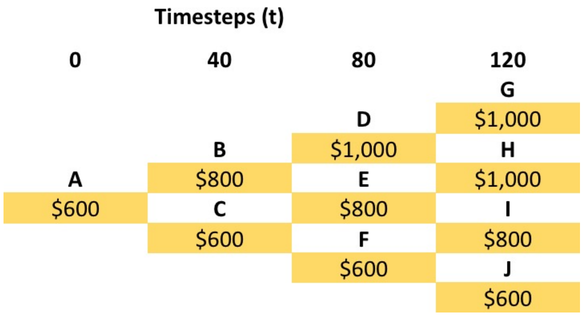
(b) Utility Tree for Network Usage of dao_7.



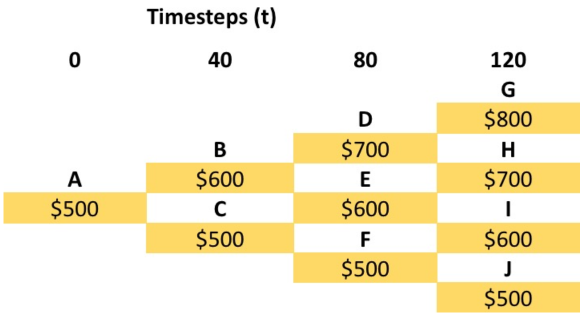
(c) Utility Tree for Response time of dao_8.



(d) Utility Tree for Network Usage of dao_8.



(e) Utility Tree for Response time of dao_9.



(f) Utility Tree for Network Usage of dao_9.

Figure B.22 Utility Tree of dao_7 to dao_9 for Case 2 (Worst Scenario).

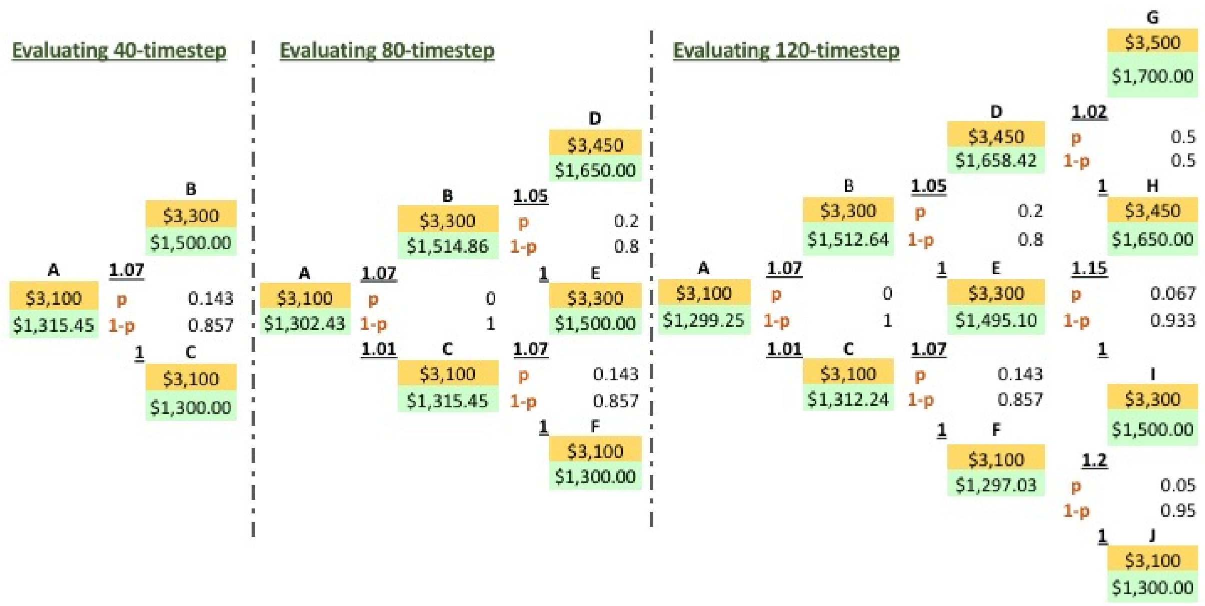


Figure B.23 Binomial Tree for dao_1 for Case 2 (Worst Scenario).

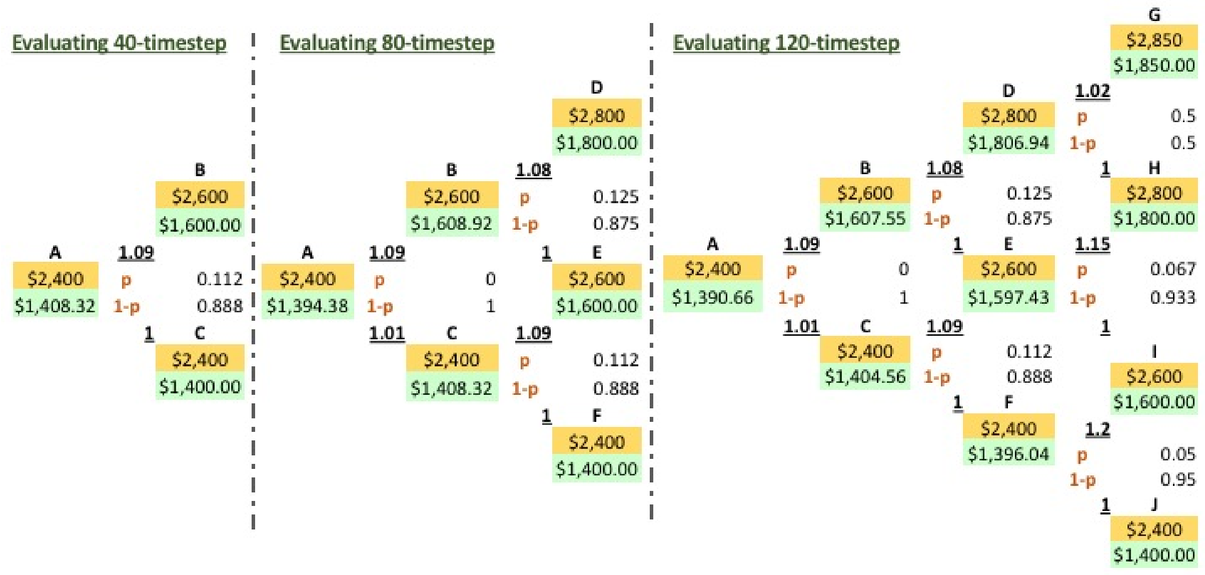


Figure B.24 Binomial Tree for dao_2 for Case 2 (Worst Scenario).

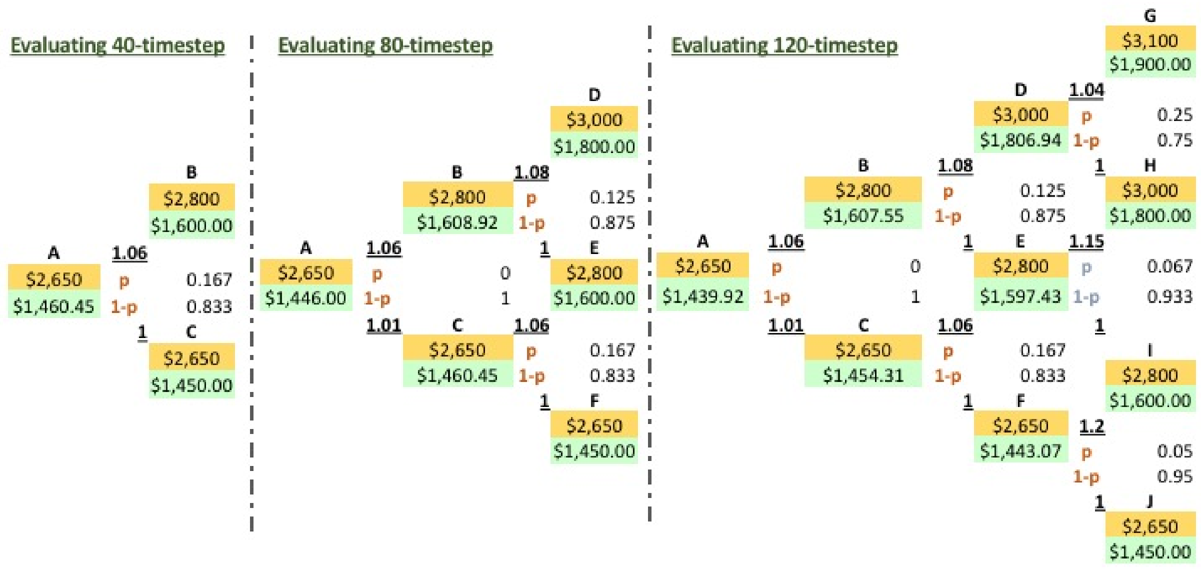


Figure B.25 Binomial Tree for dao_3 for Case 2 (Worst Scenario).

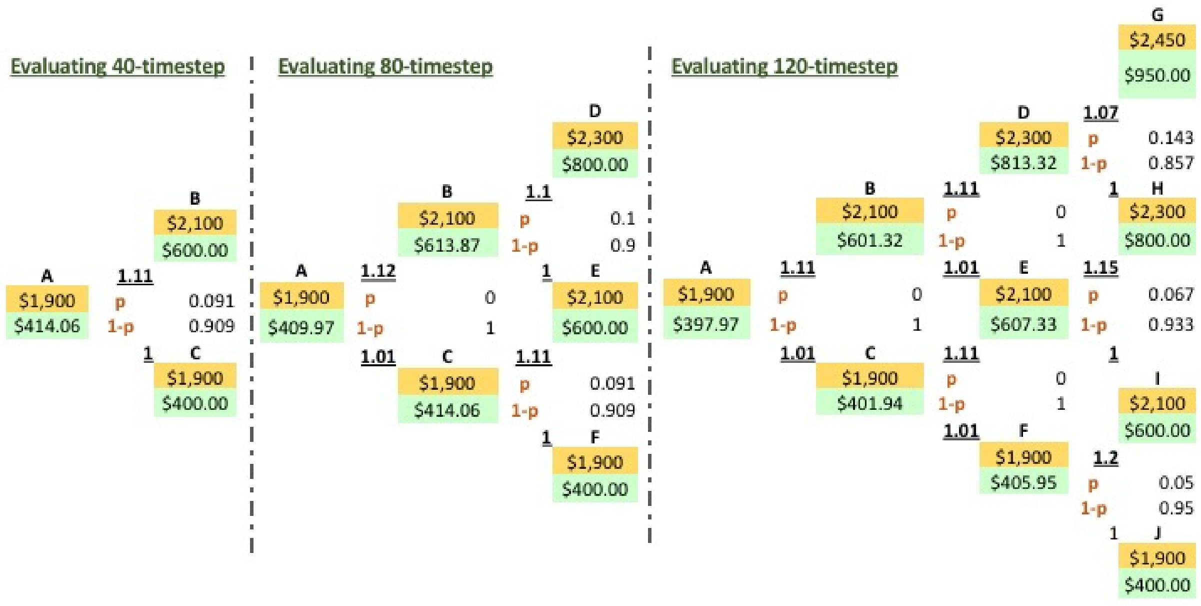


Figure B.26 Binomial Tree for dao_4 for Case 2 (Worst Scenario).

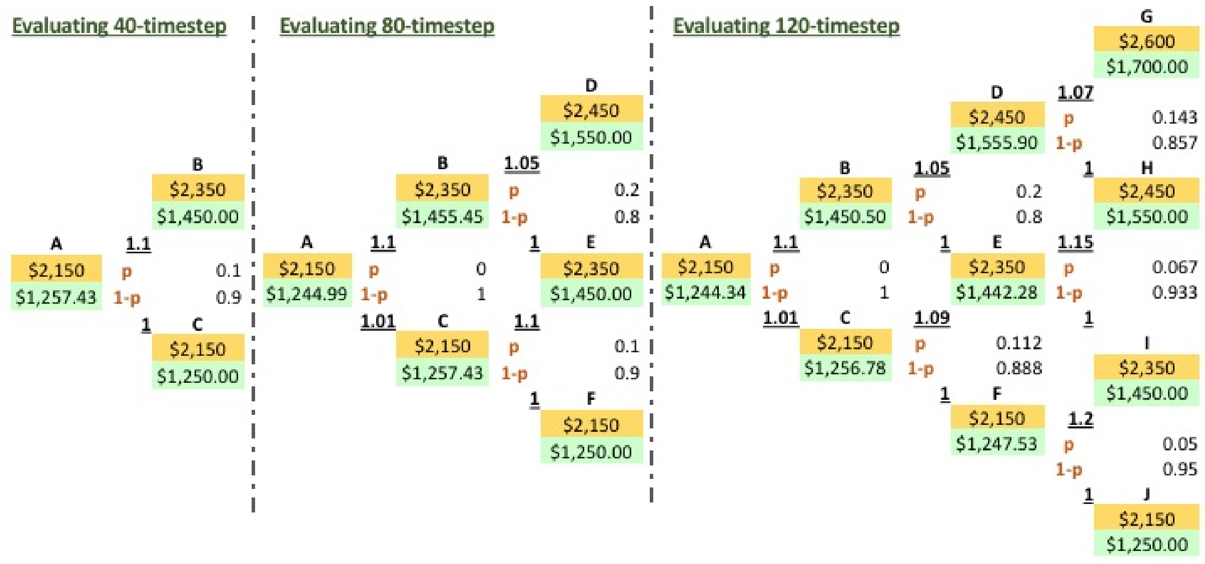


Figure B.27 Binomial Tree for dao_5 for Case 2 (Worst Scenario).

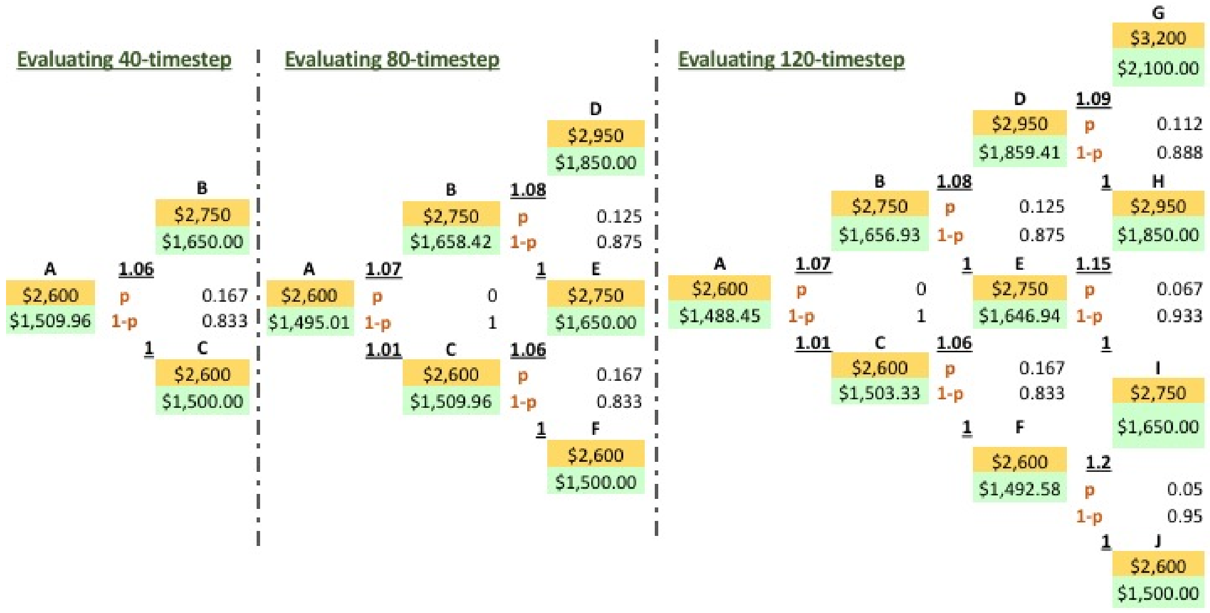


Figure B.28 Binomial Tree for dao_6 for Case 2 (Worst Scenario).

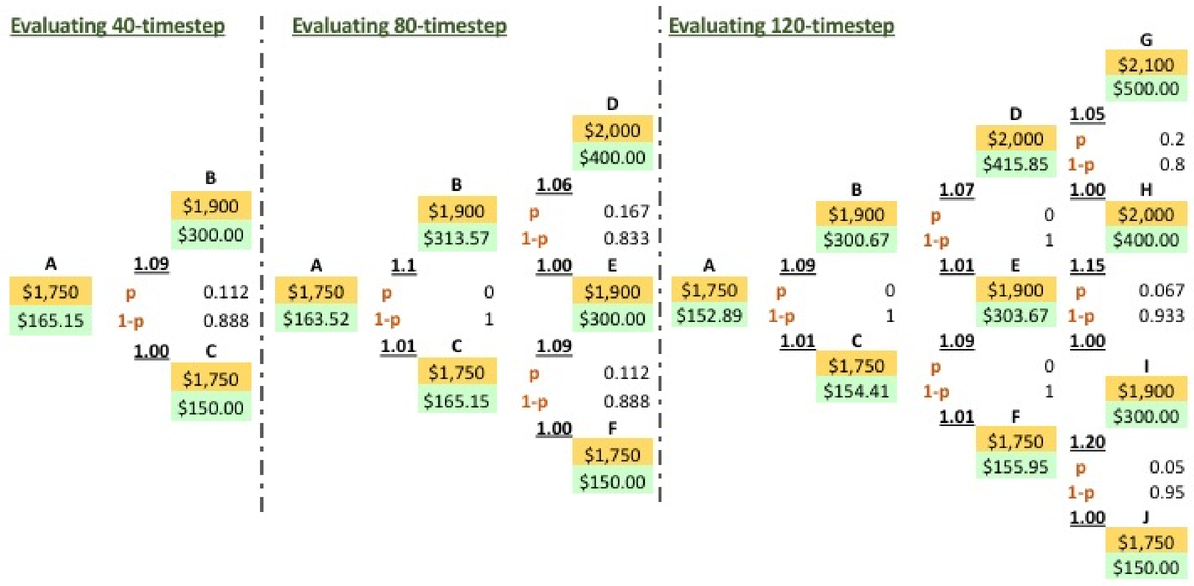


Figure B.29 Binomial Tree for *dao*$_7$ for Case 2 (Worst Scenario).

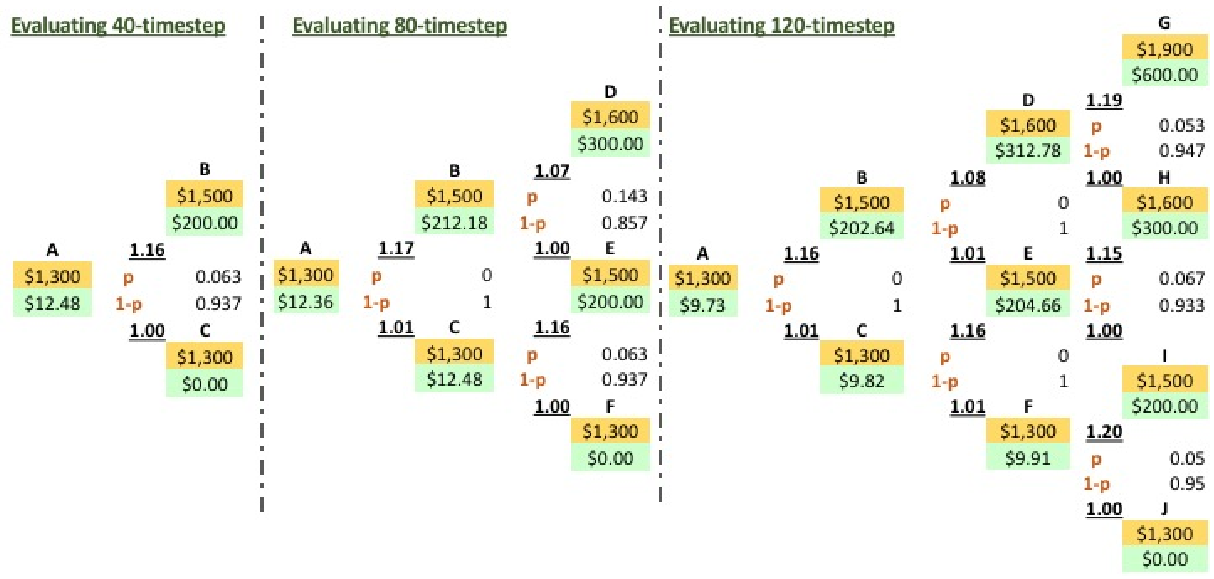


Figure B.30 Binomial Tree for *dao*$_8$ for Case 2 (Worst Scenario).

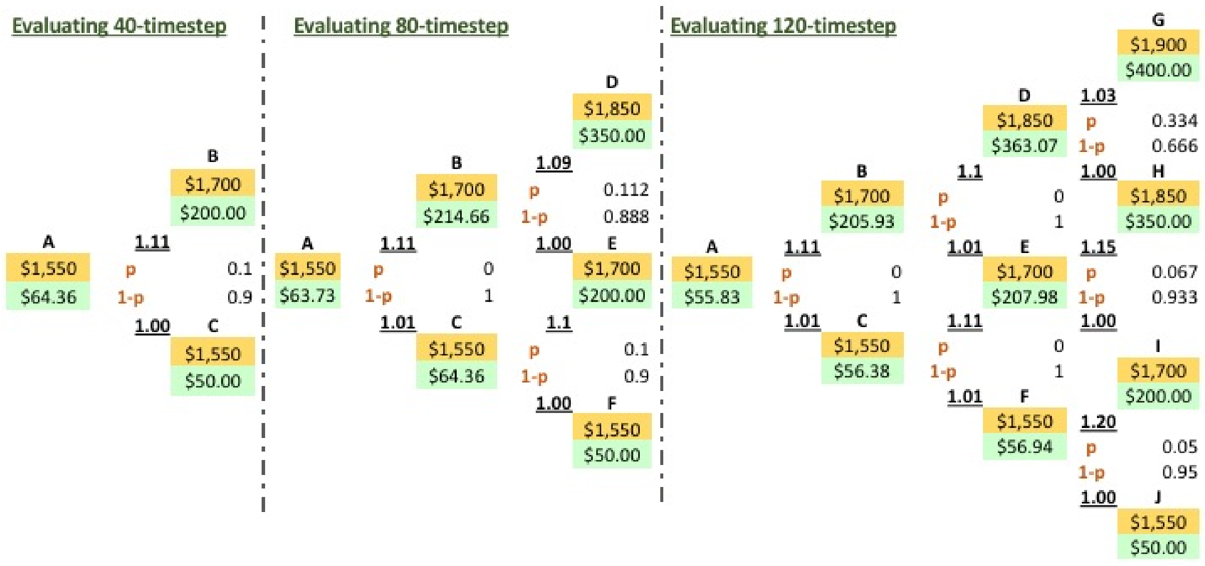


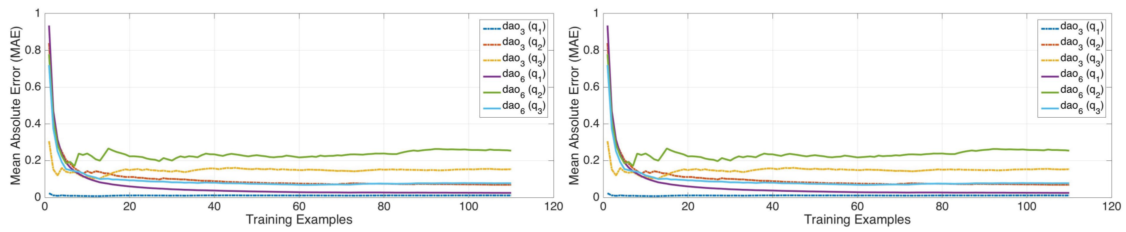
Figure B.31 Binomial Tree for *dao*$_9$ for Case 2 (Worst Scenario).

Appendix C

Other Evaluation Results For The Proactive Approach

C.1 Plots for RQ4.1.2

In this section, we plot a representation for MAE and RMSE for three quality attributes to determine the number of training examples for kNN Learner for t+h, where $h \in \{5, 8, 15\}$.



(a) MAE **(b)** RMSE

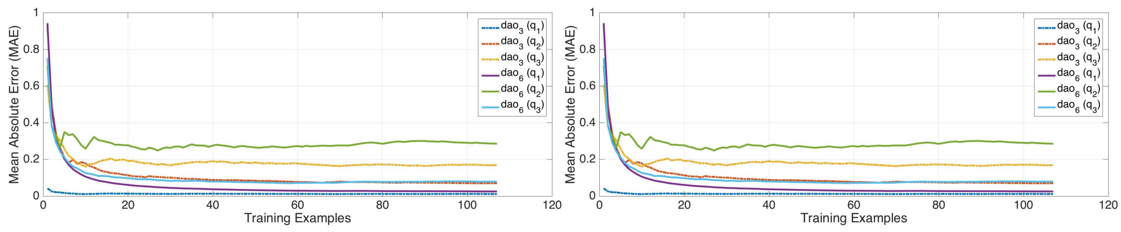
Figure C.1 The error (MAE and RMSE) over time for 3 quality attributes to determine the number of training examples for kNN Learner for **t+5**.

C.1.1 Learning Parameters

In this section, we aim to tabulate the parameters of learners adopted for experiment (RQ 4.1.3,4.2.1, and 4.2.2) in Table C.1.

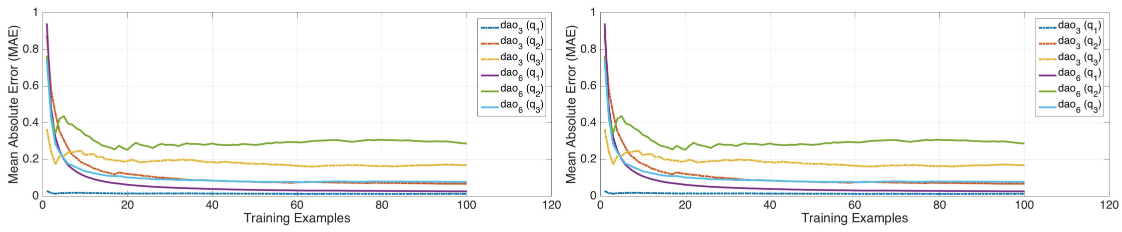
C.2 Additional Experimental Evaluation

In this section, we show the analysis of RQ4.2.2 in detail for Normal, Case 1 and Case 3.



(a) MAE **(b)** RMSE

Figure C.2 The error (MAE and RMSE) over time for 3 quality attributes to determine the number of training examples for kNN Learner for **t+8**.



(a) MAE **(b)** RMSE

Figure C.3 The error (MAE and RMSE) over time for 3 quality attributes to determine the number of training examples for kNN Learner for **t+15**.

Table C.1 The parameters for the single and ensemble learners.

Learner	Parameter	Value
kNN	Number of neighbours	3
Perceptron	Learning rate	0.025
	Fading factor	0.9
SGD Multiclass	Learning rate	0.01
FIMTDD	Tie threshold	0.5
	Learning rate	constant
	Tree type	regression tree
FTM	Fading factor	0.9
AddExp	Factor for decreasing the weights of experts	0.5
	Initial expert weight	0.1
	Loss threshold	0.01

In this section, we show the analysis of RQ4.2.2 in detail for Normal, Case 1 and Case 3. We also present the analysis of the behaviour of the static, predefined and random approaches.

C.2.1 Analysis of Static, Predefined, and Random Approaches Overall Behaviour:

- *Static-Selection:*
  - Generally, for the best case scenario, the static-selection for all the cases provided better mean exponential benefit with lower cost as compared with predefined-selection and random-selection approaches (Table 6.9). However, its mean exponential benefit and cost were much worse than reactive and proactive approaches, except for Case 3 (Table 6.9 and 6.10).
  - The static-selection approach is ignoring the run-time changes. In this context, it has deployed only one *dao* over time (i.e it has zero switches as seen in Table 6.9) based on the design-time knowledge (i.e. benefit estimations from the architects). So no matter how this *dao* will behave over time in terms of benefit, the static-selection will keep using it. For example, for the best case scenario, a *dao* has been selected by the architects, which seemed to provide good mean exponential benefit (Table 6.9), as compared to predefined and random approaches. However, for the worst case scenario, the selected *dao* was always violating the QoS constraints over time except for Case 2, resulting in a zero mean exponential benefit (Table 6.9).
- *Predefined selection:* In all the cases (Table 6.9 and 6.10), the predefined selection behaved the same. It is providing the lowest mean exponential benefit, highest mean cost, and highest number of switches (8 switches). This is because it has predefined *DAO* for deployment, regardless of their benefit at run-time, it will keep using them.
- *Random-Selection:* In most of the cases, the random-selection approach produces low mean exponential benefit with high cost and increased number of switches (Table 6.9) as compared with static-selection, reactive and proactive approaches. Though the random-selection approach detects the changes in *dao*'s benefit, it suggests replacements to random *DAO*. In particular, the ad hoc selection in Random-selection approach caused the approach to recommend *DAO* which are not always the best, resulting in poor exponential benefit over time. More detailed representative analysis

for the behaviour of the random-selection approach is shown after the summary of results.

- *Overall:* The static-selection and predefined-selection approaches are biased towards their design-time decisions, which may (not) provide good mean exponential benefit and cost (Table 6.9). Further discussion related to how these approaches operate over time is shown in the representative example (i.e. Case 2) explained after the summary of results. Finally, the predefined- and random-selection approaches showed the worst number of switches, mean exponential benefit and cost, as compared with other approaches (Table 6.9 and 6.10).

C.2.2 Analysis (Best/Worst Case Scenario for Normal):

- For the static-selection approach, it will always use the same option (i.e. dao_6) regardless of exponential benefit changes at run-time (Figure C.5b). In this context, the current *dao* showed a gradual reduction with high fluctuation in the exponential benefit (from about 0.96 to 0.5) till t=75. After that, a significant rise in exponential benefit to 0.82 has occurred at $t = 90$, followed by a noticeable oscillation in benefit till the end. As for the cost, it kept constant (i.e. 0.5) from the beginning till the end. (Figure C.5c).

- The predefined-selection is similar to static-selection in terms of ignoring the run-time changes (Figure C.5a and C.5b, and Table 6.9). This has resulted in having a very low exponential benefit with an average of 0.5268, and relatively high mean cost of 0.5439 in comparison to reactive and proactive approaches, with very high number of switches (i.e. 8 switches).

- The random-selection approach selects an ad hoc *dao*, which results in a significant drop in exponential benefit (from 0.95 to 0.3) till t=25 (Figure C.5b). The random-selection approach then detected changes in exponential benefit different from the reactive and proactive approach (Figure C.5b). For instance, in one of 30 runs, it kept using dao_5 from $t = 25$ to $t = 70$ because it did not detect any change for this period. This is due to the gradual rise in exponential benefit till t=30, followed by a steady rate (i.e. low standard deviation in exponential benefit, which caused no change detection) till t=70.

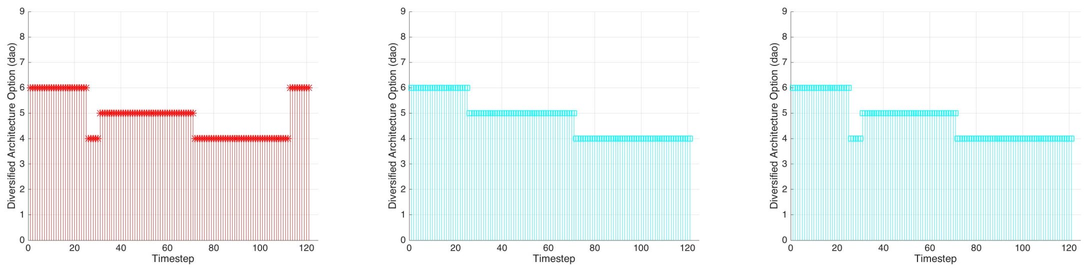
- At t=70, the random approach has detected a significant decrease in benefit, which led to a switch to a random *dao* (i.e. not always the optimal). This has also resulted in a noticeable decrease in benefit over time (from 0.62 to 0.42), as seen in Figure C.5b and C.5e. Similar changes and selections of *dao* happened till t=120.

- The reactive and proactive approaches recommended using the same *dao* (Figure C.4) as it is still the best one until the 25^{th} timestep.

- After $t = 30$ (Figure C.4), the reactive and proactive approaches were evaluating dao_5 till $t = 70$. They then detected a significant drop in dao_5's exponential benefit from about 0.8 to 0.6. Both approaches have analyzed the cost-benefit trade-offs of the *DAO* and hence recommended a replacement to dao_4.

- From $t = 70$ to 112, the reactive and proactive approaches provided the best steady exponential benefit, approximately 0.85, with a cost of 0.52 (Figure C.5b and C.5c).

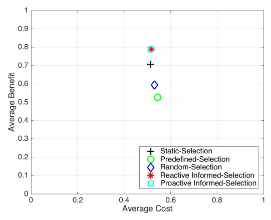
- At $t = 112$, the reactive and proactive approaches found that the current *dao* provides poorer exponential benefit than previous (from 0.85 to 0.75), as depicted in Figure C.5b. The reactive approach thus suggested a switch to dao_6 (Figure C.4a). The proactive approach avoided these unnecessary replacements (Figure C.4b), because it has the ability to evaluate the *DAO* based on their future benefits. For instance, from $t = 70$ till the end, the proactive approach noticed that dao_4 provides the most balanced cost-benefit trade-off.



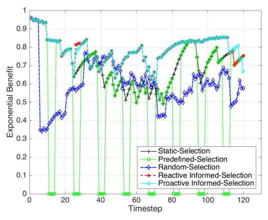
(a) The selected *DAO* by reactive approach **(b)** The selected *DAO* by proactive approach $(t+1)$ **(c)** The selected *DAO* by proactive approach (Average Forecasts)

Figure C.4 The selected *DAO* by reactive/proactive $(t+1)$ approach for **Normal** for the *best* scenario.

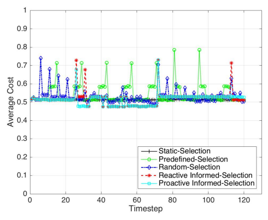
To this regard, the proactive informed-selection produced almost the same average exponential benefit and cost as the reactive approach but with less number of switches for the *normal* case (Table 6.10).



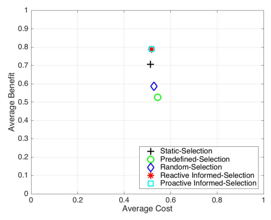
(a) Overall behaviour ($t+1$, Best).



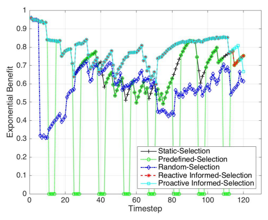
(b) Exponential Benefit ($t+1$, Best).



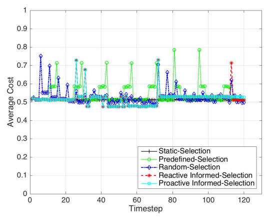
(c) Average Cost ($t+1$, Best).



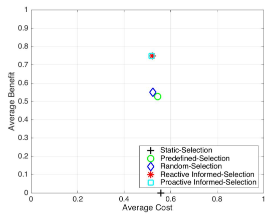
(d) Overall Behaviour (Avg Forecasts, Best).



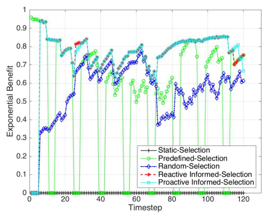
(e) Exponential Benefit (Avg Forecasts, Best).



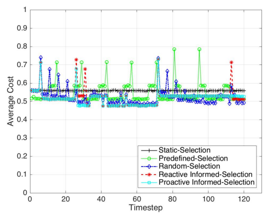
(f) Average Cost (Avg Forecasts, Best).



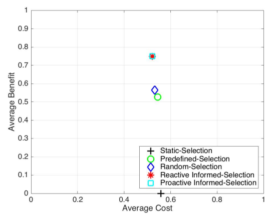
(g) Overall Behaviour ($t+1$, Worst).



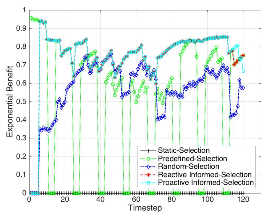
(h) Exponential Benefit ($t+1$, Worst).



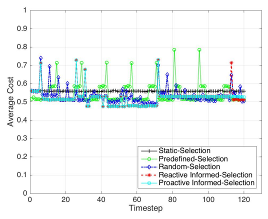
(i) Average Cost ($t+1$, Worst).



(j) Overall Behaviour (Avg Forecasts, Worst).

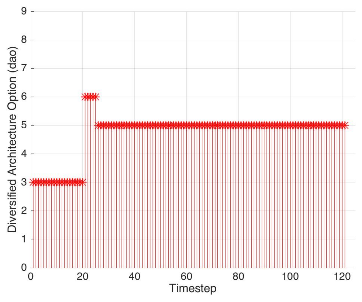


(k) Exponential Benefit (Avg Forecasts, Worst).

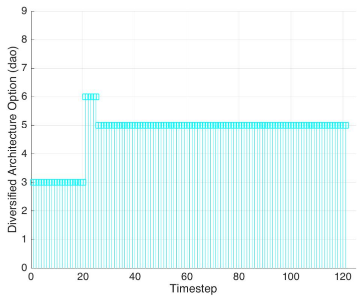


(l) Average Cost (Avg Forecasts, Worst).

Figure C.5 The evaluation of the five approaches' decision-making process for **Normal** ($t+1$, Average Forecasts) for the *best/worst* scenario.



(a) The selected *DAO* by reactive approach



(b) The selected *DAO* by proactive approach

Figure C.6 The selected *DAO* by reactive/proactive (Average Forecasts) approach for Case 1 for the *best* scenario.

C.2.3 Analysis (Best/Worst Case Scenario for Case 1)

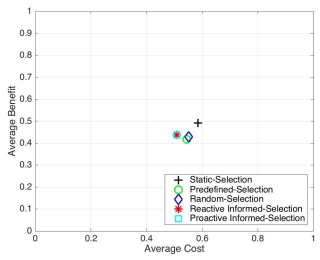
Consider the case where the architect has selected a *dao*, which seemed to initially provide the *most balanced* cost-benefit trade-off to handle the energy consumption strict constraint (i.e. dao_3). We also analyze the results with respect to varying values of h and average forecast of $t+h$, where $h \in \{1,5,8,15\}$ (Table 6.10). In the *best* case, the proactive approach for varying h values and average forecasts behaved the same. Therefore, we plotted only the average forecasts in Figure C.7a-C.7c. Next, we discuss the Behaviour of the architecture evaluation approaches, as follows:

- The reactive and proactive (for all $t+h$ and their average) approaches initially evaluated dao_3, which imposed a significant degradation in exponential benefit at $t = 20$ (Figure C.7b). For that, both approaches suggested a replacement to dao_6 (Figure C.6a and C.6b), which provided as well poor benefit (Figure C.7b). Therefore, at $t = 25$, the reactive and proactive approaches quickly handled that and thus suggested a replacement to dao_5. This *dao* provided the most balanced cost-benefit trade-off (Figure C.7b and C.7c). However, since the fluctuation in QoS was considerably low, both approaches did not detect any significant deviations in benefit after $t = 25$.

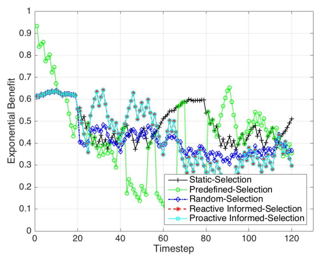
- Further, we did not benefit greatly from future forecasts. This is due to the too strict constraint for energy consumption (i.e. various constraint violation has occurred). This explains why the proactive approach for all $t+h$ and their average provided similar results to the reactive approach (Figure C.7a-C.7c).

- As for random- and predefined-selection approaches (Figure C.7a), they provided very low exponential benefit (i.e. 0.45) with high cost (i.e. 0.55), in comparison to reactive and proactive approaches.

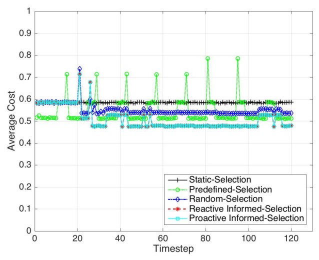
- The static-selection also provided lower benefit (i.e. 0.7) and higher cost (i.e. 0.5) than reactive and proactive approaches (Figure C.7a).



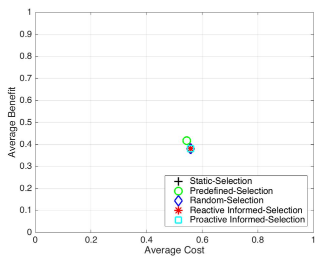
(a) Overall Behaviour (Avg Forecasts, Best).



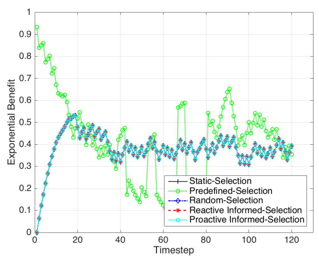
(b) Exponential Benefit (Avg Forecasts, Best).



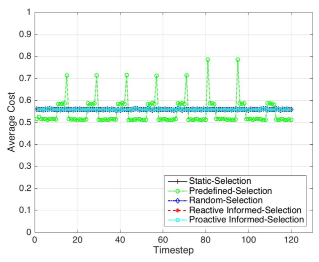
(c) Average Cost (Avg Forecasts, Best).



(d) Overall Behaviour (Avg Forecasts, Worst).



(e) Exponential Benefit (Avg Forecasts, Worst).



(f) Average Cost (Avg Forecasts, Worst).

Figure C.7 The evaluation of the five approaches' decision-making process for **Case 1** (Average Forecasts) for the *best/worst* scenario.

For the *worst* case scenario, the proactive approach for varying $t+h$ produced different results (Figure C.9, Table 6.10), and hence We will report the results below (Figure C.8a-C.8f):

- The predefined-selection approach (Figure C.8a) provided similar output to best case scenario.

- Additionally, the static, random, and reactive approach produced the same average exponential exponential (i.e. 0.38) and average cost (i.e. 0.55), as seen in Figure C.8a. This is because the fluctuation in QoS for dao_2 was considerably low. Consequently, the random and reactive approaches did not detect any changes in benefit (Figure C.8b and C.8c).

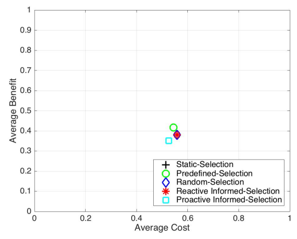
- The proactive approach for varying h values, where $h \in \{1,5,8,15\}$ behaved differently. Particularly, they triggered changes in benefit and replacements to other dao at different timesteps. For instance, at $63^t h$ timestep, the proactive approach using $t+1$ suggested a replacement to dao_5 (Figure C.9b), which resulted in a significant reduction in cost, with slight decrease in benefit (Figure C.8b and C.8c). Further, the proactive approach using $t+15$ recommended three switches to DAO (Figure C.9e), which caused a gradual improvement in benefit with a slight rise cost at some times, and in others it caused a slight decrease in benefit with a significant reduction in cost (Figure C.8k and C.8l). The proactive approach for other $t+h$ values had similar underlying Behaviour (Figure C.8e, C.8f, C.8h, and C.8i). In this context, there is a trade-off between the application's stability and improvement in QoS, which could be adjusted by the architect to determine the suitable h value with respect to the application's requirements.

- The average of forecasts have produced similar results to reactive (i.e. both approaches did not trigger any changes in benefit). This may be due to the use of mean forecasts.

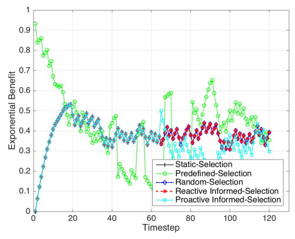
C.2.4 Analysis (Best/Worst Case Scenario for Case 3)

From this experiment, we can investigate that the proactive approach for $t+h$, where $h \in \{1,5,8,15\}$ almost showed same results, except that $t+5$ recommended lower number of replacements (i.e three switches instead of four) and average of forecasts suggested two switches instead of four. This is the second time for the average forecasts to produce different outcome (i.e. Case 2) than $t+h$, where $h \in \{1,5,8,15\}$. Therefore, we present an illustration for the overall Behaviour of proactive approaches (for $t+1$, $t+5$, and average forecasts), their exponential benefit, and their average cost (Figure C.10- C.11). We also plot the DAO replacements over time in Figure C.12 for $t+1$ (as an exemplar for $t+8$ and $t+15$), $t+5$, and average of forecasts. Similar to Case 1, the proactive approach has evaluated the DAO over time and the decision-making has benefited from forecasts. In particular, the proactive approach provided more promising results as compared to reactive and baseline architecture evaluation approaches. Therefore, in this analysis, we will report a summary of the results. For instance, for the best case, the approaches act as follows (Figure C.10- C.12 and Table 6.10):

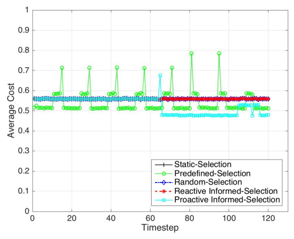
- For the first time, the design-time knowledge has favored the static-selection approach. The latter has produced very high overall exponential benefit with considerable average



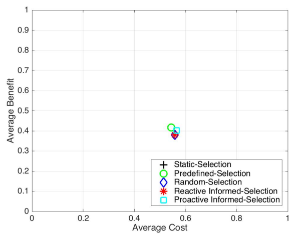
(a) Overall Behaviour ($t+1$, Worst).



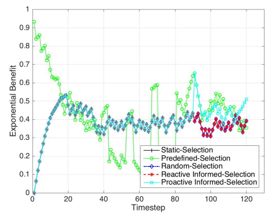
(b) Exponential Benefit ($t+1$, Worst).



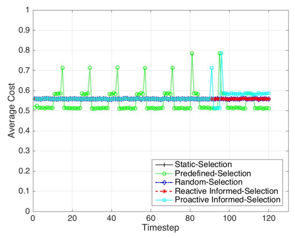
(c) Average Cost ($t+1$, Worst).



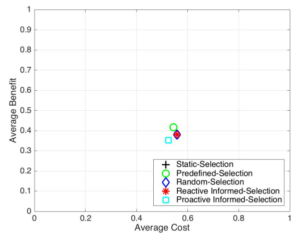
(d) Overall Behaviour ($t+5$, Worst).



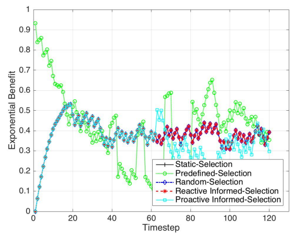
(e) Exponential Benefit ($t+5$, Worst).



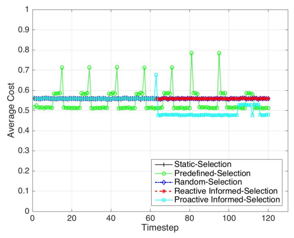
(f) Average Cost ($t+5$, Worst).



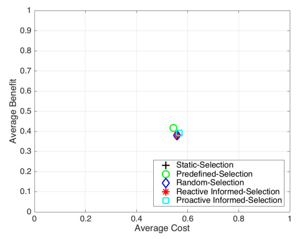
(g) Overall Behaviour ($t+8$, Worst).



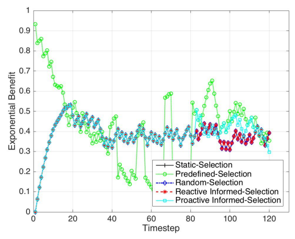
(h) Exponential Benefit ($t+8$, Worst).



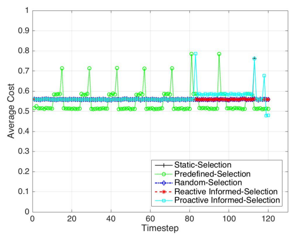
(i) Average Cost ($t+8$, Worst).



(j) Overall Behaviour ($t+15$, Worst).

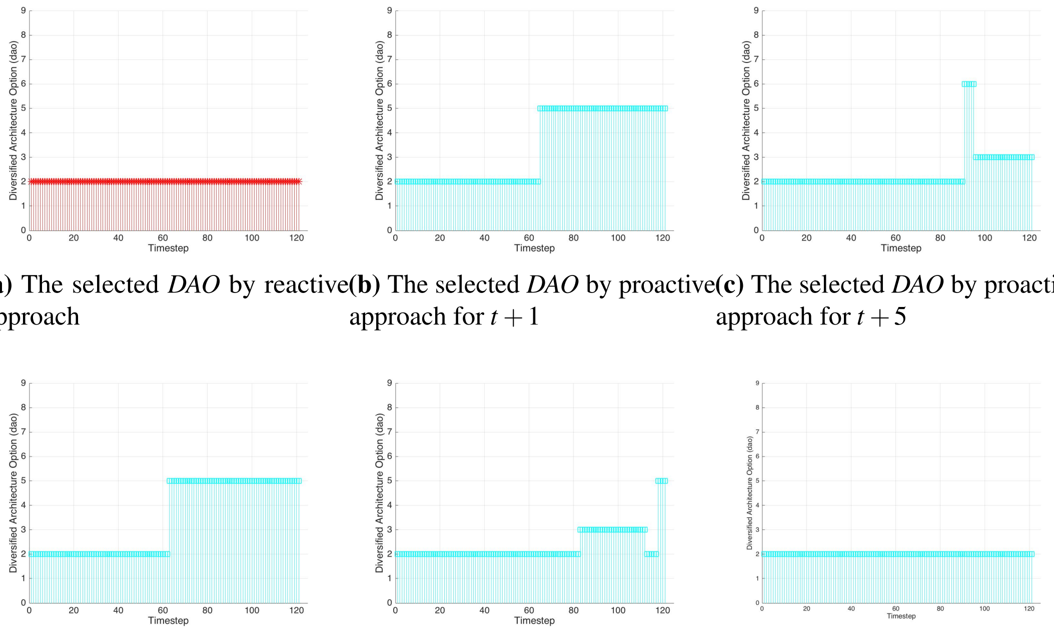


(k) Exponential Benefit ($t+15$, Worst).



(l) Average Cost ($t+15$, Worst).

Figure C.8 The evaluation of the five approaches' decision-making process for **Case 1** ($t+1$, $t+5$, $t+8$, and $t+15$) for the *worst* scenario.



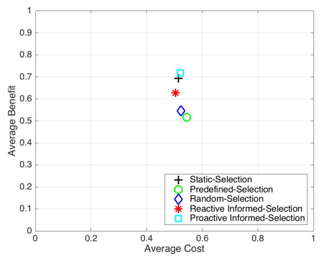
(a) The selected *DAO* by reactive approach **(b)** The selected *DAO* by proactive approach for $t+1$ **(c)** The selected *DAO* by proactive approach for $t+5$

(d) The selected *DAO* by proactive approach for $t+8$ **(e)** The selected *DAO* by proactive approach for $t+15$ **(f)** The selected *DAO* by proactive approach for average forecasts

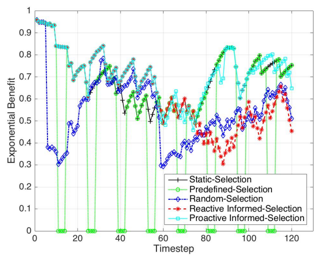
Figure C.9 The selected *DAO* by proactive approach ($t+1$, $t+5$, $t+8$, $t+15$, and average forecasts) for **Case 1** for the *worst* scenario.

cost (i.e. similar to proactive approach, but without replacements), as compared with other approaches (Figure C.10a).

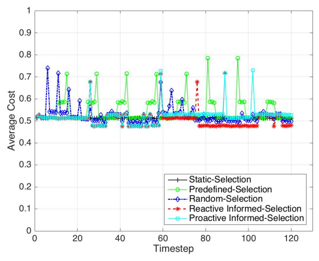
- The predefined-selection and random-selection have reacted similarly to what happened in previous cases (i.e. Normal, Case 1, and Case 2).
- The *DAO* evaluation conducted by the proactive approach for $t+1$, $t+8$, and $t+15$ provided the same choices for *DAO* over time (Figure C.12b).
- The major difference between $t+5$ (Figure C.10d) and other h values (e.g. Figure C.10a) is that the former provided almost the same average exponential benefit and slightly lower average cost, with also lower number of replacements (i.e. three instead of four).
- On the contrary, the average of forecasts produced slightly lower benefit and cost (Table 6.10), but with less number of replacements as well (i.e. two switches) as compared with other $t+h$ (Figure C.11 and C.12d).



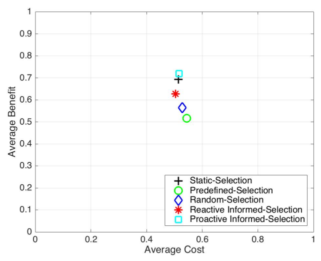
(a) Overall Behaviour ($t+1$, Best).



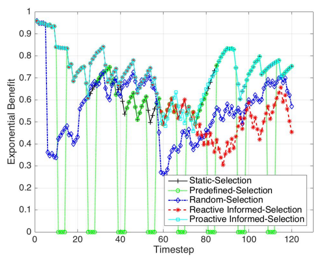
(b) Exponential Benefit ($t+1$, Best).



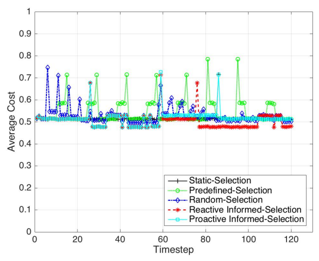
(c) Average Cost ($t+1$, Best).



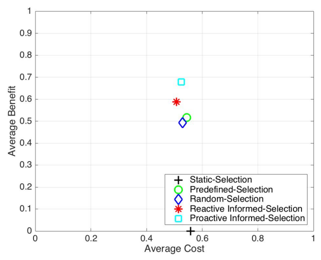
(d) Overall Behaviour ($t+5$, Best).



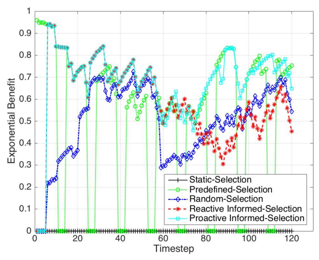
(e) Exponential Benefit ($t+5$, Best).



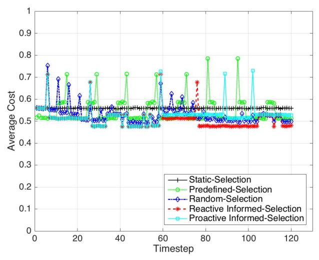
(f) Average Cost ($t+5$, Best).



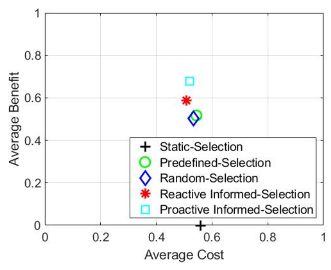
(g) Overall Behaviour ($t+1$, Worst).



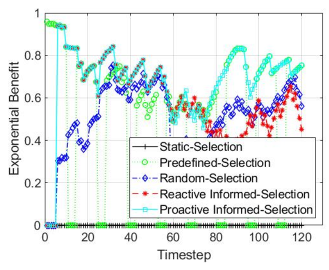
(h) Exponential Benefit ($t+1$, Worst).



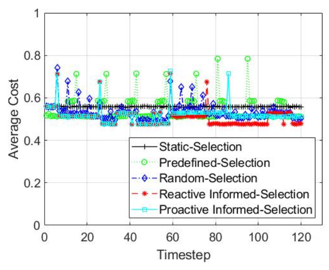
(i) Average Cost ($t+1$, Worst).



(j) Overall Behaviour ($t+5$, Worst).

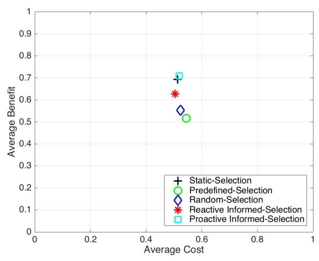


(k) Exponential Benefit ($t+5$, Worst).

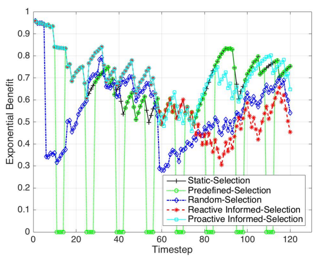


(l) Average Cost ($t+5$, Worst).

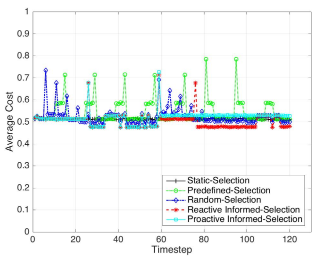
Figure C.10 The evaluation of the five approaches' decision-making process for **Case 3** ($t+1$,$t+5$) for the *best/worst* scenario.



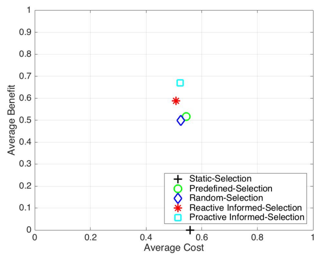
(a) Overall Behaviour (Average Forecasts, Best).



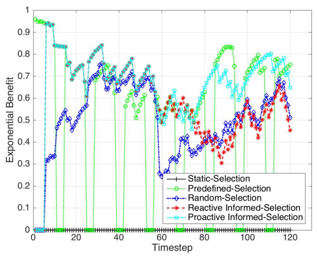
(b) Exponential Benefit (Average Forecasts, Best).



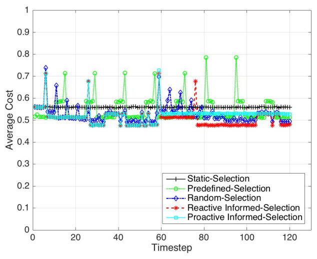
(c) Average Cost (Average Forecasts, Best).



(d) Overall Behaviour (Average Forecasts, Worst).

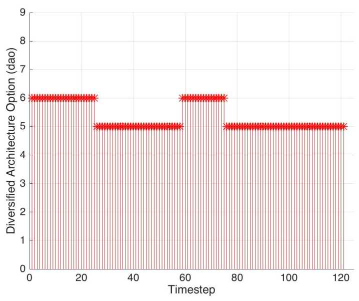


(e) Exponential Benefit (Average Forecasts, Worst).

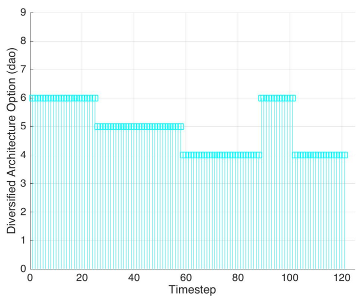


(f) Average Cost (Average Forecasts, Worst).

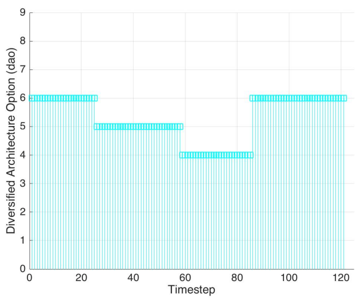
Figure C.11 The evaluation of the five approaches' decision-making process for **Case 3** (Average Forecasts) for the *best/worst* scenario.



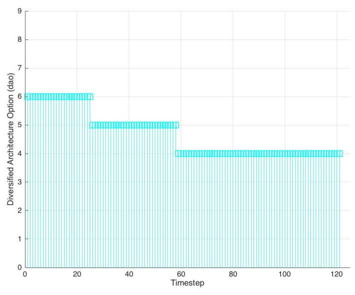
(a) The selected *DAO* by reactive approach



(b) The selected *DAO* by proactive approach for $t+1$



(c) The selected *DAO* by proactive approach for $t+5$



(d) The selected *DAO* by proactive approach for Average forecasts

Figure C.12 The selected *DAO* by reactive/proactive approach for Case 3 ($t+1$, $t+5$, and average forecasts) for the *best* scenario.

References

[1] L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. F. Rana, and M. Parashar, "Mobility-aware application scheduling in fog computing," *IEEE Cloud Computing*, vol. 4, no. 2, pp. 26–35, 2017.

[2] R. Kazman, M. Klein, and P. Clements, "Atam: Method for architecture evaluation," DTIC Document, Tech. Rep., 2000.

[3] J. Asundi, R. Kazman, and M. Klein, "Using economic considerations to choose among architecture design alternatives," DTIC Document, Tech. Rep., 2001.

[4] L. E. Brandão, J. S. Dyer, and W. J. Hahn, "Using binomial decision trees to solve real-option valuation problems," *Decision Analysis*, vol. 2, no. 2, pp. 69–88, 2005.

[5] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Advances in artificial intelligence–SBIA 2004*. Springer, 2004, pp. 286–295.

[6] L. Da Xu, W. He, and S. Li, "Internet of things in industries: A survey," *Industrial Informatics, IEEE Transactions on*, vol. 10, no. 4, pp. 2233–2243, 2014.

[7] K. Nahrstedt, H. Li, P. Nguyen, S. Chang, and L. Vu, "Internet of mobile things: Mobility-driven challenges, designs and implementations," in *Internet-of-Things Design and Implementation (IoTDI), 2016 IEEE First International Conference on*. IEEE, 2016, pp. 25–36.

[8] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[9] R. Kazman, J. Asundi, and M. Klein, "Quantifying the costs and benefits of architectural decisions," in *Proceedings of the 23rd international conference on Software engineering*. IEEE Computer Society, 2001, pp. 297–306.

[10] P. Bengtsson, N. Lassing, J. Bosch, and H. van Vliet, "Architecture-level modifiability analysis (alma)," *Journal of Systems and Software*, vol. 69, no. 1, pp. 129–147, 2004.

[11] V. Issarny, G. Bouloukakis, N. Georgantas, and B. Billet, "Revisiting service-oriented architecture for the iot: a middleware perspective," in *International Conference on Service-Oriented Computing*. Springer, 2016, pp. 3–17.

[12] L. Dobrica and E. Niemela, "A survey on software architecture analysis methods," *IEEE Transactions on software Engineering*, vol. 28, no. 7, pp. 638–653, 2002.

[13] M. M. Bersani, F. Marconi, D. A. Tamburri, P. Jamshidi, and A. Nodari, "Continuous architecting of stream-based systems," in *Software Architecture (WICSA), 2016 13th Working IEEE/IFIP Conference on*. IEEE, 2016, pp. 146–151.

[14] D. Sobhy, R. Bahsoon, L. Minku, and R. Kazman, "Diversifying software architecture for sustainability: A value-based perspective," *Proceedings of 2016 the European Conference on Software Architecture (ECSA)*, 2016.

[15] A. Avizienis and J. P. Kelly, "Fault tolerance by design diversity: Concepts and experiments," *Computer*, no. 8, pp. 67–80, 1984.

[16] B. Baudry, M. Monperrus, C. Mony, F. Chauvel, F. Fleurey, and S. Clarke, "Diversify: Ecology-inspired software evolution for diversity emergence," in *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on*. IEEE, 2014, pp. 395–398.

[17] B. Baudry, S. Allier, and M. Monperrus, "Tailored source code transformations to synthesize computationally diverse program variants," in *Proceedings of the 2014 International Symposium on Software Testing and Analysis*. ACM, 2014, pp. 149–159.

[18] R. Pooley and A. Abdullatif, "Cpasa: continuous performance assessment of software architecture," in *Engineering of Computer Based Systems (ECBS), 2010 17th IEEE International Conference and Workshops on*. IEEE, 2010, pp. 79–87.

[19] L. Bass, I. Weber, and L. Zhu, *DevOps: A Software Architect's Perspective*. Addison-Wesley Professional, 2015.

[20] J. Humble and D. Farley, *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education, 2010.

[21] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli, "Model evolution by run-time parameter adaptation," in *Proceedings of the 31st International Conference on Software Engineering*. IEEE Computer Society, 2009, pp. 111–121.

[22] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *Journal of management information systems*, vol. 24, no. 3, pp. 45–77, 2007.

[23] R. L. Nord, M. R. Barbacci, P. Clements, R. Kazman, and M. Klein, "Integrating the architecture tradeoff analysis method (atam) with the cost benefit analysis method (cbam)," DTIC Document, Tech. Rep., 2003.

[24] L. Trigeorgis, *Real options: Managerial flexibility and strategy in resource allocation*. MIT press, 1996.

[25] M. Amram, N. Kulatilaka *et al.*, "Real options:: Managing strategic investment in an uncertain world," *OUP Catalogue*, 1998.

[26] E. S. Schwartz and L. Trigeorgis, *Real options and investment under uncertainty: classical readings and recent contributions*. MIT press, 2004.

[27] S. Wang, L. L. Minku, and X. Yao, “Resampling-based ensemble methods for online class imbalance learning,” *Knowledge and Data Engineering, IEEE Transactions on*, vol. 27, no. 5, pp. 1356–1368, 2015.

[28] J. Branke, K. Deb, H. Dierolf, and M. Osswald, “Finding knees in multi-objective optimization,” in *International Conference on Parallel Problem Solving from Nature*. Springer, 2004, pp. 722–731.

[29] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: Nsga-ii,” *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[30] G. Ditzler, M. Roveri, C. Alippi, and R. Polikar, “Learning in nonstationary environments: A survey,” *IEEE Computational Intelligence Magazine*, vol. 10, no. 4, pp. 12–25, 2015.

[31] J. Z. Kolter and M. A. Maloof, “Using additive expert ensembles to cope with concept drift,” in *Proceedings of the 22nd international conference on Machine learning*. ACM, 2005, pp. 449–456.

[32] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, “MOA: massive online analysis,” *Journal of Machine Learning Research*, vol. 11, pp. 1601–1604, 2010. [Online]. Available: http://portal.acm.org/citation.cfm?id=1859903

[33] M. J. Hawthorne and D. E. Perry, “Applying design diversity to aspects of system architectures and deployment configurations to enhance system dependability,” in *DSN 2004 Workshop on Architecting Dependable Systems,(DSN-WADS 2004)*, 2004.

[34] N. Narendra and P. Misra. (2016, March) Research challenges in the internet of mobile things. [Online]. Available: https://iot.ieee.org/newsletter/march-2016/research-challenges-in-the-internet-of-mobile-things.html

[35] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, “ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments,” *Software: Practice and Experience*, vol. 47, no. 9, pp. 1275–1296, 2017.

[36] C. Y. Baldwin and K. B. Clark, *Design rules: The power of modularity*. MIT press, 2000, vol. 1.

[37] K. J. Sullivan, P. Chalasani, S. Jha, and V. Sazawal, “Software design as an investment activity: a real options perspective,” *Real options and business strategy: Applications to decision making*, pp. 215–262, 1999.

[38] H. Erdogmus and J. Vandergraaf, “Quantitative approaches for assessing the value of cots-centric development,” 1999.

[39] K. J. Sullivan, W. G. Griswold, Y. Cai, and B. Hallen, “The structure and value of modularity in software design,” in *ACM SIGSOFT Software Engineering Notes*, vol. 26, no. 5. ACM, 2001, pp. 99–108.

[40] H. Erdogmus, "A real options perspective of software reuse," in *International Workshop on Reuse Economics "Redirecting Reuse Economics" Tuesday*, 2002.

[41] R. Bahsoon and W. Emmerich, "Evaluating architectural stability with real options theory," in *Software Maintenance, 2004. Proceedings. 20th IEEE International Conference on*. IEEE, 2004, pp. 443–447.

[42] I. Ozkaya, R. Kazman, and M. Klein, "Quality-attribute based economic valuation of architectural patterns," in *Economics of Software and Computation, 2007. ESC'07. First International Workshop on the*. IEEE, 2007, pp. 5–5.

[43] B. Tansey and E. Stroulia, "Valuating software service development: integrating cocomo ii and real options theory," in *Economics of Software and Computation, 2007. ESC'07. First International Workshop on the*. IEEE, 2007, pp. 8–8.

[44] N. Esfahani, S. Malek, and K. Razavi, "Guidearch: guiding the exploration of architectural solution space under uncertainty," in *Software Engineering (ICSE), 2013 35th International Conference on*. IEEE, 2013, pp. 43–52.

[45] E. Letier, D. Stefan, and E. T. Barr, "Uncertainty, risk, and information value in software requirements and architecture," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 883–894.

[46] R. De Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel *et al.*, "Software engineering for self-adaptive systems: A second research roadmap," in *Software Engineering for Self-Adaptive Systems II*. Springer, 2013, pp. 1–32.

[47] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and experience*, vol. 41, no. 1, pp. 23–50, 2011.

[48] I. Meedeniya, I. Moser, A. Aleti, and L. Grunske, "Architecture-based reliability evaluation under uncertainty," in *Proceedings of the joint ACM SIGSOFT conference–QoSA and ACM SIGSOFT symposium–ISARCS on Quality of software architectures–QoSA and architecting critical systems–ISARCS*. ACM, 2011, pp. 85–94.

[49] C. Becker, R. Chitchyan, L. Duboc, S. Easterbrook, M. Mahaux, B. Penzenstadler, G. Rodriguez-Navas, C. Salinesi, N. Seyff, C. Venters *et al.*, "The karlskrona manifesto for sustainability design," *arXiv preprint arXiv:1410.6968*, 2014.

[50] D. Sobhy, L. Minku, R. Bahsoon, and R. Kazman, "Evaluating software architectures for uncertainty: A systematic literature review," *IEEE Transactions on Software Engineering*, vol. (To be submitted), 2019.

[51] D. Sobhy, L. Minku, R. Bahsoon, T. Chen, and R. Kazman, "Run-time evaluation of architectures: A case study of diversification in iot," *Journal of Systems and Software*, vol. (Second Review Cycle), 2018.

[52] D. Sobhy, L. Minku, R. Bahsoon, and R. Kazman, “Continuous and proactive software architecture evaluation: An iot case,” *ACM Transactions on Software Engineering and Methodology*, vol. (Review Cycle), 2019.

[53] S. E. I. (SEI), “Reduce risk with architecture evaluation,” SEI/CMU, Tech. Rep., February 2018.

[54] H. Sundmaeker, P. Guillemin, P. Friess, and S. Woelfflé, “Vision and challenges for realising the internet of things,” 2010.

[55] B. Roy and T. N. Graham, “Methods for evaluating software architecture: A survey,” *School of Computing TR*, vol. 545, p. 82, 2008.

[56] H. P. Breivold, I. Crnkovic, and M. Larsson, “A systematic review of software architecture evolution research,” *Information and Software Technology*, vol. 54, no. 1, pp. 16–40, 2012.

[57] J. S. Bradbury, J. R. Cordy, J. Dingel, and M. Wermelinger, “A survey of self-management in dynamic software architecture specifications,” in *Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems*. ACM, 2004, pp. 28–33.

[58] C. Krupitzer, F. M. Roth, S. VanSyckel, G. Schiele, and C. Becker, “A survey on engineering approaches for self-adaptive systems,” *Pervasive and Mobile Computing*, vol. 17, pp. 184–206, 2015.

[59] M. Szvetits and U. Zdun, “Systematic literature review of the objectives, techniques, kinds, and architectures of models at runtime,” *Software & Systems Modeling*, vol. 15, no. 1, pp. 31–69, 2016.

[60] S. Mahdavi-Hezavehi, V. H. Durelli, D. Weyns, and P. Avgeriou, “A systematic literature review on methods that handle multiple quality attributes in architecture-based self-adaptive systems,” *Information and Software Technology*, vol. 90, pp. 1–26, 2017.

[61] B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, “Systematic literature reviews in software engineering–a systematic literature review,” *Information and software technology*, vol. 51, no. 1, pp. 7–15, 2009.

[62] B. Kitchenham, R. Pretorius, D. Budgen, O. P. Brereton, M. Turner, M. Niazi, and S. Linkman, “Systematic literature reviews in software engineering–a tertiary study,” *Information and Software Technology*, vol. 52, no. 8, pp. 792–805, 2010.

[63] K. Petersen, S. Vakkalanka, and L. Kuzniarz, “Guidelines for conducting systematic mapping studies in software engineering: An update,” *Information and Software Technology*, vol. 64, pp. 1–18, 2015.

[64] J. Bosch, “Software architecture: The next step,” in *European Workshop on Software Architecture*. Springer, 2004, pp. 194–199.

[65] A. Jansen and J. Bosch, “Software architecture as a set of architectural design decisions,” in *Software Architecture, 2005. WICSA 2005. 5th Working IEEE/IFIP Conference on*. IEEE, 2005, pp. 109–120.

[66] J. S. van der Ven, A. G. Jansen, J. A. Nijhuis, and J. Bosch, "Design decisions: The bridge between rationale and architecture," in *Rationale management in software engineering*. Springer, 2006, pp. 329–348.

[67] H. Cervantes and R. Kazman, *Designing software architectures: a practical approach*. Addison-Wesley Professional, 2016.

[68] D. E. Perry and A. L. Wolf, "Foundations for the study of software architecture," *ACM SIGSOFT Software Engineering Notes*, vol. 17, no. 4, pp. 40–52, 1992.

[69] M. Shaw and D. Garlan, *Software architecture: perspectives on an emerging discipline*. Prentice Hall Englewood Cliffs, 1996, vol. 1.

[70] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*. Addison-Wesley Professional, 2012.

[71] S. E. S. Committee *et al.*, "Ieee standard for a software quality metrics methodology, std. 1061-1998," Technical Report, Tech. Rep., 1998.

[72] (2017, July). [Online]. Available: https://techbeacon.com/5-challenges-performance-engineering-iot-apps?amp

[73] R. Kazman, L. Bass, G. Abowd, and M. Webb, "Saam: A method for analyzing the properties of software architectures," in *Software Engineering, 1994. Proceedings. ICSE-16., 16th International Conference on*. IEEE, 1994, pp. 81–90.

[74] H. Zhang and M. Ali Babar, "On searching relevant studies in software engineering," 2010.

[75] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proceedings of the 18th international conference on evaluation and assessment in software engineering*. ACM, 2014, p. 38.

[76] S. Abrahão and E. Insfran, "Evaluating software architecture evaluation methods: An internal replication," in *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*. ACM, 2017, pp. 144–153.

[77] G. Buchgeher and R. Weinreich, "Continuous software architecture analysis," in *Agile Software Architecture*. Elsevier, 2014, pp. 161–188.

[78] A. Janes, V. Lenarduzzi, and A. C. Stan, "A continuous software quality monitoring approach for small and medium enterprises," in *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*. ACM, 2017, pp. 97–100.

[79] P. Zimmerer, "Strategy for continuous testing in idevops," in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. ACM, 2018, pp. 532–533.

[80] P. Clements, R. Kazman, M. Klein *et al.*, *Evaluating software architectures*. Tsinghua University Press Beijing, 2003.

[81] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli, "Dynamic qos management and optimization in service-based systems," *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 387–409, 2011.

[82] P. Bengtsson and J. Bosch, "Scenario-based software architecture reengineering," in *Software Reuse, 1998. Proceedings. Fifth International Conference on*. IEEE, 1998, pp. 308–317.

[83] G. Tesauro, "Reinforcement learning in autonomic computing: A manifesto and case studies," *IEEE Internet Computing*, vol. 11, no. 1, pp. 22–30, 2007.

[84] S.-W. Cheng, "Rainbow: cost-effective software architecture-based self-adaptation," Ph.D. dissertation, IBM, 2004.

[85] T. Al-Naeem, I. Gorton, M. A. Babar, F. Rabhi, and B. Benatallah, "A quality-driven systematic approach for architecting distributed software applications," in *Proceedings of the 27th international conference on Software engineering*. ACM, 2005, pp. 244–253.

[86] N. Esfahani, E. Kouroshfar, and S. Malek, "Taming uncertainty in self-adaptive software," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. ACM, 2011, pp. 234–244.

[87] L. Zhu, A. Aurum, I. Gorton, and R. Jeffery, "Tradeoff and sensitivity analysis in software architecture evaluation using analytic hierarchy process," *Software Quality Journal*, vol. 13, no. 4, pp. 357–375, 2005.

[88] R. Calinescu and M. Kwiatkowska, "Using quantitative analysis to implement autonomic it systems," in *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*. IEEE, 2009, pp. 100–110.

[89] M. A. Babar, L. Zhu, and R. Jeffery, "A framework for classifying and comparing software architecture evaluation methods," in *Software Engineering Conference, 2004. Proceedings. 2004 Australian*. IEEE, 2004, pp. 309–318.

[90] P. Berander, L.-O. Damm, J. Eriksson, T. Gorschek, K. Henningsson, P. Jönsson, S. Kågström, D. Milicic, F. Mårtensson, K. Rönkkö *et al.*, "Software quality attributes and trade-offs," *Blekinge Institute of Technology*, 2005.

[91] M. Rohr, S. Giesecke, M. Hiel, W.-J. van den Heuvel, H. Weigand, and W. Hasselbring, "A classification scheme for self-adaptation research," 2006.

[92] M. Handte, G. Schiele, V. Matjuntke, C. Becker, and P. J. Marrón, "3pc: System support for adaptive peer-to-peer pervasive computing," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 7, no. 1, p. 10, 2012.

[93] H. Koziolek, "Sustainability evaluation of software architectures: a systematic review," in *Proceedings of the joint ACM SIGSOFT conference–QoSA and ACM SIGSOFT symposium–ISARCS on Quality of software architectures–QoSA and architecting critical systems–ISARCS*. ACM, 2011, pp. 3–12.

[94] L. G. Jones and A. J. Lattanze, "Using the architecture tradeoff analysis method to evaluate a wargame simulation system: A case study," DTIC Document, Tech. Rep., 2001.

[95] M. T. Ionita, P. America, D. K. Hammer, H. Obbink, and J. J. Trienekens, "A scenario-driven approach for value, risk, and cost analysis in system architecting for innovation," in *Software Architecture, 2004. WICSA 2004. Proceedings. Fourth Working IEEE/IFIP Conference on*. IEEE, 2004, pp. 277–280.

[96] R. Kazman, L. Bass, and M. Klein, "The essential components of software architecture design and analysis," *Journal of Systems and Software*, vol. 79, no. 8, pp. 1207–1216, 2006.

[97] F. Faniyi, R. Bahsoon, A. Evans, and R. Kazman, "Evaluating security properties of architectures in unpredictable environments: A case for cloud," in *2011 Ninth Working IEEE/IFIP Conference on Software Architecture*. IEEE, 2011, pp. 127–136.

[98] S. Al-Azzani and R. Bahsoon, "Using implied scenarios in security testing," in *Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems*. ACM, 2010, pp. 15–21.

[99] T. L. Saaty, "Decision making with the analytic hierarchy process," *International journal of services sciences*, vol. 1, no. 1, pp. 83–98, 2008.

[100] J. Lee, S. Kang, and C.-K. Kim, "Software architecture evaluation methods based on cost benefit analysis and quantitative decision making," *Empirical Software Engineering*, vol. 14, no. 4, pp. 453–475, 2009.

[101] C.-K. Kim, D.-H. Lee, I.-Y. Ko, and J. Baik, "A lightweight value-based software architecture evaluation," in *Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2007. SNPD 2007. Eighth ACIS International Conference on*, vol. 2. IEEE, 2007, pp. 646–649.

[102] M. Osterlind, P. Johnson, K. Karnati, R. Lagerstrom, and M. Valja, "Enterprise architecture evaluation using utility theory," in *2013 17th IEEE International Enterprise Distributed Object Computing Conference Workshops*. IEEE, 2013, pp. 347–351.

[103] L. Grunske, "Identifying good architectural design alternatives with multi-objective optimization strategies," in *Proceedings of the 28th international conference on Software engineering*. ACM, 2006, pp. 849–852.

[104] Y. Liu, M. A. Babar, and I. Gorton, "Middleware architecture evaluation for dependable self-managing systems," in *International Conference on the Quality of Software Architectures*. Springer, 2008, pp. 189–204.

[105] H. B. Christensen, K. M. Hansen, and B. Lindstrøm, "Lightweight and continuous architectural software quality assurance using the asqa technique," in *European Conference on Software Architecture*. Springer, 2010, pp. 118–132.

[106] V.-P. Eloranta, U. van Heesch, P. Avgeriou, N. Harrison, and K. Koskimies, "Lightweight evaluation of software architecture decisions," in *Relating System Quality and Software Architecture*. Elsevier, 2015, pp. 157–179.

[107] L. G. Williams and C. U. Smith, “Pasa sm: a method for the performance assessment of software architectures,” in *Proceedings of the 3rd international workshop on Software and performance*. ACM, 2002, pp. 179–189.

[108] S. R. Cellini and J. E. Kee, “Cost-effectiveness and cost-benefit analysis,” *Handbook of practical program evaluation*, vol. 4, 2015.

[109] B. W. Boehm *et al.*, *Software engineering economics*. Prentice-hall Englewood Cliffs (NJ), 1981, vol. 197.

[110] B. W. Boehm and K. J. Sullivan, “Software economics: a roadmap,” in *Proceedings of the conference on The future of Software engineering*. ACM, 2000, pp. 319–343.

[111] T. A. Luehrman, “Strategy as a portfolio of real options,” *Harvard business review*, vol. 76, pp. 89–101, 1998.

[112] S. E. Elmaghraby, W. S. Herroelen *et al.*, “The scheduling of activities to maximize the net present value of projects.” *European Journal of Operational Research*, vol. 49, no. 1, pp. 35–49, 1990.

[113] H. Markowitz, “Portfolio selection, cowles foundation monograph no. 16,” *John Wiley, New York. S. Moss (1981). An Economic theory of Business Strategy, Halstead Press, New York. TH Naylor (1966). The theory of the firm: a comparison of marginal analysis and linear programming. Southern Economic Journal (January)*, vol. 32, pp. 263–74, 1959.

[114] J. Favaro, “A comparison of approaches to reuse investment analysis,” in *Software Reuse, 1996., Proceedings Fourth International Conference on*. IEEE, 1996, pp. 136–145.

[115] M. Denne and J. Cleland-Huang, “The incremental funding method: Data-driven software development,” *IEEE software*, vol. 21, no. 3, pp. 39–47, 2004.

[116] B. Ojameruaye, R. Bahsoon, and L. Duboc, “Sustainability debt: a portfolio-based approach for evaluating sustainability requirements in architectures,” in *Proceedings of the 38th International Conference on Software Engineering Companion*. ACM, 2016, pp. 543–552.

[117] R. Bahsoon and W. Emmerich, “An economics-driven approach for valuing scalability in distributed architectures,” in *Software Architecture, 2008. WICSA 2008. Seventh Working IEEE/IFIP Conference on*. IEEE, 2008, pp. 9–18.

[118] S.-W. Cheng, *Rainbow: cost-effective software architecture-based self-adaptation*. ProQuest, 2008.

[119] J. Yang, G. Huang, W. Zhu, X. Cui, and H. Mei, “Quality attribute tradeoff through adaptive architectures at runtime,” *Journal of Systems and Software*, vol. 82, no. 2, pp. 319–332, 2009.

[120] W. Heaven, D. Sykes, J. Magee, and J. Kramer, “A case study in goal-driven architectural adaptation,” in *Software Engineering for Self-Adaptive Systems*. Springer, 2009, pp. 109–127.

[121] D. Cooray, E. Kouroshfar, S. Malek, and R. Roshandel, "Proactive self-adaptation for improving the reliability of mission-critical, embedded, and mobile software," *IEEE Transactions on Software Engineering*, vol. 39, no. 12, pp. 1714–1735, 2013.

[122] N. Esfahani, A. Elkhodary, and S. Malek, "A learning-based framework for engineering feature-oriented self-adaptive software systems," *IEEE transactions on software engineering*, vol. 39, no. 11, pp. 1467–1493, 2013.

[123] C. Ghezzi and A. M. Sharifloo, "Dealing with non-functional requirements for adaptive systems via dynamic software product-lines," in *Software Engineering for Self-Adaptive Systems II*. Springer, 2013, pp. 191–213.

[124] E. M. Fredericks, B. DeVries, and B. H. Cheng, "Towards run-time adaptation of test cases for self-adaptive systems in the face of uncertainty," in *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. ACM, 2014, pp. 17–26.

[125] T. Chen, K. Li, R. Bahsoon, and X. Yao, "Femosaa: Feature-guided and knee-driven multi-objective optimization for self-adaptive software," *ACM Trans. Softw. Eng. Methodol.*, vol. 27, no. 2, pp. 5:1–5:50, Jun. 2018. [Online]. Available: http://doi.acm.org/10.1145/3204459

[126] M. J. Van Der Donckt, D. Weyns, M. U. Iftikhar, and R. K. Singh, "Cost-benefit analysis at runtime for self-adaptive systems applied to an internet of things application." in *Proceedings of ENASE 2018, Portugal*, 2018.

[127] G. Blair, N. Bencomo, and R. B. France, "Models@ run. time," *Computer*, vol. 42, no. 10, 2009.

[128] B. H. Cheng, K. I. Eder, M. Gogolla, L. Grunske, M. Litoiu, H. A. Müller, P. Pelliccione, A. Perini, N. A. Qureshi, B. Rumpe *et al.*, "Using models at runtime to address assurance for self-adaptive systems," in *Models@ run. time*. Springer, 2014, pp. 101–136.

[129] R. Calinescu, "Emerging techniques for the engineering of self-adaptive high-integrity software," in *Assurances for Self-Adaptive Systems*. Springer, 2013, pp. 297–310.

[130] D. El Kateb, F. Fouquet, G. Nain, J. A. Meira, M. Ackerman, and Y. Le Traon, "Generic cloud platform multi-objective optimization leveraging models@ run.time," in *Proceedings of the 29th Annual ACM Symposium on Applied Computing*. ACM, 2014, pp. 343–350.

[131] D. Garlan and B. Schmerl, "Using architectural models at runtime: Research challenges," in *European Workshop on Software Architecture*. Springer, 2004, pp. 200–205.

[132] J. Kramer and J. Magee, "Self-managed systems: an architectural challenge," in *2007 Future of Software Engineering*. IEEE Computer Society, 2007, pp. 259–268.

[133] F. Chauvel, N. Ferry, B. Morin, A. Rossini, and A. Solberg, "Models@ runtime to support the iterative and continuous design of autonomic reasoners." in *MoDELS@ Run. time*, 2013, pp. 26–38.

[134] B. Morin, O. Barais, J.-M. Jezequel, F. Fleurey, and A. Solberg, "Models@ run. time to support dynamic adaptation," *Computer*, vol. 42, no. 10, 2009.

[135] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya, "Software architecture optimization methods: A systematic literature review," *IEEE Transactions on Software Engineering*, vol. 39, no. 5, pp. 658–683, 2013.

[136] S.-W. Cheng, "Rainbow: Cost-effective software architecture-based self-adaptation," Ph.D. dissertation, Pittsburgh, PA, USA, 2008, aAI3305807.

[137] D. Kim and S. Park, "Reinforcement learning-based dynamic adaptation planning method for architecture-based self-managed software," in *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*. IEEE, 2009, pp. 76–85.

[138] R. Calinescu, K. Johnson, and Y. Rafiq, "Using observation ageing to improve markovian model learning in qos engineering," in *Proceedings of the 2nd ACM/SPEC International Conference on Performance engineering*. ACM, 2011, pp. 505–510.

[139] G. A. Moreno, J. Cámara, D. Garlan, and B. Schmerl, "Efficient decision-making under uncertainty for proactive self-adaptation," in *Autonomic Computing (ICAC), 2016 IEEE International Conference on*. IEEE, 2016, pp. 147–156.

[140] S. Mahdavi-Hezavehi, M. Galster, and P. Avgeriou, "Variability in quality attributes of service-based software systems: A systematic literature review," *Information and Software Technology*, vol. 55, no. 2, pp. 320–343, 2013.

[141] D. E. Seborg, D. A. Mellichamp, T. F. Edgar, and F. J. Doyle III, *Process dynamics and control*. John Wiley & Sons, 2010.

[142] A. Van Lamsweerde and E. Letier, "From object orientation to goal orientation: A paradigm shift for requirements engineering," in *Radical Innovations of Software and Systems Engineering in the Future*. Springer, 2004, pp. 325–340.

[143] T. C. Lethbridge and R. Laganiere, *Object-oriented software engineering*. McGraw-Hill New York, 2005.

[144] A. Van Lamsweerde, "Goal-oriented requirements engineering: A guided tour," in *Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on*. IEEE, 2001, pp. 249–262.

[145] A. Immonen and E. Niemelä, "Survey of reliability and availability prediction methods from the viewpoint of software architecture," *Software & Systems Modeling*, vol. 7, no. 1, p. 49, 2008.

[146] D. Falessi, G. Cantone, R. Kazman, and P. Kruchten, "Decision-making techniques for software architecture design: A comparative survey," *ACM Computing Surveys (CSUR)*, vol. 43, no. 4, p. 33, 2011.

[147] A. Tang, M. A. Babar, I. Gorton, and J. Han, "A survey of architecture design rationale," *Journal of systems and software*, vol. 79, no. 12, pp. 1792–1804, 2006.

[148] M. A. Babar and I. Gorton, "Comparison of scenario-based software architecture evaluation methods," in *Software Engineering Conference, 2004. 11th Asia-Pacific*. IEEE, 2004, pp. 600–607.

[149] S. T. Redwine Jr and W. E. Riddle, "Software technology maturation," in *Proceedings of the 8th international conference on Software engineering*. IEEE Computer Society Press, 1985, pp. 189–200.

[150] Y. Deswarte, K. Kanoun, and J.-C. Laprie, "Diversity against accidental and deliberate faults," in *csda*. IEEE, 1998, p. 171.

[151] H. Song, A. Elgammal, V. Nallur, F. Chauvel, F. Fleurey, and S. Clarke, "On architectural diversity of dynamic adaptive systems," *The 37th International Conference on Software Engineering, NIER track*, 2015.

[152] F. Chauvel, H. Song, and F. Fleurey, "Diversity: A heuristic to improve robustness of self-adaptive cloud architectures," in *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*. IEEE, 2015, pp. 132–141.

[153] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah, and P. Merle, "Elasticity in cloud computing: state of the art and research challenges," *IEEE Transactions on Services Computing*, vol. 11, no. 2, pp. 430–447, 2018.

[154] L. Chen, "Microservices: architecting for continuous delivery and devops," in *2018 IEEE International Conference on Software Architecture (ICSA)*. IEEE, 2018, pp. 39–397.

[155] D. Zhao and Z. Hu, "Supervised adaptive dynamic programming based adaptive cruise control," in *2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*. IEEE, 2011, pp. 318–323.

[156] A. Bennaceur, V. Issarny, D. Sykes, F. Howar, M. Isberner, B. Steffen, R. Johansson, and A. Moschitti, "Machine learning for emergent middleware," in *International Workshop on Eternal Systems*. Springer, 2012, pp. 16–29.

[157] K. Ashton, "That 'internet of things' thing," *RFiD Journal*, vol. 22, no. 7, pp. 97–114, 2009.

[158] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, "Internet of things: Vision, applications and research challenges," *Ad Hoc Networks*, vol. 10, no. 7, pp. 1497–1516, 2012.

[159] E. Borgia, "The internet of things vision: Key features, applications and open issues," *Computer Communications*, vol. 54, pp. 1–31, 2014.

[160] N. PINEDA, "5 industry problems that can be solved by iot," February 2018.

[161] B. Littlewood, P. Popov, and L. Strigini, "Modeling software design diversity: a review," *ACM Computing Surveys (CSUR)*, vol. 33, no. 2, pp. 177–208, 2001.

[162] I. Schaefer, R. Rabiser, D. Clarke, L. Bettini, D. Benavides, G. Botterweck, A. Pathak, S. Trujillo, and K. Villela, "Software diversity: state of the art and perspectives," 2012.

[163] A. Avizienis and J. P. J. Kelly, "Fault tolerance by design diversity: Concepts and experiments," *Computer*, vol. 17, no. 8, pp. 67–80, Aug. 1984. [Online]. Available: http://dx.doi.org/10.1109/MC.1984.1659219

[164] A. Avizienis, "The n-version approach to fault-tolerant software," *Software Engineering, IEEE Transactions on*, vol. SE-11, no. 12, pp. 1491–1501, Dec 1985.

[165] M. R. Lyu, *Software fault tolerance*. John Wiley & Sons, Inc., 1995.

[166] A. Avizienis and J.-C. Laprie, "Dependable computing: From concepts to design diversity," *Proceedings of the IEEE*, vol. 74, no. 5, pp. 629–638, 1986.

[167] A. Avižienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *Dependable and Secure Computing, IEEE Transactions on*, vol. 1, no. 1, pp. 11–33, 2004.

[168] M. A. Vouk, D. F. McAllister, D. E. Eckhardt, and K. Kim, "An empirical evaluation of consensus voting and consensus recovery block reliability in the presence of failure correlation," *Journal of Computer and Software Engineering*, vol. 1, no. 4, pp. 367–388, 1993.

[169] L. Chen and A. Avizienis, "N-version programming: A fault-tolerance approach to reliability of software operation," in *Digest of Papers FTCS-8: Eighth Annual International Conference on Fault Tolerant Computing*, 1978, pp. 3–9.

[170] A. J. O'Donnell and H. Sethu, "On achieving software diversity for improved network security using distributed coloring algorithms," in *Proceedings of the 11th ACM conference on Computer and communications security*. ACM, 2004, pp. 121–131.

[171] M. Franz, "E unibus pluram: massive-scale software diversity as a defense mechanism," in *Proceedings of the 2010 workshop on New security paradigms*. ACM, 2010, pp. 7–16.

[172] F. Fleurey, B. Baudry, B. Gauzens, A. Elie, and K. Yeboah-Antwi, "Emergent robustness in software systems through decentralized adaptation: an ecologically-inspired alife approach," in *European Conference on Artificial Life 2015*, 2015.

[173] R. Mahmud, R. Kotagiri, and R. Buyya, "Fog computing: A taxonomy, survey and future directions," in *Internet of Everything*. Springer, 2018, pp. 103–130.

[174] R. Betts, "Architecting for the internet of things," O'reilly, Tech. Rep., June 2016.

[175] F. Jalali, K. Hinton, R. Ayre, T. Alpcan, and R. S. Tucker, "Fog computing may help to save energy in cloud computing," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 5, pp. 1728–1739, 2016.

[176] J. C. Hull, *Options, futures, and other derivatives*. Pearson Education India, 2006.

[177] A. Elkhodary, N. Esfahani, and S. Malek, "Fusion: a framework for engineering self-tuning self-adaptive software systems," in *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*. ACM, 2010, pp. 7–16.

[178] Z.-K. Zhang, M. C. Y. Cho, C.-W. Wang, C.-W. Hsu, C.-K. Chen, and S. Shieh, "Iot security: Ongoing challenges and research opportunities," in *Service-Oriented Computing and Applications (SOCA), 2014 IEEE 7th International Conference on*. IEEE, 2014, pp. 230–234.

[179] J.-C. Laprie, J. Arlat, C. BeOunes, and K. Kanoun, "Architectural issues in software fault tolerance," *Software fault tolerance*, pp. 47–80, 1995.

[180] R. Kazman and L. Bass, "Categorizing business goals for software architectures," DTIC Document, Tech. Rep., 2005.

[181] E. H. Bowman and G. T. Moskowitz, "Real options analysis and strategic decision making," *Organization Science*, vol. 12, no. 6, pp. 772–777, 2001.

[182] F. Black and M. Scholes, "The pricing of options and corporate liabilities," *The journal of political economy*, pp. 637–654, 1973.

[183] J. C. Cox, S. A. Ross, and M. Rubinstein, "Option pricing: A simplified approach," *Journal of financial Economics*, vol. 7, no. 3, pp. 229–263, 1979.

[184] M. Wasilewska, "Comparison between portfolios of real options and portfolios of financial options," *Prace Naukowe Uniwersytetu Ekonomicznego we Wrocławiu*, no. 290 Performance Measurement and Management, pp. 128–135, 2013.

[185] B. Nuseibeh, J. Kramer, and A. Finkelsteiin, "Viewpoints: meaningful relationships are difficult!" in *Proceedings of the 25th International Conference on Software Engineering*. IEEE Computer Society, 2003, pp. 676–681.

[186] G. Fischer, "Communities of interest: Learning through the interaction of multiple knowledge systems," in *Proceedings of the 24th IRIS Conference*, vol. 1. Department of Information Science, Bergen, 2001, pp. 1–13.

[187] N. Rozanski and E. Woods, *Software systems architecture: working with stakeholders using viewpoints and perspectives*. Addison-Wesley, 2011.

[188] M. Jorgensen and M. Shepperd, "A systematic review of software development cost estimation studies," *IEEE Transactions on software engineering*, vol. 33, no. 1, pp. 33–53, 2007.

[189] C. F. Lee and A. C. Lee, *Encyclopedia of finance*. Springer, 2006.

[190] R. Geske, "The valuation of compound options," *Journal of financial economics*, vol. 7, no. 1, pp. 63–81, 1979.

[191] H. S. Herath and C. S. Park, "Multi-stage capital investment opportunities as compound real options," *The Engineering Economist*, vol. 47, no. 1, pp. 1–27, 2002.

[192] H. Erdogmus, "Management of license cost uncertainty in software development: a real options approach," in *Proc. 5th Annual Conference on Real Options: Theory Meets Practice, UCLA, Los Angeles, CA*, 2001.

[193] "Fog computing and the internet of things: Extend the cloud to where the things are," Cisco Systems, Tech. Rep., 2016.

[194] "The internet of things consortium," 2018.

[195] A. aws, "Cloud watch," October 2018.

[196] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* MIT press Cambridge, 1998, vol. 1, no. 1.

[197] C. C. Holt, "Forecasting seasonals and trends by exponentially weighted moving averages," *International journal of forecasting*, vol. 20, no. 1, pp. 5–10, 2004.

[198] L. I. Kuncheva, "Classifier ensembles for changing environments," in *International Workshop on Multiple Classifier Systems.* Springer, 2004, pp. 1–15.

[199] J. rey Horn, N. Nafpliotis, and D. E. Goldberg, "A niched pareto genetic algorithm for multiobjective optimization," in *Proceedings of the first IEEE conference on evolutionary computation, IEEE world congress on computational intelligence*, vol. 1. Citeseer, 1994, pp. 82–87.

[200] K. Deb and S. Gupta, "Understanding knee points in bicriteria problems and their implications as preferred solution principles," *Engineering optimization*, vol. 43, no. 11, pp. 1175–1204, 2011.

[201] S. Opel, "Aws streamlines amazon ec2 spot instance pricing model and operational complexity," January 2018.

[202] L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "Qos-aware middleware for web services composition," *IEEE Transactions on software engineering*, vol. 30, no. 5, pp. 311–327, 2004.

[203] A. Ramírez, J. A. Parejo, J. R. Romero, S. Segura, and A. Ruiz-Cortés, "Evolutionary composition of qos-aware web services: a many-objective perspective," *Expert Systems with Applications*, vol. 72, pp. 357–370, 2017.

[204] J. Gorbold. (April 2010) Power consumption, amd phenom ii x6 1090t be. [Online]. Available: https://www.bit-tech.net/reviews/tech/cpus/amd-phenom-ii-x6-1090t-black-edition/7/

[205] T. Guérout, T. Monteil, G. Da Costa, R. N. Calheiros, R. Buyya, and M. Alexandru, "Energy-aware simulation with dvfs," *Simulation Modelling Practice and Theory*, vol. 39, pp. 76–91, 2013.

[206] C. Reporter. (2016, March) Research challenges in the internet of mobile things. [Online]. Available: http://www.circleid.com/posts/20160811_internet_data_growth_iot_leading_to_unlimited_energy_consumption

[207] R. Kazman, S. Haziyev, A. Yakuba, and D. A. Tamburri, "Managing energy consumption as an architectural quality attribute," *IEEE Software*, vol. 35, no. 5, pp. 102–107, 2018.

[208] Y. Chen and T. Kunz, "Performance evaluation of iot protocols under a constrained wireless access network," in *Selected Topics in Mobile and Wireless Networking (MoWNeT), 2016 International Conference on.* IEEE, 2016, pp. 1–7.

[209] aws, “Deploy an end-to-end iot application,” https://aws.amazon.com/getting-started/projects/deploy-iot-application/services-costs/, March 2018.

[210] R. N. Taylor, N. Medvidovic, and E. M. Dashofy, *Software architecture: foundations, theory, and practice.* Wiley Publishing, 2009.

[211] S. S. Gokhale, “Architecture-based software reliability analysis: Overview and limitations,” *IEEE Transactions on dependable and secure computing*, vol. 4, no. 1, 2007.

[212] F. Wagner, F. Ishikawa, and S. Honiden, “Robust service compositions with functional and location diversity,” *IEEE Transactions on Services Computing*, vol. 9, no. 2, pp. 277–290, 2016.

[213] A. Arcuri and L. Briand, “A practical guide for using statistical tests to assess randomized algorithms in software engineering,” in *Software Engineering (ICSE), 2011 33rd International Conference on.* IEEE, 2011, pp. 1–10.

[214] X. Zhu and X. Wu, “Class noise vs. attribute noise: A quantitative study,” *Artificial intelligence review*, vol. 22, no. 3, pp. 177–210, 2004.

[215] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, “Energy consumption in mobile phones: a measurement study and implications for network applications,” in *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement.* ACM, 2009, pp. 280–293.

[216] J. Dejun, G. Pierre, and C.-H. Chi, “Ec2 performance analysis for resource provisioning of service-oriented applications,” in *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops.* Springer, 2010, pp. 197–207.

[217] S. J. Pan, Q. Yang *et al.*, “A survey on transfer learning,” *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[218] P. Jamshidi, M. Velez, C. Kästner, N. Siegmund, and P. Kawthekar, “Transfer learning for improving model predictions in highly configurable software,” in *Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems.* IEEE Press, 2017, pp. 31–41.

[219] G. H. Oliveira, R. C. Cavalcante, G. G. Cabral, L. L. Minku, and A. L. Oliveira, “Time series forecasting in the presence of concept drift: A pso-based approach,” in *Tools with Artificial Intelligence (ICTAI), 2017 IEEE 29th International Conference on.* IEEE, 2017, pp. 239–246.

[220] M.-L. Zhang and Z.-H. Zhou, “Ml-knn: A lazy learning approach to multi-label learning,” *Pattern recognition*, vol. 40, no. 7, pp. 2038–2048, 2007.

[221] K. Gurney, *An introduction to neural networks.* CRC press, 2014.

[222] L. Bottou and O. Bousquet, “The tradeoffs of large scale learning,” in *Advances in neural information processing systems*, 2008, pp. 161–168.

[223] J. Langford, L. Li, and T. Zhang, “Sparse online learning via truncated gradient,” *Journal of Machine Learning Research*, vol. 10, no. Mar, pp. 777–801, 2009.

[224] S. Shalev-Shwartz and A. Tewari, “Stochastic methods for l1-regularized loss minimization,” *Journal of Machine Learning Research*, vol. 12, no. Jun, pp. 1865–1892, 2011.

[225] H. Robbins and S. Monro, “A stochastic approximation method,” in *Herbert Robbins Selected Papers*. Springer, 1985, pp. 102–109.

[226] Z. Wang, K. Crammer, and S. Vucetic, “Breaking the curse of kernelization: Budgeted stochastic gradient descent for large-scale svm training,” *Journal of Machine Learning Research*, vol. 13, no. Oct, pp. 3103–3131, 2012.

[227] L. Massaron and A. Boschetti, *Regression Analysis with Python*. Packt Publishing Ltd, 2016.

[228] E. Ikonomovska, J. Gama, and S. Džeroski, “Learning model trees from evolving data streams,” *Data mining and knowledge discovery*, vol. 23, no. 1, pp. 128–168, 2011.

[229] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, “A survey on concept drift adaptation,” *ACM computing surveys (CSUR)*, vol. 46, no. 4, p. 44, 2014.

[230] J. Demšar, “Statistical comparisons of classifiers over multiple data sets,” *Journal of Machine learning research*, vol. 7, no. Jan, pp. 1–30, 2006.

[231] “Application performance monitoring and management | appdynamics,” February 2018.

[232] “New relic,” 2018.

[233] R. Snijders, F. Dalpiaz, M. Hosseini, A. Shahri, and R. Ali, “Crowd-centric requirements engineering,” in *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*. IEEE, 2014, pp. 614–615.

[234] M. Hosseini, A. Shahri, K. Phalp, J. Taylor, R. Ali, and F. Dalpiaz, “Configuring crowdsourcing for requirements elicitation,” in *2015 IEEE 9th International Conference on Research Challenges in Information Science (RCIS)*. IEEE, 2015, pp. 133–138.

[235] M. Almaliki, C. Ncube, and R. Ali, “The design of adaptive acquisition of users feedback: An empirical study,” in *2014 IEEE Eighth International Conference on Research Challenges in Information Science (RCIS)*. IEEE, 2014, pp. 1–12.

[236] ——, “Adaptive software-based feedback acquisition: A persona-based design,” in *2015 IEEE 9th International Conference on Research Challenges in Information Science (RCIS)*. IEEE, 2015, pp. 100–111.

[237] L. M. Hilty, P. Arnfalk, L. Erdmann, J. Goodman, M. Lehmann, and P. A. Wäger, “The relevance of information and communication technologies for environmental sustainability–a prospective simulation study,” *Environmental Modelling & Software*, vol. 21, no. 11, pp. 1618–1629, 2006.

[238] P. Avgeriou, M. Stal, and R. Hilliard, “Architecture sustainability [guest editors’ introduction],” *Software, IEEE*, vol. 30, no. 6, pp. 40–44, 2013.

[239] H. Koziolek, D. Domis, T. Goldschmidt, and P. Vorst, “Measuring architecture sustainability,” *Software, IEEE*, vol. 30, no. 6, pp. 54–62, 2013.

[240] J. Savolainen, N. Niu, T. Mikkonen, and T. Fogdal, “Long-term product line sustainability with planned staged investments,” *Software, IEEE*, vol. 30, no. 6, pp. 63–69, 2013.

[241] C. Becker, R. Chitchyan, L. Duboc, S. Easterbrook, B. Penzenstadler, N. Seyff, and C. C. Venters, “Sustainability design and software: the karlskrona manifesto,” in *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, vol. 2. IEEE, 2015, pp. 467–476.

[242] B. Penzenstadler, V. Bauer, C. Calero, and X. Franch, “Sustainability in software engineering: A systematic literature review,” in *Evaluation & Assessment in Software Engineering (EASE 2012), 16th International Conference on.* IET, 2012, pp. 32–41.

[243] P. Lago, S. A. Koçak, I. Crnkovic, and B. Penzenstadler, “Framing sustainability as a property of software quality,” *Communications of the ACM*, vol. 58, no. 10, pp. 70–78, 2015.