A Qualitative Approach to Effort Judgment for Web Service Composition based SOA Implementations

By

Zheng Li
Abstract

Before developing any Service-Oriented Architecture (SOA) based software, effort estimation will be inevitably required for many purposes ranging from budget analysis to cost-benefit balance. Therefore, effort estimation plays a vital role in SOA implementations. Nevertheless, effort estimation for SOA implementation confronts many challenges due to both the complexity of SOA and the diversity of SOA projects. The current approaches based on traditional techniques are normally not adequate for estimating the effort of SOA implementations. Meanwhile, there is a lack of work on effort estimation for SOA implementation within the published literature.

To decrease the complexity and difficulty of the problems of effort estimation before building SOA systems, this book concentrates on one particular type of SOA projects: service composition based SOA implementations. As enterprises move to having more and more services, and business applications increasingly rely on subscribing services, the major problem in SOA implementations will be service composition and may be less on development of new services. Moreover, it has been identified that the benefits of SOA cannot be fully realized until reaching the level of service composition. Therefore, it is worthwhile to narrow down our concerns from generic SOA implementations to service composition based SOA implementations that do not take into account service migration or new service development. Considering Web service is the de facto representation of service in practice, we can further focus on Web service composition (WSC) based SOA implementations.

Considering the limitations to empirically implementing SOA projects by using various WSC approaches, “learning by doing” is nearly an impossible way to collect development data for quantitatively unfolding the research into effort estimation for WSC-based SOA implementations. Fortunately, “learning by analogy” is a feasible alternative to investigate distributed systems like SOA systems. After comprehending generic SOA implementations from an organizational perspective and viewing WSC-based SOA systems as mechanistic organizations, we borrow Divide-and-Conquer (D&C) from the organization theory domain as a generic strategy of effort judgment for WSC-based SOA implementations. Through the generic strategy D&C, the work on effort judgment for a whole WSC-based SOA implementation can be further narrowed down to that for individual WSCs.
When it comes to the effort judgment for individual WSC approaches, it is necessary to draw some classification, and thereby facilitate the effort judgment for different type of WSC approaches. To overcome the deficiencies of existing classifications for WSC, this book introduces a novel classification matrix aimed at putting the influence on the effort involved in WSC. The matrix uses clarified terminology, and differentiates the classifications between the Context and Process dimensions. The Context dimension includes five effort-related contexts, while the Process dimension is divided into three process models. Considering the different influences of different contexts and processes on the composition effort, those process models and context types in the classification matrix can be viewed as effort factors when composing Web services. Therefore, we can use this classification matrix to facilitate the effort judgment for different WSC approaches.

After applying a set of qualitative effort-related hypotheses to WSC effort factors that are identified by the aforementioned classification matrix, this book employs several rules to assign effort scores to different factors and different types of WSC approaches. These effort scores are used to facilitate qualitatively judging different effort between different types of WSC approaches, and eventually construct an effort checklist for WSC approaches. For a certain WSC-based SOA implementation, designers and developers can use this effort checklist together with the D&C algorithm to further calculate effort scores of different implementation proposals, and therefore conveniently and qualitatively compare the effort required by different proposals.
Contents

Abstract .................................................................................................................................................. i
Contents .................................................................................................................................................. iii
List of Figures ......................................................................................................................................... v
List of Tables .......................................................................................................................................... vii
Chapter 1: Introduction .......................................................................................................................... 1
  1.1 Research Background and Problem Statement .............................................................................. 2
  1.2 Research Questions ....................................................................................................................... 3
  1.3 Research Objectives and Contributions ....................................................................................... 3
  1.4 Book Outline ................................................................................................................................. 6
Chapter 2: Background and Literature Review ........................................................................................ 8
  2.1 Service-Oriented Architecture (SOA) and Web Service .............................................................. 9
  2.2 Software Cost and Effort Estimation ............................................................................................ 11
    2.2.1 Data-Intensive Approaches to Software Effort Estimation .................................................. 12
    2.2.2 Human-Intensive Approaches to Software Effort Estimation .......................................... 21
  2.3 Existing Approaches to Cost and Effort Estimation for SOA Projects ......................................... 23
  2.4 Summary ..................................................................................................................................... 26
Chapter 3: Generic Strategy of Effort Judgment: Divide-and-Conquer (D&C) from an Organizational Perspective .................................................................................................................... 27
  3.1 Think of SOA from the Organizational Perspective .................................................................. 28
    3.1.1 Viewing an SOA System as an Organization .................................................................... 28
    3.1.2 Treating an SOA System as a Mirror of Real Organization ............................................. 30
    3.1.3 Analogue between SOA Implementation and Organization Design ................................... 32
  3.2 Benefits from Thinking of SOA from an Organizational Perspective .......................................... 39
    3.2.1 Towards Technology Independent Strategies of SOA Implementation ............................. 39
    3.2.2 Towards SOA Implementation Complexity Measurement ............................................... 45
    3.2.3 Towards Effort Judgment for SOA Implementation ........................................................... 49
  3.3 D&C Approach to Effort Judgment for SOA Implementation ...................................................... 49
    3.3.1 Brief Background of D&C Approach .................................................................................. 49
    3.3.2 Effort Estimation for Generic SOA Implementation .............................................................. 52
  3.4 Summary ..................................................................................................................................... 57
Chapter 4: Prerequisite to Effort Judgment: Identification of Effort Factors of Web Service Composition

4.1 Context Classification of Web Service Composition Approaches ................................................................. 60
  4.1.1 Pattern: Orchestration and Choreography .......................................................... 61
  4.1.2 Semiotics: Syntactic and Semantic Compositions ..................................................... 62
  4.1.3 Mechanism: RESTful and SOAP-based Compositions ............................................. 64
  4.1.5 Runtime: Static and Dynamic Compositions .............................................................. 69

4.2 Process Classification of Web Service Composition Approaches ........................................................... 70
  4.2.1 One-Stop Approach Model ...................................................................................... 70
  4.2.2 Bridge Approach Model .......................................................................................... 72
  4.2.3 Double-Bridge Approach Model .............................................................................. 74
  4.2.4 Clarification against Similar Concepts ...................................................................... 76

4.3 Effort Factors of Web Service Composition .............................................................................................. 79

4.4 Summary .................................................................................................................................................. 82

Chapter 5: Implementation of Effort Judgment: A Qualitative Approach .......................................................... 83

5.1 Circumstantial-Evidence-Based Judgment for Various Influences of Different Effort Factors .......................................................... 84
  5.1.1 Effort Judgment with Circumstantial Evidence .......................................................... 85
  5.1.2 Circumstantial Evidences for Software Development ................................................ 87
  5.1.3 Circumstantial-Evidence-Based Judgment around Effort Factors of WSC .......... 89

5.2 Effort Judgment for Individual Web Service Composition Approaches ...................................................... 95
  5.2.1 Comparison between Partial Composition Efforts Determined by Comparable Effort Factors .............................................................................................................. 96
  5.2.2 Qualitative Effort Comparison between Different Web Service Composition Approaches .................................................................................................................................. 99

5.3 Effort Judgment for Web Service Composition based SOA Implementations .......... 103

5.4 Summary .................................................................................................................................................. 105

Chapter 6: Discussion and Conclusion .............................................................................................................. 107

6.1 Research Achievements ......................................................................................................................... 107

6.2 Response to Research Questions .......................................................................................................... 109

6.3 Limitations ............................................................................................................................................. 110

6.4 Future Work ......................................................................................................................................... 111

Bibliography ...................................................................................................................................................... 113
List of Figures

Figure 2.1: Roles of Services and Operations ................................................................. 9
Figure 2.2: One Classification of Software Cost and Effort Estimation Methods ........ 12
Figure 2.3: An Example of Perceptron of ANN .............................................................. 18
Figure 2.4: General Cyclical Case-Based Reasoning Process ....................................... 19
Figure 3.1: View SOA System as an Instance of Organization ....................................... 29
Figure 3.2: A Company and its Infrastructural SOA System .......................................... 31
Figure 3.3: The Pentagonal Process of SOA Implementation / Organization Design ...... 32
Figure 3.4: The Matrix Structure of SOA Systems ........................................................... 35
Figure 3.5: Service Orchestration and Choreography ..................................................... 38
Figure 3.6: Tall Structure vs. Flat Structure for Implementing the Same Business Process ................................................................. 41
Figure 3.7: An SOA-based Application of Hotel Booking in a Travel Agency ............... 43
Figure 3.8: The Travel Agency Application after Applying Four Proposed Strategies .... 45
Figure 3.9: Complexity Measurement Framework for SOA Implementation ................. 46
Figure 3.10: Principle of Divide-and-Conquer ............................................................... 50
Figure 3.11: Procedure of D&C based Effort Estimation for SOA Projects .................. 53
Figure 3.12: Redesigned Automation System of RailCo Ltd. ......................................... 55
Figure 3.13: Procedure of Cost and Effort Estimation for Developing an Automated System (RailCo Ltd. case study) ................................................................. 56
Figure 4.1: Syntactic Web Service and Semantic Web Service ....................................... 63
Figure 4.2: Stages of a Web Service Composition Scenario .......................................... 68
Figure 4.3: One-Stop Approach Model ............................................................................. 71
Figure 4.4: Bridge Approach Model .................................................................................. 73
Figure 4.5: Double-Bridge Approach Model ..................................................................... 75
Figure 4.6: A Sample of Web Service Composition Model (Dumez et al. 2008) ......... 77
Figure 4.7: One Process and Five Context-related Effort Factors of Web Service Composition ........................................................................................................... 80
Figure 5.1: The Inference Process of Diagnostic Reasoning .......................................... 86
Figure 5.2: The Hockey Stick Function ............................................................................ 88
Figure 5.3: A Sample of Consistent Comparisons for $E_{F2} > E_{F1}$ .......................... 100
Figure 5.4: WSC Tree of Automation System of RailCo Ltd. ............................................ 104
List of Tables

Table 2.1: Quantitative vs. Qualitative Research Directions ........................................... 26
Table 3.1: Comparison between SOA System and Organization ....................................... 30
Table 3.2: Algorithm of D&C based Effort Estimation for SOA Projects ............................ 54
Table 4.1: Classification Matrix of WSC and Sample WSC Approaches ............................. 81
Table 5.1: Circumstantial-Evidence-Based Effort Judgment around Different Effort Factors of Web Service Composition ................................................................. 94
Table 5.2: Effort Comparison between Orchestration and Choreography .......................... 96
Table 5.3: Effort Comparison between Syntactic and Semantic Web Service Composition Approaches .............................................................................................................. 96
Table 5.4: Effort Comparison between SOAP-based and RESTful Web Service Composition Approaches ............................................................................................................... 97
Table 5.5: Effort Comparison between Manual, Semi-Automated and Automated Design-Time Web Service Composition Approaches ......................................................... 97
Table 5.6: Effort Comparison between Static and Dynamic Runtime Web Service Composition Approaches ............................................................................................................. 98
Table 5.7: Effort Comparison between One-Stop, Bridge and Double-Bridge Process based Web Service Composition Approaches ................................................................. 98
Table 5.8: Effort-Judgment-Assistant Table for Web Service Composition .......................... 101
Table 5.9: A Sample of Effort-Judgment Checklist for Web Service Composition ............... 102
Table 5.10: Algorithm of D&C based Effort Judgment for WSC-based SOA Projects ........ 103
Table 5.11: Effort Comparison between Implementation Proposals for the Automation System of RailCo Ltd............................................................................................................ 105
CHAPTER 1

Introduction

The rapidly changing global environment is becoming a notable characteristic of every aspect of society. To adapt the changing environment, Service-Oriented Architecture (SOA) has been accepted broadly and developed rapidly to bridge the gap between frozen information technology (IT) and constantly fluid business. Forrester Research shows that companies can reduce at least 30% of costs for the integration of projects and maintenance by implementing SOA. Gartner even predicts that 40% of capital expenditures will be used for service infrastructure development by 2011, and at least one third of business application software spending will be on subscribing services instead of on achieving product licenses (Demirkan, Kauffman et al. 2008). Unfortunately they do not mention how much or how to determine the effort and cost that are required to establish those SOA based software projects.

In fact, effort estimation plays a vital role in software development, especially in the emerging SOA environment. Before SOA implementations, the requirement of effort estimation ranges from budget analysis to the cost-benefit balance. Nevertheless, effort estimation for SOA implementation confronts many challenges mainly due to two reasons: one is the complexity of SOA, and another is the diversity of SOA projects. Considering it is difficult to collect enough data from real SOA implementations, this research tries to adopt “learning by analysis” (Fox 1981) as a research methodology and use a qualitative approach to realize effort judgment between different proposals of a certain SOA implementation.

This book only concentrates on Web service composition (WSC) based SOA implementations that do not take into account development of new services or migration of legacy systems, and thereby identifies a set of effort factors when composing Web services. A qualitative checklist for WSC approaches is eventually generated and can be used to judge the effort between different implementation proposals of a WSC-based SOA project. This chapter introduces the research background, promotes the research questions based on the problem statement, and specifies the
research objectives and contributions.

1.1 Research Background and Problem Statement

For properly balancing the benefit and cost in SOA system investment or project bidding, it is crucial to determine the required effort before developing an SOA application. However, there is little work published about cost or effort estimation for SOA implementations. Most organizations even “do not have a clue how to approach the cost estimation process, and in many cases grossly underestimate the cost of their SOA” (Linthicum 2007). Generally, the challenges of effort estimation for SOA implementations are twofold:

- **The complexity of SOA.** Josuttis (2007) has pointed out that distributed processing would be inevitably more complicated than non-distributed processing, and any form of loose coupling would increase complexity. Practically, building a true heterogeneous SOA for a wide range of operating environments may take years of development time if the company does not have sufficient SOA experience and expertise. Meanwhile, the more complexity involved in a system, the more difficulty the designers or engineers have to understand the implementation process and thus the system itself (Cardoso 2005). In other words, people have to devote more effort to accurate manipulations when performing more complicated tasks.

- **The diversity of SOA projects.** O’Brien (2009) proposes a framework for scope, cost and effort estimation for SOA projects, which reveals the diversity of SOA projects. In general, the main types of SOA projects engaged in an organization may include: service development, service mining, service composition, SOA application development, SOA infrastructure, SOA Governance, and SOA Architecture Analysis. For each of these types of project there are various activities and other factors that will involve expenditure of effort and cost. Moreover, developing SOA projects may face numerous and diverse challenges. A systematic literature review (Gu and Lago 2009) has concluded 413 challenges of service-oriented software engineering in total under 45 topics, 8 types, and 2 dimensions.

To decrease the complexity and difficulty of this effort estimation problem within SOA environment, we can start from and mainly focus on the effort estimation for a particular type of SOA project: service composition based SOA implementations. In SOA, composition of services is the concept with which we provide support for business processes in a flexible and feasible way. Through this
way of business support, business processes in SOA are essentially a composition of service invocations in a certain order with rules that influence the execution and other constructs, such as parallel invocations, transformations of data, dependencies, and correlations. In practice, considering the de facto implementation of service concept is Web services (Newcomer and Lomow 2004), we can focus on the effort estimation for WSC-based SOA implementations.

In fact, it is reasonable and suitable to choose WSC-based SOA implementations as a problem context for the research into effort estimation. On the one hand, as organizations move to having more and more services, and business application software will increasingly rely on subscribing services (Demirkan, Kauffman et al. 2008), then the major problem in SOA implementation will be service composition and may be less on development of new services. Therefore, our concerns can be narrowed down from generic SOA implementation to service composition based SOA implementation. On the other hand, it has been identified that the benefits of SOA can not be fully realized until reaching the level of service composition (Sarang, Jennings et al. 2007). Consequently, it is worthwhile to narrow down our concerns from generic SOA implementations to WSC-based SOA implementations that do not take into account service migration or new service development.

1.2 Research Questions

According to the problems discussed previously, the following four research questions are investigated in this research.

- Q1: How to unfold research into the topics related to service-oriented computing?
- Q2: What is the generic strategy for effort judgment for WSC-based SOA implementations?
- Q3: What factors should be of concern when judging the effort of individual WSC approaches?
- Q4: How can these effort factors be used to realize the effort judgment for WSC-based SOA implementations?

1.3 Research Objectives and Contributions

Following the proposed research questions, the research objectives are separated into answers to each of them, while the contributions of our work can then be specified respectively.
Q1: How to unfold research into the topics related to service-oriented computing?

Considering “learning by analogy” is an efficient way to unfold research into distributed systems, we propose to borrow the achievements from organization theory domain to inspire the investigation of this research question. In our work, we suggest thinking of SOA from an organizational perspective according to the definition of organization: if viewing both people and services as organizational units that are arranged together to achieve a specific goal, we can treat an SOA system as an organization. Moreover, more common features are identified and compared between SOA systems and organizations. Through the comparison and analogy, we can be convinced that exploring the knowledge in organization theory is a possible and reasonable way to unfold research into the topics related to service-oriented computing.

The main contribution of this part of work is that new interdisciplinary research opportunities are revealed in a connection between service-oriented theory and organization theory. To the best of our knowledge, this is the first time to use organization theory to inspire the research into service-oriented software engineering. Through thinking of SOA from an organizational perspective, we have tried to identify technology independent strategies for implementing SOA, measure SOA system complexity, and finally investigate the strategies of effort judgment for WSC-based SOA implementations.

Q2: What is the generic strategy for effort judgment for WSC-based SOA implementations?

In organization theory, a mechanistic organization uses multiple levels of management to insure proper and centralized decision making. The hierarchical structure of a mechanistic organization can be viewed as a global goal tree: the global goal is stepwise decomposed into a set of sub-tasks, while the goal is achieved through recomposing the sub-solutions to those sub-tasks. Consequently, the notion of decomposition and recomposition directly engenders a Divide-and-Conquer (D&C) approach that allows the organization to use larger groups of work units more efficiently and address problems on a larger scale. Following the answer to the first research question, composite services play integrative roles in an SOA system as analogous to full-time managers who are in charge of orchestrating work across units in an organization. Hence, WSC-based SOA systems can also be thought of a D&C way to achieve business goals. When building WSC-based SOA systems, we can therefore follow the D&C way to analyze
and judge the effort.

The main contribution of this part of work is that we firstly apply the D&C strategy to effort judgment for WSC-based SOA implementations. The D&C strategy, different from Top-Down or Bottom-Up strategies, emphasizes a recursive process when resolving a problem. In fact, the D&C strategy is suitable for generic issues of SOA implementation, and can be helpful for simplifying the complexity of SOA effort estimation. Following the D&C principle, consequently, our concentration of effort judgment for a complete WSC-based SOA implementation can be narrowed down to individual WSCs.

- Q3: What factors should be concerned when judging the effort of individual WSC approaches?

Before identifying effort factors, it is necessary to get familiar with existing WSC approaches. Since it is difficult to exhaustively explore all the published composition approaches, we can inductively classify the existing WSC works, and thereby facilitate the comprehension of related knowledge and the effort estimation work. To answer this research question, we present a novel classification matrix aimed at the influence on the effort of WSC. This matrix uses clarified terminology, and differentiates the classifications between the Context and Process dimensions. The Context dimension includes major effort related contexts that are Pattern, Semiotics, Mechanism, Design Time and Runtime. The Process dimension is divided into three process models, namely One-Stop, Bridge and Double-Bridge. As such, considering the different influences on the composition effort, different context types and process models can be viewed as different effort factors of WSC.

The contributions of this part of work are manifold. Firstly, a new perspective of classification of WSC is developed, which can help researchers explore the knowledge space in the WSC domain, and help developers choose suitable techniques when composing Web services. Secondly, we propose a new way, using the effort-oriented classification matrix, to reveal effort factors of WSC. Finally yet importantly, new research opportunities could be identified when comparing and analyzing different composition approaches through the classification matrix. For example, there is a lack of research concentration on the gap between automated composition at design time and dynamic composition at runtime.

- Q4: How can these effort factors be used to realize the effort judgment for WSC-based
Chapter 1: Introduction

SOA implementations?

As discussed in Chapter 2, it could be difficult to collect enough WSC data from industry, so that human-intensive effort estimation approaches would be more feasible for this research topic. For human-intensive effort estimation approaches, usually referred to as expert judgment, we propose to re-consider the justification from an evidence-based perspective. According to the classification of evidence, the results of the Case-Based Reasoning (CBR) based mental processes in traditional expert judgment can be regarded as direct evidence: experts act as witnesses and adduce previous cases for the current one. Considering the limitation of direct evidence collection – the requirement of availability of experts with experience from similar projects, we can adopt circumstantial-evidence-based effort judgment for WSC-based SOA implementation. In the context of software effort estimation, unlike direct evidences that are similar projects’ or activities’ actual effort, the circumstantial evidences are development experiences deposited in human knowledge. When judging effort of WSC instances, we can apply the circumstantial evidences to suitable effort factors and then give qualitative comparison between effort estimates of different WSC proposals.

The most important contribution here is the novel idea that circumstantial-evidence-based judgment can be used to facilitate qualitative software effort estimation. In fact, the circumstantial-evidence-based judgment can not only act as complementary to expert judgment for effort estimation, but also help settle implementation design for software projects. Moreover, the circumstantial evidences in the context of effort judgment can be accumulated and deposited as general knowledge to further guide and assess individual expert judgments. Another contribution is that we have developed a method of assigning scores to effort factors of WSC, which can be used to facilitate the qualitative effort judgment between different effort factors, between different types of WSC approaches, and even between different WSC-based SOA implementation proposals.

1.4 Book Outline

This book is organized as follow. Chapter 2 describes the background of WSC-based SOA implementation and the corresponding knowledge of software cost and effort estimation. Through distinguishing between data-intensive and human-intensive approaches to software effort estimation, this chapter reviews the characteristics of four estimation types, namely Estimation by Predefined
Modelling, Estimation by Un-Predefined Modelling, Estimation by Analogy, and Expert Judgment. After exhaustively exploring the existing effort estimation approaches for SOA implementation, the initial analysis suggests that this research topic should be unfolded along the qualitative direction.

Chapter 3 discusses the generic strategy of effort judgment for WSC-based SOA implementations. Through viewing SOA from an organizational perspective, this chapter shows the interdisciplinary research opportunities such as building complexity framework and identifying technology independent strategies for SOA implementations. More importantly, the mechanism of an organization is used to inspire the investigation of effort estimation for WSC-based SOA implementation. The D&C approach is finally employed as the generic strategy to tackle the effort judgment issue in this research topic.

Chapter 4 explains the process and result of identification of WSC effort factors. Following the D&C strategy, the work on effort judgment can be narrowed down from a complete WSC-based SOA implementation to individual WSCs. Therefore, this chapter focuses only on individual WSC approaches, and classifies the published approaches into a matrix composed of context dimension and process dimension. Different context types and process models are naturally regarded as different effort factors of WSC due to their different influences on the composition effort. Correspondingly, the characteristics of different factors are also reviewed respectively.

Chapter 5 develops a circumstantial-evidence-based method to realize the effort judgment for WSC-based SOA implementations. Through treating existing development experiences as circumstantial evidence, this chapter firstly uses circumstantial-evidence-based effort judgment to give qualitative comparison between different influences of different WSC effort factors. Secondly, a set of symbols and rules are used to build up a checklist for effort judgment for different types of WSC approaches. Benefiting from the effort-judgment checklist, the effort tradeoff before WSC-based SOA implementations can be finally determined following the D&C strategy.

Chapter 6 discusses and concludes this book by using three sections to respectively summarize our research achievements, the limitations of our current work, and the future work. Each section is structured following the sequence of our work.
CHAPTER 2

Background and Literature Review

Service-Oriented Architecture (SOA) is a booming computing paradigm that utilizes services as fundamental elements for developing software applications/solutions. The concept of a service is proposed as a self-describing, platform-independent computational element that performs functions ranging from simple requests to complicated business processes and supports rapid, low-cost composition of distributed systems. In practice, the de facto implementation of the service concept is Web services that allow organizations to expose their reusable business process fragments over the Internet using open standards and protocols.

Considering the increasing adoption of SOA and the architectural difference between SOA implementations and traditional software developments, it is necessary to investigate the issue of effort estimation for SOA project development particularly. Before tackling this particular issue, it is vital to get familiar with the existing effort estimation techniques and understand the state of the practice of current approaches to effort estimation for SOA implementations. Through distinguishing between data- and human-intensive approaches to software effort estimation, this chapter reviews the characteristics of four estimation types, namely Estimation by Predefined Modelling, Estimation by Un-Predefined Modelling, Estimation by Analogy, and Expert Judgment. Meanwhile, this chapter also tries to exhaustively explore the existing effort estimation approaches for SOA implementation. Note that, to arrange this book better and logically, the literature review about WSC is combined with Chapter 4 rather than being elaborated here.

This chapter is organized as follow. The concepts SOA and Web service are briefly introduced in Section 2.1. Section 2.2 separates the existing cost and effort estimation techniques into two types, namely data- and human-intensive approaches. The data-intensive approaches can be further categorized into Estimation by Predefined Modelling, Estimation by Un-Predefined Modelling, and Estimation by Analogy. Section 2.3 reviews the published works about cost and effort estimation for SOA implementation. A summary of the literature review work is given in Section 2.4.
2.1 Service-Oriented Architecture (SOA) and Web Service

It is hard to find out who developed the term SOA firstly, Josuttis (2007) introduced that it is Alexander Pasik, a former analyst at Gartner, that coined the term SOA for a class on middleware that he was teaching in 1994. The basic SOA principles have not changed even if they appeared before XML or Web Services were invented. But it is generally accepted that SOA was firstly reported by Gartner in 1996 (Schulte and Natis 1996). SOA is a flexible, standardized software architecture style of design that guides all aspects of creating and using business services throughout their lifecycle (from conception to retirement). Meanwhile, SOA is also a way to define and provision an IT infrastructure to allow different applications to exchange data and participate in business processes, regardless of the operating systems or programming languages underlying those applications (Newcomer and Lomow 2004). Thus, service-oriented architecture is not limited to just Web services, or technology or technical infrastructure either. Instead, it reflects a new way of thinking about processes that reinforces the value of commoditization, reuse, semantics and information, and creates business value (Bieberstein, Bose et al. 2005).

At the core of an SOA is a service (Lewis, Morris et al. 2005). Brown et al. (Brown, Johnston et al. 2002) specified that a service is a coarse-grained, discoverable, and self-contained software entity that interacts with applications and other services through a loosely coupled, often asynchronous, message-based communication model. A formal definition of service can be found in the W3C Web Services Glossary (W3C 2004): A service is an abstract resource that represents a capability of performing tasks that form a coherent functionality from the point of view of provider entities and
requester entities. To be used, a service must be realized by a concrete provider agent (service registry). This definition also reveals three roles involved in SOA (IBM Redbooks 2006), as illustrated in Figure 2.1.

- **Service Provider** - creates a service and in most cases publishes its interface and access information to a Service Registry. Each Service Provider must decide which services to expose, evaluate trade-offs between security and easy availability, determine how to price the services or determine how to exploit the value of the services if they are free. The Service Provider also has to decide in which category the service should be listed, and what sort of trading partner agreements are required to use the service.

- **Service Registry** - is responsible for making the service interface and implementation access information available to service consumers. The implementers of a Service Registry must consider the scope with which the registry will be implemented. For example, there are public Service Registries available over the Internet to an unrestricted audience, as well as private Service Registries that are only accessible to users within a company-wide intranet.

- **Service Requester** - locates (discovers) entries in the Service Registry and then binds to the Service Provider in order to invoke the defined service. Before finding and consuming services, the Service Requester interrogates the Service Registry to find Service Providers for a particular service description. If suitable Service Providers exist in the Service Registry, the registry returns a reference to one or more Service Providers so that the Service Requester can select and bind to any of them. After the requester-provider binding, the bound Service Provider returns a reference to a service object that implements the service functionality.

There are significant differences between SOA and Web Service concepts: SOA is a type of architecture, while Web Service specifications define an interoperable platform supporting a SOA (Rotem-Gal-Oz 2007). Also SOA is not limited to Web services nor do Web services imply SOA (Solutions 2010). Nevertheless, contemporary SOA is intrinsically reliant on Web services so much so that Web services concepts and technology used to actualize service-orientation have influenced and contributed to a number of the common SOA characteristics (Erl 2005). The real momentum for SOA was created by Web services, which, initially driven by Microsoft, reached a broader public in 2000 (Levitt 2001), and the major advantages of implementing an SOA using Web services are that Web services are pervasive, simple, and platform-neutral (Newcomer and Lomow 2004). Web services provide the potential of fulfilling these primitive requirements, but they need to be
intentionally designed to do so. This is because the Web services framework is flexible and adaptable. Web services can be designed to duplicate the behavior and functionality found in proprietary distributed systems, or they can be designed to be fully SOA-compliant. This flexibility has allowed Web services to become part of many existing application environments and has been one of the reasons behind their popularity. It also reveals the fact that Web services are not necessarily inherently service-oriented. Following the operations shown in Figure 2.1, a service-oriented architecture based on SOAP (W3C 2003), Web services Description Language (WSDL) (W3C 2007), and Universal Description, Discovery, and Integration (UDDI) (Bellwood, Clement et al. 2002) requires the following interactions:

- A Web service publishes its WSDL definition into a UDDI registry.
- The client discovers the service’s definition in the registry.
- The client uses information from the WSDL definition to send messages or requests directly to the service via SOAP.

### 2.2 Software Cost and Effort Estimation

Software cost and effort estimation, started as early as the late 1950s and 1960s (Norden 1958; Nelson 1966), is the process of predicting the total cost required to develop a software system. Effort estimation techniques have drawn upon a variety of fields, statistics, machine learning, and knowledge acquisition. Given the diversity of estimation techniques one is faced with the difficult exercise of determining which technique would be the best in given circumstances. In order to assess a technique’s appropriateness, the underlying assumptions, strengths, and weaknesses have to be known and its performances must be assessed. The ability to accurately estimate the time/cost taken for a software project to come to its successful conclusion has been a challenge for software engineers (Wikipedia 2009), which has been researched widely, and many estimation measures have been proposed for traditional software development over the last 30 years (Sommerville 2006), such as Algorithmic Models, Expert Judgment, Machine Learning based Metrics, Parkinson's Law and Pricing to Win (Sommerville 2006; Briand and Wieczorek 2002; Leung and Fan 2009). Therefore, one possible classification of software effort and cost estimation methods can be illustrated as shown in Figure 2.2.

More generally, the approaches to software effort estimation can be divided into two categories: data-intensive approaches and expert-intensive approaches. As the name suggests, the data-intensive approaches require a large amount of data from past projects to support software effort estimation.
When estimating the effort of a new software project, mathematical equations or models may be built on existing data sets. In contrast, the expert-intensive approaches estimate effort of a new project according to the opinions of experts who have knowledge or experiences of similar project developments. The data-intensive effort estimation approaches have been well addressed in academia, while the dominant estimation method adopted in industry is based on expert-intensive approaches (Jørgensen and Shepperd 2007). Before identifying or proposing a suitable effort estimation method for WSC-based SOA implementation, it is necessary to comprehensively understand existing approaches to software effort estimation.

![Figure 2.2: One Classification of Software Cost and Effort Estimation Methods](image)

### 2.2.1 Data-Intensive Approaches to Software Effort Estimation

According to the extensive study during the previous decades, we can find that data-intensive effort estimation approaches mainly focus on three types of techniques: Estimation by Predefined Modelling, Estimation by Un-Predefined Modelling, and Estimation by Analogy.

**1) Effort Estimation by Predefined Modelling**

Effort estimation by predefined modelling uses an explicit functional form to relate effort to one or more independent effort drivers of software development, for example regression-based estimation approaches and the well documented COCOMO model (Boehm, Abts et al. 2000). In particular, regression-based techniques are the most popular way of estimating software development effort because they are convenient to use (Mair and Shepperd 2005). It has been revealed that roughly half of existing relevant research work concentrates on regression-based estimation approaches including the most common parametric estimation models (Jørgensen and Shepperd 2007).
However, the realization of the predefined modelling is not a trivial undertaking. Once a model is proposed, the data set used to fit the model must comprise a large enough number of observations to guarantee covering the independent parameters that appear in the model. Meanwhile, the assumptions of the independence of parameters need to be tested; otherwise the model could be still invalidated if the input parameters are not mutually independent. When deriving regression models, the statistical techniques assume that the statistical constrains are satisfied, such as the data set has certain properties, and the data are from the same underlying population. Last but not least, it is difficult to justify the quality of observations that may be collected from a number of different sources, because the methods in which software projects are developed and the ways in which effort-related data are collected may vary greatly in different organizations. For example, the standard regression method uses Ordinary Least Squares Regression to build linear parametric models for effort estimation (Mittas and Angelis 2008). The past data set are used to refine necessary parameters by fitting the data to the predefined model in an attempt to minimise the overall sum of squared errors. Nevertheless, this standard regression method inevitably suffers from two weaknesses. On the one hand, the regression lines can be sensitive and susceptible to statistical outliers. On the other hand, the true data may behave in a curvilinear manner and therefore the predefined regression lines are not applicable.

Although there are inevitable pitfalls in the methods of effort estimation by predefined modelling, we should not despise the significance of these methods. As previously mentioned, predefined modelling like regression has been arguably viewed as the simplest and more elucidating way to investigate relationships between effort and effort drivers empirically (Walkerden and Jeffery 1997). In this book, we do not plan to exhaustively examine all the existing techniques of effort estimation by predefined modelling. Instead, we only review two typical models, the SIZE based model and COCOMO II model, so as to facilitate comprehending the general and essential mechanism of this effort estimation strategy.

**The SIZE based model**

The SIZE based model is one of the best-known effort models, which can be expressed as a formulation:

\[
EFFORT = a \times SIZE^b \times m
\]

(2.1)

where \( EFFORT \) is the labour of a software development task that is generally accounted by calculating how long and how many workers are needed to finish the task, and the unit can be person-day, person-month, or person-year. The constant factor, \( a \), depends on local organisational
experiences and the type of software project that is developed. SIZE may be either an assessment of the code size of the software project, for example the number of lines of code (LOC), or a measurement of functionality estimate of the project, for example function or object points. The value of the exponent, \( b \), indicates either economies or diseconomies of scale as the size of the software increases. According to the empirical research, such as (Kitchenham 1992; Banker, Chang et al. 1994), the value of \( b \) usually lies between 1 and 1.5 (Sommerville 2006). The multiplier, \( m \), is a synthetic parameter that reflects the combination of process, product and development attributes, for example the dependability requirements for the software and the experience of the development team.

As the name suggests, the substantial component of the SIZE based estimation model is the measurement of the size of the software to be developed. Lines of source code have been widely accepted as a software size metric, because the number of LOC is easy to count especially when programming in the first generation language like assembly language or FORTRAN. However, there is not a simple relationship between program statements and the true size of a software project when we use modern languages. Moreover, LOC cannot be exactly measured until the system is completed. Accurate LOC is difficult to estimate at an early stage of a project because it can vary due to different design decisions and implementation strategies. Consequently, some other options appear to replace the measurement of code size with that of the functionality of the code, for example Function Points (FPs).

FPs is designed to measure system size by quantifying the amount of functionality provided to the user in terms of the number of inputs, outputs, and files (Albrecht and Gaffney 1983; Kemerer 1993). The idea originates from Albrecht’s work on Function Points at IBM (Albrecht 1979). The FPs counts are based on system features seen by the end user. In 1982, DeMarco also published a description of a different kind of functional measures that he termed the “bang metrics” (DeMarco 1982). These metrics are based on counts derived from data flow diagrams, entity-relationship models, and state diagrams. A frequently mentioned functional size measure is IFPUG Function Points (International Function Point Users Group) (IFPUG 1994). It defines five function types that can be counted from early life cycle documents and then stipulates a linear weighting scheme for each of the five function types depending on their complexity. The five function types are classified as follows:

- An external input is any elementary process of an application that processes data or control information that enters from outside the boundary of the application.
- An external output is an elementary process of an application that generates data or control
information that exits the boundary of the application.

- An internal logical file is a user identifiable group of logically related data or control information maintained through an elementary process of the application.
- An external interface file is a user identifiable group of logically related data or control information references by the application but maintained within the boundary of a different application.

An external inquiry is an elementary process of the application that is made of an input-output combination that results in data retrieval. The input side is the control information that defined the request for data. The output side contains no derived data.

In practice, FP's measures must be extracted from design documents (Moser, Henderson-Sellers et al. 1999), and are considered more programming language independent (Jones 1991). FP's can be used continuously throughout the entire systems development life cycle (SDLC), furthermore, not only can FP's be estimated early in the life cycle, but they can also be recalculated as development continues (Fenton and Pfleeger 1996). Validity is often mentioned as a necessary property for successful software measures (Henderson-Sellers 1996). FP's has been criticized because they depend on subjective estimates rather than objective data (Mahmood, Pettingell et al. 1996) and the difference between counts provided by different individuals can result in variances of up to 30% (Kitchenham 1997). The procedures for counting FP's are labor-intensive, require expertise and experience, and do not easily lend themselves to automatic data collection (Kusumoto, Imagawa et al. 2002), although some attempts have been made to try to automate portions of the FP's calculation process (Lamma, Mello et al. 2004).

**COCOMO II model**

In fact, the COCOMO II model can also be traced back to SIZE based estimation model. The model with size measurement by LOC has been investigated as the basic version of the COCOMO model (Walkerden and Jeffery 1997). COCOMO II uses Object Points that only concerns screens, reports and modules in conventional programming languages, which gives effort estimation more easily from a high-level software specification (Sommerville 2006).

However, COCOMO II is more developed and more complicated than the SIZE based estimation model. The COCOMO II research started in 1994 and is initially described (Boehm, Clark et al. 1995). COCOMO II has a tailorable mix of four sub-models, Applications Composition, Early Design, Reuse, and Post Architecture.

- Application Composition Model - This assumes that systems are created from reusable
components, scripting or database programming. It is designed to make estimates of prototype development. Software size estimates are based on application points, and a simple size/productivity formula is used to estimate the effort required. Application points are the same as object points, but the name was changed to avoid confusion with objects in object-oriented development.

- Early Design Model - This model is used during early stages of the system design after the requirements have been established. Estimates are based on function points, which are then converted to number of lines of source code. The formula follows the standard form discussed above with a simplified set of seven multipliers.

- Reuse Model - This model is used to compute the effort required to integrate reusable components and/or program code that are automatically generated by design or program translation tools. It is usually used in conjunction with the Post Architecture model.

- Post Architecture model - Once the system architecture has been designed, a more accurate estimate of the software size can be made. Again this model uses the standard formula for cost estimation discussed above. However, it includes a more extensive set of 17 multipliers reflecting personnel capability and product and project characteristics.

As can be seen, reuse and reengineering are firstly taken into account with other new capabilities in COCOMO II (Boehm, Madachy et al. 1996). The reuse model is used to estimate the effort required to integrate reusable or generated code, which is a nonlinear model. Some effort is required if reuse is considered to make an assessment of whether reuse is possible. Furthermore, as more and more reuse is contemplated, the costs per code unit reused drop as the fixed understanding and assessment costs are spread across more lines of code.

COCOMO II considers reused code to be of two types. Black-box code is code that can be reused without understanding the code or making changes to it. The development effort for black-box code is taken to be zero. Code that has to be adapted to integrate it with new code or other reused components is called white-box code. Some development effort is required to reuse this because it has to be understood and modified before it can work correctly in the system. In addition, many systems include automatically generated code from program translators that generate code from system models. This is a form of reuse where standard templates are embedded in the generator. The system model is analyzed, and code based on these standard templates with additional details from the system model is generated. The COCOMO II reuse model includes a separate model to estimate the costs associated with this generated code.
2) Effort Estimation by Un-Predefined Modelling

Unlike effort estimation by predefined modelling, the techniques without predefined modelling are generally input-output metrics and they do not require proposing explicit functional form for effort estimation. The typical techniques of effort estimation by un-predefined modelling can be found as machine learning approaches (Srinivasan and Fisher 1995), for example decision trees and artificial neural network (ANN). Although un-predefined modelling does not need a data set to cover and work out the independent parameters that appear in predefined models, those machine learning techniques still require a large number of past data to train effort estimation systems. Here we employ ANN as an example to show the input-output type of technique for effort estimation.

Artificial neural network (ANN) based effort estimation

ANN is inspired by biological nerve net that is a basic knowledge of how the human brain works. With similar biological neurons structures, ANN can be viewed as simplified mathematical techniques of human brain. Each neuron is like a mathematical function with some inputs, a mathematical formula, and outputs. A collection of neurons or process elements with internal connection, and they function as parallel distributed computing networks. In contrast to existing computer systems that are programmed by a programmer to perform sequential commands, ANN must be trained before being used. The main idea in ANN is to produce intelligent systems capable of sophisticated computations by learning from previous project information, experiences and details to provide new data, rules, and experiences based on the inferring from learnt data.

ANN includes some active and hidden layers. Each layer has interconnected nodes, neurons, where each node generates a non-linear function of its input (Mitchell 1997). The data processing in ANN starts by generating the structure of the network and selecting the adequate leaning technique to train, test and validate the generated network by using existing data sets. Each ANN involves three main entities:

- Neurons or nodes.
- Layers and interconnection structure.
- A learning algorithm.

The most widely used training techniques in the ANN for estimation are known as back-propagation and feed-forward techniques. In fact, ANN has been used in software architecture, modelling, reliability as well as software risk management (Srinivasan and Fisher 1995). Figure 2.3 shows an example of perceptron of ANN.
Each artificial perceptron produces the weighted sum of its M inputs, $x_j$, where $j = 1, 2, \ldots m$, and generates an output of 1 if this result is above the defined threshold $u$. Otherwise, an output of 0 will be generated. The obtained formula is shown as Equation 2.2.

$$y = \theta \times \left( \sum_{j=1}^{m} W_j x_j - u \right)$$

(2.2)

In this equation, $\theta$ is a unit step function at 0 and $w_j$ is the synapse weight associated with the j-th input. $U$ is considered as another weight, i.e. $w_0 = -u$ attached to the neuron with a constant input of $x_0 = 1$. Positive weights model excitatory synapses, while negative weights model inhibitory ones. The step function $\theta$ can be realized as a Sigmoid, Gaussian, or Linear function. The Sigmoid function is the most widely used in ANNs.

Architectures of ANN are divided into two categories:

- Feed-forward architecture without loops occurring in the network path.
- Feedback architecture with recursive loops.

Combined architecture is also a possible architecture for ANN. Among the different ANN architectures combined model, the feed-forward back-propagation is the most widely used model (Moløkken and Jørgensen 2003; Huang and Chiu 2009).

In practice, since ANN has been successfully applied to several problems in software engineering, researchers also try to apply and use the advantages of ANN to realize accurate, reliable and flexible software cost and effort estimation. For example, using back propagation learning algorithm on a multilayer perceptron is proposed by Witting and Finnie (1994) to estimate software development effort. In other research, Karunanithi et al. (1992) suggests using ANN techniques such as the feed forward and Jordon networks with expert experiments to estimate software flexibility and reliability. Samson (1997) uses another soft computing technique, an Albus multiplayer perceptron, to estimate software development time and cost.
3) Effort Estimation by Analogy

Analogy, as one of the basic human reasoning processes, is used by almost every individual on a daily basis to solve new problems based upon past experiences. Effort estimation by analogy is also data-intensive because the performance of analogy depends on the availability of a suitable dataset. The process of analogy generally follows the procedure of the case-based reasoning (CBR), while one general CBR procedure comprises a 4-stage cycle (Aamodt and Plaza 1994), as shown in Figure 2.4. The 4-stage cycle consists of:

- **RETRIEVE**: the most similar case or cases to the target problem.
- **REUSE**: the past information and solution to solve the new problem.
- **REVISE**: the proposed solution and to better adapt the target problem.
- **RETAIN**: the parts of current experience in the case-base for future problem solving.

![Figure 2.4: General Cyclical Case-Based Reasoning Process](image)

In the general cyclical CBR process, an initial problem is described as a new case. Following the new case, we can RETRIEVE a case from the previous cases. The retrieved case is combined with the new case through REUSE into a solved case. The REVISE process is then used to test the solution for the new case. Finally, useful experience is RETAINed for future reuse, and the dataset of previous cases will be updated by a new learned case, or by modification of some existing cases.
When applying the analogy technique to effort estimation, an estimate of the effort to complete a new software project is made by analogy with one or more previously completed projects. Therefore, estimating software project effort by analogy can be viewed as an example of a CBR strategy (Walkerden and Jeffery 1999), and can be correspondingly divided into a set of steps:

- Measuring or estimating the values of project metrics for the target project.
- Searching a repository of completed projects for projects similar to the target and selecting one or more projects as source analogues.
- Using the effort value of the source analogue(s) as an initial estimate for the target project.
- Comparing the known metric values for the target and source projects.
- Adjusting the effort estimate in light of the differences between the target and source projects.

Since analogy refers to the relation between the source and the target measured by their similarity, the major focus of analogy-based study is a pattern mapping problem, that is, how to measure the similarity so as to find and reuse similar past cases. Different methods of measuring similarity have been proposed for different measurement contexts. The dominant similarity measurement used in CBR problems is probably the Euclidean distance metric, which is based on the principle of Pythagorean Theorem (Gullberg 1997) to derive a straight line distance between two points in n-dimensional space. For example, as two of the representative analogy-based effort estimation tools, both ESTOR (Mukhopadhyay, Vicinanza et al. 1992) and ANGEL (Shepperd, Schofield et al. 1996) select an analogue for the target project by calculating the Euclidean distance between completed projects and the target and selecting the nearest neighbour.

Overall, software effort estimation by analogy has been demonstrated as a viable alternative to other estimation methods. Walkerden and Jeffery (1999) identified six potential advantages offered by analogy-based estimation:

- It is easy to understand the basis for an estimate.
- It is useful where the domain is difficult to model.
- It can be used with partial knowledge of the target project.
- It has the potential to mitigate problems with calibration.
- It has the potential to mitigate problems with outliers.
- It offers the chance to learn from past experience.

However, considering the general deficiencies of the CBR algorithm (Aha 1991), effort estimation by analogy still suffers some inevitable issues. The major issue is that analogy-based methods are
computationally expensive because they save and compute similarities to all training cases, moreover:

- They are intolerant of noise.
- They are intolerant of irrelevant features.
- They are sensitive to the choice of the algorithm’s similarity function.
- There is no simple way they can process categorical feature values.
- They give little usable information regarding the structure of the data.

Furthermore, Walkerden and Jeffery (1999) also identified four important factors that influence prediction accuracy:

- The availability of an appropriate analogue (Dataset Relevancy).
- The soundness of the strategy for selecting it (Feature Subset Selection).
- The differences between the analogue and targets (Distance Measures).
- The accuracy of the data used (Quality of the Dataset).

2.2.2 Human-Intensive Approaches to Software Effort Estimation

Human-intensive approaches to software effort estimation have been also recognized as a prediction method, namely expert judgment (Boehm 1984). Expert judgment relies on the accumulated experiences of one or more experts in order to directly derive new project effort estimate (Hughes 1996). In contrast to model-based methods, the procedure of expert judgment may appear more “intuitive” to some practitioners. Basically, experts use their experience and understanding of available information about past projects to derive an estimate of a new project. Examples of the information used can be design requirements, source code, software tools available, rules of thumb, resources available, size/complexity of the new functions, data from past projects, or feedback from past estimates.

Unfortunately, some weaknesses inevitably exist when using any human memory-based techniques, for example, past projects can be forgotten, details confused and important factors accidentally ignored (Mukhopadhyay, Vicinanza et al. 1992). Therefore, the main disadvantage of expert judgment is that the estimate is heavily dependent upon the availability of the expert and may be subjective. Moreover, the very nature of expert judgement means that deriving an estimate is not a repeatable process (Mendes, Watson et al. 2002). There may be very limited visibility into the judgment process and factors that the expert considered in developing the estimate, thus making the
estimate itself difficult to accept and even more difficult to document appropriately. Even when it is clear how one project differs from another completed project, it is not always apparent how the differences affect the effort. The proportional-effort strategy used generally in expert judgment is unreliable because project efforts are not necessarily linear, for example, two people cannot produce code twice as fast as one. Extra time may be needed for coordination and communication or to accommodate for differences in ability, interest, and experience.

Nevertheless, the human-intensive approaches are useful when no quantified, empirical data is available (Boehm, Abts et al., 2000a), and a practical and low-cost estimation process has been suggested (Johnson, Moore et al., 2000). Although expert judgment comes with much criticism for the reliance on human memory and the lack of repeatability of such memory-based approaches (Mukhopadhyay, Vicinanza et al. 1992; Mendes, Watson et al. 2002), reports have proven it to be the dominant strategy in software development estimation (Jørgensen 2004; Höst and Wohlin 1997; Moløkken and Jørgensen 2003; Moløkken-Østvold, Jørgensen et al. 2004). In particular, expert judgment is most suitable when experts have relevant experience, knowledge, and an understanding of the current project. Moreover, a variety of tools can be used to assist experts and facilitate judgement. For example, a database of historical information from past projects can be adopted to help experts understand where the current project fits in. Statistical or artificial-intelligence techniques based modelling tools can be employed to assist analysts in finding a similar project or in distinguishing one project from another.

Jørgensen (2005) gives a practical guideline for expert-judgment-based software effort estimation, which provides two golden rules for estimating effort on the basis of expert input:

- Never accept effort estimation based solely on gut feelings.
- Always require justification of effort estimates, so that they can be reviewed.

Following these two golden rules, effort estimate should not be generated from a high-level description of a proposed software system. Instead, the estimation may be derived from either a top-down or a bottom-up analysis of the proposed system’s size and functionality. Meanwhile, the experts can be required to give three predictions: a pessimistic one, an optimistic one, and a most likely guess. The final estimate of effort can then be calculated as the mean of the beta probability distribution, as shown in Equation 2.3.

$$\hat{E} = \frac{1}{6} \times \left(\text{pessimistic} + 4\text{most likely} + \text{optimistic}\right)$$

(2.3)

Furthermore, the effect of expert judgment can be improved by estimating as a group. Doing the
estimation in this way can bring some advantages such as the overestimate from some people may be balanced by the underestimate from others. However, this approach also involves some risks, for example people with stronger personalities may dominate the final estimate (Baron and Greenberg 2002). To reduce potential risks and bias of individual experts, the adoption of the Delphi technique is suggested (Boehm 1984; Chulani, Boehm et al. 1999). Firstly, an estimate form to be filled out with corresponding specification is supplied by a group coordinator. Secondly, experts anonymously and independently use the form to record their estimates for a new project. Thirdly, a summary of the experts’ responses can be drawn on a new form by the group coordinator. At last, the average estimate will be calculated and presented to the group. In general, the experts in the group may be given opportunities to revise their estimates and explain the rational reasoning behind the estimate by anonymously filling out the forms in more iterations. The process can be iterated for as many rounds as appropriate. Since the estimates are made anonymously and independently, no discussion among the experts is taking place during individual iterations. To combine the advantages of free discussion with the advantages of anonymous estimation, group meetings can be held after each iteration, which focuses on discussing large variations in the estimates.

2.3 Existing Approaches to Cost and Effort Estimation for SOA Projects.

As frequently mentioned, the reuse model in COCOMO II can be used to satisfy some simple cost and effort estimation cases for SOA based software development. COCOMO II considers two types of reused code (Boehm, Abts et al. 2000). As the name suggests, black-box code can be reused without knowing the detailed code or making any change to it, while white-box code has to be adjusted with new code or integrated with other reused components before its reuse. Similarly, within the SOA context, there are black-box service, provided by the third party or inherited from the existing or legacy projects, that can be adopted directly, and white-box service, built through migrating or wrapping some reusable business process component, that has to modify the internal of the original component (Grefen, Ludwig et al. 2006; Demirkan, Kauffman et al. 2008). Nevertheless, the significant difference between code-level and service-level reuse is that, even merely taking black-box ones for instance, whether a code-level component is suitable for reuse in the current environment should be still understood and revealed to ensure its dependability by using reverse engineering or re-engineering (Sommerville 2006; Hall 1992), whilst the contractually reusable and loosely coupled service can be identified and mined through service discovery techniques, for example, semantic annotation (Shadbolt, Berners-Lee et al. 2006) and quality of
The approach applying COCOMO II to cost estimation for service oriented computing is attempted by Tansey and Stroulia (2007). In the authors' work of the proposed integrated model for cost/value estimation, merely taking into account the internally created or modified services, the cost is examined by using COCOMO Post-Architecture or Early Design model with the cooperation of project status and user calibration. In the meantime it is acknowledged that, however, the COCOMO II model should be extended to accommodate new characteristics of SOA based development, and fundamentally different development process may be involved in SOA projects considering the declarative composition specifications.

As one of the key areas in software cost estimation domain, size estimation is significant for SOA applications. Santillo has tried to use a Function Point method to measure software in SOA environment (Santillo 2007), through which many issues are also revealed. In addition to the main one, "boundary positioning", the basic issue is that SOA is functionally different from the other software architectures, because the "function" of a service should present a real-world self-contained business activity (Josuttis 2007). Another inevitable issue is that the effort of wrapping legacy code and data to work as services cannot be assigned any functional size. Following the theory of Function Point, Liu et al. (Liu, Xu et al. 2009) use service points to measure the size of SOA based software. Precisely, the entire software size estimation is proposed as the sum of the size of each service point.

\[ Size = \sum_{i=1}^{n} (P_i \times P) \]  

(2.4)

where \( P_i \) is an infrastructure factor with empirical value that is related to the supporting infrastructure, technology and governance processes. While \( P \) presents single specific service's estimated size that varies with different service types. The equation of this approach is straightforward, yet the weakness is that how to calculate \( P \) for varied service points is not elaborated.

An ongoing framework, SMAT-AUS, is designed and developed to determine the scope and estimate cost and effort for SOA projects (O'Brien 2009). Within the framework Service Mining, Service Migration, Service Development and Service Integration are treated as separate SOA project types. For each project type a set of methods, templates and functions are supposed to be available to support the cost and effort estimation work in organizations. Unfortunately, except the
Software Engineering Institute's Service Migration and Reuse Technique (SMART) method (Lewis, Morris et al. 2005) that can be adopted to estimate the cost and effort of identifying and migrating reusable legacy or existing component, currently there is few metrics suitable for the different projects beneath this framework, especially for Service Mining with numerous service discovery and matchmaking techniques (Crasso, Zunino et al. 2008; Fenza, Loia et al. 2008). Umar and Zordan (2009) warn that both gradual and sudden migration would be expensive and risky so that costs and benefits must be carefully weighed. Bosworth (2001) gives a full consideration about complexity and cost when developing Web services, while Liu et al. (Liu, Xu et al. 2009) simply suggest traditional methods for computing the effort and cost for services that have to be build from scratch. Since utilizing solutions based on inter-operable services is part of service-oriented integration (SOI) and results in a service-oriented integration architecture, Erl (2005) gives a bottom line of effort and cost estimation for cross-application integration: "The cost and effort of cross-application integration is significantly lowered when applications being integrated are SOA-compliant."

Moreover, a generic SOA application should be sophisticated and comprise a combination of project types listed above, whereas the Application Development module of SMAT-AUS framework only introduces the main activities and rough cost factors, without identifying the complexity or specifying the procedure for estimation process.

The discussion about cost estimation for SOA implementation also appears in industry circles. Linthicum (2007) outlines some general guidelines for estimating entire cost of an SOA application. After understanding the domain in detail, the calculation of SOA cost can be expressed as a sum of several cost analysis procedures.

\[
Cost\ of\ SOA = (\ Cost\ of\ Data\ Complexity + \ Cost\ of\ Service\ Complexity + \ Cost\ of\ Process\ Complexity + \ Enabling\ Technology\ Solution )
\]

Although a few of detailed information are provided, for example to set Relational as 30%, Object-Oriented as 60%, and ISAM as 80% when figuring Complexity of the Data Storage Technology that is the basic element of Data Complexity, the other aspects are merely suggested to follow similar means without clarifying essential matters. Meanwhile, the notable problem is that, as Linthicum reminds, this approach is not really metrics. Additionally, it is doubtful that Data Complexity, System Complexity, Service Complexity and Process Complexity are sufficient to present the complexity of SOA systems, because distributed processing would be inevitably more
complicated than non-distributed processing and loosely-coupled asynchronous SOA environment should be inherently more complex, especially regarding error handling (Josuttis 2007; Norfolk 2007).

2.4 Summary

Before unfolding the research into effort estimation for SOA implementation, we have reviewed the existing cost and effort estimation techniques which are divided into four types. In particular, this chapter tries to exhaustively present current works on cost and effort estimation for SOA projects. When it comes to the decision of research methodology, we can firstly use Table 2.1 to analyze whether data-intensive or human-intensive approach is suitable for this topic. For a research topic, quantitative research direction is suitable if enough data are available, while qualitative direction is suitable if enough experts are available. Quantitative and qualitative directions can be used together for cross-validation if both data and experts are available.

<table>
<thead>
<tr>
<th>Experts? (Yes)</th>
<th>Data? (Yes)</th>
<th>Data? (No)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experts? (Yes)</td>
<td>Quantitative / Qualitative Research Topic</td>
<td>Qualitative Research Topic</td>
</tr>
<tr>
<td>Experts? (No)</td>
<td>Quantitative Research Topic</td>
<td>Unfeasible Research Topic</td>
</tr>
</tbody>
</table>

Through the literature review, we can find that the data-intensive techniques usually require collecting data from a large number of real software projects. However, SOA is still maturing, and meanwhile data of SOA implementations are often confidential because SOA is tightly related to business processes. Therefore, it could be difficult to collect enough data from industry to do research into effort estimation for SOA implementations. On the contrary, human-intensive approaches would be more feasible in this case. As shown in Table 2.1, the qualitative direction is the only way for a research topic if there is no enough available data but experts are available. For this topic, we can start our work along a qualitative way and towards an expert judgment solution to effort estimation for WSC-based SOA implementations.
CHAPTER 3

Generic Strategy of Effort Judgment: Divide-and-Conquer (D&C) from an Organizational Perspective

According to the previous work of literature review, both academia and industry have published little work related to estimating effort for SOA implementation. In particular, existing approaches cannot satisfy the effort estimation work in service-oriented environment. Therefore, it is necessary to adequately address the problem of effort estimation for SOA implementations, and the first step can aim at one particular implementation type. In this book, for example, we focus on the qualitative effort judgment for WSC-based SOA implementations.

Before unfolding the research into the specific effort estimation work, we must comprehensively understand generic SOA implementations. Considering SOA belongs to one of the distributed system styles, there are two ways (Fox 1981) to get familiar with SOA systems: One is “learning by doing” that requires building more practical systems by ourselves, and another one is “learning by analogy” that draws upon ideas from the considerable experiences in distributed systems in other fields. This chapter adopts the second way that borrows the fruitful work from organization theory area to the research into SOA domain. Benefiting from viewing SOA from an organizational perspective, we can build complexity framework and identify technology independent strategies for SOA implementations. More importantly, the working mechanism of an organization can be used to inspire a generic strategy to investigate effort judgment for SOA implementations.

In this chapter, thinking of SOA from an organizational perspective is justified in difference ways in section 3.1. Section 3.2 introduces several benefits from the inspiration from organization theory to SOA implementation, and then derives the generic strategy of effort estimation for SOA implementation. Section 3.3 explains this D&C based strategy and demonstrates how it works.
particularly for effort judgment for SOA implementation. The full scope of work in this chapter is summarized in section 3.4.

3.1 Think of SOA from the Organizational Perspective

Organizations emerged as early as ancient civilizations appeared. Today, organizations have become indispensable and pervasive components of human beings’ society, for example, from schools to hospitals and from armies to governments. When it comes to the SOA area, we can similarly regard service-oriented systems as organizations that are composed of services. There are three ways of thinking of SOA from the organizational perspective. The first way is to view an SOA system as an organization, the second one is to treat SOA system as a mirror of its corresponding organization, and the third is to parallel the process of SOA implementation with that of organization design.

3.1.1 Viewing an SOA System as an Organization

Viewing SOA systems as organizations is to use the organization concept to cover SOA systems, as shown in Figure 3.1. Under the same umbrella of organization concept, both traditional organizations and SOA systems consist of organizational units. Organizational units in an SOA system are services, while that in a traditional organization are individuals. Furthermore, different organizational units have different skills and play different roles in an organization. For example, composite services play integrative roles in an SOA system, which parallels the responsibilities of managers in a traditional organization.

In fact, there is no single agreement on definition of an organization. Fierce debates about the organization concept are still underway, though theorists have traditionally consented that organizations are collectivities of people who are socially arranged to pursue specific purposes and achieve explicit goals (McAuley, Duberley et al. 2007). However, this classical consensus makes it possible to think of SOA from the organizational perspective due to two reasons.

On the one hand, it is suitable to think of SOA representing organization architecture. The Organization for the Advancement of Structured Information Standards (OASIS) (Service-Oriented Architecture Reference Model Technical Committee 2006) defines SOA as “a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to
produce desired effects consistent with measurable preconditions and expectations.” When it comes to implementation, SOA is used to build up a collection of independent services that can be quickly and easily integrated into different, high-level business services and business processes to create business value and achieve business strategies (Rosen, Lublinsky et al. 2008). To summarize, SOA both in theory and in practice is proposed for organizing services to attain some particular goals. Therefore, SOA can be set under the umbrella of organization theory in terms of the suggestion of traditional organization concept: if the organizing process is about goal attainment, the organization theory could be followed to conceptualize, explain and ultimately guide individuals’ activities that should be united together to achieve desirable, common organizational goals (McAuley, Duberley et al. 2007).

On the other hand, it is reassuring to think of SOA from the organizational perspective. In fact, conceptual challenge might appear when talking of organizations based on having a goal, because the agreement about an organization’s purpose amongst members may not exist. In the SOA area, however, this disagreement issue can be ignored. Within SOA systems, a service is a well-defined unit of functionality realized by a service interface and a service implementation (Papazoglou and Heuvel 2007). A service interface identifies a service and exposes the semantic description of the service’s invocation. A service implementation realizes the work that the service is designed to perform. Unlike people in social organizations, services in SOA do not have mental or psychological attributes. Consequently, services will always obey the control from the “senior management” of the whole SOA system, and may even not be aware of the “organizational goal”.

![Figure 3.1: View SOA System as an Instance of Organization](image-url)
When thinking about SOA organizationally, the blind obedience characteristic of services can naturally avoid the challenge of defining organizations in terms of having a goal while not all members freely agree to that goal (Bakan 2005).

Moreover, SOA system and organization have similar features. Campbell and Craig (Campbell and Craig 2005) identified seven general characteristics of the organizations, as shown in the left column of Table 3.1. We can then list the comparable features of SOA system in the right column.

Table 3.1: Comparison between SOA System and Organization

<table>
<thead>
<tr>
<th>No.</th>
<th>General features of organizations</th>
<th>Corresponding features of SOA systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>They all contain people.</td>
<td>They all contain services.</td>
</tr>
<tr>
<td>2</td>
<td>The people in an organization perform different roles and their continued membership of the organization is dependent upon such performance.</td>
<td>The services in an SOA system perform different units of functionality and their continued adoption in the SOA system depends on such working performance.</td>
</tr>
<tr>
<td>3</td>
<td>The organization has a collective goal to which all members subscribe.</td>
<td>The SOA system has a particular goal to which all services freely agree.</td>
</tr>
<tr>
<td>4</td>
<td>All of the roles, taken together, help the organization achieve its collective goal.</td>
<td>All of the units of functionality, taken together, help the SOA system achieve its particular goal.</td>
</tr>
<tr>
<td>5</td>
<td>Difference tasks are distributed to different individuals according to their expertise, interest or specialism.</td>
<td>Different services undertake different work according to their functional capability and non-functional performance.</td>
</tr>
<tr>
<td>6</td>
<td>There is a clearly defined hierarchy of authority so that each member of the organization is aware of where he or she ‘fits in’.</td>
<td>Whether a service is aware of its contribution to the SOA system or not depends on the mechanism of service composition. The services involved in choreography will understand their positions, timing of operations, and the interaction with other participants. However, the services involved in orchestration even need not be aware of their involvement.</td>
</tr>
<tr>
<td>7</td>
<td>The limits or borders of an organization are usually clearly defined. This means that there is usually no doubt whether a particular person is ‘inside’ or ‘outside’ of the organization.</td>
<td>For the whole SOA system, the limits or borders are indeed clearly defined. In other words, there is no doubt whether a particular service is included in or excluded from the SOA system.</td>
</tr>
</tbody>
</table>

3.1.2 Treating an SOA System as a Mirror of Real Organization

Unlike Conway’s Law (Conway 1968), treating SOA systems as copies of real organizations herein
implies that an SOA system reflects the features of an organization for which the system is implemented. Conway's Law emphasizes the organization who designs a system, which states that “any organization that designs a system will inevitably produce a design whose structure is a copy of the organization's communication structure” (Conway 1968). However, here we emphasize the organization for which a system is designed and implemented. Figure 3.2 illustrates such a sample that a departmentalized company and its SOA based information infrastructure have the same functional structures.

Figure 3.2: A Company and its Infrastructural SOA System

In fact, current information infrastructures for organizations are normally realized by third-party software companies, but the organizations themselves are also closely involved in the activities from requirements analyses to system implementations. Considering the information technology (IT) must be aligned with business when establishing information infrastructure for an organization, any SOA based information system will inevitably copy the features of the organization for which the SOA system is designed and built. For example, services will be grouped functionally to support different departments in a functional structure based organization, while services should be grouped separately according to different product lines in an organization that employs product-based structure; businesses with narrow span of control will result in tall-hierarchy control flows among services, while businesses adopting broad span of control will bring flat-hierarchy control flows
among services.

Overall, benefitting from thinking of SOA from the organizational perspective, no matter treating SOA system as a mirror of real organization or viewing SOA system as an organization, it is possible to use organization concept to comprehend SOA systems, and further borrow the existing work from organization theory domain to SOA environment.

### 3.1.3 Analogue between SOA Implementation and Organization Design

![Diagram of the Pentagonal Process of SOA Implementation/Organization Design](image)

The similarity between SOA systems and organizations is not a coincidence. We can find common ground on a pentagonal process of SOA implementation (Rosen, Lublinsky et al. 2008; Josuttis 2007; Lawler and Howell-Barber 2007; Krafzig, Banke et al. 2004) and organization design (Daft 2009; Burton, DeSanctis et al. 2006; Davis and Weckler 1996; Kates and Galbraith 2007), which is identified through refining the waterfall process of organization design (Burton, DeSanctis et al. 2006). In the pentagonal process, five steps focusing on the **Goal and Strategy**, **Environment and Scope**, **Structure**, **Process** and **Coordination and Control** are executed generally along the clockwise sequence arrowed in Figure 3.3. Meanwhile, each step has influence on as well as is under influence of the other four steps. For example, the goal and strategy together determine the whole process of
organization design or SOA implementation, while they will be refined gradually as the process is unfolded.

1) Goal and Strategy
As mentioned previously, an organization must have a collective goal according to the traditional consensus of organization concept. Although different parts of the organization may have their own objectives, an overall collective goal can be established by aggregating all the separated objectives together. The overall goal is a desired direction that the organization will head. In practice, an organization’s overall goal embodies a set of specified goals, each of which focuses on different aspect of the organization. Daft (2009) distinguishes organization’s goals into official goals and operative goals. The official goals formally define the business, values and outcomes that the organization attempts to achieve, while the operative goals are more explicit and scattered in different facets such as performance, efficiency, innovation and profit.

Goals of an organization introduce the target that the organization wants to pursue, while strategies define how the organization can pursue its target. Therefore, strategies can be treated as the operationalization of organization’s goals (Burton, DeSanctis et al. 2006). Following the analysis of organization’s goals, we can also distinguish organization’s strategies into official strategies and operative strategies. The official strategies are essential plans of actions that can realize the corresponding official goals, for example the cost-leadership strategy or differentiation strategy. On the other hand, the operative strategies will aim at different detailed tasks like how to improve working efficiency or increase product profits. As different tasks may have resource conflict with each other, the strategy set should be carefully balanced.

Goals and strategies are in the first phase of organization design and essentially influence how an organization should be designed. Similarly, the first step of SOA implementation is to identify the business strategies and goals, and we can adopt the technique, namely business value chain, to help identify the specific goals and strategies for concrete SOA projects (Rosen, Lublinsky et al. 2008). Since service-oriented computing emerged from the requirement of addressing the rapid and usually unpredictable changes that modern enterprises are confronting, SOA systems contribute more promises than the traditional software infrastructures. Therefore, common goals and strategies can be extracted among the general SOA implementations.

2) Environment and Scope
The environment is the surroundings of a system, and the system influences and is influenced by its
environment. Meanwhile, the environment is not static but can be changing continuously and dynamically. Generally, there are five environment patterns interacting with any system, including asymptotic variation, interfering variation, periodic variation, phase-transition variation, and random variation (Peng, Liu et al. 2009).

Both SOA systems and organizations cannot be isolated from their external environments. The environment surrounding an SOA system or organization has a set of factors relating to resources or vulnerabilities. For example, the suppliers, customers, competitors, culture and government are organizations’ environmental factors, while the developers, users, legacy system, existing service pool and state of current technology are SOA systems’. Building organization and implementing SOA are highly dependent on the environmental factors. In practice, the number of factors that constitute environment might be considerable. All these factors together reflect the boundary that an organization or SOA system, and then outline a scope, which determines the capability, applicability, competitive advantages and business range for the organization or SOA system.

For organization design, environment restricts organizations within certain scopes, and further influences their processes, structures and controls. For SOA implementation, analyzing the external environment and determining the applicable scope are particularly significant. SOA-based software infrastructure is supposed to be adaptive within an increasingly changing and complex environment. However, the loosely coupled asynchronous SOA systems are inherently more complex than the traditional architecture based systems. Josuttis (2007) has pointed out that distributed processing would be inevitably more complicated than non-distributed processing, and any form of loose coupling increases complexity. In practice, building a true heterogeneous SOA for a wide range of operating environments may take years of development time if the company does not have sufficient SOA experience and expertise (Jamil 2009). Since the more complexity involved in a system, the more difficulty the designers or engineers have to understand the implementation process and thus the system itself (Cardoso 2005), SOA should be adopted only in the suitable environment and only when its benefits outweigh any extra costs due to the increased complexity.

3) Structure
Structures of both organizations and SOA systems are established to divide up work into manageable and measurable units with clear responsibility boundaries. Organization’s structure is normally a hierarchy that allocates roles, power, authorities and responsibilities, and determines working relationships and communication channels. Generally, organizational units are arranged around functions, products/services, customers/geographies, or business processes. Therefore, the
organization structures can be typically divided in five basic styles: functional, product- or service-based, customer- or geographical area based, business process based, and matrix structure (Davis and Weckler 1996). Each kind of structure has specific advantages and disadvantages. Unsuitable organization structure will result in formidable obstacle to align the other design elements with the organization’s strategy (Kates and Galbraith 2007). Consequently, large organizations always build hybrid structures to achieve the combination of the advantages.

SOA systems normally adopt matrix structure, which simultaneously groups services in two directions: functional direction and business process based direction, as shown in Figure 3.4.

The functional direction is to classify services according to the type of logic they encapsulate. Although there are quite a few service classifications we can identify from literature, most of the existing classifications can be unified and layered as Basic Services, Business Entity Services, Process-centric Services, and Enterprise Services. The Basic Services, settled at the bottom layer of SOA systems, provide reusable, technical, and foundational functionalities. The Business Entity Services represent the entities in business activities, such as employee, customer, contract, and product. Through composing relevant Business Entity Services, a Process-centric Service encapsulates a sequence of activities to complete a specific business task. The Enterprise Services provide endpoints to access the corresponding SOA systems, which could have less reuse potential but enable cross-enterprise integrations.
On the other hand, to obtain the reuse of services in multiple business processes, the landscape of different services within different business process contexts should be described to show how the services work together (Rosen, Lublinsky et al. 2008). The business process based direction of SOA systems’ structure can then be outlined. Along the business process based direction, the overall services are grouped according to different roles and responsibilities within the real business. Each group may contain from an individual service to multiple services. Moreover, the relationships to each other among the groups and the places of these groups in the business processes are also described.

4) Process
The organization design has been viewed from an information processing perspective (Galbraith 1974). The term process herein means not only business processes that produce products or services for customers, but also non-profit routines that constitute organizational actions. Generally, a process in an organization is a series of connected activities that transfer and transform information and resources through the organization (Kates and Galbraith 2007), for example approving an application, submitting a report, and managing work progress. From the mid-1990s, many modern enterprises began to evolve into process-focused organizations in order to achieve higher performance and survive the market competition (Seltsikas 2001). Currently, process-focused organization has become the new organizational form with business process as the core concern. As a result, the design of processes significantly impacts on how well the organizational goals can be achieved.

Moreover, processes in an organization have close relationship with the organization’s structure and coordination. All kinds of organization’s structures inevitably create barriers to collaboration, because boundaries will appear as soon as organizational units are grouped under the structure. To fulfill the collaborative targets in organization, however, processes are required to flow cross the boundaries. Therefore, processes can force the related organizational units to work together.

Like process-focused organizations, SOA systems can also be regarded as process-focused systems comprising such as management process, coordination process, and traditional work process. Among all kinds of processes, SOA instinctively concentrates on business process. Essentially, SOA is aligned with business process management (BPM) in business firms in which the criticality of business processes is concerned (Lawler and Howell-Barber 2007). The emphasis of SOA is the functional infrastructure and the business services instead of the technical infrastructure and technical services. A business service encapsulates a piece or an entity of a business process. When
implementing SOA, it is crucial to analyze business processes before identifying and developing services (Rosen, Lublinsky et al. 2008). Following the analysis of business process, those potentially and even partially suitable services should be identified first. These existing services provide constraints that frame the future SOA system as much as possible. The business processes are then broken down into business pieces that can be implemented by developing new services.

5) Coordination and Control

The coordination problem is one of the central topics in organizational studies (Heath and Staudenmayer 2000). As mentioned previously, individual actions of large numbers of interdependent roles and specialists must be coordinated to constitute processes to fulfill global tasks in an organization. On the other hand, the coordination will increase organizations’ information processing capabilities when encountering increasing amount of uncertainty (Galbraith 1974). In practice, the activity of coordinating overlaps the activity of controlling, because the appearance of coordination usually implies the occurrence of some control (Davis and Weckler 1996). To coordinate and control organizational work, organizations should adopt suitable techniques and mechanisms. Unfortunately, there is not a fixed prescription of methods for coordinating and controlling work. The coordination and control, for example, can be simply related to the structures (Kates and Galbraith 2007), be utilized by goal setting, hierarchy, and rules (Galbraith 1974), or be executed by using four basic techniques: Supervision, Standardization, Building employee commitment, and Teams (Davis and Weckler 1996). However, the principle of these techniques and mechanisms is uniform: to make sure organizational units work appropriately and find out to what extent they are reaching the goals and targets.

When implementing SOA, services must be composed to fully realize the benefits of SOA (Sarang, Jennings et al. 2007), which also relies on the coordinating and controlling activities. According to the cooperation fashions among component services, the mechanism of coordination and control can be distinguished between orchestration and choreography.

Orchestration, as shown in Figure 3.5(a), describes and executes a centralized process flow that normally acts as an intermediary to the involved services. The central intermediary explicitly specifies the business logic and controls the order of invocation of services. As a result, the intermediation defines a long-term, cross-organization, transactional process. The involved services, on the other hand, do not need to be aware of whether they are involved in an orchestrated process. Orchestration represents coordination from the perspective of a single participant that can be another composite service.
Choreography, as shown in Figure 3.5(b), describes multi-party collaboration and focuses on the peer-to-peer message exchange. The collaboration is decentralized, which means that all participating services work equally and do not rely on a central controller. Each service involved in choreography knows exactly its contribution to a business process: operation, timing of operation, and the interaction with other participants. Choreography represents collaboration from a global perspective.

![Service Orchestration and Choreography](image)

**Figure 3.5: Service Orchestration and Choreography**
3.2 Benefits from Thinking of SOA from an Organizational Perspective

3.2.1 Towards Technology Independent Strategies of SOA Implementation

When it comes to information system architecture, service-orientation establishes a universal model in which functionalities and business logics are cleanly partitioned and consistently represented. Therefore, in addition to the various targets that satisfy the business requirements of respective SOA projects, SOA implementations possess general promises and goals. The motivations and expectations of the people who are engaged in SOA activities can be used to empirically and efficiently assess the universal goals of SOA. With reference to a 2006 survey conducted by the Cutter Consortium (Rosen, Lublinsky et al. 2008), we can identify the most common and general goals of SOA are agility, flexibility, reuse, data rationalization, integration, and reduced costs. In fact, plenty of technical strategies have been developed to help realize SOA’s goals. Examples of technology based strategies are standard service contract that facilitates integration, loosely coupling that supports flexibility, and service autonomy that provides reliable and predictable performance. As supplementary, here we focus on the technology independent strategies of SOA implementation that are inspired by existing research into organization design.

*Strategy 1: Use Total Quality Management (TQM) to Accommodate to Changing Environment*

After the first introduction by Dr. Deming in the late 1950s, TQM was conceptualized as an appropriate resource to promote organizational innovation. Innovation has been widely recognized as a competitive instrument essential for organizations’ long-term success and survival. In other words, organizations can use innovation to adapt and fit the changing conditions of environment like technology and market (Santos-Vijande and Alvarez-Gonzalez 2007). Therefore, TQM can be employed as a strategic guidance to instruct organizations to be aligned with their environment. Just as the name implies, TQM is a holistic level management for quality, because it can be achieved only if the total quality concept is utilized from the acquisition of resources to the customer satisfaction (Kaynak 2003).

In the discipline of SOA, the quality management has also been emphasized to satisfy the unique characteristics of service-oriented computing. Nevertheless, to the best of our knowledge, existing research into quality management in SOA area is mainly at the service level, which is limited around the Quality of Service (QoS). The overall QoS of an SOA system is determined by all the
QoS of component services who compose the SOA system (Yau, Ye et al. 2008). Based on the QoS management, SOA systems generally replace component services with higher quality services to realize adaptations. Hence, the focus of QoS management is on individual services in an SOA environment.

When applying TQM to SOA domain, Deming’s 14 points (Walton 1988) can be used as a framework to guide SOA implementations. For example, service suppliers and SOA system users should be taken into account when measuring the total quality of an SOA implementation. Here we focus on the quality of interaction and cooperation process among services. With reference to the explanation of TQM by Deming, in any circumstance, processes should be constantly analyzed to determine what changes can be made to bring improvement. Therefore, TQM introduces a new angle of view to SOA systems when adapting environment. However, employing TQM does not indicate abandoning QoS management. There is no conflict between TQM and QoS management. On the contrary, they are two complementary approaches for SOA to accommodate the changing environment: (1) TQM can be used to adjust the process of interaction and cooperation among services. (2) QoS management can be used to switch services based on the latest quality requirement.

**Strategy 2: Keep the Structure as Flat as Possible**

In an organizational hierarchy, the number of levels typically increases along with that the organization grows and becomes more complex. However, every level in a hierarchy will inevitably involve more operating costs (George and Jones 2007). Moreover, an organization with a higher hierarchy may come with longer decision making chains and slower responsiveness to customers. Therefore, organizations can increase efficiency by keeping their structures as flat as possible. Furthermore, flat structure can decentralize responsibility and control to lower-level employees to take greater advantage of the skills and experience of organization members.

In an SOA system’s hierarchy, the number of levels can increase along with the growing cascade of service composition. If some business logic involves several business pieces that reside in different services, it can be realized by composing these services and exposed as a new composite service. In general, a composite service is recursively defined as an aggregation of elementary and composite services. When thinking of SOA from the organizational perspective, composite services play integrative roles in an SOA system. In organizations, an integrative role is a full-time manager who is in charge of orchestrating work across units (Kates and Galbraith 2007). These managers have accountability for results but are not directly responsible for the resource achieving and specific
work that should be accomplished by staff. If the organizational structure is flattened, the number of integrative roles will be reduced. The similar scenario of SOA implementation can be simply illustrated in Figure 3.6.

![Diagram of Tall Structure vs. Flat Structure](image)

**Figure 3.6: Tall Structure vs. Flat Structure for Implementing the Same Business Process**

Figure 3.6(a) shows a tall structure example of some business process implementation by inserting two composite services that respectively compose two of the elementary services. Suppose only the three elementary services are available at the beginning, we can flatten the structure and move the additional functionality of original composite services upward, as shown in Figure 3.6(b), to reduce the composition cost and simplify the complexity of the business process implementation. However, we should keep the tall structure if the composite service already exists or its reusability is to be achieved. Therefore, when applying this strategy, the value and cost should be well balanced to determine the extent of flattening structure.

**Strategy 3: Take Measurements at Interim Steps in Process**
When generating products following certain working processes or designing the working processes themselves in an organization, the outcomes are always measured. On the other hand, it is also valuable to take measurements at interim steps in processes. The research and practice in organizations during the past decades reveal that it is increasingly important to ensure the work finishes properly the first time instead of having to be redone (Davis and Weckler 1996). Consequently, the inspections and measurements can be applied to different steps in processes to save the cost of rework and avoid flaws in the end product.

When applying this strategy to SOA implementation, the inspiration is to confirm the individual work of each service in processes. The idea behind this strategy is to clearly define connected subtasks in a process, and specify and measure the result of each subtask. It should be noted that measuring interim task mainly concerns the result rather than how the task is performed. Considering a service is such an entity that performs some task while hiding technical details, we can use the interim task measurement to help identify the most suitable services. Once all the services are determined, the relevant business process can then be correctly implemented.

**Strategy 4: Build Business Process Teams to Facilitate Coordination and Control**

In organizations, teams are cross-functional structures that bring people outside the scope of traditional departments to work together and share collective responsibility for special and complex assignments. A business process team is established around one business process and includes people who can collectively perform all the major activities to carry out the business process from beginning to end. Business process teams exist as a characteristic of “horizontal organizations” (Davis and Weckler 1996). Horizontal organizations instinctively tend to flatten their structures by focusing on all the units involved in completing certain work, rather than the coordinate activities relying on a vertical hierarchy. As a result, horizontally established business process teams have advantages of reduced management costs and less need for coordination. An organization can build different teams for different business processes. Meanwhile, team members can still remain in their conventional department positions, and simultaneously join different business process teams according to their knowledge and skills.

Building business process teams in an SOA system should be a virtual division without many real actions. In other words, all the services involved in a business process logically constitute a team without changing the existing structure of the SOA system. Through virtual business process teams, the focus of coordination and control can be balanced between inward IT and outward business during SOA implementations. Furthermore, considering one service can be involved in different
business process teams like the same scenario of organizational teams, we can identify and scale services’ dependency of business processes in an SOA system. The more dependency a service has, the more carefully it should be controlled especially when planning to change or adapt this service. Additionally, the philosophy behind business process team is that all the team members must understand the whole working process, and how they should contribute and cooperate with each other to achieve the final target. Therefore, virtually building business process teams may facilitates the choreography mechanism in particular among involved services when implementing SOA.

**Demonstration of Applying Technology Independent Strategies**

Here we use a practical case to demonstrate how the technology independent strategies can be applied to and improve an SOA implementation. Imagine there is an SOA-based application in a travel agency, as illustrated in Figure 3.7. The travel agency books hotel through BPay on behalf of a group of tourists, and will rent a car by money transfer if the number of the tourists is more than ten. Suppose there are three online banking services, two hotel booking services and one car rental service. Each service can fulfill its corresponding business function, while Online Banking Service B1 and Hotel Booking Service H1 are selected according to their reliability and response time. Moreover, the business rule “rent a car by money transfer if the number of the tourists is more than ten” is encapsulated in a composite service composed by Hotel Booking Service H1 and Car Renting Service. The composite service is implemented as a Reinforced Hotel Booking Service.
following the technology based strategy of using service composition to “fulfill a large extent of future business automation requirements” (Erl 2007). After applying the four proposed strategies, this travel agency application will evolve as shown in Figure 3.8.

**Apply Technology Independent Strategy 1.** After a period of operation, the travel agency receives many complaints from small groups of tourists about inconvenience without cars. Hence, the travel agency decides to change the business rule into “rent a car by money transfer if the number of the tourists is more than five”. When applying TQM to the SOA system to check the cooperation among services, we can find that the invocation of Car Rental Service is inflexible because old business rule is hardcoded in the Reinforced Hotel Booking Service. Therefore, the number of tourists should be set as a variable and exposed as an input parameter of the composite service. The operation of this composite service is then adjusted by accepting one threshold parameter to improve the flexibility of the SOA system.

**Apply Technology Independent Strategy 2.** When analyzing the structure of the travel agency’s SOA system, we find that the Reinforced Hotel Booking Service does not have any reuse opportunity. Furthermore, the encapsulated business rule can be easily transformed into control flow logic of invoking two component services, and moving the control flow logic into upper business logic will have little increase complexity for the latter. Therefore, we can flatten the structure as Figure 3.6(b) by removing the Reinforced Hotel Booking Service to reduce the service maintenance effort.

**Apply Technology Independent Strategy 3.** Suppose both BPay and money transfer will result in commission charges, and the charges vary depending on different bank and different time. We can then use the criterion “choose bank with the lowest commission charges” to constantly and simultaneously measure the three candidate online banking services. The service of the bank that charges the lowest fee will be dynamically employed by the SOA system to help the travel agency save money.

**Apply Technology Independent Strategy 4.** Based on the business logic behind the travel agency application, we can identify there are two atomic business processes: one is hotel booking through BPay, and another is car rental through money transfer. Consequently, the selected Online Banking Service, Hotel Booking Service and Car Rental Service can be logically grouped into two business process teams. The coordination and control among services in the hotel booking business process team follows the BPay rules, while in the car rental business process team obeys the money transfer
rules. Through the team building, we can naturally arrange different cooperation for services in different teams, and in this case we may further identify the Online Banking Service is the key service when implementing the SOA system.

3.2.2 Towards SOA Implementation Complexity Measurement

Based on the definition of organizational complexity (Dooley 2002), it is not difficult to reformulate the concept of complexity of SOA implementation as the amount of differentiation that exists within different facets affecting the SOA implementation. Scherrer-Rathje et al. (2009) have revealed that different facets of complexity need to be analyzed differently. Inspired by the existing research into organizational complexity (Dooley 2002; Goold and Campbell 2002; Hornby 2007; Peng, Liu 2009; Peng 2001; Siahpush 1991; Efstathiou, Calinescu et al. 2002), we can identify four different dimensions that should be analyzed separately when measuring the complexity of SOA implementation: Structure, Environment, Business and Resource. The four dimensions then constitute a framework for the complexity measurement of SOA implementation, as illustrated in Figure 3.9.
Figure 3.9: Complexity Measurement Framework for SOA Implementation

1) Structural Complexity
Structure is the fundamental content when understanding any type of system, including organizations and SOA systems. Goold and Campbell (2002) suggest that the organization is a structured network that has features of both network and structure. The network is woven by using largely self-managing units, and meanwhile sufficient structure and hierarchy are necessary to ensure that the responsibilities and relationships are clear, and that the collaborations among units are successful. Similarly, SOA is supposed to adopt coarse-grained and self-contained services to build structural and hierarchical business processes. Therefore, structural complexity is the first and important facet that influences SOA implementation.

Generally, structural complexity is closely related to the elements within the structure and the interconnections among the elements. The density of interconnections depends on the number of elements and their diversity, while the number of elements also drives the diversity to some extent (Dooley 2002). In other words, the more the elements and their types, the more complicated the interconnections, and the more complex the structure. For SOA systems, more complex structures inevitably require more effort on cooperation and collaboration among services. For instance, an SOA-based online shopping platform is more complex than one online shop on the platform, because the shop only needs to invoke part of services provided by the platform while the platform is constituted through interconnecting more services and more types of services. Measuring the structural complexity of SOA implementation can borrow Hornby’s proposal (Hornby 2007) that
takes into account modularity, reuse and hierarchy characteristics when measuring the structural complexity of evolutionary design systems.

2) *Environmental Complexity*

As mentioned previously, both an SOA system and organization are systems sharing some common features, and both cannot be isolated from external environment. The environment is the surroundings of a system, and the system and its environment may impact on each other. In reality, the system’s environment will always be more complex than the system itself. However, the system’s complexity can be considered as a response to the complexity of the environment, and be measured through differentiation (Dooley 2002). In fact, SOA emerged from the requirement of satisfying the increasingly changing business environment, such as expanded market, growing competitors, and switched government policies. The complex environment normally requires complex SOA systems within the business institutions, which unavoidably results in complex SOA implementations. For example, the SOA implementation for some international business will be more complex than that for the same scale of local business, because different social, cultural, political and information infrastructural factors must be also taken into account when implementing SOA for international business. On the other hand, the robust adaptability of the SOA systems should be emphasized when facing frequently and rapidly changing environment, which also contributes complexity to the SOA implementation.

Peng et al. (2009) summarize five environment variations that differently impact on the organizational change, containing asymptotic variation, interfering variation, periodic variation, phase-transition variation, and random variation, which can also be used to inspire the research into environmental complexity of SOA implementation.

3) *Business Complexity*

The ‘business’ here implies not only commercial processes but also non-profit routines that constitute organizational actions and that can be supported by corresponding SOA systems. Dooley (2002) treats business as an internal environment that makes organizations internally differ in complexity. Similarly, the business radically determines the implementation complexity of the corresponding SOA system. For example, building a stock transaction system will be absolutely more complex than establishing an online pizza-order system, because the stock trade requires more sophisticated calculation, more complicated procedure and more trading rules than ordering food.

Business complexity is closely associated with the structure of the business. There is no denying
that increasing calculation brings increasing data flows, complicated procedure implies large-scale business process, and more rules result in more control flows. Therefore, there is a tight correlation between business complexity and structural complexity within an SOA implementation. The measurement for structural complexity can be switched to satisfy the measurement of business complexity. Furthermore, different implementation strategies for the same business may lead to varying structural complexities. Holding constant business complexity, adopting coarser grained services will reduce the number of services and interconnections when implementing SOA, which can correspondingly decrease the structural complexity. However, we cannot infinitely reduce the structural complexity by continuously scaling up services, because in turn the larger scale services may spoil the flexibility of SOA system when experiencing dramatic environment changes.

4) Resource Complexity

Resources are essential components and play significant roles in organizations. The management research field has emerged one key development that is to look at organizations in the Resource-Based View (RBV) (Peng 2001). In RBV, the organizations are viewed as a bundle of assets and resources that can create competitive advantages if the resources are employed in proper ways. By separating workers from the general resource concept, Siahpush (1991) suggests that the organizations are composed by resources (or vulnerabilities) and actors including employees and employers. Although the resource concept is broad and may be intangible in RBV, the suggestion of composite organization can be migrated to view SOA systems as combinations of resources and services from the organizational perspective. Inspired by Efstathiou et al.’s work (Efstathiou, Calinescu et al. 2002), we can use tangible resource related metrics to entropically measure the complexity of SOA implementations. To simplify the calculation, the resource complexity can be quantitatively measured with the amount of resource requisitions when implementing SOA, for example, the external storage capacity, the response time of transaction, and the message size of communication. External storage for example due to data backup will incur manipulating data synchronization and employing more facilities; the limit of response time requires stable network and fast calculation; large message size may result in additional CPU load and peak memory utilization. Moreover, the involved personnel that are treated as actors in organizations can be also viewed as resource in the SOA environments, because services will play actors’ roles when organizationally observing SOA.
3.2.3 Towards Effort Judgment for SOA Implementation

Mechanistic organization, studied by Burns and Stalker (Campbell and Craig 2005), has a rigid management system. With a strict hierarchy of authority, generally, a mechanistic organization uses multiple levels of management to insure proper and centralized decision making. In daily routine, the information and decisions are propagated through each level of the hierarchy. Meanwhile, the tasks in mechanistic organizations are precisely defined and broken down into separately specialized parts. These separate parts will be allocated to members who are the most suitable for them. The individual members are then coordinated through the management system to achieve global goals. Consequently, the hierarchical structure of a mechanistic organization can be viewed as a global goal tree: the global goal is stepwise decomposed into a set of sub-tasks, while the goal is achieved through recomposing the sub-solutions to those sub-tasks. The notion of decomposition and recomposition directly engenders D&C approach that allows the organization to use larger groups of work units more efficiently and address problems on a larger scale (Yadgar, Kraus et al. 2003). Likewise, SOA systems can also be thought of a D&C way to achieve business goals. When building SOA systems, we can therefore follow the D&C way to analyze and judge the effort.

Since the D&C strategy of effort judgment for SOA implementation is the essential notion among the inspirations from organization theory in this book, the content of this part is separately elaborated in section 3.3.

3.3 D&C Approach to Effort Judgment for SOA Implementation

3.3.1 Brief Background of D&C Approach

As stated in section 3.2.3, D&C can be adopted to analyze and judge the effort when implementing WSC-based SOA systems. What is the detailed effort judgment process and why does it work? It is then necessary to introduce some background of the D&C approach.

The history of D&C method can be traced back to as early as 200BC (Knuth 1998), when the Babylonian reciprocal table of Inakibit-Anu was used to facilitate searching and sorting numerical values. However, the first description of the D&C algorithm appears in John Mauchly's article discussing its application in computer sorting (Knuth 1998). Nowadays, the D&C approach is applied widely in areas such as Parallel Computing (Bai and Ward 2007), Clustering Computing
Chapter 3: Generic Strategy of Effort Judgment: Divide-and-Conquer from an Organizational Perspective

(Khalilian, Boroujeni et al. 2009), Granular Computing (Lin 2005), and Huge Data Mining (Hu and Wang 2008).

The principle underlying D&C is shown in Figure 3.10. That is to recursively decompose the problem into smaller sub-problems until all the sub-problems are sufficiently simple enough, and then to solve the sub-problems. Resulting solutions are then recomposed to form an overall solution. Adopting the principle, the D&C procedure will lead to different subroutines for different sub-problems. Normally, some or all of the sub-problems are of the same type as the input problem, thus D&C procedure can be naturally expressed recursively. The QuickSort (Knuth 1998) algorithm is also such a procedure.

![Figure 3.10: Principle of Divide-and-Conquer](image)

The advantages of applying the D&C approach to suitable problems are multifold, and can be classified as the following:

- **Structural Simplicity**: Profiting from perhaps the simplest structuring technique, D&C has been identified as a high prior strategy to resolve problems not only in the computer science field but also in politics and sociology fields. No matter where the D&C approach is applied the solution structure can be expressed explicitly in a program-like function such as:
Chapter 3: Generic Strategy of Effort Judgment: Divide-and-Conquer from an Organizational Perspective

\[
Solution(x) \equiv \\
IF \ IsBase(x) \\
THEN \ SolveDirectly(x) \\
ELSE \ Compose(Solution(Decompose(x)))
\]

(3.1)

Where \( x \) is the original problem that will be solved through \( Solution \) procedure. \( IsBase \) is used to verify whether the problem \( x \) is primitive or not, which returns TRUE if \( x \) is a basic problem unit, or FALSE otherwise. \( SolveDirectly \) presents the conquer procedure. \( Decompose \) is referred to as the decomposing operation, while \( Compose \) is referred to as the composing operation.

- **Computational Efficiency.** D&C is frequently used for designing fast algorithms. In appropriate application scenarios, the D&C approach leads to asymptotically optimal cost for solving the problems. Assume a problem of size \( N \) can be broken into a bounded number \( P \) of sub-problems of size \( N/P \) step by step, and all the basic sub-problems have constant-bounded size. Then the D&C algorithm will have \( O(N \log N) \) worst-case program execution performance. Normally, the consequence is more flexible because the size and the number of tasks can be decided at run-time.

- **Parallelism.** Since sub-problems in the individual division stage are logically and physically independent, the D&C approach can be naturally executed in parallel procedures. For computing problems, D&C is suitable for application in parallel machines due to not only the independent problem grains but also the efficient use of cache and deep memory hierarchies (Zhang and Xue 2009). In fact, D&C has been considered as one of the well-known parallel programming paradigms.

- **Capability of Solving Complexity.** Through breakdown of an overall goal into smaller and independent sub-problems, the D&C strategy provides adaptation scalability and variability, and is frequently used in the areas of engineering to reduce and manage complexity. Those complicated cases, such as resolutions for conceptually difficult problems, and approximate algorithms for NP-hard problems, are usually based on the D&C principle.

Given these merits, D&C can be considered a suitable and effective approach to accommodate complex problems such as effort estimation for SOA-based software development, where individual measures must be carried out independently. The following sections discuss its applications in SOA effort estimation.
3.3.2 Effort Estimation for Generic SOA Implementation

When it comes to the effort estimation for SOA implementation by using D&C, the first problem is how we can divide an SOA system. The advantages of SOA are mainly reusability and composability with an emphasis on extensibility and flexibility, at a high level of granularity and abstraction. In other words, SOA-based software can be naturally divided into a set of loosely coupled services. These services can then be classified through their different features. Krafzig et al. (2004) has identified that distinguishing services into classes is extremely helpful when properly estimating the implementation and maintenance cost, and the cost factors may vary depending on the service type. However, there is not a standard way to categorize services. Service classification can be different for different purposes, for example differentiating services according to their target audience (Josuttis 2007), categorizing services through their business roles and responsibilities (Erl 2005), and classifying services by using their background techniques and protocols (Davies, Schorow et al. 2008). As we focus on the development process, services in our work are distinguished as follows:

- **Available Service** (basic service type), is the service already existing i.e. is it provided by a third party or inherited from legacy SOA based systems.
- **Migrated Service** (basic service type), is the service to be generated through modifying or wrapping reusable traditional software component(s).
- **New Service** (basic service type), is the service to be developed from scratch.
- **Composite Service**, is the service any composition of above three types of basic services or other composite services.

Through this type of classification, four different development areas are identified in SOA projects. These areas present both a decomposition process that results in Service Discovery, Service Migration, and Service Development, and a recomposition process that is Service Composition. The effort estimation for overall SOA-based software development can then be separated into these smaller areas with corresponding metrics. Therefore, the D&C approach is a feasible attempt for SOA-based software cost estimation following this development oriented service classification.

The notion of the effort estimation approach for generic SOA implementations closely follows the D&C principle. Firstly, through the service-oriented analysis, the SOA project is divided into basic services recursively. Secondly, different sets of metrics are adopted to satisfy the cost and effort estimation for different service development processes. The total cost and effort of the SOA project will be calculated through the service integration procedure as shown in Figure 3.11.
Figure 3.11: Procedure of D&C based Effort Estimation for SOA Projects

Where E1 is the cost estimation model or software size measurement used to accomplish modelling or sizing work for discovering available services, E2 represents migrating potential services, E3 represents developing new services, and E4 is the cost estimation model or size measurement for calculating the service integration effort. As for the Decomposability condition, particularly, it depends on the design and real situations whether the current service should be further divided or developed as a whole. The procedure in Figure 3.11 presents the generic process of SOA cost estimation using the D&C method. The process can lead to both a model tree if applying D&C to modelling the development cost estimation for SOA-based software, and a sizing tree if applying D&C to measuring the size of an SOA-based application. To calculate the ultimate cost and effort, the predicted size should be combined as a parameter with the estimation model.

To precisely describe the D&C based effort estimation for SOA-based software development, the complete process can be expressed in the following pseudo code. Here we define the stage where that service division occurs as the service levels, and the composite service stands in a higher level next to its successive component services.

As shown in Table 3.2, the SOA project itself is treated as the highest-level coarse-grain service, which is also the initial input parameter of SoaEffortEstimation function. Within the body of SoaEffortEstimation function, the effort of the input service development will be estimated directly.
if the service belongs to those three basic types, or recursively calculated by analysing and composing the cost and effort of the development for component services. When composing individual service development effort into the overall SOA-based software development effort, the strategy of supposed service composition is progressed level-by-level instead of integrating the services all at once. The reason of adopting such a strategy is that, according to our work, service composition occurring in different levels will make different contributions to the total cost and effort of the project development.

**Table 3.2: Algorithm of D&C based Effort Estimation for SOA Projects**

```
1) //Treat the project as the highest-level service S to analyze.
2) double SoaEffortEstimation(service S) {
3)    double effort = 0;
4)    Determine the type of S according to the design and real situations.
5)    switch (the type of S) {
6)        case AVAILABLE:
7)           effort += The effort of service discovery;
8)           break;
9)        case MIGRATABLE:
10)           effort += The effort of service migration (service wrapping);
11)           break;
12)        case NEW:
13)           effort += The effort of service development;
14)           break;
15)        default:
16)           Divide S into component services at lower level.
17)           foreach component service in S
18)           effort += SoaEffortEstimation(component service);
19)           effort += The effort of service integration for component services
                in S;
20)           break;
21)    }
22)    return effort;
23) }
```

A real example can be used to show the application process of the D&C based effort estimation for SOA implementation in practice. We employ the RailCo Ltd. case study presented in (Erl 2005). There are two reasons for choosing this case: The RailCo Ltd. case study characterizes all the service types listed in the previous section, and there are a limited number of services, which are adequate for illustrative purposes in this book.
RailCo Ltd. is a railway parts supplier company specializing in air brakes and related installation tools. To improve the working efficiency of this company, a service-oriented analysis was conducted, which decomposed the business process logic into a series of service candidates. RailCo Ltd. revealed the requirements of two business services in higher level and four application services in lower level. The redesigned automation system is represented in Figure 3.12 following current disciplines:

1) Legacy System Service is migrated from the previous project.
2) Polling Notification Service and Transform Service are new services that should be developed from scratch.
3) Metadata Checking Service is an available service provided by a third party.
4) Invoice Processing Service and PO Processing Service are both combined services containing all or some of above basic services.

The procedure of cost and effort estimation for developing this redesigned service-oriented project is illustrated in Figure 3.13, while the detailed steps are elaborated as follow.
1) Divide the Automation System into an Invoice Processing Service and a PO Processing Service. 
2) Divide the Invoice Processing Service into its four basic component services. 
3) Estimate the cost and effort of discovering the available Metadata Checking Service by using corresponding metrics E1. 
4) Estimate the cost and effort of migrating the Legacy System Service by using corresponding metrics E2.
5) Estimate the cost and effort of developing the Polling Notification Service and Transform Service by using corresponding metrics E3.
6) Estimate the cost and effort of integrating the above four component services into the Invoice Processing Service by using corresponding metrics E4.
7) Divide the PO Processing Service into its two basic component services.
8) Notice that Legacy System Service and Transform Service have both been taken into account.
9) Estimate the cost and effort of mining the Legacy System Service and Transform Service by using corresponding metrics E1. Since these two services are in the same project and can be directly identified, the cost and effort here can be treated as zero in this special case.
10) Estimate the cost and effort of integrating the above two component services into the PO Processing Service by using the corresponding metrics E4.
11) Estimate the cost and effort of integrating the Invoice Processing Service and PO Processing Service into the Automation System by using the corresponding metrics E4.
12) Sum up all the estimation results to calculate the total cost and effort of the Automation System development.

Through the demonstration of the RailCo Ltd. case, the D&C approach is proven helpful for simplifying and regulating the SOA-based software effort estimation. Moreover, all the simplified effort estimation problems are independent enough to be solved in parallel. The uniform and explicit working procedure based on D&C is then a feasible attempt to service-oriented software cost and effort estimation.

Note that this D&C strategy is generic for effort estimation for any type of SOA implementation, including the WSC-based SOA implementations. Since a WSC-based SOA implementation only comprises WSC-related human activities, we can simply suppose that the pre-defined basic services are all available. As such, our concerns about the effort of a complete WSC-based SOA implementation can be narrowed down to that of individual WSCs.

3.4 Summary

SOA appears as a concept for large distributed systems (Josuttis 2007). When it comes to the research into distributed systems like service-oriented system, there are generally two methods: one is “learning by doing” while another is “learning by analogy” (Fox 1981). Through viewing SOA
from an organizational perspective, we can use the knowledge in organization theory domain to inspire the research in SOA. For example, the technology independent strategies for developing service-oriented system can be investigated by analogizing organization design with SOA implementations, and complexity measurement of SOA implementations can be enlightened by existing work of organizational complexity.

This chapter focuses on using “learning by analogy” to inspire the research into effort judgment for SOA implementations. In the aforementioned work that views SOA systems from an organizational perspective, services are regarded as organizational units of an SOA system, while composite services play integrative roles that have the similar responsibilities of managers in human organizations. Therefore, a generic SOA system can be viewed as a mechanistic organization that has a clear hierarchical structure. Considering that mechanistic organizations usually adopts D&C approach to efficiently employ larger groups of work units and address problems on a larger scale (Yadgar, Kraus et al. 2003), we can also follow the D&C way to analyze and judge the effort before building SOA systems, particularly WSC-based SOA systems.

Since the pre-defined basic services: Available Service, Migrated Service and New Service are all supposed to be available before implementing WSC-based SOA systems, composing Web services is the only activity that results in effort in this particular SOA implementation type. Following the D&C principle, consequently, our concentration of effort judgment for a complete WSC-based SOA implementation can be narrowed down to individual WSCs. Hence, next chapter attempts to investigate the existing WSC approaches and identify a set of WSC effort factors. The identified factors will be used to facilitate the effort judgment for WSC-based SOA implementations.
Chapter 4

Prerequisite to Effort Judgment: Identification of Effort Factors of Web Service Composition

Since the effort judgment for a WSC-based SOA project can be spread over component composition activities by using a D&C strategy, the initial focus is on individual WSC approaches. To conquer the problem of judging effort of individual and different WSC approaches, the prerequisite should be the identification of effort factors through comprehensively understanding existing WSC approaches. Considering numerous works for composing Web services have been developed and reported in the literature, it is difficult to investigate different effort factors by exhaustively examining all the published composition approaches. However, we can inductively classify the existing WSC works, and thereby facilitate the comprehension of related knowledge and the identification of composition effort factors.

Existing classification work on WSC can be found in several survey papers (Dustdar and Schreiner 2005; Rao and Su 2005). These classifications are either incomplete or ambiguous, which causes many issues when using them to categorize and analyze new composition approaches. Firstly, none of the existing classifications distinguishes between the composition technologies and the composition contexts. For example, Dustdar and Schreiner (2005) list model-driven approaches as a separate composition category, while they combine AI planning approaches with the automated design process and ontology environment. Secondly, the terminology is vague in some composition classifications. For example, Rao and Su (2005) use “static composition” to cover those approaches having manual workflow generation, even though the component Web service selection and binding are accomplished automatically. Finally, the lack of clear classification targets is the most significant weakness of existing classification work of WSC. Current classification work generally surveys composition types through subjective identification without objective constraints. The
resulting classification is then difficult to be associated with other specific research topics such as software cost and effort estimation. For example, the declarative service composition class (Dustdar and Schreiner 2005) focuses on its irregular composition architecture that is almost irrelevant to the composition effort and cost.

This chapter uses clarified terminology, and differentiates the classifications between the Context and Process dimensions. The Context dimension includes major effort related contexts that are Pattern (orchestration and choreography), Semiotics (semantics and syntax), Mechanism (SOAP and REST), Design Time (manual, semi-auto, and automated) and Runtime (static and dynamic). The Process dimension is divided into three process models, namely One-Stop, Bridge and Double-Bridge. Considering the different influences on the composition effort, different context types and development processes can be viewed as different effort factors of WSCs. Meanwhile, those two dimensions present a novel effort-oriented classification matrix to accommodate WSC approaches for facilitating the identification of effort factors.

This chapter is organized as follows. Section 4.1 introduces the classification of WSC approaches along the Context dimension, while Section 4.2 introduces the Process classification dimension by modelling existing WSC approaches. In Section 4.3, the effort-oriented classification matrix of WSC approach is demonstrated, which also outlines the effort factors we have identified through classification. The work of identifying effort factors of WSC is finally summarized in Section 4.4.

### 4.1 Context Classification of Web Service Composition Approaches

There is no unique definition of the term “context” according to the literature. For example, the meaning of “context” can be either broad or narrow (Tosic, Karunaratne 2010). In the definition with the broad meaning, context may be any and all information that characterizes a situation of a WSC instance. With the narrow meaning, on the other hand, context can be the information about external run-time circumstances that characterize the situation of a composite Web service and influence its execution or behavior. We define that the context discussed here has broad meaning and mainly refers to the environment and different stages when composing Web services. Through analyzing the lifecycle of WSC, we have identified several contexts: Pattern, Semiotics, Mechanism, Design Time, and Runtime that have obvious influence on composition effort.
4.1.1 Pattern: Orchestration and Choreography

According to different cooperation modes among component Web services, the WSC patterns can be distinguished between orchestration and choreography. The concepts of orchestration and choreography have been introduced and explained in section 3.1.3. In brief, orchestration and choreography describe two aspects of WSC for creating business processes (Liu and Özsu 2010). Orchestration concentrates on the interactions of a single Web service with its environment, while choreography concentrates on the exchange of messages among all the involved Web services. Consequently, an orchestration can be broken down into a series of primitive workflow logic activities, which invoke Web services following the determined execution sequence based on the central controller’s enactment; whereas a choreography can be broken down into a series of message exchanges, which is not to control but to make autonomous participants cooperate based on their agreement.

In most cases, the pattern to which WSC belongs can be identified easily through the adopted standards or flow languages. For example, the current de facto standard for Web service orchestration is the Business Process Execution Language also known as BPEL. BPEL is an executable business process modeling language that can be used to describe the execution logic by defining the control flow and prescribing the rules for managing the non-observable data. The BPEL engine can then execute the description and orchestrate the pre-specified activities. Whereas one of the most widespread W3C recommended protocols for choreography is Web Services Choreography Description Language (WS-CDL). WS-CDL is designed to describe the common and collaborative observable behavior of multiple Web services that interact with each other to achieve their common goal. In other words, WS-CDL description offers the specification of collaborations between the participants involved in choreography.

Therefore, we can conveniently identify that the BPEL description related WSCs normally have orchestration context, e.g. (Anzboeck and Dustdar 2005), while WS-CDL description involved WSCs generally have choreography context, e.g. (Valero, Cambronero et al. 2009). Nevertheless, the WSC pattern should not be judged merely through these keywords, because the technique can be adapted to satisfy different scenarios. For example, some people advocate the use of abstract BPEL as a choreography language. Consequently, the most reliable judgment should be still based on the understanding of the WSC process.
4.1.2 Semiotics: Syntactic and Semantic Compositions

The semiotic environment is becoming a significant context for WSC along with the evolution of the Web. Semiotics is the general science of signs, which studies both human language and formal languages. Syntax and Semantics are two of fundamental components of semiotics. Syntax relates to the formal or structural relations between signs and the production of new ones, while semantics deals with the relations between the sign combinations and their inherent meaning.

Currently, the World Wide Web can be mainly considered as syntactic Web that uses Hyper Text Markup Language (HTML) to compose documents and publish information. When it comes to Web services, the syntactic level XML standards, for example Simple Object Access Protocol (SOAP), Web Service Description Language (WSDL) and Universal Description, Discovery and Integration (UDDI) have been used extensively to address corresponding e-business activities and research issues in industry and academia. By using human-oriented metadata, SOAP is designed to provide descriptions of message transport mechanisms; WSDL is for describing the interfaces of Web services; while UDDI registers Web services by their physical attributes such as name, address and functional categorization. However, the syntactic Web was designed primarily for human interpretation and conveying information, a syntactic web page does not contain special tagging and the meaning of information is not readable by a computer program. The lack of machine-readable semantics then requires human intervention for Web service discovery and composition, and therefore hampers the usage of Web services in complex business environment.

To overcome the obstacles of interpretability and interoperability between traditional systems and applications, the semantic Web was proposed through incremental and information-added adjustments. These adjustments make the Web ontological. Ontology was originally developed to facilitate knowledge sharing and reuse (Gruber 1993). Benefiting from ontology, greater ability of expression is provided for knowledge modeling and communicating knowledge between heterogeneous and distributed application systems. Therefore, the semantic Web can be viewed as a version of the Web with ontological content and services, which includes machine-readable and human-transparent descriptions to the existing data and documents on the syntactic Web. In addition, the semantic Web supplies the necessary infrastructure and techniques for publishing, resolving and reasoning about ontological descriptions of the content and services.
As for semantic Web services, besides the syntactic description, the information needed to select, compose, and respond to services is also encoded with semantic markup by service providers. These efforts of service augmentation can then facilitate automated service discovery, composition, dynamic invocation and binding without human assistance or highly constrained agreements on protocols. Figure 4.1 illustrates the differences between syntactic and semantic Web services. Informally, a Web service can be characterized by its name, required inputs, the produced outputs, and the operations it will take (Kona, Bansal et al. 2008). The inputs and outputs may be further subject to pre-conditions and post-conditions respectively. With descriptions in the syntactic level, as shown in the unfilled nodes of syntactic Web service in Figure 4.1, it is difficult for service providers and consumers to represent or interpret the meaning of inputs, outputs and other applicable constraints. A semantic Web service relaxes such limitation by augmenting the service description with a rich set of formally semantic annotations of the service’s capabilities, as shown in the grey nodes of semantic Web service in Figure 4.1. Accordingly, new standards and languages of semantic markup, like Web Ontology Language for Web Services (OWL-S) and Web Service Modeling Ontology (WSMO), should be investigated and used to give meaning to Web services.

Overall, the XML-based standards are for syntax, whilst the ontology-based standards are for semantics. Both share unified Web infrastructure and together provide capability for developing Web applications that deal with data and semantics. Nevertheless, one of the most important characteristics of ontology-based techniques is that they allow a richer integrability and
interoperability of data in communications between domains. As previously mentioned, driven by the semantic markup and agent technologies, semantic Web service discovery, selection, composition, and execution are all supposed to be automatic tasks. Although fully automating these processes is still a challenge, accomplishing parts of this goal can still be achieved. For example, the semantic description is useful for the translation between WSC problems and AI-planning systems (Rao, Küngas et al. 2006), while the semantic matchmaking can be used to facilitate the automated Web service discovery (Sirin, Hendler et al. 2003). Considering these outstanding characteristics, WSCs can be categorized according to syntactic and semantic context, while the context can be also identified through employed standards and techniques.

4.1.3 Mechanism: RESTful and SOAP-based Compositions

Concentrating on the technologies and architectures, nowadays there are two main mechanism paradigms of building composite Web services, namely RESTful composition and SOAP-based composition.

Basically, REpresentational State Transfer (REST) and SOAP are not directly comparable with each other and not necessarily opposite. REST is an architectural style originally designed for building large-scale distributed hypermedia systems, whereas SOAP is a general protocol used as one foundation of numerous WS-* technologies. Within the REST environment, the Web is considered as a universal storage medium for publishing globally accessible information. In contrast, SOAP treats the Web as the universal transport mechanism for message exchange. When building Web services, traditional SOAP/WS-* environment requires relatively heavyweight open standards than that used in the RESTful context. Although the SOAP vs. REST debate has been an ongoing discussion for some time, there is an implicit consensus that REST is more suitable for basic, ad-hoc, client-driven scenarios, while SOAP/WS-* are more suitable to address the quality of services requirements in highly interactive Web applications.

Although the concepts REST and SOAP are not comparable, RESTful and SOAP-based Web services are indeed comparable. We can identify the differences between RESTful and SOAP-based Web services mainly through their interfaces, the operations and Message Exchange Patterns (MEPs) behind interfaces, and their QoS support techniques.
1) Interface differences

The interface of a RESTful Web service comprises a variable set of Uniform Resource Identifiers (URIs). Each URI uses a globally unique address to identify a specific resource. Unfortunately, to the best of our knowledge, there is no standard and machine-processable way of describing RESTful interfaces. Using WSDL 2.0 description to wrap the RESTful Web services has been revealed as a burden for service consumers (Pautasso 2009). The Web Application Description Language (WADL) and other dedicated interface definition languages for RESTful services like RESTful Interface Definition and Declaration Language (RIDDLL) (Mangler, Schikuta et al. 2009) are not yet widely employed. Consequently, most of the time the interfaces of RESTful Web services are described through natural, informal, and more human-oriented documentations. When it comes to SOAP-based Web services, as mentioned previously, WSDL has gained widespread adoption to syntactically define the service interfaces. In a WSDL document, SOAP-based Web services are described as collections of network endpoints, or ports. A port associates a network address with a reusable binding. The reusable WSDL binding contains the concrete transport protocol and data format specifications for a particular port type. A port type is a set of abstract operations that are related to some abstract messages representing the data for exchange. Benefiting from the abstract interfaces described by WSDL, technical details of SOAP-based Web services can be concealed, for example, the implementation language, deployment platform and underlying communication protocol.

2) Operation differences

Since “REST is in many ways a retrospective abstracting of the principles that make the World Wide Web scaleable” (zur Muehlen, Nickerson et al. 2005), RESTful Web services requires little technology support apart from well accepted HTTP and XML infrastructures. As a result, the manipulations of resources are completely constrained in the RESTful environment through a fixed set of four operations associated with HTTP: GET, PUT, DELETE, and POST. GET is used to retrieve a representation of the current state of a resource. PUT can either update the state of existing resource or create a new resource with the requested URI if it does not previously exist. DELETE is used to delete a URI-identified resource and also invalidate the URI itself. POST creates subordinate resources to which new URIs are assigned by service provider. In contrast to the standard operations among RESTful Web services, the operations provided by SOAP-based Web services are ad hoc. Various APIs defined in different WSDL documentations represent different sets of operations for communication and interaction between service providers and consumers. The operations of SOAP-based Web services essentially are functional components that are located on remote machines and can be invoked through APIs over the network.
3) MEPs differences
MEPs are patterns or templates that abstract the sequences of message transmission in the Web service context. Since REST is associated closely with HTTP, and HTTP is stateless request-response application protocol, RESTful Web services only have the synchronous request-response pattern under the HTTP mechanism. SOAP-based Web services allow rich patterns ranging from traditional request-response to broadcasting and sophisticated message exchanges. The latest WSDL 2.0 has been published with supporting eight MEPs (Lewis 2007). Each MEP describes a bilateral message exchange between two involved services from a service point’s perspective.

- In-Only – The service receives a message.
- Robust In-Only – The service receives a message and will return a fault message only when meeting a fault.
- In-Out – The service receives a message and returns a response message.
- In-Optional-Out – The service receives a message and optionally returns a response message.
- Out-Only – The service sends a message.
- Robust-Out-Only – The service sends a message and will receive a fault message only when its partner service meets a fault.
- Out-In – The service sends a message and receives a response message.
- Out-Optional-In – The service sends a message and optionally receives a response message.

4) QoS support technique differences
Quality of Service (QoS) indicates a certain performance level of services that will be delivered to consumers, and can be evaluated through corresponding parameters like response time, throughput, cost, etc. As REST is usually used in conjunction with HTTP, the QoS of RESTful services are supported generally through basic protocols and techniques. For example, services’ interactions can be secured at the transport layer using the Secure Sockets Layer (SSL) protocol, while the security of messages can be guaranteed by encryption and digital signatures. On the contrary, SOAP-based Web services adopt more complicated mechanisms to cover QoS features. On the one hand, the header of an SOAP document contains message-layer infrastructure information that can be used for QoS configurations. On the other hand, the WS-* technology stack is employed to satisfy the large scope of QoS requirements such as transactions, security, and reliability. Benefiting from SOAP and WS-* technologies, QoS aspects of SOAP-based Web services are protocol transparent and
independent. In other words, the QoS of Web service can be provided end to end without taking into account the variety of middleware systems transported.

All these differences between RESTful and SOAP-based Web services make the problem of RESTful WSC fundamentally different from the composition problem of SOAP-based Web service. SOAP-based WSC is a collection of related, structured activities or tasks that produce a specific service or product for a particular customer. Within the relatively complex SOAP-based environment, a large number of standards and tools have been developed to facilitate the service composition activities. Dissimilarly, RESTful WSC integrates normally disparate Web resources to create a new application. These resources can be the exposure of pure data or traditional application functionality. With the constraint of lightweight technologies adopted in RESTful environment, service compositions mainly focus on the Web 2.0 Mashups that usually imply simple and fast integration of data/content from different sources on the Internet.

### 4.1.4 Design Time: Manual, Semi-Automated and Automated Compositions

Generally, there are four fundamental activities when composing a Web service, namely Planning, Discovery, Selection, and Execution (Cardoso and Sheth 2006). Planning is to determine a composition plan including the execution sequence of tasks. Each task corresponds to either the functionality or activity of a service. Discovery is to find all the candidate services that can satisfy the tasks in the plan. The aim of Selection is to choose optimal subset from all the discovered services by using non-functional attributes. Execution builds a real composite Web service. In practice, the sequence of Planning, Discovery, and Selection can be diverse. For example, the theorem proving approach in (Rao, Küngas et al. 2006) is based on the pre-determined Web services to generate the composition plan. Moreover, during the service composition procedure, the network configurations and non-functional factors may change, and existing Web services may be updated or terminated. As a result, some pre-identified services may not be available, and the new ones need to be discovered and selected. In other words, Discovery and Selection can also take place during or even after Execution. Therefore, we can define a potential Adaptation activity at the end of the procedure of WSC.
Based on the previous analysis, the process of WSC can be separated into design time and runtime stages. Figure 4.2 shows one of the possible composition scenarios. Depending on the real practices, the design time stage comprises various activities from only Planning to the combination of Planning, Discovery, and Selection. According to the extent to which human intervention is involved, the design time procedure can be manual, semi-automated, and automated. Considering that there is still a long way to realize the complete automation of WSC even at design time, we mainly concentrate on the Planning activity when unfolding classification. Therefore, we can draw the outline of these three types of composition approaches during design time as:

**1) Manual approach**

In general, the manual Planning activity implies manual design-time WSC. Two different scenarios of manual approaches can be further identified respectively as primitive level and abstract level respectively. In primitive manual design-time composition approaches, developers have to specify every detailed activity in the composition processes. The resulting specifications are executable composition programs. For example, we can use BPEL to describe the procedure of WSC following the logic of corresponding business process, and the finalized description is executable with the support of the BPEL engine. As for the manual design-time composition approaches at an abstract level, the WSC plans are usually drawn into abstract workflows or models instead of specific programs. In such approaches the manual planning results cannot be executed directly, but can be transformed into executable specifications and finally executed by some tools or engines. Examples can be found in most of the UML related model-driven approaches.
2) Automated approach
In general, the automated *Planning* activity implies automated design-time WSC. In the manual approaches discussed above, although we can decrease the effort of WSC through abstraction rather than programming, the planning phase still has to be realized manually. How to automatically generate the composition model or workflow then becomes a subsequent research topic. The current trend is to use Artificial Intelligence (AI) planning to satisfy the automation of the generation of a WSC plan. Benefiting from existing AI planning systems, the prerequisite effort of WSC is only to encode the requirements into dedicated, formal, and mathematical expressions.

3) Semi-automated approach
We treat an instance of WSC as semi-automated approach, if one of the following cases is met: (1) there are specifically automated *Discovery/Selection* activities to facilitate manual *Planning*; or (2) there are specifically manual *Discovery/Selection* activities that constrain automated Planning. Taking (Sirin, Hendler et al. 2003) as an example of the former case, semantic matchmaking technique is used to realize the semi-automated approach by automatically filtering and presenting matching services to the user at each step of a composition. An example of the latter case can be found in (Rao, Künugas et al. 2006), the theorem proving technique requires manually pre-determining Web services before automatically generating the composition plan.

4.1.5 Runtime: Static and Dynamic Compositions
The *Execution* and potential *Adaptation* activities remain at the runtime stage of WSC. By focusing on the *Adaptation* activity, we can define that the WSC is *dynamic* at runtime if it is adaptive with minimal user intervention, otherwise it is *static*. In detail, static runtime WSC has to manually adapt the changing environment. In the worst case, static runtime composition does not have adaptability at all. On the contrary, dynamic runtime composition can still discover and select new services without requiring any human assistance when practically generating composite services, for instance eFlow (Casati, Ilnicki et al. 2000).

Benefiting from the division between the design time and runtime of WSCs, we can clearly distinguish the two concepts: automated design-time and dynamic runtime compositions that are confusing in the existing literature. Furthermore, it can be found that there is no relationship between automated composition at design time and dynamic composition at runtime. On the one
hand, automated design-time composition does not imply dynamic runtime composition, for example, most of the AI planning approaches only concentrates on the automated Planning process while leaving the planning result executed statically. On the other hand, dynamic runtime composition does not require automated design-time composition, for example, the visual language UML Profile for composing Web services (UML-WSC) (Thone, Depke et al. 2002) supports dynamically composing Web services at runtime although the composition model is still built manually at design time.

4.2 Process Classification of Web Service Composition Approaches

The processes and behaviors of different WSC approaches can be abstracted and categorized by the modeling work. Therefore, we can try to use different approach models to classify WSC approaches along the Process dimension. According to the analysis in Section 4.1.4, any WSC approach will comprise of four fundamental activities: Planning, Discovery, Selection and Execution. In practice, the Discovery and Selection activities cannot be always isolated from the Planning and Execution activities. In other words, the Discovery and Selection activities could be part of the Planning or Execution activities. Therefore, taking into account all of the four activities may complicate the modeling work and result in overlapping approach models. Considering that a “model is an abstraction of reality in the sense that it cannot represent all aspects of reality” (Rothenberg 1989), it is not necessary to distinguish detailed techniques between different composition approaches. If we ignore the Discovery and Selection activities, we can abstract three approach models from existing WSC research: One-Stop Approach Model, Bridge Approach Model, and Double-Bridge Approach Model.

4.2.1 One-Stop Approach Model

The One-Stop approach model, as shown in Figure 4.3, describes those approaches that need only one step between composition request and execution. Unfortunately, one step does not imply less effort. We can find three approach scenarios having different efforts, but all belonging to the One-Stop approach model.
Scenario I: The planning result is an executable program. In primitive manual WSC approaches, developers have to specify every detailed activity in the composition processes. The resulting specifications are executable composition programs. Obviously, the composition process can be programmed from scratch by using traditional languages and standards. The current universal technique is to use a dedicated, process-oriented language like BPEL to specify the transition interactions among Web services at a macro-level state. The composition engine can then execute the description and orchestrate the pre-specified activities.

To supplement the approach model illustrations and facilitate comprehending different approach scenarios, we also propose formal models to mathematically describe different WSC approaches. Scenario I herein can be formalized as:

\[ CS = E(P < \text{statement}>) \]  

(4.1)

where a composite service CS is generated by Executing (E) program P that consists of a series of statements. Note that \(< >\) symbol represents a vector that comprises an element sequence.

Scenario II: The planning result is an executable, abstract workflow or model. Supported by some tools or engines, the workload of WSC can be relieved by drawing the abstract, executable workflow or model, instead of programming executable specifications. Furthermore, the visualized descriptions for WSCs can facilitate their understandability. For example, eFlow (Casati, Ilnicki et al. 2000) adopts a graph-oriented method to define the interaction and order of execution among the nodes in an abstract composition process. Although the graph of composition flow structure has to be drawn manually, the eFlow engine can update the graph and bind the nodes with concrete services automatically at runtime, and eventually contact the providers in order to execute those services. As a result, the pre-drawn abstract composition workflow will be directly executed. Another example is the UML-WSC profile (Thone, Depke et al. 2002). The UML-WSC profile is a well-defined UML extension, which uses a static model and extended variant of activity diagrams to define the process-oriented WSC. The static model describes the available Web services and
components, while the extended variant of activity diagrams describes the composition processes. The composition model specified through UML-WSC profile can be executed automatically by a process engine. Therefore, the UML-WSC profile is also considered as an alternative to non-visualized languages like BPEL.

Similarly, the formal model of Scenario II is:

\[ CS = E(M < (\langle Action \rangle \times \langle State \rangle) \wedge \langle ControlFlow \rangle>) \]  \hspace{1cm} (4.2)

where a composite service CS is generated by directly Executing (E) abstract model M. Model M is built in the space of cross join (\(\times\)) of possible Action set with possible State set, and filtered (\(\wedge\)) by the process of Control Flow. A cross join returns the Cartesian product (Warner 1990) of the sets of items from the two joined vectors. Note that control flow exists not only in coordination of orchestration mechanism but also in collaboration of orchestration mechanism when composing Web services.

**Scenario III: The planning result is an executable requirement.** The growing semantic Web and semantic Web service technologies open up prospects for the full automation of WSC. The full automation implies that the composite Web services can be built directly according to user’s requirements. Driven by the semantic markup and agent technologies, the discovery, selection, composition, and execution of semantic Web service are all supposed to be automated tasks. Unfortunately, none of these tasks is completely realized within the current Web environment. The requirement-driven, full automation of WSC is still the object of ongoing research.

Although full automated design-time WSC has not been realized, Scenario III can still be formalized as:

\[ CS = E(R < (\langle Input \rangle \rightarrow \langle Output \rangle) \wedge \langle BusinessRule \rangle \wedge \langle Resource \rangle>) \]  \hspace{1cm} (4.3)

where a composite service CS is generated by directly Executing (E) user’s requirement R. Requirement R is a morphism (\(\rightarrow\)) from the Input set to the Output set filtered (\(\wedge\)) by Business Rules and Resources. Business Rules are regulations of certain operation or calculation, while Resources are essential components for practical compositions for example existing Web services.

### 4.2.2 Bridge Approach Model

The Bridge approach model of WSC, as shown in Figure 4.4, is an evolution from the One-Stop approach model. As mentioned earlier, the most common scenario of the One-Stop approach is to
Chapter 4: Prerequisite to Effort Judgment: Identification of Effort Factors of Web Service Composition

manually compose Web services. The user must provide inputs at choice points, decide the interoperation among component Web services, and specify the composition procedure. In other words, manually composing Web services is hard and time consuming for users. To simplify the work of generating composite Web services, the composition can be designed at a more abstract level than that of technology-specific implementation during the planning process. The Bridge approach model describes such approaches that plan composition at an abstract level, while the planning result cannot be directly executed and has to be transformed into an executable specification.

![Figure 4.4: Bridge Approach Model.](image)

The fundamental characteristic of the Bridge approach model is that the abstract composition definitions are independent of, but can be transformed into, executable composition specifications. Therefore, unlike the second scenario of One-Stop approach model, the Bridge approach model uses a transformation process for the mapping between the planning result and executable specification. The notion of Bridge approach model is that the planning phase of WSC does not need to be tied to any particular composition language and execution engine, thereby the same planning result can be transformed into more than one executable description.

The typical example of the Bridge approach model is the model-driven WSC approaches. In the model-driven approaches, models are used to describe user requirements, information structures, abstract business processes, component services and component service interactions. The techniques used in the modeling work normally follow the standards provided by the Object Management Group (OMG). The standards mainly refer to the UML and MDA. Numerous discussions related to UML-based modeling of WSC can be found in the literature (Orriëns, Yang et al. 2003; Dustdar and Schreiner 2005). The generic scenario is to use UML class diagrams to represent the state parts of compositions, while the behavior parts are represented through UML activity diagrams. The state parts can be Web service interface, the structure of composite Web service, and QoS characteristics. Meanwhile, the behavior parts describe the composition operations,
interactions of component Web services, and control and data flow. Eventually, the well-defined composition model can be transformed into an executable composition specification by using, for example, the open source UML Model Transformation Tool (UMT).

When formalizing the Bridge approach model, the detailed descriptions of composition model and program can be hidden to simplify the formal expression. Therefore, the Bridge approach model may be formalized as:

\[ CS = E(M \Rightarrow P) \]  \hspace{1cm} (4.4)

where a composite service CS is generated by Executing (E) program P that is transformed from the abstract model M. The \( \Rightarrow \) symbol is used to represent the process of transformation. Benefiting from these formal expressions, the difference between the Bridge and One-Stop approach models can be conveniently identified: The former is different from Scenario I of the latter because WSC starts from abstraction work, and different from Scenario II because the composite Web service is still generated by executing a program.

### 4.2.3 Double-Bridge Approach Model

The Double-Bridge approach model can be treated as a further evolution from the Bridge approach model. In the Bridge approach model, although the effort of WSC has decreased through abstraction, the planning phase still has to be realized manually. How to automatically generate the composition model or workflow then becomes a subsequent research topic. The current trend is to use Artificial Intelligence (AI) planning to satisfy the automation of generation of WSC plan. Since AI planning systems generally adopt dedicated, formal, and mathematical techniques, the initial information and composition requirement must be transformed for input into a planning system, and the planning result should be transformed again into executable specification to build composite Web service. In the Double-Bridge approach model, correspondingly, the planning phase is settled between two transformation processes as Figure 4.5 illustrates.

In fact, the Double-Bridge approach model is the modeling result of AI planning based composition approaches. Like most AI topics, AI planning seeks to use intelligent systems to generate a plan that can be one possible solution to a specified problem. Here, plan is an organized collection of operators within the given application domain. AI planning is essentially a search problem. The underlying basis of planning relies on state transition system with states, actions and observations. Benefiting from the state transition system, the planner explores a potentially large search space and
produces a plan that is applicable to bridge the gap between the initial state and the goal. Therefore, AI planning in WSC normally comprises of five attributes: all the available services, the initial state, the state change functions, all the possible states, and the final goal. The initial state and the final goal are specified in the requirements for composing Web service. The state change functions define the preconditions and effects when invoking Web services.

A large amount of research has been reported about the AI planning related WSC. These works apply techniques ranging from Situation Calculus (Chifu, Salomie et al. 2008), Automata Theory (Mitra, Kumar et al. 2007), Rule-based Planning (Medjahed, Bouguettaya et al. 2003), Query Planning (Thakkar, Knoblock et al. 2003), Theorem Proving (Rao, Künegs et al. 2006), Petri Nets (Gehlot and Edupuganti 2009), to Model Checking (Traverso and Pistore 2004). After encoding the requirement of WSC, generally, these techniques convert the composition problems into generating execution workflows using the respectively dedicated expression. The workflows can then be transformed into executable specifications like BPEL documents or other XML-based descriptions, and executed through the corresponding engines.

Similar to the formalization of the Bridge approach model, the Double-Bridge approach model can be formalized as:

\[ CS = E(R \Rightarrow M \Rightarrow P) \]

(4.5)

where a composite service CS is generated by Executing (E) program P that is transformed from abstract model M, while M here is normally a composition plan that is built automatically after inputting a set of well-regularized data that is transformed from requirement R. Through the formal expression of the Double-Bridge approach model, we can conveniently find that the Double-Bridge approach will be a candidate solution to fully automated WSCs supposing every part of the approach is fulfilled automatically.

\[ \text{Figure 4.5: Double-Bridge Approach Model.} \]
4.2.4 Clarification against Similar Concepts

The approach models of WSC are different from composition models and composition patterns. We can distinguish these similar concepts by analyzing their concerns, targets and methodologies. Both the composition models and composition patterns are generally used during design time for composing Web services, whereas the composition approach models are abstractions and summary of the numerous existing implementations of WSC. The essential objective of modeling composition approaches is to summarize and classify the current WSC strategies, instead of guiding or facilitating the generation of composite Web services.

**Clarification 1: Composition Approach Model is not Composition Model**

The first notion that should be distinguished from composition approach model is the composition model. Generally, a model is an abstract summary of some concrete object or activity in reality. Modeling the complex reality is usually to facilitate analysis, comprehension, and further study. In practice, a model cannot and need not be a complete reflection of one thing being modeled. Several models might need to be weaved together to reflect the full reality, while each model addresses one of the aspects in the same domain (Mellor, Clark et al. 2003). As a matter of fact, the approach model of composition and the composition model are abstractions of different aspects of the WSC.

The WSC models are abstractions of the participant services and the composition workflow. In other words, composition models are used to describe user requirements, information structures, abstract business processes, component services and component service interactions. Modeling WSC is normally considered as a simpler, safer and cheaper way for model-driven development of composite Web services (Orriëns, Yang et al. 2003). Numerous efforts, especially graphical representations, have been applied to building composition models. For example, the Unified Modeling Language (UML) and Model-Driven Architecture (MDA) have been widely adopted to describe WSCs (Thone, Depke et al. 2002; Orriëns, Yang et al. 2003). The Petri Nets based algebra is also introduced as a graphical modeling technique to composing Web services (Hamadi and Benatallah 2003; Gehlot and Edupuganti 2009). Particularly, executable models can even be used to stimulate and quantitatively analyze the WSC. Overall, the composition models are to help developers and decision-makers propose and understand the possible solutions to WSCs, and give better insights into those candidate solutions at design time. Therefore, the composition modeling work is at the beginning of an abstract-to-concrete development procedure. For example, we can use *UML for Service* (UML-S) to describe and model WSC (Dumez, Gaber et al. 2008), and a sample WSC model is illustrated in Figure 4.6.
Chapter 4: Prerequisite to Effort Judgment: Identification of Effort Factors of Web Service Composition

Unlike composition models, the approach models of WSC are abstractions of real implementation.

Figure 4.6: A Sample of Web Service Composition Model (Dumez et al. 2008).
Chapter 4: Prerequisite to Effort Judgment: Identification of Effort Factors of Web Service Composition

processes when composing Web services. In other words, approach models focus on the composition lifecycle, while ignoring the technical details. The purpose of modeling composition approaches is to help researchers better and more quickly understand current strategies used to compose Web services. Since the composition approach models are built based on numerous existing implementations of WSC, the modeling work is at the end of a concrete-to-abstract induction procedure.

Clarification 2: Composition Approach Model is not Composition Pattern

Another phrase confused with composition approach model is composition pattern. Traditionally, patterns are used to represent system design and architecture ranging from customizable code fragments to the reusable business process logic. A pattern is described as a three-part rule that comprises a certain context, a problem and a solution. Through their use in this way, patterns help conserve and reuse abstract knowledge of the solutions to the related problems within particular contexts. The WSC patterns are then the abstract descriptions that suggest methodologies for composing Web services. There are three main types of WSC patterns:

- **Composite pattern based WSC patterns.** The composite pattern is one of the most popular patterns used to construct compound components. Considering a composite Web service is built from component Web services, the composite pattern can be naturally applied to WSC. Following the two alternatives composite patterns, composing Web services can be abstracted into containment composition and aggregate composition (Byun 2007). Obviously, the composite pattern based WSC patterns represent the inside structures of composite Web services.

- **Workflow pattern based WSC patterns.** Both WSC and workflow management take into account executable processes, therefore, some workflow patterns are also suitable for composing Web services. As the Business Process Execution Language (BPEL) is the current de facto standard for WSC, and a BPEL process specification is a kind of flow-chart, Wohed et al. (Wohed, van der Aalst et al. 2003) discussed how and to what extent the workflow patterns can be captured in BPEL. Each pattern here represents one kind of control flow or message flow in the whole composition procedure, for example the Multi-Choice pattern and Broadcast pattern.

- **Activity based WSC patterns.** Benatallah et al. (Benatallah, Dumas et al. 2003) proposed a full scope of WSC patterns towards bilateral service-based interactions, multilateral service composition, and execution of composite services. These patterns are to tackle
separately the important issues involved in the design, construction and maintenance of composite services. In other words, each composition pattern here describes one activity related to composing Web services, for example service discovery or service wrapping.

According to these specifications of WSC patterns in the literature, we can find the differences between composition approach models and composition patterns. Firstly, composition patterns scatter in different aspects and details of WSC, and as a result the expression of the composition lifecycle normally requires a collection of related patterns. Nevertheless, every composition approach model will describe a complete WSC procedure. Secondly, the composition patterns are used during the planning stage to instruct developers on how to build composite Web services from generic to specific (Tut and Edmond 2002). On the contrary, the approach models are the summary of the existing WSC implementations. Thirdly, the composition patterns emphasize the related contexts, while the same information will not be considered in approach models.

**4.3 Effort Factors of Web Service Composition**

Through classification and initial analysis, we can find that different type of WSC approaches have different characteristics that exert different influences on the development effort. When considering different composition *Patterns* for the same target, orchestration deals with a central mediator while choreography is a collaboration of all the participant Web services. Within the *Semiotics* context, semantic Web services have more descriptions than syntactic Web services, which can facilitate service discovery and matchmaking. When adopting different composition *Mechanism*, RESTful WSCs are relatively lightweight compared with SOAP-based WSCs. According to the manipulation procedure before generating a real composite Web service, there can be obviously different effort related to manual, semi-automated, or automated compositions at *Design Time*. During *Runtime*, the dynamic and static compositions have different effort due to the requirement of adaptability of WSC. Using different *Process* models, different WSC approaches involve different amounts of information that brings different levels of effort to developers.

Therefore, these WSC types essentially determine the required effort of corresponding WSC approaches. Considering different WSC types reflect different characteristics when implementing WSC, we can classify a particular WSC approach by all the applicable types to understand the approach’s characteristics. As previously specified, our classification of WSC is established along two directions: one is Context and another Process. These two directions can then be used as two
dimensions to build an effort-oriented classification matrix for WSC, as shown in Table 4.1. When a WSC approach comes, we can settle it into this classification matrix and conveniently find its characteristics so as to facilitate the effort judgment. Meanwhile, these different WSC types we have identified and listed in the matrix can be treated as effort factors of WSC, as illustrated in Figure 4.7.

**Figure 4.7: One Process and Five Context-related Effort Factors of Web Service Composition.**
### Chapter 4: Prerequisite to Effort Judgment: Identification of Effort Factors of Web Service Composition

#### Table 4.1: Classification Matrix of WSC and Sample WSC Approaches

<table>
<thead>
<tr>
<th>Process</th>
<th>Detailed Technique</th>
<th>Pattern</th>
<th>Semiotics</th>
<th>Mechanism</th>
<th>Design Time</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Orchestration</td>
<td>Choreography</td>
<td>Syntax</td>
<td>Semantics</td>
<td>SOAP</td>
</tr>
<tr>
<td>One-Stop</td>
<td>Workflow: BPEL Programming</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td></td>
<td>Workflow: eFlow (Casati, F., Ilnicki et al. 2000)</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Workflow: Bite (Rosenberg, Curbera et al. 2008)</td>
<td></td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Bridge</td>
<td>Model-Driven: UML + MDA (Skogan, Groenmo et al. 2004)</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td></td>
<td>Model-Driven: MD Mashup (Mosser 2008)</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td></td>
<td>Model-Driven: MoSCoE (Pathak, Basu et al. 2006)</td>
<td>√</td>
<td></td>
<td>√</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>Double-Bridge</td>
<td>AI Planning: SHOP2 (Sirin, Parsia et al. 2004)</td>
<td>√</td>
<td></td>
<td>√</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AI Planning: Linear Logic Theorem Proving (Rao, Küngas et al. 2006)</td>
<td>√</td>
<td></td>
<td>√</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AI Planning: AIMO (Tabatabaei, Kadir et al. 2008)</td>
<td>√</td>
<td></td>
<td>√</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AI Planning: Situation Calculus for REST (Zhao and Doshi 2009)</td>
<td>√</td>
<td></td>
<td>√</td>
<td>√</td>
<td></td>
</tr>
</tbody>
</table>
4.4 Summary

To identify effort factors of WSC, we first must get familiar with existing WSC approaches. However, the territory of WSC has been researched so broadly that it becomes difficult to explore every existing composition approach. Through investigation of a limited number of approaches and inductive analysis, we can deliver a set of general classifications of WSC to facilitate comprehending the state of the art of WSC approaches. Unlike existing classification work, we propose to classify different WSC approaches along two dimensions: Context and Process. Several pairs of effort-related contexts are picked in the Context dimension, while three technique categories are paralleled according to their Process models. These two dimensions then constitute an effort-oriented classification matrix of WSC. When accommodating a new WSC approach, this effort-oriented classification matrix can simultaneously outline different characteristics of the approach that have different influences on the WSC effort. Consequently, the WSC types hosted in the classification matrix can be naturally used as effort factors for composing Web services.
CHAPTER 5

Implementation of Effort Judgment: A Qualitative Approach

Given identified effort factors of WSC, we can now step by step investigate effort judgment around effort factors, for individual WSCs, and for WSC-based SOA implementations. Considering the various and numerous WSC approaches that have been revealed in Chapter 4, it is nearly impossible to collect enough real development data from experiments or industry. Consequently, it is difficult to unfold research into effort judgment along a quantitative way. By treating existing development experiences as circumstantial evidence, therefore, we propose circumstantial-evidence-based effort judgment to give qualitative comparison between different influences of different factors on different partial effort when composing Web services.

Furthermore, we propose a method to reflect the combination of partial influence of factors on WSC effort: firstly, a set of symbols and rules are used to facilitate representing the comparison result of circumstantial-evidence-based judgment mentioned above; secondly, an effort score is assigned to each factor of WSC according to the effort comparison result; thirdly, inspired by the Addition and Multiplication principles in Combinatorics, we define that the scores for different context types are accumulable in the Context dimension while the effort scores are multipliable across different dimensions in the classification matrix. Through this proposed method, we can then build up a checklist for effort judgment for different types of WSC approaches.

Finally, following the generic strategy of effort judgment for SOA implementations that we have discussed in Chapter 3, we can use the effort-judgment checklist together with the D&C algorithm to further calculate effort scores of different implementation proposals of a certain WSC-based SOA project, and therefore conveniently compare the qualitative effort required by different proposals. As such, although there could be a lack of quantitative effort estimation, the qualitative judgment result can still be used to help determine effort tradeoff before WSC-based SOA implementations.
This chapter is organized as follow. After contrasting circumstantial evidence with direct evidence in the context of software cost estimation, Section 5.1 proposes the circumstantial-evidence-based judgment for different influences of WSC effort factors. Section 5.2 uses a set of symbols and rules to represent those judgment results and combine partial effort judgments into whole, which results in an effort-judgment-checklist for individual WSC approaches. Benefiting from the effort-judgment-checklist of WSC, Section 5.3 realizes the effort judgment for WSC-based SOA implementations following the strategy discussed in Chapter 3. The summary of this chapter is drawn in Section 5.4.

### 5.1 Circumstantial-Evidence-Based Judgment for Various Influences of Different Effort Factors

As previously mentioned in Chapter 2, mathematical effort estimation models have been well documented in academia for many years, while the pervasive estimation method in industry is still based on expert judgment (Jørgensen and Shepperd 2007). One possible reason is that the mental processes software professionals use to unfold estimation are more closely related to a case-based reasoning (CBR) approach than a regression-based model (Menzies and Hihn 2006). However, expert judgment considerably depends on experts’ availability and experience, and experts’ knowledge is hardly accessed by others (Keung 2007). Therefore, expert opinion may be not reliable if it is not supported by objective or scientific evidences. To reduce the possible bias and uncertainty that happens in expert judgment, practical guidelines claim that estimation experts should be selected based on their experience from similar projects (Jørgensen 2005). Following the practical guidelines, unfortunately, the expert judgment approach could still be infeasible if the experts or the past data are not available.

Basically, expert judgment-based software effort estimation must comply with a golden rule: the expert judgment should always require justification rather than gut feelings (Jørgensen 2005). Inspired by Evidence-Based Software Engineering (EBSE) that is “to provide the means by which current best evidence from research can be integrated with practical experience and human values in the decision making process regarding the development and maintenance of software” (Dybå, Kitchenham et al. 2005), we can re-consider the justification of expert judgment from an evidence-based perspective. According to the classification of evidence (Siegel, Knupfer et al. 2000), the results of the CBR-based mental processes in traditional expert judgment can be regarded
as direct evidence: experts act as witnesses and adduce previous cases for the current one. Considering the aforementioned limitation of direct evidence collection – the requirement of availability of experts with experience from similar projects, we propose circumstantial-evidence-based judgment for various influences of effort factors of WSC. Benefiting from existing software development experiences as circumstantial evidence, we can use diagnostic reasoning to qualitatively infer partial implementation effort caused by different effort factors in different WSC proposals. As a result, circumstantial-evidence-based judgment can be used to facilitate and improve the final effort judgment for complete WSC approaches.

5.1.1 Effort Judgment with Circumstantial Evidence

As an analogue of similar forensic scenarios, the traditional CBR-based expert judgment can be viewed as using direct evidence to estimate implementation effort of a software project. In forensic science, as the name suggests, direct evidence is evidence that proves a fact without requiring inference or presumption (Siegel, Knupfer et al. 2000). In other words, direct evidence immediately and precisely establishes a bridge between judge and fact. An example of direct evidence could be a witness's observation or personal knowledge of a certain fact. The defendant involved in the past fact is the exact one involved in the judgment. In the context of effort estimation, different from law, the new project to be judged is obviously none of the past ones. Nevertheless, to some extent, experts inevitably view the similar projects as the same one when doing effort estimation based on their experiences. For example, as suggested by Jørgensen (2005), the actual effort of similar projects or similar activities in other projects will be referred to as justifications for expert judgment for new project. Therefore, it is reasonable to consider that in traditional expert judgment experts use their observation on past projects as direct evidences to estimate effort of new project.

Contrasted with direct evidence, circumstantial evidence does not prove fact in a straightforward sense, while it requires the intervening or additional evidence inference to confirm the fact. In forensic science, the most obvious difference between direct and circumstantial evidence is that “direct evidence is a verbal representation of a crime itself, whereas circumstantial evidence is an abstract statement about the connection between the defendant and an incriminating physical trace of the crime” (Heller 2006). Usually, circumstantial evidence is not sufficient, but increases the probability of the defendant’s guilt, for example blood or fingerprints. Similarly, unlike the direct evidence in expert judgment that directly gives the estimated effort, circumstantial evidence for effort judgment must be effort-related abstract statements. Suppose each finished software project
deposits some development experience in the human knowledge, similar projects or similar activities should have similar development experiences. Different experiences can then be abstracted into different assertions as scattered fingerprints of existing software projects. As such, different from human beings, similar software projects or development activities may share the same fingerprints.

To sum up, when estimating effort for a new project, similar projects’ or activities’ actual effort can be viewed as direct evidence, while existing development experiences can be considered as circumstantial evidence. In forensic science, both direct and circumstantial evidences are used to draw categorical, yes-no type, conclusions. In the context of software effort estimation, direct evidence brings quantitative effort estimate for a particular project proposal, while circumstantial evidence can give qualitative comparison between effort estimates of different development proposals.

As emphasized previously, circumstantial evidences cannot be used without rational inference to proven facts. The rational inference can be realized as a cascaded process of diagnostic reasoning. A possible guideline for using circumstantial evidences to do diagnostic reasoning is the theory of propositional learning (TPL) (Carlson and Dulany 1988). TPL is originally used for belief revision, which comprises three elements: (1) the association between a clue and a possible cause; (2) the forward implication from the possible cause to the clue; (3) the backward implication from the clue to the possible cause. The clues are circumstantial evidences like fingerprints, while the causes are suspects’ actions by which the fingerprints are left. When implementing diagnostic reasoning with TPL, the inference process of diagnostic reasoning can then be established through the linkage of aforementioned elements, as illustrated in Figure 5.1.

![Figure 5.1: The Inference Process of Diagnostic Reasoning.](image-url)
For software effort judgment, the first-hand clues are existing software projects, while the actual cause is the requirement of a new project. When doing backward implication, benefiting from the techniques of EBSE (Dybå, Kitchenham et al. 2005), the original clues can be collected and used to extract effort-related assertions. Note that, different from the work in (Menzies and Hihn 2006) that collects the actual effort of similar projects, EBSE used for circumstantial-evidence-based judgment focuses on the generic relationships between effort and different development actions, namely development experiences. When doing forward implication, on the other hand, the profile of a concrete project will be explored to identify possible development actions. The identified possible development actions can be used to build an association between previous projects and the current one to further facilitate effort judgment. In general, the association is built by a cascaded inference. In a cascaded inference, the conclusion of one inference acts as a premise for the subsequent inference, while the final conclusion will be the qualitatively estimated effort.

5.1.2 Circumstantial Evidences for Software Development

In software engineering, effort of a task is generally accounted by calculating how long and how many workers are needed to finish the task, and the unit can be person-day, person-month, or person-year. In other words, the amount of human activities in a project is proportional to the amount of effort required to finish the project. Therefore, for a certain software project, one basic circumstantial evidence (CE) can be:

- CE1. In general, the increase of required human activities in a project will have a proportional impact on the final effort.

Human activities include both physical and mental activities. Since software development is a knowledge-intensive undertaking, software product/service is a type of intellectual property produced by human mental activities. Unfortunately, within a given time span people have limited mental capability to deal with information (Globerson 1983). For every single person, the increased amount of information beyond a certain point may even defeat his/her mental ability, and hence result in errors (Miller 1956). As a result, the more information that exists in a project, the more people and human activities might be required to perform accurate manipulations. Together with CE1, therefore, we can find a new circumstantial evidence:
• CE2. In general, the increase of information in a project will have a proportional impact on the required human activities.

• CE2’. In general, the increase of information in a project will have a proportional impact on the final effort.

Additionally, complexity has been proved to be a significant and non-negligible factor that influences software development and maintenance (Francalanci and Merlo 2008). Meanwhile, the more complexity involved in a system, the more difficulty the designers or engineers have to understand the implementation process and thus the system itself (Cardoso 2005), and hence the greater mental effort people have to exert to solve the complexity (Globerson 1983).

![Hockey Stick Function](image)

**Figure 5.2: The Hockey Stick Function.**

The hockey stick function (Josuttis 2007) vividly depicts the relationship between complexity and effort of a software project, as illustrated in Figure 5.2. The amount of required effort may suddenly increase when the corresponding project exceeds a certain level of complexity. Overall, the circumstantial evidence related to complexity can be summarized as:

• CE3. In general, the increase of complexity in a project will have a proportional impact on the final effort.

When it comes to project complexity, one of the main contributors is the complexity of the methods that regard achieving the project goals (Turner and Cochrane 1993). The methods mentioned herein
generally consist of processes, tools, and techniques that are used to complete the corresponding project (Camci and T. Kotnour 2006). In particular, processes and techniques have been viewed as internal environment of a system (organization), while the system’s complexity is considered a response to the environmental complexity (Dooley 2002). Consequently, the complexity of processes and techniques involved in a software project will positively influence the complexity of the project. As for the tools, although the adoption of sophisticated tools usually implies a complex project, tools are essentially developed and used to save human activities. For a certain project, the more work the tools can fulfil, the less human activities the project will require. Overall, together with CE3, we can identify the circumstantial evidences:

- **CE4.** In general, the increase of process complexity in a project will have a proportional impact on the project complexity.

- **CE4’**. In general, the increase of process complexity in a project will have a proportional impact on the final effort.

- **CE5.** In general, the increase of difficulty of techniques in a project will have a proportional impact on the project complexity.

- **CE5’**. In general, the increase of difficulty of techniques in a project will have a proportional impact on the final effort.

- **CE6.** In general, the increase of work that tools can fulfill in a project will have an inversely proportional impact on the human activities.

- **CE6’**. In general, the increase of work that tools can fulfill in a project will have an inversely proportional impact on the final effort.

### 5.1.3 Circumstantial-Evidence-Based Judgment around Effort Factors of WSC

Following the inference process of circumstantial-evidence-based judgment explained in Section 5.1.1, the essential of judgment is to build association between circumstantial evidences and actual actions. To judge the partial effort around different factors of WSC, therefore, we should also investigate development actions related to the effort factors so as to associate them with those
identified circumstantial evidences. As a set of Context- and Process-related effort factors have been identified in Chapter 4, we can then explore profiles of these factors one by one.

**Orchestration and Choreography:** As analyzed previously, orchestration describes a centralized coordination of the involved Web services, while choreography represents decentralized multiparty collaborations. Since distributed processing would be inevitably more complicated than non-distributed processing (Josuttis 2007), generally speaking, for the same WSC project the choreography-based implementation will be more complex than the orchestration-based implementation. Meanwhile, as the current de facto standard of orchestrating Web services, BPEL stemmed from existing languages and tools and has been widely accepted, whereas the choreography language WS-CDL was developed without any prior implementation and is still far from mature (Barros, Dumas et al. 2006). Considering this technical influence, the implementation of choreography will be more difficult than that of orchestration. Consequently, if holding the other aspects of one particular WSC project constant, development actions (DA) can be compared between orchestration and choreography:

- **DA1.** In general, the implementation of choreography is more complex than that of orchestration.

- **DA2.** In general, the techniques used for choreography are more difficult than that for orchestration.

**Syntactic and Semantic compositions:** Since semantic Web and semantic Web services are proposed to automate service discovery, selection, composition and execution by adding the inherent meanings, human activities within semantic compositions will be decreased while the involved information will be increased. Considering the increased information is for machine interpretation rather than human intervention, developers do not need to deal with the increased information when composing Web services. Meanwhile, syntactic and semantic Web services share the unified Web infrastructure and both use markup language based techniques to describe information. It can then be stated that the difficulty levels of techniques adopted in both syntactic and semantic service compositions are similar. To summarize, we can assert:

- **DA3.** In general, syntactic WSC requires more human activities than semantic WSC.

- **DA4.** In general, the difficulty of the techniques used for syntactic WSC is similar to
that for semantic WSC.

RESTful and SOAP-based compositions: RESTful WSC integrates normally disparate Web resources to create a new application. SOAP/WS-* based WSC is a collection of related, structured activities or tasks that produce a specific service or product for a particular customer. Compared with RESTful WSCs, SOAP-based WSCs employ more sophisticated techniques including heavyweight protocols, a set of WS-* stack, and more Message Exchange Patterns (MEPs), which can satisfy more QoS requirements while also dealing with more information. Therefore, if the requirement of a particular WSC project can be satisfied by using either RESTful or SOAP-based approach, we can assert:

- **DA5.** In general, the techniques used for the SOAP-based implementation are more difficult than that for the RESTful implementation.

- **DA6.** In general, the SOAP-based implementation deals with more information than the RESTful implementation does.

Manual, Semi-Automated and Automated design-time compositions: During the design time of WSCs, the more automated the design processes are, the less human activities the compositions would require, and the less detailed information developers need to be concerned with. Considering the realization of automation usually requires assistant tools and more techniques, for example the Semantic Matching approach (Sirin, Hendler et al. 2003), we can also conclude that the more automated the design processes are, the more tools and more difficult techniques the compositions may adopt. Through investigating these development profiles, for a same composition task we can assert:

- **DA7.** In general, manual design-time composition requires more human activities than semi-automated design-time composition, while semi-automated design-time composition requires more human activities than automated design-time composition.

- **DA8.** In general, manual design-time composition deals with more information than semi-automated design-time composition does, while semi-automated design-time composition deals with more information than automated design-time composition does.
- DA9. In general, the techniques used for manual design-time composition are less difficult than that for semi-automated design-time composition, while the techniques used for semi-automated design-time composition are less difficult than that for automated design-time composition.

- DA10. In general, automated design-time composition adopts more tools than semi-automated design-time composition does, while semi-automated design-time composition adopts more tools than manual design-time composition does.

**Static and Dynamic runtime compositions:** If we emphasize the adaptation in both static and dynamic compositions during runtime, we can draw the same conclusions through the similar analysis as above. Note although dynamic runtime composition intuitively involves more complex concerns than static one does, in practice, dynamic runtime composition is generally supported by existing engines or tools while static runtime composition has to realize adaptation by adjusting Web services manually. Since the complexity of manual adaptation depends on the unpredictable situation in the future, we cannot simply claim that static runtime composition is more complex than dynamic runtime composition. Overall, we can assert:

- DA11. In general, static runtime composition requires more human activities than dynamic runtime composition.

- DA12. In general, static runtime composition deals with more information than dynamic runtime composition does.

- DA13. In general, the techniques used for static runtime composition are less difficult than that for dynamic runtime composition.


**One-Stop, Bridge and Double-Bridge process based compositions:** Considering that the One-Stop process delivers executable specifications, the Bridge process focuses on the abstract modeling, and the Double-Bridge process focuses on the composition requirement, approaches adopting the
One-Stop process have to deal with the most information while the Double-Bridge process based approaches deal with the least information for one certain task of WSC. Meanwhile, the Double-Bridge approaches have the longest processes while the One-Stop approaches have the shortest. However, we can imagine that both One-Stop and Bridge processes also contain two transformation procedures as well as the Double-Bridge process does. The intangible transformation procedures essentially take place as mental activities, while the tangible ones can be supported by tools. Therefore, it can be found that the Double-Bridge approaches require less human activities and use more tools, the One-Stop approaches require more human activities and use less tools, while the Bridge approaches are in the middle. When it comes to techniques, it is nearly impossible to compare the difficulty levels behind the process models with each other. Consequently, here we simply treat their difficulties similarly. Then for one certain task of WSC, the development actions we can find are:

- **DA15.** In general, One-Stop composition requires more human activities than Bridge composition, while Bridge composition requires more human activities than Double-Bridge composition.

- **DA16.** In general, One-Stop composition deals with more information than Bridge composition does, while Bridge composition deals with more information than Double-Bridge composition does.

- **DA17.** In general, Double-Bridge composition has more complex process than Bridge composition, while Bridge composition has more complex process than One-Stop composition.

- **DA18.** In general, the difficulty of the techniques used for One-Stop composition is similar to that for Bridge composition, and the difficulty of the techniques used for Bridge composition is similar to that for Double-Bridge composition.

- **DA19.** In general, Double-Bridge composition adopts more tools than Bridge composition does, while Bridge composition adopts more tools than One-Stop composition does.

To summarize, DA1~DA19 are analysis results drawn from different influences of factors on partial efforts of WSC. These analysis results can be viewed as abstracts of different development actions,
and act as possible inference bridges between real development actions and the identified circumstantial evidences. Benefiting from TPL based diagnostic reasoning, therefore, we can conveniently and qualitatively judge the effort of these WSC factors. For example, the forward implication from (DA1, DA2) to (CE3, CE4) can infer that choreography has more probability of requiring more effort than orchestration when implementing a particular WSC project, or “$E_{Ch} > E_{Or}$” for short. Similarly, we can also give qualitative effort judgment for the other WSC factors mentioned in this section, as shown in Table 5.1.

As circumstantial evidences increase the probability of judgments they support, we can adopt the result supported by major circumstantial evidences. Note here we suppose circumstantial evidences have the same weight. Therefore, we can give final judgments by analyzing the effort judgments listed in the right column of Table 5.1. For example, for a particular task of WSC, we believe that manual design-time composition requires more effort than semi-automated design-time composition while semi-automated design-time composition requires more effort than automated design-time composition, because such a conclusion is supported by more evidences than the others do.

### Table 5.1: Circumstantial-Evidence-Based Effort Judgment around Different Effort Factors of Web Service Composition

<table>
<thead>
<tr>
<th>Development Actions</th>
<th>Circumstantial Evidences</th>
<th>Effort Judgment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orchestration</td>
<td>(DA1, DA2)</td>
<td>$E_{Ch} &gt; E_{Or}$</td>
</tr>
<tr>
<td>Choreography</td>
<td>(DA3)</td>
<td>$E_{Sy} &gt; E_{Se}$</td>
</tr>
<tr>
<td></td>
<td>(DA4)</td>
<td>$E_{Sy} \approx E_{Se}$</td>
</tr>
<tr>
<td>Syntactic</td>
<td>(DA5, DA6)</td>
<td>$E_{So} &gt; E_{RE}$</td>
</tr>
<tr>
<td>Semantic</td>
<td>(DA7, DA8, DA10) (DA9)</td>
<td>$E_{Ma} &gt; E_{SA} &gt; E_{Au}$</td>
</tr>
<tr>
<td>RESTful</td>
<td>(DA11, DA12, DA14) (DA13)</td>
<td>$E_{Sa} &gt; E_{Dy}$</td>
</tr>
<tr>
<td>SOAP-based</td>
<td>(DA15, DA16, DA19) (DA17)</td>
<td>$E_{OS} &gt; E_{Br} &gt; E_{DB}$</td>
</tr>
<tr>
<td>Manual</td>
<td>(DA15, DA16, DA19) (DA17)</td>
<td>$E_{OS} &lt; E_{Br} &lt; E_{DB}$</td>
</tr>
<tr>
<td>Semi-Automated</td>
<td>(DA17)</td>
<td>$E_{OS} \approx E_{Br} \approx E_{DB}$</td>
</tr>
<tr>
<td>Automated</td>
<td>(DA18)</td>
<td></td>
</tr>
</tbody>
</table>

Note: $E_{Ch}$, $E_{Or}$, $E_{Sy}$, $E_{Se}$, $E_{So}$, $E_{RE}$, $E_{Ma}$, $E_{SA}$, $E_{Au}$, $E_{Sa}$, $E_{Dy}$, $E_{OS}$, $E_{Br}$, $E_{DB}$, $E_{OS}$, $E_{Br}$, $E_{DB}$.
5.2 Effort Judgment for Individual Web Service Composition Approaches

Given previous analysis, we can now judge the partial effort around those effort factors of WSC. Nevertheless, to further give effort judgment for a complete WSC approach, we have to take into account all those effort factors of WSC simultaneously. In fact, it is possible to combine some of the partial effort judgments for WSC according to the addition operation rule of inequalities – if two inequalities are of the same type, i.e., both greater of both less, adding the respective side results in the same type of inequality. For example, if holding the other aspects of one particular WSC project constant, we can claim that the implementation of manual design-time choreography requires more effort than that of automated design-time orchestration (manual design-time composition and choreography require more effort than automated design-time composition and orchestration respectively).

Following above idea of the effort judgment combination, we can build a checklist constituted by a set of qualitative effort judgment statements for individual and complete WSC approaches. In fact, using a checklist has been considered a simple way of utilizing experience and advocated as an efficient method of improving expert judgment processes when doing estimation (Furulund and Molokken-Østvold 2007). To facilitate building this qualitative effort judgment checklist, here we propose some symbols and rules as follow. For one certain task of WSC, we use $E_{F,CE}$ to represent the effort $E$ determined by factor $F$ when applying circumstantial evidence $CE$. Moreover, a score $S$ will be set for $E_{F,CE}$ to flag different effort determined by different but comparable factors when applying some circumstantial evidences. For convenience of calculation, the rules of score setting can be:

\[
\begin{align*}
S(E_{F1,CE}) &= 2, S(E_{F2,CE}) = 1 \quad \text{if } E_{F1,CE} > E_{F2,CE} \\
S(E_{F1,CE}) &= 1, S(E_{F2,CE}) = 1 \quad \text{if } E_{F1,CE} \approx E_{F2,CE} \\
S(E_{F1,CE}) &= 1, S(E_{F2,CE}) = 2 \quad \text{if } E_{F1,CE} < E_{F2,CE}
\end{align*}
\]

(5.1)

Note that if we use $E_F$ to represent the effort $E$ determined by factor $F$ under all the different but applicable hypotheses, then all the scores for $E_F$ under corresponding hypotheses can be summed up and represented as $S(E_F)$. Given these symbols and rules, we can qualitatively judge effort of WSC approaches from parts to whole along a quantitative way.
5.2.1 Comparison between Partial Composition Efforts Determined by Comparable Effort Factors

Benefiting from the symbols and rules proposed previously, we can conveniently obtain the comparison between partial composition efforts by qualitatively representing the analysis in Section 5.1.3 and calculating the corresponding effort scores.

**Orchestration and Choreography:** By using $For$ for representing the effort factor Orchestration and $Fch$ for Choreography, the effort comparison and scores can be listed in Table 5.2.

<table>
<thead>
<tr>
<th>Applied Circumstantial Evidence</th>
<th>Comparison</th>
<th>Scores</th>
</tr>
</thead>
<tbody>
<tr>
<td>CE3</td>
<td>$E_{For-CE3} &lt; E_{Fch-CE3}$</td>
<td>$S(E_{For-CE3})=1, S(E_{Fch-CE3})=2$</td>
</tr>
<tr>
<td>CE5'</td>
<td>$E_{For-CE5'} &lt; E_{Fch-CE5'}$</td>
<td>$S(E_{For-CE5'})=1, S(E_{Fch-CE5'})=2$</td>
</tr>
<tr>
<td>Total</td>
<td>$E_{For} &lt; E_{Fch}$</td>
<td>$S(E_{For})=2, S(E_{Fch})=4$</td>
</tr>
</tbody>
</table>

**Syntactic and Semantic compositions:** By using $Fsy$ for representing the effort factor Syntax and $Fse$ for Semantics, the effort comparison and scores can be listed in Table 5.3.

<table>
<thead>
<tr>
<th>Applied Circumstantial Evidence</th>
<th>Comparison</th>
<th>Scores</th>
</tr>
</thead>
<tbody>
<tr>
<td>CE1</td>
<td>$E_{Fsy-CE1} &lt; E_{Fse-CE1}$</td>
<td>$S(E_{Fsy-CE1})=1, S(E_{Fse-CE1})=2$</td>
</tr>
<tr>
<td>CE5'</td>
<td>$E_{Fsy-CE5'} = E_{Fse-CE5'}$</td>
<td>$S(E_{Fsy-CE5'})=1, S(E_{Fse-CE5'})=1$</td>
</tr>
<tr>
<td>Total</td>
<td>$E_{Fsy} &lt; E_{Fse}$</td>
<td>$S(E_{Fsy})=2, S(E_{Fse})=3$</td>
</tr>
</tbody>
</table>
RESTful and SOAP-based compositions: By using $F_{so}$ for representing the effort factor SOAP and $F_{re}$ for REST, the effort comparison and scores can be listed in Table 5.4.

Table 5.4: Effort Comparison between SOAP-based and RESTful Web Service Composition Approaches

<table>
<thead>
<tr>
<th>Applied Circumstantial Evidence</th>
<th>Comparison</th>
<th>Scores</th>
</tr>
</thead>
<tbody>
<tr>
<td>CE2'</td>
<td>$E_{F_{so}-CE2} &gt; E_{F_{re}-CE2}$</td>
<td>$S(E_{F_{so}-CE2})=2$, $S(E_{F_{re}-CE2})=1$</td>
</tr>
<tr>
<td>CE5'</td>
<td>$E_{F_{so}-CE5} &gt; E_{F_{re}-CE5}$</td>
<td>$S(E_{F_{so}-CE5})=2$, $S(E_{F_{re}-CE5})=1$</td>
</tr>
<tr>
<td>Total</td>
<td>$E_{F_{so}} &gt; E_{F_{re}}$</td>
<td>$S(E_{F_{so}})=4$, $S(E_{F_{re}})=2$</td>
</tr>
</tbody>
</table>

Manual, Semi-Automated and Automated design-time compositions: By using $F_{ma}$ for representing the effort factor Manual, $F_{sa}$ for Semi-Automated and $F_{au}$ for Automated, the effort comparison and scores can be listed in Table 5.5.

Table 5.5: Effort Comparison between Manual, Semi-Automated and Automated Design-Time Web Service Composition Approaches

<table>
<thead>
<tr>
<th>Applied Circumstantial Evidence</th>
<th>Comparison</th>
<th>Scores</th>
</tr>
</thead>
<tbody>
<tr>
<td>CE1</td>
<td>$E_{F_{ma}-CE1} &gt; E_{F_{sa}-CE1}$</td>
<td>$S(E_{F_{ma}-CE1})=2+2=4$</td>
</tr>
<tr>
<td></td>
<td>$E_{F_{ma}-CE1} &gt; E_{F_{au}-CE1}$</td>
<td>$S(E_{F_{ma}-CE1})=1+2=3$</td>
</tr>
<tr>
<td></td>
<td>$E_{F_{sa}-CE1} &gt; E_{F_{au}-CE1}$</td>
<td>$S(E_{F_{sa}-CE1})=1+1=2$</td>
</tr>
<tr>
<td>CE2'</td>
<td>$E_{F_{ma}-CE2} &gt; E_{F_{sa}-CE2}$</td>
<td>$S(E_{F_{ma}-CE2})=2+2=4$</td>
</tr>
<tr>
<td></td>
<td>$E_{F_{ma}-CE2} &gt; E_{F_{au}-CE2}$</td>
<td>$S(E_{F_{ma}-CE2})=1+2=3$</td>
</tr>
<tr>
<td></td>
<td>$E_{F_{sa}-CE2} &gt; E_{F_{au}-CE2}$</td>
<td>$S(E_{F_{sa}-CE2})=1+1=2$</td>
</tr>
<tr>
<td>CE5'</td>
<td>$E_{F_{ma}-CE5} &lt; E_{F_{sa}-CE5}$</td>
<td>$S(E_{F_{ma}-CE5})=1+1=2$</td>
</tr>
<tr>
<td></td>
<td>$E_{F_{ma}-CE5} &lt; E_{F_{au}-CE5}$</td>
<td>$S(E_{F_{ma}-CE5})=2+1=3$</td>
</tr>
<tr>
<td></td>
<td>$E_{F_{sa}-CE5} &lt; E_{F_{au}-CE5}$</td>
<td>$S(E_{F_{sa}-CE5})=2+2=4$</td>
</tr>
<tr>
<td>CE6'</td>
<td>$E_{F_{ma}-CE6} &gt; E_{F_{sa}-CE6}$</td>
<td>$S(E_{F_{ma}-CE6})=2+2=4$</td>
</tr>
<tr>
<td></td>
<td>$E_{F_{ma}-CE6} &gt; E_{F_{au}-CE6}$</td>
<td>$S(E_{F_{ma}-CE6})=1+2=3$</td>
</tr>
<tr>
<td></td>
<td>$E_{F_{sa}-CE6} &gt; E_{F_{au}-CE6}$</td>
<td>$S(E_{F_{sa}-CE6})=1+1=2$</td>
</tr>
<tr>
<td>Total</td>
<td>$E_{F_{ma}} &gt; E_{F_{sa}} &gt; E_{F_{au}}$</td>
<td>$S(E_{F_{ma}})=14$, $S(E_{F_{sa}})=12$, $S(E_{F_{au}})=10$</td>
</tr>
</tbody>
</table>
**Static and Dynamic runtime compositions:** By using $Fst$ for representing the effort factor Static and $Fdy$ for Dynamic, the effort comparison and scores can be listed in Table 5.6.

**Table 5.6: Effort Comparison between Static and Dynamic Runtime Web Service Composition Approaches**

<table>
<thead>
<tr>
<th>Applied Circumstantial Evidence</th>
<th>Comparison</th>
<th>Scores</th>
</tr>
</thead>
<tbody>
<tr>
<td>CE1</td>
<td>$E_{Fst-CE_1} &gt; E_{Fdy-CE_1}$</td>
<td>$S(E_{Fst-CE_1})=2$, $S(E_{Fdy-CE_1})=1$</td>
</tr>
<tr>
<td>CE2'</td>
<td>$E_{Fst-CE_2'} &gt; E_{Fdy-CE_2'}$</td>
<td>$S(E_{Fst-CE_2'})=2$, $S(E_{Fdy-CE_2'})=1$</td>
</tr>
<tr>
<td>CE5'</td>
<td>$E_{Fst-CE_5'} &lt; E_{Fdy-CE_5'}$</td>
<td>$S(E_{Fst-CE_5'})=1$, $S(E_{Fdy-CE_5'})=2$</td>
</tr>
<tr>
<td>CE6'</td>
<td>$E_{Fst-CE_6'} &gt; E_{Fdy-CE_6'}$</td>
<td>$S(E_{Fst-CE_6'})=2$, $S(E_{Fdy-CE_6'})=1$</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>$E_{Fst} &gt; E_{Fdy}$</td>
<td>$S(E_{Fst})=7$, $S(E_{Fdy})=5$</td>
</tr>
</tbody>
</table>

**One-Stop, Bridge and Double-Bridge process based compositions:** By using $Fos$ for representing the effort factor One-Stop model, $Fbr$ for Bridge model and $Fdb$ for Double-Bridge model, the effort comparison and scores can be listed in Table 5.7.

**Table 5.7: Effort Comparison between One-Stop, Bridge and Double-Bridge Process based Web Service Composition Approaches**

<table>
<thead>
<tr>
<th>Applied Circumstantial Evidence</th>
<th>Comparison</th>
<th>Scores</th>
</tr>
</thead>
<tbody>
<tr>
<td>CE1</td>
<td>$E_{Fos-CE_1} &gt; E_{Fbr-CE_1}$</td>
<td>$S(E_{Fos-CE_1})=2+2=4$</td>
</tr>
<tr>
<td>CE2’</td>
<td>$E_{Fos-CE_2'} &gt; E_{Fbr-CE_2'}$</td>
<td>$S(E_{Fos-CE_2'})=2+2=4$</td>
</tr>
<tr>
<td>CE4’</td>
<td>$E_{Fos-CE_4'} &lt; E_{Fbr-CE_4'}$</td>
<td>$S(E_{Fos-CE_4'})=1+1=2$</td>
</tr>
<tr>
<td>CE5’</td>
<td>$E_{Fos-CE_5'} = E_{Fbr-CE_5'}$</td>
<td>$S(E_{Fos-CE_5'})=1+1=2$</td>
</tr>
<tr>
<td>CE6’</td>
<td>$E_{Fos-CE_6'} &gt; E_{Fbr-CE_6'}$</td>
<td>$S(E_{Fos-CE_6'})=2+2=4$</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>$E_{Fos} &gt; E_{Fbr} &gt; E_{Fdb}$</td>
<td>$S(E_{Fos})=16$, $S(E_{Fbr})=14$, $S(E_{Fdb})=12$</td>
</tr>
</tbody>
</table>
5.2.2 Qualitative Effort Comparison between Different Web Service Composition Approaches

To realize the effort comparison between different WSC approaches, we should find a way to reflect the combined influences of different factors on the composition effort. Here we use the Addition and Multiplication principles in Combinatorics to inspire how to calculate the combined influences of different WSC factors. Considering different types of contexts can be superposed together to influence WSC, we define that the scores for different context types are accumulable in the Context dimension. Meanwhile, considering the Context and Process dimensions are independent of each other, we can further define that the effort scores are multipliable across different dimensions. As such, the effort score of a WSC approach can be expressed through the sum and product of the scores of the relevant effort factors. After filling the applicable circumstantial evidences and scores to the WSC classification matrix, we can achieve an effort-judgment-assistant table, as shown in Table 5.8.

Moreover, according to the combination of different effort factors identified in the classification matrix, there are totally 144 different types of WSC approaches \((3 \times 2 \times 2 \times 2 \times 3 \times 2 = 144)\). Therefore, if filling real WSC approaches to the effort-judgment-assistant table, we can further achieve an effort-judgment checklist of 144 types of WSC approaches, as demonstrated in Table 5.9. Benefiting from this checklist, we can conveniently judge the qualitative effort between different approaches for a certain WSC instance: one composition approach requires more effort than another if the former’s effort score is bigger than the latter’s.

However, it should be noted that the scores do NOT indicate any count of the amount of effort. Instead, these quantitative numbers are only used to facilitate qualitatively judging the effort of different composition approaches before implementing a WSC. Meanwhile, some other meaning of the effort scores can be further revealed. By investigating the result and procedure of the calculation of effort scores, as shown in Figure 5.3, we can find that the amount of applicable circumstantial evidences implies the times of comparisons, while the times of consistent comparisons is proportional to the difference between resulting scores of effort factors.
Here we define different comparisons are *consistent* when the same conclusion can be drawn in these comparisons by applying different hypotheses. For example, there are two consistent comparisons when applying circumstantial evidences CE3 and CE5’ to the compare between Orchestration and Choreography in Table 5.2. Since the consistent comparisons can help to confirm and reinforce the comparison result, the difference between scores of effort factors reflects the extent of our confidence in the effort judgment. When it comes to composition approaches, the difference between total effort scores is further magnified through the summation and multiplication by scores of effort factors. Therefore, to summarize, the larger difference that exists between two approach effort scores, the more confidence we will have in the judgment result.
## Table 5.8: Effort-Judgment-Assistant Table for Web Service Composition

<table>
<thead>
<tr>
<th>Category</th>
<th>Process</th>
<th>Pattern</th>
<th>Semiotics</th>
<th>Mechanism</th>
<th>Design Time</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Orchestration</td>
<td>Choreography</td>
<td>Syntax</td>
<td>Semantics</td>
<td>SOAP</td>
</tr>
<tr>
<td>One-Stop</td>
<td>CE1, 2', 3', 4', 5', 6'</td>
<td>S(E_{total})=16</td>
<td>S(E_{total})=2</td>
<td>S(E_{total})=4</td>
<td>S(E_{total})=2</td>
<td>S(E_{total})=3</td>
</tr>
<tr>
<td>Bridge</td>
<td>CE1, 2', 3', 4', 5', 6'</td>
<td>S(E_{total})=14</td>
<td>S(E_{total})=2</td>
<td>S(E_{total})=4</td>
<td>S(E_{total})=2</td>
<td>S(E_{total})=3</td>
</tr>
<tr>
<td>Double-Bridge</td>
<td>CE1, 2', 3', 4', 5', 6'</td>
<td>S(E_{total})=12</td>
<td>S(E_{total})=2</td>
<td>S(E_{total})=4</td>
<td>S(E_{total})=2</td>
<td>S(E_{total})=3</td>
</tr>
</tbody>
</table>

- 101 -
### Table 5.9: A Sample of Effort-Judgment Checklist for Web Service Composition

<table>
<thead>
<tr>
<th>Process</th>
<th>Category</th>
<th>Applied Evidences</th>
<th>Score</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Pattern</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Orchestration</td>
</tr>
<tr>
<td>One-Stop</td>
<td>WSC Approach 1 (e.g. BPEL programming)</td>
<td>CE1, 2’, 4’, 5’, 6’</td>
<td>16</td>
<td>√</td>
</tr>
<tr>
<td></td>
<td>WSC Approach 2 (e.g. Casati et al. 2000)</td>
<td>CE1, 2’, 4’, 5’, 6’</td>
<td>16</td>
<td>√</td>
</tr>
<tr>
<td></td>
<td>WSC Approach 3 (e.g. Lathem et al. 2007)</td>
<td>CE1, 2’, 4’, 5’, 6’</td>
<td>16</td>
<td>√</td>
</tr>
<tr>
<td>Bridge</td>
<td>WSC Approach 4 (e.g. Skogan et al. 2004)</td>
<td>CE1, 2’, 4’, 5’, 6’</td>
<td>14</td>
<td>√</td>
</tr>
<tr>
<td></td>
<td>WSC Approach 5 (e.g. Timm and Gannod 2007)</td>
<td>CE1, 2’, 4’, 5’, 6’</td>
<td>14</td>
<td>√</td>
</tr>
<tr>
<td></td>
<td>WSC Approach 6 (e.g. Anzboeck and Dustdar 2005)</td>
<td>CE1, 2’, 4’, 5’, 6’</td>
<td>14</td>
<td>√</td>
</tr>
<tr>
<td>Double- Bridge</td>
<td>WSC Approach 7 (e.g. Sirin et al. 2004)</td>
<td>CE1, 2’, 4’, 5’, 6’</td>
<td>12</td>
<td>√</td>
</tr>
<tr>
<td></td>
<td>WSC Approach 8 (e.g. Valero et al. 2009)</td>
<td>CE1, 2’, 4’, 5’, 6’</td>
<td>12</td>
<td>√</td>
</tr>
<tr>
<td></td>
<td>WSC Approach 9 (e.g. Tabatabaei et al. 2008)</td>
<td>CE1, 2’, 4’, 5’, 6’</td>
<td>12</td>
<td>√</td>
</tr>
</tbody>
</table>

### Calculation Examples

- S(WSC Approach 1) = S(E\text{or}) × (S(E\text{or}) + S(E\text{fa}) + S(E\text{for}) + S(E\text{fa})) = 16 × (2 + 2 + 4 + 14 + 7) = 16 × 29 = 464
- S(WSC Approach 2) = S(E\text{or}) × (S(E\text{or}) + S(E\text{fa}) + S(E\text{for}) + S(E\text{fa})) = 16 × (2 + 2 + 4 + 14 + 5) = 16 × 27 = 432
- S(WSC Approach 3) = S(E\text{or}) × (S(E\text{or}) + S(E\text{fa}) + S(E\text{for}) + S(E\text{fa})) = 16 × (2 + 3 + 2 + 12 + 7) = 16 × 26 = 416
- S(WSC Approach 4) = S(E\text{or}) × (S(E\text{or}) + S(E\text{fa}) + S(E\text{for}) + S(E\text{fa})) = 14 × (2 + 2 + 4 + 14 + 7) = 14 × 29 = 406
- S(WSC Approach 5) = S(E\text{or}) × (S(E\text{or}) + S(E\text{fa}) + S(E\text{for}) + S(E\text{fa})) = 14 × (2 + 3 + 4 + 14 + 5) = 14 × 28 = 392
- S(WSC Approach 6) = S(E\text{or}) × (S(E\text{or}) + S(E\text{fa}) + S(E\text{for}) + S(E\text{fa})) = 14 × (2 + 2 + 4 + 14 + 5) = 14 × 27 = 378
- S(WSC Approach 7) = S(E\text{or}) × (S(E\text{or}) + S(E\text{fa}) + S(E\text{for}) + S(E\text{fa})) = 12 × (2 + 3 + 4 + 10 + 7) = 12 × 26 = 312
- S(WSC Approach 8) = S(E\text{or}) × (S(E\text{or}) + S(E\text{fa}) + S(E\text{for}) + S(E\text{fa})) = 12 × (4 + 3 + 4 + 10 + 7) = 12 × 28 = 336
- S(WSC Approach 9) = S(E\text{or}) × (S(E\text{or}) + S(E\text{fa}) + S(E\text{for}) + S(E\text{fa})) = 12 × (4 + 3 + 4 + 10 + 5) = 12 × 26 = 312

- **Note:** The formulas provided are illustrative examples and may not reflect the exact calculations used in the table. The specific formulas used in the table may vary based on the criteria and context provided.
5.3 Effort Judgment for Web Service Composition based SOA Implementations

Following the analysis in Chapter 3, we first narrow down our focus on individual WSCs when judging effort of WSC-based SOA projects. Since we have been able to qualitatively compare different effort judgments of different WSC approaches with each other through a checklist, we can now switch our focus on complete WSC-based SOA implementations.

As mentioned previously, WSC-based SOA implementations are one particular SOA implementation type that does not take into account the migration of legacy systems or development of new Web services. Accordingly, we can only concentrate on the effort of composing Web services when doing effort judgment for WSC-based SOA implementations. The corresponding algorithm of D&C based effort judgment shown in Table 3.2 can then be simplified as shown in Table 5.10.

| Table 5.10: Algorithm of D&C based Effort Judgment for WSC-based SOA Projects |
|-----------------------------|-------------------|
| 1)  //Treat the project as the highest-level service S to analyze. |
| 2)  double SoaEffortJudgment (service S) { |
| 3)    double effort = 0; |
| 4)    Determine the type of S according to the design and real situations. |
| 5)    if (S is decomposable) { |
| 6)        Divide S into component services at lower level. |
| 7)        foreach component service in S |
| 8)            effort += SoaEffortJudgment (component service); |
| 9)            effort += The effort of service integration for component services in S; |
| 10)   } |
| 11)  return effort; |
| 12) } |

Similarly, within the body of SoaEffortJudgment function, unless the input service is already available, the effort of the service implementation will be recursively calculated by analysing and judging the effort of the compositions of component Web services. When applying the simplified algorithm to the aforementioned RailCo Ltd. case, the procedure of effort judgment follows a WSC tree as illustrated in Figure 5.4, and narrows down to following seven steps:

1) Divide the Automation System into an Invoice Processing Service and a PO Processing Service.
2) Divide the Invoice Processing Service into its four basic component services.
3) Judge the cost and effort of composing the above four component services into the Invoice Processing Service.
4) Divide the PO Processing Service into its two basic component services.
5) Judge the cost and effort of composing the above two component services into the PO Processing Service.
6) Judge the cost and effort of integrating the Invoice Processing Service and PO Processing Service into the Automation System.
7) Sum up all the judgment results to calculate the total cost and effort of the Automation System development.

![Figure 5.4: WSC Tree of Automation System of RailCo Ltd.](image)

Since different basic services defined in Section 3.3.2 are all supposed available in the WSC-based SOA implementation type, composing Web services is the only activity that brings development effort. Benefiting from the way of D&C, our focus on effort judgment for the whole WSC-based SOA implementations has been narrowed down to that for individual WSCs. Consequently, the problem of effort judgment for WSC-based SOA implementations can be eventually tackled by using the D&C strategy. Here we still adopt the RailCo Ltd. case to show how to realize the final effort judgment.

By applying the algorithm listed in Table 5.10 and using the aforementioned effort-judgment checklist, we can conveniently calculate the complete effort score for the WSC-based SOA implementation of RailCo Ltd. Considering that Web service uses an abstract interface to conceal
the technical details like the implementation language, deployment platform and underlying communication protocol, in practice, different composite Web services in one SOA project can be implemented with different techniques under different contexts by different people. Consequently, we use arbitrary combinations of WSC approaches to represent different implementations of this WSC-based SOA system.

Table 5.11: Effort Comparison between Implementation Proposals for the Automation System of RailCo Ltd.

<table>
<thead>
<tr>
<th>Proposal</th>
<th>Implementation Effort Scores</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposal 1</td>
<td>$S(\text{WSC approach 1}) + S(\text{WSC approach 1})$ + $S(\text{WSC approach 1}) = 464 + 464 + 464 = 1392$</td>
</tr>
<tr>
<td>Proposal 2</td>
<td>$S(\text{WSC approach 7}) + S(\text{WSC approach 7})$ + $S(\text{WSC approach 7}) = 312 + 312 + 312 = 936$</td>
</tr>
<tr>
<td>Proposal 3</td>
<td>$S(\text{WSC approach 2}) + S(\text{WSC approach 5})$ + $S(\text{WSC approach 9}) = 432 + 406 + 312 = 1150$</td>
</tr>
</tbody>
</table>

Suppose there are three candidate implementation proposals listed in Table 5.11: Proposal 1 uses WSC approach 1 three times, Proposal 2 uses WSC approach 7 three times, whereas Proposal 3 uses a combination of WSC approach 2, 5 and 9. Through the implementation effort scores, we can find that the second one requires the least effort among three proposals. Moreover, compared with the Proposal 3, we have more confidence to say that the Proposal 1 requires more effort than the Proposal 2. As such, developers can give qualitative effort judgment for different proposals before implementing a WSC-based SOA project.

5.4 Summary

Considering that it is difficult to collect enough real data of various WSC-based SOA implementations, we try to use general software development experiences as circumstantial evidences to do qualitative effort judgment. Based on the circumstantial-evidence-based effort judgment, we develop a method of assigning scores to effort factors of WSC, which can be used to facilitate the qualitative effort judgment between different effort factors, between different type of WSC approaches and even between different SOA implementation proposals. Note that the calculation rule here for counting the effort scores of different types of WSC approaches are mainly inspired by the Addition and Multiplication principles in Combinatorics: (1) We apply an Addition-principle-like method to the effort factors in the Context dimension of the classification
matrix, considering that different partial efforts of one WSC within different contexts are mutually exclusive, while different contexts are accumulable. (2) We apply a Multiplication-principle-like method to the effort factors across those two dimensions of the classification matrix, considering that the Process dimension is independent of the Context dimension, and one process of WSC can be realized within any combination of contexts. However, this calculation rule still suffers from intuition, and will be further validated and revised through empirical study in our future work.
CHAPTER 6

Discussion and Conclusion

With more and more availability of services, WSC-based SOA implementations have increasingly become a significant type of SOA projects in practice. However, effort estimation for such a type of SOA project can still be limited because of the numerous and various approaches to WSC. Through viewing WSC-based SOA system from a perspective of mechanistic organization, this book borrows D&C as the generic strategy to narrow down the problem of effort judgment for the entire SOA implementation to that for individual WSCs. Moreover, benefiting from an effort-oriented classification matrix and a set of effort-related evidences, we assign scores to effort factors of WSC assisted by a set of rules. These effort scores are used to facilitate qualitatively judging different effort between different types of WSC approaches, and eventually construct an effort checklist for WSC approaches. Finally, this effort checklist can be used together with D&C algorithm to realize the qualitative effort judgment for WSC-based SOA implementations.

To discuss and conclude this book, this chapter uses three sections to summarize our research achievements, the limitations of our current work, and the future work. Each section is structured following the sequence of our work.

6.1 Research Achievements

Following the structure of this book, our research achievements can be separated into three parts. Firstly, the strategy of effort judgment for generic SOA implementations is proposed. To propose this strategy, we try to borrow ideas from organization theory domain to SOA domain, which is inspired by the “learning by analogy” research method (Fox 1981). Consequently, in addition to the strategy itself, a comprehensive analogy between SOA systems and organizations is drawn and justified. Benefiting from the analogy, a set of detailed work we have done is: (1) Four strategies that have been successfully applied to organizational area are identified to support SOA
implementations. Moreover, the novel methodology of investigating technology independent strategies for implementing SOA is outlined by treating SOA implementation as organizational activities that follow the procedure of organization design. (2) A complexity measurement framework is established for comprehending and measuring the organizational complexity of SOA implementation that comprises four dimensions: Structure, Environment, Business and Resource. (3) More importantly, inspired by the working mechanism of mechanistic organizations, we propose to follow the D&C way to analyze and judge the effort when implementing SOA. Overall, to the best of our knowledge, this is the first time to use organization theory to inspire the research into service-oriented software engineering.

Secondly, a set of effort factors of WSC is identified. To identify effort factors of WSC, we build up a brand new classification scheme – effort-oriented classification matrix – for WSC approaches. This classification matrix uses clarified terminology to overcome some weaknesses of existing WSC classification work, which distinguishes between the Context and Process dimensions. The Context dimension is aimed at analyzing the environment influence on the effort of WSC, while the Process dimension focuses on the influence of different composition procedures on the effort. In particular, through abstracting the common procedures and activities of WSCs, we have developed four different composition approach models that we name One-Stop, Bridge, Double-Bridge, and Adaptation to familiarize researchers and engineers with current strategies for composing Web services.

Thirdly, a method is developed to realize the qualitative effort judgment for WSC-based SOA implementations. This method is based on circumstantial-evidences-based judgment and it is developed along three progressive steps: effort judgment around separated effort factors, for individual WSC approaches, and finally for WSC-based SOA implementations. When it comes to circumstantial-evidences-based effort judgment, we treat the development experiences deposited in human knowledge as circumstantial evidences in the context of software engineering, and then use them together with rational inference to give qualitative effort estimates of different implementation influences of different WSC effort factors. To judge the effort of individual WSC approaches, we use a set of symbols and rules to assign scores to effort factors and then combine these factor scores into WSC approach scores. As a result, an effort-judgment checklist is established for 144 types in total of WSC approaches. Benefiting from this effort-judgment checklist, we are able to use the D&C strategy to calculate different implementation effort scores for different proposals of a certain WSC-based SOA system. Moreover, the difference between effort scores of implementation proposals reflects the extent of our confidence when determining the effort tradeoff for the
WSC-based SOA system.

6.2 Response to Research Questions

Based on the achievements of this research, we can now make response to those four predefined research questions. To avoid duplication of effort, here we give brief and precise answers to those questions.

- Q1: How to unfold research into the topics related to service-oriented computing?

Considering the limitations to empirically implementing various SOA projects by ourselves, we have employed learning by analogy with organization theory as an efficient way to inspire the research into SOA domain.

- Q2: What is the generic strategy for effort judgment for WSC-based SOA implementations?

Inspired by the management mechanism of traditional mechanistic organizations, WSC-based SOA systems can be thought of a D&C way to achieve business goals. When building WSC-based SOA systems, we can therefore follow the D&C way as a generic strategy to analyze and judge the implementation effort.

- Q3: What factors should be concerned when judging the effort of individual WSC approaches?

Through presenting a novel classification matrix of WSC approaches, we can view different context types and process models as different factors that have different influences on the composition effort. The context types are Pattern, Semiotics, Mechanism, Design Time and Runtime. The process models include One-Stop, Bridge and Double-Bridge.

- Q4: How can these effort factors be used to realize the effort judgment for WSC-based SOA implementations?

By distinguishing direct evidences (similar projects’ or activities’ actual effort) and
circumstantial evidences (development experiences deposited in human knowledge) in the context of software effort estimation, we have proposed the circumstantial-evidence-based method to facilitate qualitatively judge the effort of different proposals before implementing a WSC-based SOA project.

### 6.3 Limitations

The limitations of our work can be also specified following the book structure. Firstly, the research into SOA domain by analogy of organization theory is still at an initial stage. This book only focuses on some fundamental studies of the organization theory, and we have not reached the essential and sophisticated organization knowledge, for example organizational behaviour and organization development. Meanwhile, the evaluation would be a challenge to aforementioned analogy work. The studies in organization theory domain, for example the strategies of organization design, have been accepted for general organizations. However, it could be not convincible to use only one SOA implementation or two to validate that the analogues in SOA domain are generally acceptable.

Secondly, there could be some other WSC effort factors that we have not found. Considering the effort factors are identified from the classification of WSC approaches, we cannot guarantee the effort factors have been exhaustively examined because it is difficult to exhaustively explore existing approaches to give a full scope of classification scheme for WSC.

Thirdly, the implementation of effort judgment discussed in Chapter 5 still suffers from two weaknesses of subjectiveness. (1) The circumstantial evidences are collected subjectively when judging effort around different WSC factors. In fact, the evidence collection plays a vital role in the proposed circumstantial-evidence-based effort judgment. However, there is a lack of guidance for evidence collection in the context of software effort estimation. Considering the number of evidences will directly influence the effort scores calculated in this book, subjective service collection may inevitably result in noisy score data. Therefore, it is necessary to employ an objective procedure to collect evidences before doing judgment. (2) The rule of combining partial effort factor scores into a whole is established intuitively. Inspired by the Addition and Multiplication principles in Combinatorics, an Addition-principle-like method is applied to the effort factors in the Context dimension while a Multiplication-principle-like method is applied to the effort factors across two dimensions of the WSC classification matrix. Although some initial
analyses have been given to justify this calculation rule, the result would be more reliable if the rule can be supported empirically.

### 6.4 Future Work

To overcome the limitations of our current work, some future work we plan to do can be listed respectively as follow. Firstly, the deeper research will be unfolded into inspiration from organization theory to service-oriented computing, which will mainly focus on the analogy of organizational behaviour and organization development. For example:

- To use well-known human organizational structures to inspire the deployment of SOA systems.
- To choose social laws to simplify SOA systems.
- To use dependency theory of social interaction to explain how to achieve social goals of SOA systems.

In particular, we plan to pay more attention to the research opportunity of the interdisciplinary area between WSC patterns (Orchestration and Choreography) and organization types (Mechanical and Organic organizations). Through the initial investigation, we have found that the fundamental ideas and notions behind these different concepts in two disciplines are nearly the same. In other words, we can precisely view Orchestration and Choreography from the perspective of Mechanical organization and Organic organization respectively, and the concepts still make sense even if we simply replace their descriptions with each other:

Secondly, given the importance of understanding the art of the state of WSC and the lack of a full scope of classification scheme, we propose a systematic literature review (SLR) of existing WSC approaches. Through this SLR, we expect to verify and validate current classification work, and reveal a complete classification scheme of existing WSC approaches. The result of this SLR can also be used to inspire new research opportunities in academia and assist implementation analysis in industry.

Thirdly, we will continue to develop and improve the circumstantial-evidences-based effort judgment. With regard to the circumstantial evidences, we plan to adopt two methods to realize the objective evidence collection: On the one hand, we can use data mining to achieve effort-related assertions from the public data sets through qualitative comparison between real project data. On the other hand, SLR, as the main methodology applied for EBSE, can be another effective approach
to the secondary empirical evidence collection. As a result, the circumstantial evidences in the context of effort judgment will be accumulated and deposited as general knowledge to further guide and assess individual expert judgments. Moreover, we also plan to use propositional calculus to formalize the rational inference taking place during judgment processes. Overall, the essential target of this part of our work is to use the circumstantial-evidences-based effort judgment as complementary to expert judgment to facilitate and revise the final quantitative effort estimation for new software projects.
Bibliography


Jamil, E. 2009. SOA in asynchronous many-to-one heterogeneous bi-directional data syn-
chronization for mission critical applications. WeDoWebSphere.
http://wedowebsphere.de/news/1528/-SOA%20in%20Asynchronous%20Many-toone%20Hete-

Proceedings of International Conference on Computer Science and Software Engineering

Johnson, P. M., Moore, C. A., Dane, J. A. and Brwer, R. S. 2000. Empirically guided software effort

McGraw-Hill.

Media, Inc.


Kaynak, H. 2003. The relationship between total quality management practices and their effects on


Presentations of Australian Conference on Software Measurement (ACOSM 2007).

Khalilian, M., Boroujeni, F. Z., Mustapha, N. and Sulaiman, M. N. 2009. K-means divide and
conquer clustering. Proceedings of the 2nd International Conference on Computer and


IEEE Computer Society, pp. 70-79.
Pautasso, C. 2009. RESTful Web service composition with BPEL for REST. Data and Knowledge


Bibliography


