ESSAYS ON E-SERVICE MANAGEMENT: IT SERVITIZATION UNDER SOA AND CRM DOMAINS

by

Noyan Ilk

A Dissertation Submitted to the Faculty of the

DEPARTMENT OF MANAGEMENT

In Partial Fulfillment of the Requirements
For the Degree of

DOCTOR OF PHILOSOPHY
WITH A MAJOR IN MANAGEMENT INFORMATION SYSTEMS

In the Graduate College

THE UNIVERSITY OF ARIZONA

2012

THE UNIVERSITY OF ARIZONA
GRADUATE COLLEGE

As members of the Dissertation Committee, we certify that we have read the dissertation

prepared by Noyan Ilk

entitled Essays on E-service Management: IT Servitization under SOA and CRM Domains

and recommend that it be accepted as fulfilling the dissertation requirement for the

Degree of Doctor of Philosophy

_____ Date: 07/30/2012
Paulo B. Goes

_____ Date: 07/30/2012
J. Leon Zhao

_____ Date: 07/30/2012
Benn R. Konsynski

_____ Date: 07/30/2012
David E. Pingry

_____ Date: 07/30/2012
Mingfeng Lin

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copies of the dissertation to the Graduate College.

I hereby certify that I have read this dissertation prepared under my direction and recommend that it be accepted as fulfilling the dissertation requirement.

_____ Date: 07/30/2012
Dissertation Director: Paulo B. Goes

STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at the University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

SIGNED: Noyan Ilk

ACKNOWLEDGEMENTS

First of all, I would like to thank my advisors, Professors Paulo Goes and J. Leon Zhao for their invaluable guidance and supervision as well as their friendship during my doctoral studies. It has been a great privilege to have studied under their collective mentorship and I am indebted to them for all the efforts to help me become a better academic and researcher.

I would also like to thank Professor Benn Konsynski for his continuous support on my research as a committee member throughout the last five years. I would also like to thank my other committee members Professors David Pingry and Mingfeng Lin for their valuable feedback and contributions to my progress in the doctoral program. Additionally, I would like to acknowledge Professor Mohan Tanniru for initiating the service-oriented transformation project that I have been involved in.

Finally, I would like to express gratitude to my parents Yuksel and Aytac Ilk and to my brother Dr. Dilhan Ilk for their unconditional love and limitless support throughout my life. I am especially grateful to my brother for pioneering the path to higher education in our family. I have been merely following his steps on this everlasting path.

TABLE OF CONTENTS

TABLE OF CONTENTS − *Continued*

TABLE OF CONTENTS − *Continued*

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES − *Continued*

ABSTRACT

We are living in a world of service economy. Global markets have radically transformed from product-based industrial structures to service-based post-industrial ones over the past fifty years. Information Technology (IT) has catalyzed a significant portion of this transformation. Advances in IT have not only alleviated the accessibility of existing service systems, but also enabled Servitization of products and commodities that were delivered through traditional (mostly physical) mediums.

Ironically, IT itself has been a commodity that has met its own share of Servitization. Hardware computing resources such as storage and processors have been virtualized, whereas software and media content have been delivered as service contracts through distributed networks. Causing a paradigm shift on how IT is delivered and used, Servitization of IT is expected to impose technical, economical and managerial challenges in various business domains of organizations.

In this dissertation, I develop novel methods and policies to overcome such challenges. By conducting four closely related studies, I address common IT Servitization problems encountered in the service-oriented architecture (SOA) and customer relationship management (CRM) domains. Specifically, I make the following contributions: (1) in study one, I work on the efficient creation of software services out of legacy software, by developing an approach to annotate source code components of an enterprise IT system with business semantics. The approach facilitates source code reuse in developing new web services. (2) In study two, I develop a financial valuation model for SOA investments. The model quantifies evident and elusive costs and benefits of

SOA and supports managerial decision making regarding the investment. (3) In study three, I investigate the deployment of online service channels by evaluating the impact of priority-based admission control policies for live-chat communication services. I show that under imperfect profiling of customer types, reserve-type admission control policies may have negative consequences for the entire system. (4) In study four, I investigate the value of adding flexibility to the infrastructure of online services in live-chat contact centers. Contrary to widely accepted results from the service management literature, I find out that cross-training of agents in large contact centers suffers from switching costs as well as capacity shifting inefficiencies.

Methods, models and policies proposed in this dissertation are expected to contribute towards understanding the short-term applicability and long-term impact of service-orientation and IT Servitization in business organizations.

1 INTRODUCTION

Over the last 50 years, there has been a substantial growth in value contribution of service industries to the economies of industrialized nations. A wide range of industries including education, healthcare and business operations have been witnessing skyrocketing demands for the services that they provide (Dietrich & Harrison, 2006). As of 2011, the service sector is largest sector in the U.S. economy, accounting for 80 percent of its gross domestic product. This accounts roughly for $11.5 trillion worth of output, and a growth of 50 percent over the last 15 years (Central Intelligence Agency, 2012).

A considerable portion of this growth has been attributed to the advances in Information Technology (IT) (Montoya, Massey, & Khatri, 2010). In particular, the rise of network based technologies such as the Internet has alleviated the accessibility of service systems while increasing their productivity and efficiency. In addition, technologies such as web services and distributed computing have played a prominent role in transforming the product based IT functionalities into service based ones − a phenomenon that we call IT Servitization. For example, a wide range of commodities from media content to hardware computing capacity can now be packaged as services and delivered over the Internet through a variety of service contracts.

The term Servitization first originated in the manufacturing domain (Vandermerwe & Rada, 1988). Within this literature, it is commonly defined as the process of adding service components to products that are offered by manufacturing firms (Baines, Lightfoot, Benedettini, & Kay, 2009; Vandermerwe & Rada, 1988). From an

Information Systems (IS) perspective, this concept resembles the notion of service orientation, which provides a framework for the commoditization of IT hardware and software as well as IT-enabled business processes (Demirkan, Kauffman, Vayghan, Fill, Karagiannis, & Maglio, 2008). In simple terms, service orientation help transform silo based monolithic IT systems into network based services that are autonomous, loosely coupled and reusable. Each service represents an IT function that can be published and discovered via standard interfaces. These services can be composed together and reused in different logical flows to support a number of business processes.

The potential benefits of such an approach are vast in theory, ranging from reduced IT maintenance to increased business agility (Tiwana & Konsynski, 2010), from improved interoperability to seamless application integration (Zhao, Tanniru, & Zhang, 2007). Yet, compared to its importance on facilitating the growth of service-based IT economies, relatively little systematic research has been conducted on service orientation and IT Servitization. In fact, a group of scholars have been cultivating the term "Service Science, Management and Engineering" (SSME) as a scientific discipline to promote research in this direction (Chesbrough & Spohrer, 2006; Spohrer & Maglio, 2007; Chesbrough H. , 2005).

My work has been an answer to this call for research. This dissertation consists of a collection of closely related studies in which I have developed novel methods and policies that can enable and/or expedite the Servitization of IT in business organizations. Within the broad context of IT functionality in organizations, I have focused my attention into two distinct business domains, namely Service-oriented Architectures (SOA) and

Customer Relationship Management (CRM). These domains were not selected arbitrarily. The former one possesses technical and economical challenges to IT Servitization, whereas the latter one includes challenges from operational and managerial perspectives. More specifically, SOA domain requires solutions that can alleviate the difficulty of creating services from existing IT components, while justifying the costs associated with the SOA implementation. On the other hand, CRM domain requires solutions that can facilitate the use of new IT based communication channels (e.g. live-chat, e-mail) to improve customer services.

In parallel with the multi-disciplinary nature of service science, I have employed research methodologies from technical, financial and operational perspectives and conducted four studies to provide solutions to these challenges. An overview of these four studies including their challenges, methods and contributions is given in Figure 1.

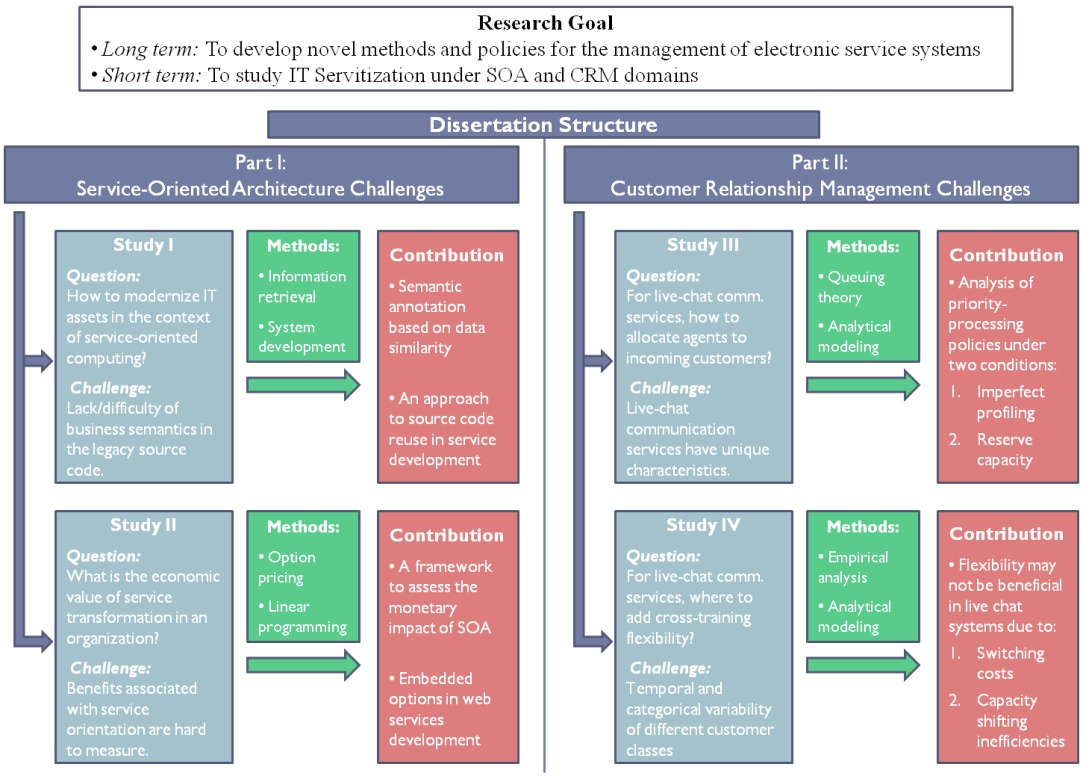

Figure 1. Overview of the dissertation

The first two studies fall under the domain of SOA. In study one, I have proposed an information retrieval based approach to enrich existing source code components of enterprise IT systems with semantics acquired from business process definitions. Enterprise systems are mission critical software platforms that need to be modernized continuously in order to respond to changing business demands. Service-based transformation is an appealing approach that can introduce flexibility and standardization into an enterprise system, thereby easing the future modernization efforts. SOA offers the promise of leveraging the value of enterprise systems through source code reuse. However, one problem to overcome is the lack of business semantics in the existing

source code. The method I propose uses the textual similarity between data entities used in the source code of the enterprise system and WSDL representations of existing business services. Data entities are collected by parsing the source code and WSDL documents and compared with each other via a string similarity algorithm. For identified matches, the source code is annotated with respect to its relevant service definition. The outcome of this process is an enriched source code model that can be queried for business semantics in developing new web services. I have validated the applicability of the approach using a case study on SAP's Enterprise System Framework. The results have indicated that the proposed approach is useful for creating service-based knowledge in the source code components of an existing enterprise system.

In study two, I have proposed a financial valuation model for SOA investment in an organization. SOA requires significant up-front investments, and in return, promises a vast array of benefits. Unfortunately, in contrast to the costs of the investment, monetary benefits associated with SOA are more difficult to measure. For one reason, benefits such as increased agility are elusive in nature, making it harder to define metrics for their calculation. For another, SOA value is realized in long term under managerial flexibility, and traditional capital budgeting methods often fail to capture flexibility when valuing investments. I have used a dynamic model based on a set of embedded real options to value four types of benefits from SOA: 1) follow-up application implementations, 2) reusability of developed IT services, 3) reduced time to market of applications and 4) substitution of modular IT components. A unique feature of the model was to consider multi-stage interaction between reusability flexibility in an initial application and reduced

costs of development in consequent applications. To illustrate its applicability, I have employed the financial valuation model at a real life investment decision. The results have showed that the model is valuable for supporting managerial decision making with respect to SOA investments.

The third and fourth studies fall under the domain of CRM. In study three, I have investigated priority based admission control policies for live-chat services that are employed at e-commerce websites. Live-chat is a new communication channel that is becoming popular for customer – company interactions. With increasing traffic on e-commerce websites, providing such live-chat services requires a good allocation of service resources to serve the customers. When resources are limited, firms may consider employing priority-processing and reserving resources for high-value customers. First, I have modeled a reserve-based priority-processing policy for e-commerce systems that have imperfect customer classification. Specifically, the model considered two policy decisions: 1) the number of agents exclusively reserved for high-value customers, and 2) the configuration of the classification system. I have derived explicit expressions for average waiting times of high-value and low-value customer classes and defined a total waiting cost function. Through numerical analysis, I have studied the impact of these two policy decisions on average waiting times and total waiting costs. Initial results have shown that reserving agents for high-value customers may have negative consequences for such customers under imperfect classification. I have also studied the interaction between the two policy decisions and discussed how one decision should be modified with respect to a change in the other one. The findings have shown that a firm may

increase its reserve capacity along with classifier sensitivity in order to keep the waiting costs at minimum.

In study four, I have investigated the impact of introducing agent flexibility to live-chat queueing systems. A unique feature of live-chat service is the multi-tasking ability. Under multi-tasking, a consequence of increasing agent flexibility is to process not just multiple customers but also multiple classes of customers together. Through empirical analysis, I have first showed that agent flexibility under multi-tasking property can lead to depreciation in agents' performances. Considering this effect, I have developed a model for a multi-class, multi-server live-chat system with processor sharing discipline. This model can be used to analyze the impact of flexibility in the system under varying conditions. A particular point of interest is to observe the interaction between task switching costs and average waiting times. The results have shown that, while increasing agent utilization, flexibility can also generate negative effects on the system due to excessive task switching as well as inefficient capacity sharing.

In the following chapters of the dissertation, I discuss each of these studies in more detail. In the 6th and the final chapter, I conclude the dissertation by highlighting the contributions and outlining a future research plan.

2 STUDY ONE – DISCOVERING SEMANTICS IN THE SOURCE CODE FOR ENTERPRISE IT SYSTEM MODERNIZATION

2.1 Introduction

In order to reuse an item – whether it is a material, a part, a tool or a piece of software – in a different structure, one first needs to learn about the meaning and the functionality (i.e. semantics) of the item. While this sounds trivial for daily items and objects, it is harder for source code components of large-scale software such as an enterprise IT system. Enterprise systems are mission critical software platforms that enable organizations to integrate and coordinate their business processes. Consisting of hundreds of business functions, these systems provide a single architecture that is central to the organizations and ensure that information can be shared across all functional levels and management hierarchies (Stohr & Zhao, 2001).

Enterprise systems need to evolve continuously to support changing business practices. System modernization is a way to facilitate this evolution. Modernization refers to the changes in the system architecture (e.g.., system restructuring, functional enhancements or new software attributes) while conserving significant portions of the software. Modernization of enterprise systems creates new opportunities for organizations to transform their existing IT assets into the service-oriented architectures (SOA). SOA provide a design framework to encapsulate business logic through loosely coupled, independent services. These high level services represent repeatable business tasks and can be accessed on demand over a network (Peltz, 2003). Under the SOA

notion, individual services can be assembled together to compose complex business functionality, enabling organizations to quickly adapt to changing conditions and requirements.

However, there are practical challenges to realize this notion. The most important issue is the absence of business semantics in the low-level system source code. Source code is a collection of statements and declarations – a means of communication with the computing machine through structure and syntax. Analyzing the source code may reveal information about the structural aspects and the architecture of the system (i.e., how the code is working), but not about the reason for the existence of a particular piece of code (i.e., what the code is about) (Kuhn, Ducasse, & Girha, 2005). Unfortunately, without sufficient semantic knowledge on code functionality in the context of business processes, it would be impossible to utilize source code components in new services development.

In this study of the dissertation, I propose an approach to overcome this problem. The approach is based on the idea that the gap between the two ends of an enterprise system – (1) services as processes and (2) source code – can be bridged via the similarity of data definitions used in both ends. In the next section, I describe the approach in more detail including the steps required and how to automate these steps. In Section 2.3, I provide a case study for the validation of the approach.

2.2 A Model for Semantic Enrichment of Source Code

Effective enterprise system modernization requires understanding the business logic as well as discovering structural aspects of the system. Software reverse engineering has so far focused on the second issue. Advancements in this area have made

it possible to recover structure from even the most complicated systems. By examining the source code, it is possible to obtain a system's data structure and control flow, identify functional dependencies and interrelationships between code components and create representations of the system at a higher abstraction level (Chikofsky & Cross, 1990).

However, as mentioned by DeBaud et al. (1994), knowledge of system structure alone is not sufficient to understand the business logic hidden inside the system, just as knowing the rules of grammar for English are not sufficient to understand essays or stories. This is because the source code itself may not provide any semantics for the business services supported by the system. To deal with this problem of reverse engineering, I propose the semantic enrichment model shown in Figure 2.

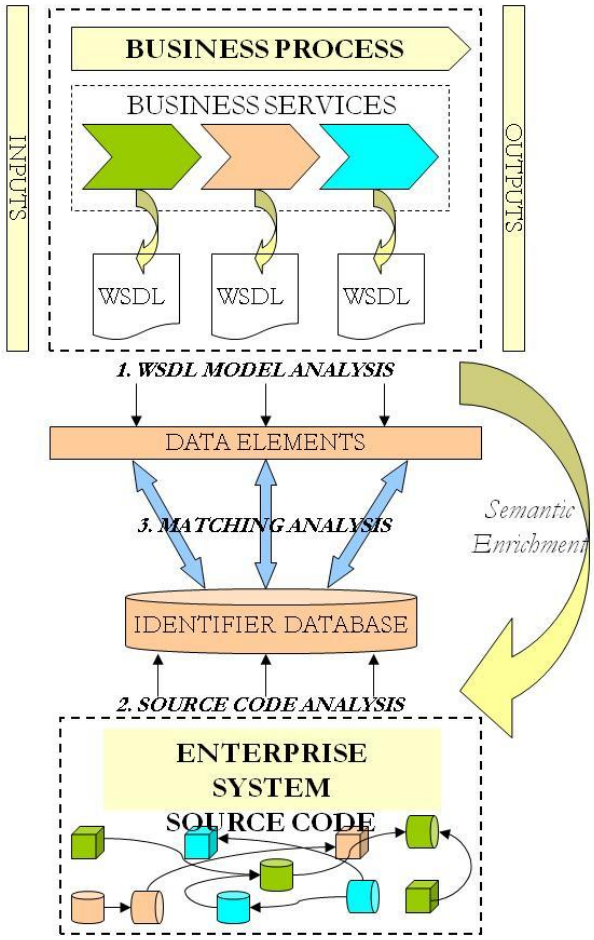

Figure 2. Semantic enrichment of enterprise system source code

The process starts with analyzing the formal models of business services. In SOA, each activity in the business process is represented by a business service in the service model. By analyzing business services, it is possible to obtain information about the inputs, outputs and operations of business processes. A reliable source to access this information is WSDL representations of business services. WSDL is an XML based language that provides a model for describing web services and how to access them (World Wide Web Consortium, 2001). A WSDL document contains the definition of

operations offered by the service, as well as the data that these operations accept and return (Vara, de Castro, & Marcos, 2005). The standard XML grammar used in WSDL enables for extraction of semantic information about services (i.e., about processes) in a structured manner. WSDL parsing tools can be developed to automate this process (Balasooriya, Padbye, Prasad, & Navathe, 2005). In the first step I identify (World Wide Web Consortium, 2001) and extract the data names being communicated in a given service. *Operation*, *message* and *types* elements in the WSDL format are used for this purpose.

In the second step, I look into the source code of the enterprise system. The purpose of this step is to develop an index for efficient source code search and component classification. In the context of reverse engineering, my point of interest is a source code domain representation (i.e., source model) that will allow for fast and accurate search of identifiers (i.e., tokens that name the components) in the code. Such a representation should also be capable of providing relational information (such as location in the code or dependencies) about search results. To accomplish this, I analyze the source code lexically and syntactically and store the extracted information in a relational database. Lexical analysis classifies the strings in the source code into tokens and syntactic analysis helps to parse the token sequence to reveal the syntactic structure of the system. The database provides access to retrieve data related source code identifiers (i.e., variables and parameters) and their relations automatically.

The final step is searching for equivalences between data names extracted from WSDL documents and data identifiers uncovered from the source code. Finding an exact

match is not expected at this point, since WSDL documents and the source code may include conceptually equivalent but syntactically different tokens. Determining the equality match between entities in different domains is a well studied problem in literature that has been explored by different communities such as statistics, databases, software comprehension and artificial intelligence (Singla & Domingos, 2006). In this case, the problem takes the shape of establishing identity equivalence between a pair of tokens where there is no other feature than the token string itself to support the solution process. I make use of string similarity metrics to overcome this problem. String similarity metrics help to quantify the similarity exhibited by two strings composed of symbols from a finite alphabet (Kondrak, 2005). There are several different types of string similarity metrics proposed in literature, including edit-distance metrics, common character comparisons, q-gram processing, tokenization-based metrics and hybrid methods (Cohen, Rayikumar, & Fienberg, 2003). Building on a common character metric called Jaro-Winkler measure; I employ a rule based algorithm to compare strings in the service and the source code domain.

Once a similarity match between tokens has been discovered, I annotate the identifier record with its relevant WSDL data specific information (i.e., the data name and the service used) in the database. At the end of this process, the database gets expanded with business semantics for matched identifiers. Below, I discuss each step in the model in more detail.

2.2.1 WSDL Model Analysis

WSDL is an XML based language that provides a model for describing web services and how to access them. WSDL describes services as collections of network endpoints or ports, which support the operations that interact by exchanging messages, as shown in Figure 3.

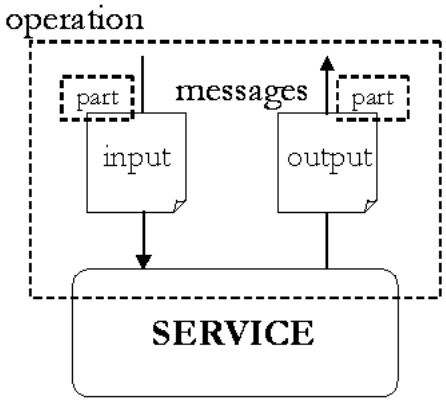

Figure 3. Service interaction

In a WSDL document, there are seven major elements that are used in the service description process (World Wide Web Consortium, 2001):

- Types: The element providing data type definitions used to describe the messages exchanged.
- Message: The element representing an abstract definition of the data being transmitted. A message consists of logical parts, each of which is associated with a definition within some type system.
- Operation: The element representing an abstract definition of an action supported by the service.

- PortType: The element which is a set of abstract operations supported by one or more endpoints.
- Binding: The element which specifies concrete protocol and data format specifications for the operations and messages defined by a particular portType.
- Port: The element which specifies an address for a binding, thus defining a single communication endpoint.
- Service: The element which is used to aggregate a set of related ports.

For the model, I am interested in finding the data names (i.e., tokens that name data elements) used in a particular service, from the service's WSDL document. The process is started by looking into the portType element, which describes the operation that the service realizes. The operation component in portType groups the set of messages that will be interchanged between the service provider and the requester. Every time an operation is used, the requester interchanges a set of messages with the service provider (Vara, de Castro, & Marcos, 2005). At this point, I identify two containers: (1) the input message tag and (2) the output message tag that carry data related information in the WSDL document.

Being used in operation, a message defines the data format to a single request or response in the communication. Each possible request or response is described by a message element in WSDL. The message element defines a logical message in terms of one or more parts. Each part might correspond to either procedure-oriented or document-oriented information that the service sends or receives. The part may represent a

procedure call and its parameters or an actual document being exchanged (Lemahieu, 2001). At this stage, I identify two containers: (1) the message element tag and (2) the part element tag that enclose data related information.

Referred from the part element, the type associated to a part can be an XML Schema Definition (XSD) base type (e.g., int, float, string, etc.), or any one of the types defined in the types element. In the last case, the data type can be defined by means of a type or an element attribute, from one of the schemas referenced in the WSDL document (Vara, de Castro, & Marcos, 2005). From this definition, I recognize that the types element tag also enclose data related information.

Overall, I identify the elements Table 1 in a WSDL document that contain information about the data used in a particular service:

Table 1. Data elements in a WSDL document

<<portType/operation/input>>
<<portType/operation/output>>
<<message>>
<<message/part>>
<<types>>

The next step is to look into the attributes of these elements. For this reason, XML syntax between the element tags needs to be understood. In WSDL, the grammar for defining a request-response operation is shown in Figure 4 (World Wide Web Consortium, 2001).

```
<wsdl:definitions .... >
  <wsdl:portType .... > *
    <wsdl:operation name="nmtoken"
parameterOrder="nmtokens">
      <wsdl:input name="nmtoken"?
message="qname"/>
      <wsdl:output name="nmtoken"?
message="qname"/>
    </wsdl:operation>
  </wsdl:portType >
</wsdl:definitions>
```

Figure 4. Request-response operation

Here, the input and output elements specify the abstract message format for the request and response, respectively. Operations refer to the messages involved using the message attribute of type QName. The name attributes of the input and output elements provide a unique name among all input and output elements within the enclosing port type.

For <<message>> and <<message/part>> elements, the syntax is shown in Figure 5.

```
<definitions .... >
  <message name="nmtoken"> *
    <part name="nmtoken" element="qname"?
type="qname"?/> *
  </message>
</definitions>
```

Figure 5. <<Message>> element

Here, the message name attribute provides a unique name among all messages defined within the enclosing WSDL document, whereas the part name attribute provides a unique name among all the parts of the enclosing message.

The <<types>> element encloses data type definitions that are relevant for the exchanged messages. Here, the type can be an XSD simpleType or a complexType using a *QName*. A general syntax for a <types>> element is shown in Figure 6.

```
<definitions .... >
  <types>
    <xsd:schema .... />*
     <element name="qname"?>
        <simpleType name="qname"?>
        </simpleType>
        <complexType name="qname"?>
          <all>
            <element name="qname"?
type="string"/>
          </all>
        </complexType>
      </element>
  </types>
</definitions>
```

Figure 6. <<Types>> element

In a WSDL document where all the elements mentioned above exist, a WSDL parser should be able to perform the following two operations:

(1) Identify and capture the following data elements:

- Input/output name
- Input/output message
- Message name
- Part name
- Part element
- SimpleType name
- ComplexType name
- Type element name

(2) Create a set from these elements with no duplicates and no namespace reference members.

To illustrate this process, I provided a sample WSDL document in Figure 7 that models a service for providing stock quotes (World Wide Web Consortium, 2001). This service supports a single operation called "*GetLastTradePrice"*, which is deployed using the SOAP 1.1 protocol over HTTP. The request takes a ticker symbol of type string, and returns the price as a float.

Figure 7. Stock quote service WSDL document

In this example, there are two messages for the <<operation>> element "*GetLastTradePrice*". The input message is "*GetLastTradePriceInput*" with a namespace assigned using "tns" prefix. The output message is "*GetLastTradePriceOutput*" with, again, a namespace assigned using "tns" prefix. These messages are elaborated in the <<message>> element section, where it is seen that message name "*GetLastTradePriceInput*" consists of a <<part>> with a name "*body*" and an element "*TradePriceRequest*" with a namespace assigned. Similarly, for the output message, the <<part>> name is "*body*" and the element name is "*TradePrice*". Finally, I refer to the <<types>> element, where element names "*TradePriceRequest*", "*tickerSymbol*", "*TradePrice*" and "*price*" can be identified.

In total, the WSDL parser should be able to capture the following data names from this WSDL document: "*tns:GetLastTradePriceInput*", "*tns:GetLastTradePriceOutput*", "*GetlastTradePriceInput*", "*body*", "*xsd1:TradePriceRequest*", "*body*", "*xsd1:TradePrice*", "*TradePriceRequest*", "*tickerSymbol*", "*TradePrice*" and "*price*". After stripping away the namespace identifiers and duplicates, the parser should be able to create the following set: S = {*GetlastTradePriceInput, GetLastTradePriceOutput, body, TradePriceRequest, TradePrice, tickerSymbol, price*}. The result of this process – the set – contains the data entities that are related to the particular service of stock quotation.

2.2.2 Source Code Analysis

Reverse engineering methodology facilitates program comprehension by creating higher-level representations of the system. A basic method of presenting higher-level

representations is based on creating source models, which comprises a collection of components (e.g., functions, files, variables, parameters, etc.), a set of relations between the components (e.g., ‘‘function calls function’’’) and a set of attributes of these components and relations (e.g., ‘‘function call includes a parameter pass’), to represent the system (Guo, Atlee, & Kazman, 1999).

A way to obtain a source model is to analyze a system’s source code structure via compiler construction techniques (van, Klint, & Verhoef, 1997). A compiler is a program that can read a program in source language and translate it into target language (Aho, Lam, Sethi, & Ullman, 2006). This translation process consists of four stages (Aho, Lam, Sethi, & Ullman, 2006; Wirth, 1996):

(1) Lexical Analysis: Reading the stream of characters building the source program and grouping these characters into token objects. Identifiers consisting of letters and digits, numbers consisting of digits, delimiters and operators consisting of special characters are recognized in this stage.

(2) Syntax Analysis (i.e., Parsing): Using the tokens produced by lexical analysis to create a tree-like intermediate representation that depicts the grammatical structure of the token stream.

(3) Semantic Analysis: Verifying the source program for semantic consistency with the language definition.

(4) Code Generation: Generating a sequence of instructions taken from the instruction set of the target language, on the basis of representations from syntax analysis.

There are many techniques and tools for static source model extraction, and these are largely divisible into two classes: those based on parsing and those based on lexical techniques (Kazman & Carriére, 1999). The source model can be created based on lexical analysis by specifying regular expressions to be matched to textual information within the code components (Pinzger, Fischer, Gall, & Jazayeri, 2002). Lexical analyzers are fast, easy to use and versatile. However, they lack the capability of recognizing syntactic constructs such as call graphs (Murphy & Notkin, 1996). Parsing techniques, on the other hand, can recognize and extract syntactic constructs that are specified by context free grammars. By using grammar specifications, parsers are able to create more abstract representations of the source code (i.e., source models). However, one difficulty with parsers is to perform a complete parse on every character stream in the source program, which makes them slow and more prone to syntax errors.

In the Source Code Analysis step, my goal is to have a source model that will allow for fast and accurate search of source code identifiers, as well as providing dependency information. A hybrid technique that combines the regular expression search characteristic of lexical analyzers with syntactic construct characteristic of traditional parsers is a promising way to accomplish this goal.

I adopt the Lexical Source Model Extraction (LSME) approach defined by Murphy and Notkin (1996) to implement for this step. LSME is a user driven approach that takes lexical specifications and the system source code as inputs to create a representation of the system source code (i.e., source model) as shown in Figure 8.

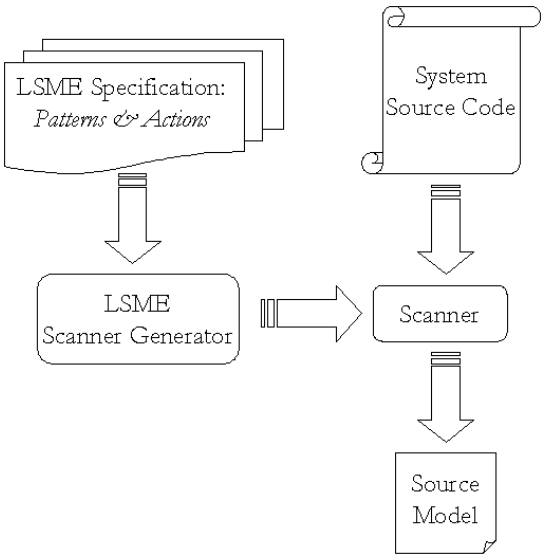

Figure 8. LSME approach (adapted from Murphy and Notkin 1996)

In the first step, LSME asks for users to create a lexical specification of the system. In particular, the users define the following two inputs:

(1) Patterns: Patterns use regular expressions to describe the constructs of interest in system source code. An example pattern can be shown as following: *[<type>] <functionName> \ ([{<formalArg> } +] \ ) [{<type> <argDecl> ; } +] \ { <calledFunctionName> \ ([{ <parm> } +] \ )*. This pattern is used to extract a static call relation between functions. It specifies that a function definition consists of an optional type specification, followed by the name of the function, a left parenthesis, an optional list of formal arguments, a right parenthesis, an optional list of declarations of the types of the formal arguments (each of which is terminated by a semicolon), and an opening curly brace for the start of the function body. The function body is child pattern to identify the called function's name.

(2) Actions: Actions are the operations to execute when a pattern is matched to information in an artifact being scanned. The action code can access the value of the scanner variables matched to scanned tokens and perform operations such as writing to the local output stream. For example, the action command: "*write (functionName, "calls", calledFunctionName)*" would write out a stream of the form "function calls function" when static calls are matched in the source.

Overall, for lexical analysis, the specification serves the purpose of creating regular expressions in terms of relevant tokens. For syntactic analysis, it is useful for describing relations between constructs that can only be recognized after a match to a regular expression definition.

In the second step, LSME proposes to generate a scanner from patterns and actions that reads in a sequence of artifact files and produces a stream of local output. In the generic LSME approach, the scanner translates each pattern into separate deterministic finite state machines. The scanner is responsible for executing the appropriate state machines and recording the paths taken through the machines. The outcome of this process would be the source model.

For this approach, I employ a different implementation of the scanning process. After the scanner's pass through the source code, I import the extracted data into a database. The attributes in the database are defined by the user as representing component types and their relations. At the end of the scanning process, no source model is generated automatically, but only the database is populated with information extracted from the source code, as shown in Figure 9.

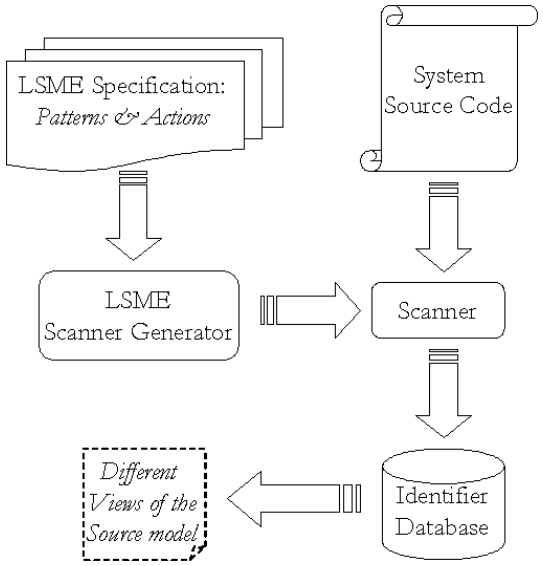

Figure 9. Modified LSME approach

Using database implementation is a design decision that comes with tradeoffs. The drawback is that it requires user intervention on two additional stages: (1) creating and modifying the database structure, (2) querying the database to retrieve the component related information. On the other hand, the advantage is that it can generate a number of different viewpoints for the same source model. Based on his/her intention, the user can query one part of the database and retrieve specific information about the source model. This functionality is useful for the approach, since only data related source code components (i.e., variables and parameters) are used in the Matching Analysis step.

2.2.3 Matching Analysis

Determining the equality match between syntactically different entities is a well studied problem in literature that has been explored by different communities such as statistics, databases, software comprehension and artificial intelligence. Depending on its

characteristics and the domain, the problem can take different formulations and names such as entity resolution, record linkage, object identification, alias detection and so on. Some of the solution methods include using more than one feature to create relations, discovering relational clues via inductive logic, or applying similarity measures (Singla & Domingos, 2006).

In the final step of the model, I am interested in establishing identity equivalence between the data names extracted from the WSDL documents and data related identifiers (i.e., variable and parameter tokens) uncovered from the source code. The solution process is restricted with the condition that there can be no other feature than name properties (i.e., strings) of entities to support the matching process. A more formal definition of the problem can be given as:

Definition 1 (Identity Equivalence Problem): Given two strings *S* and *T* from a finite alphabet $\sum$ (i.e., *S*, *T* $\in \sum^*$), and a Boolean match function *M* over $\sum^*$ x $\sum^*$; the identity equivalence problem is to determine if *M*(*S*, *T*) = *True*.

In the Matching Analysis step, I employ string similarity metrics to perform as the match function *M*. String similarity metrics help to quantify the typographical similarity exhibited by two strings composed of symbols from a finite alphabet (Kondrak, 2005). The match or no match decision is determined on whether the similarity measure value of a given string set (i.e., sim_{ST}) is greater than *r*, where *r* is an arbitrary threshold value.

There are several different types of string similarity metrics proposed in literature, including edit-distance metrics, common character comparisons, q-gram processing, token-based metrics and hybrid methods (Cohen et al. 2003). Edit-distance is based on

the minimum number of elementary edit operations (e.g., insert, delete, replace or transpose) needed to transform one string to another; whereas common character comparisons take the number and order of common characters between given strings into account (Kondrak, 2005). Levenstein distance and Monge-Elkan function can be given as examples of edit-distance metrics, while Jaro metric can be given as common character comparison example. In q-gram processing, positional q-grams are obtained and compared by sliding a window of length q over the characters of both strings (Gravano, et al., 2001). In token-based metrics such as Jaccard similarity or TF-IDF, strings are considered as multisets of words and the words' occurrence / co-occurences in these sets are considered. Hybrid methods such as level 2 Monge-Elkan or soft TF-IDF combine edit-distance measures and tokenization by breaking strings into sub-strings (Cohen et al. 2003).

Commercial or open source toolkits are available to run several types of string similarity metrics automatically. While it is easy to employ these toolkits such as SimMetrics (Chapman, 2007) or SecondString (Cohen, Rayikumar, & Fienberg, 2006) for case specific matching problems, our intention is to establish a theoretical structure for discovering semantic relations between the services and source code components of an enterprise system. Unique characteristics of both domains require for a tailored matching algorithm rather than using generic similarity metrics. For example, entities in WSDL documents are usually named with a full-text vocabulary bearing clear semantic information. The reason is simple; these documents are published in Universal Description Discovery and Integration (UDDI) registries for reusability purposes. On the

other hand, source code components are rarely named with user readability and informativeness in mind. In many systems, programmers name code components using abbreviations of full entity words (Lawrie, Field, & Binkley, 2007). In such cases when there are conceptual differences in the naming process, generic string similarity metrics may be ineffective to provide accurate matching results.

To illustrate this issue with an example, suppose that we want to compare the WSDL data name string "*TradePriceRequest*" with two source code identifiers strings, "*TradePriceRespond*" and "*TPReq*". Levenstein distance similarity scores for the pairs "*TradePriceRequest vs. TradePriceRespond*" and "*TradePriceRequest vs. TPReq*" are 0.71 and 0.29, respectively. Given a threshold value of 0.7, we establish an equivalence relationship between "*TradePriceRequest*" and "*TradePriceRespond*", whereas we reject the "*TPReq*" as a match. This decision is problematic, because while "*TPReq*" seems to be an abbreviation of "*TradePriceRequest*", there is a significant semantics difference between "*TradePriceRespond*" and "*TradePriceRequest*".

To overcome this problem, I introduce the common sub-sequence concept into the matching process of the Matching Analysis step:

Definition 2 (Sub-sequence): Given a sequence $X = x_1, x_2, \dots, x_k$; another sequence $Z = z_1, z_2, \dots, z_m$ is a sub-sequence of X if there exists a strictly increasing sequence $i_1, i_2, \dots, i_m$ of indices of X such that for all j = 1, 2, …, m, there is $x_{i_j} = z_j$. Given two sequences X and Y, a sequence Z is a common sub-sequence of X and Y if Z is a sub-sequence of both X and Y (Kondrak, 2005).

Proposition 1 (Abbreviation): All abbreviations that do not change the order of characters in a sequence are sub-sequences of that sequence. For example, "*TPReq*" is a sub-sequence of "*TradePriceRequest*", whereas *"PTReq"* is not.

Using sub-sequences can help us to avoid false positive matches – strings that carry semantics differences, but achieve high similarity scores. Avoiding false positives gives flexibility to change the *r* threshold value without flooding on results, which in turn helps to catch the abbreviated tokens. I propose the following rule to insert a common sub-sequence criterion into the matching process:

Proposition 2 (Common sub-sequence): If a string *T* from the source code identifier set is not a sub-sequence of a string *S* from the WSDL data name set, then $sim_{ST} = 0 \ \therefore M(S,T) = False.$

Combining the common sub-sequence rule with string similarity, we can increase the quality of results in matching between WSDL data names and source code identifiers. For string similarity computation, I choose the Jaro-Winkler metric to employ. Jaro-Winkler is a variant of the Jaro common character comparison, which can be defined as follows:

Definition 3 (Jaro Similarity Metric): Given strings $s = a_1, a_2, \dots, a_K$ and $t = b_1, b_2, \dots, b_L$, define *a* character a_i in *s* to be common with *t* there is a $b_j = a_i$ in *t* such that $i - H \leq j \leq i + H$, where $H = \frac{min(|s|,|t|)}{2}$. Let $s' = a'_1, \dots, a'_K$ be the characters in *s* which are common with *t* (in the same order they appear in *s*) and let $t' = b'_1, \dots, b'_L$ be analogous; define a transposition for s', t' to be in position i such that $a'_i = b'_i$. Let

$T_{S',t'}$ be half the number of transpositions for s', t'. The Jaro similarity metric for the *s* and *t* string set (Jaro, 1995):

$$Jaro(s,t) = \frac{1}{3} x \left(\frac{|s'|}{|s|} + \frac{|t'|}{|t|} + \frac{|s'| - T_{s',t'}}{|s'|} \right) \quad (2.1)$$

Jaro-Winkler extension to Jaro metric modifies the weights of poorly matching pairs *s*-*t* that share a common prefix. The output score is simply adjusted as follows:

$$Jaro - Winkler\,(s,t) = Jaro\,(s,t) + \left(\frac{P'}{10} x (1 - Jaro\,(s,t)) \right) \quad (2.2)$$

where *P'* = the longest common prefix of *s* and *t*, *P*, times a constant scaling factor, *f*, for how much the score is adjusted upwards for having a common prefix (Chapman, 2006).

Based on the definitions stated above, the function code in Table 2 can be used to determine match or no match decision for a given string set.

Table 2. Match function

```
function M(S, T): boolean
  for each pair (S, T) do
    SS: sub-sequence set of S;
    if T ∉ SS
      sim_ST = .0;
    else
      sim_ST = Jaro − Winkler(S, T);
    end if
  end for
  for each sim_ST do
    if sim_ST ≥ r  // r is the match threshold
value //
      return True;
    else
      return False;
    end if
  end for
end function
```

A *True* result returning from this function indicates that there is identity equivalence between the WSDL data name and source code identifier being compared and a mapping is established. In such a case, the approach conducts a final operation – it imports the following information into the relevant entries of the database: (1) WSDL data name, and (2) the service description and the process/activity the service corresponds to. At the end of this process, the database gets expanded with service specific knowledge for matched identifiers. After this point, any source model queried from the database including matched identifiers would carry a level of business semantics.

2.3 Validation of the Model

I employed the approach in a systems modernization project involving SAP's Enterprise System Framework. The goal of the project was to build a search portal for application developers to locate the source code components that may be reused in new services development. The input queries for the search function were the abstract service descriptions in textual form and the expected outputs were: (1) a list of data related identifiers ordered by relevancy to the search input and, (2) a list of modularization units (i.e., functions, and subroutines) that contain the top three relevant data related identifiers. The relevancy value for each identifier was solely based on match function values and is not a measure of search performance. The total relevancy was calculated by dividing the number of matched words between the input text and attribute data cells to the total number of words in the attribute data cells.

The case study involved analyzing the *Sales Order Processing (SOP)* business process component in the *Enterprise Resource Planning (ERP)* suite. *Sales Order Processing* handles customers' requests to a company for delivery of goods or services at a certain time. It enables the provision of applications that a seller can use to create, display and change sales orders (Gabriel, 2008). The three operations: (1) create sales order, (2), read sales order, and (3) change sales order can be considered as three of the elementary services that assemble the *Sales Order Processing* process.

SAP's *Enterprise Services Repository (ESR)* stores definitions and metadata of enterprise services provided by the SAP. I retrieved the WSDL representations of the services: (1) create sales order, (2), read sales order, and (3) change sales order from ESR. Parsing the WSDL documents to identify the data names used in these services produced the results in Table 3. A sample data names set for the three services is given in table 4.

Table 3. WSDL parsing results

	Create Sales Order	Read Sales Order	Change Sales Order
number of operation elements	6	6	6
number of message elements	6	6	6
number of types elements	739	1,075	477
total number of data elements	751	1,087	489

Table 4. WSDL sample data names

Create Sales Order	Read Sales Order	Change Sales Order
SalesOrderItemID	SalesOrderItemID	SalesOrderItemID
ProductPartyID	ProductPartyID	ProductPartyID
PriceDate	VersionID	LogItemNote
BuyerDocument	Status	CurrencyCode
Quantity	ParentItemID	DecimalValue
ShippingPointParty	AlternateItemID	CategoryCode
MaterialID	Telephone	schemeAgencyID
BuyerParty	FormOfAddress	RejectReasonCode

Further, I applied the Source Code Analysis step to obtain the source code domain representation of the SAP's ERP system. Patterns for the following components of the SAP source code: (1) function module, (2) subroutine, (3) import parameter, (4) export parameter, (5) internal table, and (6) global table were generated for the lexical specification input. Four actions: (1) function call, (2) subroutine perform, (3) parameter pass and (4) table access were also specified in the lexical specification. 28,228 source code files were processed in the LSME scanner generated from these patterns and actions. Extracted data from the LSME parsing were imported to an Oracle database. Results from the parsing process and a summary of database metadata are given in Table 5.

Table 5. Source code parsing results

Source Code Component	Number of occurrence in the code
Function Module	10,022
Subroutine	18,143
Parameter	72,251
Table	11,727
Relationship:	
Function Call	21,587
Subroutine Perform	30,082
Parameter Pass	39,235
Table Access	11,959

Overall, 83,978 data related identifiers from parameter and table source code components were retrieved from the database for Matching Analysis.

In the Matching Analysis step, data names extracted through WSDL parsing of create, read and change sales order services were compared with data related identifiers extracted through source code parsing using the match function in Table 2. The implementation of the match function was done under Java environment. 0.8, 0.85 and 0.9 values were used for the threshold value – *r* and results were analyzed manually to decide on the specific *r* value to be applied in the project. Eventually, 0.85 was chosen as the threshold value and the database was populated with results from the Matching Analysis. A sample set of the match function results are given in Table 6.

Table 6. Sample match function results

Data Identifier from the Source Code	Data Name from Services	*r* Value	Result
REFERENCE	ReferenceID	0.96	True
INDIC	Indicator	0.92	True
PROPVALUE	PropertyValue	0.89	True
SALESORG	SalesOrganisationID	0.88	True
STARTLINE	StartTime	0.91	False (failed sequence rule)
ROUTINE	RouteID	0.90	False (failed sequence rule)
STATE	Status	0.89	False (failed sequence rule)
SCHEMAPROG	schemeAgencyID	0.84	False (failed r threshold)
LANGMATCH	languageCode	0.82	False (failed r threshold)

The final output of the project was a web-based search portal that takes user defined textual input and searches the input text with regular expressions under the two data identifier related attributes in the database: (1) WSDL data name, and (2) the service description. The search results returned: (1) a list of data related identifiers ordered by relevancy to the search query and (2) a list of modularization units (i.e., functions, and subroutines) that contain the top three relevant data related identifiers. From the second list, the user also had the option to retrieve the code snippets of the modularization units.

2.4 Conclusion

This study presented an approach to enrich source code components with business semantics in the context of enterprise systems. The potential value of this approach is to

support code reuse for SOA based services development during system modernization efforts.

The approach makes use of data naming similarity to discover semantic relationships between high-level business services and low-level source code components. Data names are extracted from both ends through parsing techniques. Then a similarity function is applied to identify identity equivalences between the naming sets. Based on established identity matches, the source code model is annotated with service specific knowledge. The entire process can be automated via parsing tools and similarity function implementations. I have validated the applicability of the approach using a case study on SAP's Enterprise System Framework. The results have indicated that the proposed approach is useful for creating service-based knowledge in the source code components of an existing enterprise system.

3 STUDY TWO – UNDERSTANDING THE ECONOMIC VALUE OF SERVICE-ORIENTED ARCHITECTURES

3.1 Introduction

In this study, I investigate a different aspect of SOA based IT Servitization, namely its economic impact for the organizations. Since SOA is a system design style rather than a system itself, it does not dictate on any specific technology to achieve service orientation (Papazoglou & van den Heuvel, 2007). However, a certain infrastructure needs to be in place to build, deploy and manage an SOA implementation. Components of this infrastructure include, but are not limited to, a service repository to store service related data, an enterprise service bus (ESB) to route and transform service messages and an orchestration engine to compose complex business processes from available services. IBM's WebSphere, Oracle's SOA Suite and Sun's Java CAPS are some of the commercial products that provide such components required for the SOA infrastructure – with price tags up to million dollars. According to AMR Research (2008), an average of $1.4 Million on SOA software and services has been spent in 2007 per company that has active SOA initiatives. Upon licensing, installation and training costs, a full SOA implementation can turn out to be a very expensive investment for an organization.

Consistent with its high costs, potential benefits of SOA are also vast, in theory, ranging from reduced maintenance to increased reuse, from business flexibility to application integration and so on. Most importantly, SOA enables organizations to be

more responsive to the demands of the markets, because flexible applications can be developed and deployed rapidly to support business agility (Zhao, Tanniru, & Zhang, 2007).

Unfortunately, in contrast to the costs of the investment, the monetary benefits associated with SOA are more difficult to measure. One problem is that benefits such as increased agility or improved flexibility are elusive in nature, meaning that it is hard to define performance metrics for their accurate calculation (Tsourveloudis & Valavanis, 2002). A second problem with benefits measurement is the uncertain nature of future events. As an enabler software platform, the core value of SOA is to open up future payoff opportunities that can be realized throughout its lifespan. By capitalizing on such opportunities, managers have the flexibility to alter the benefits obtained from this platform. Unfortunately, traditional capital budgeting methods often fail to capture managerial flexibility when valuing investments (Trigeorgis, 2005).

In this research, I study the following question: how to support managerial decision making regarding a specific type of IT investment – the SOA platform? To answer this question, it is necessary to measure the actual economic value to be gained by SOA. To achieve this, I identify five operational and strategic issues related to SOA and make suggestions on how these issues can be modeled quantitatively. Then, I construct a model that can be applied to estimate the extended SOA investment value. The extended investment value is a measure for the economic incentive of SOA adoption, and serves as an instrument to support the managerial investment decision.

In the next section, I discuss the operational and strategic issues related to SOA in more detail. In Section 3.3, I construct a SOA valuation model based on these issues. Finally in Section 3.4, a case study is provided to validate the application of this model.

3.2 Operational and Strategic Issues in SOA Investments

3.2.1 Up-front Expenditures

As with many IT infrastructure projects similar in nature, SOA requires significant up-front expenditures during the initiation phase of the project. Up-front expenditures correspond to the negative cash flow at time zero (i.e. until the project roll-out) and include cost of installation, cost of initial development and cost of training. Specifically, SOA software and hardware purchases (e.g. enterprise service bus, service repository database, etc.) and their setup labor costs are the items to consider during the installation process. After installation of SOA components, initial core services have to be developed and deployed in order to roll-out the SOA implementation. Finally, user training has to be completed before SOA becomes operational in the organization.

It is relatively easy to measure the monetary impact of up-front expenditures. Cost figures regarding software and hardware purchases can be obtained through vendor channels. For effort based items, costs can be defined as a function of labor hours and labor rate. In this calculation, labor rate is a given constant, whereas labor hours required for setup, service development and user training are estimated by management. The Equation 3.1 below is used in the model to compute the up-front expenditures for SOA, where C_u is the total up-front expenditure, c_s is the cost of SOA package and software

purchases, c_h is the cost of hardware, L corresponds to the total labor hours spent for installation, development and training and e is the hourly labor rate.

$$C_u = c_s + c_h + (L * e) \quad (3.1)$$

3.2.2 SOA Led Changes in IT Operations

An SOA implementation represents a major architectural paradigm shift for a company — within both business and IT units (Patrick, 2005). The impact of these changes has been subject to recent research from the industry (IBM Global Business Services, 2006; IBM Global Technology Services, 2008; webMethods Inc, 2005) and academia (Beimborn & Joachim, 2009; Bieberstein, Bose, Walker, & Lynch, 2005; Luthria & Rabhi, 2008; Maurizio, Sager, Jones, Corbitt, & Girolami, 2008; Yoon & Carter, 2007). In a work by Yoon and Carter (2007), it is argued that changes that organizations experience from SOA implementation can put into two benefit categories: (1) improved business agility and (2) lowered costs. Drawing from the work of Yoon and Carter (2007), Beimborn and Joachim (2009) define the drivers for SOA benefits as: modularization, reuse, reduced complexity, integration potential, virtualization and platform independence. In addition, they also define specific risk and effort factors as the downside impact of SOA. These factors include: organizational change, increased governance efforts and technology risks.

Based on previous literature, I identify three dimensions of changes with the introduction of SOA technology: (1) technical, (2) organizational and (3) managerial, as shown in Figure 10. Technical dimension is driven by changes in application

development and maintenance efforts. Organizational dimension focus on how roles and responsibilities in the organizational structure are affected by SOA. Finally, managerial dimension involves changes in IT service management.

Figure 10. Three dimensions of change in IT operations

3.2.2.1 Technical dimension

SOA promotes a shift from writing software to assembling and integrating services. Due to the reuse of functionality, which is encapsulated in services, a service has to be implemented only once but can be used in different business processes (Beimborn & Joachim, 2009). This reuse characteristic of SOA has two important technical implications: (1) new applications can be developed in shorter time at lower costs, and (2) existing applications can be maintained more rigorously with less effort.

Application development under SOA is based on service composition mechanism. When applying composition, a new software asset is created by assembling two or more existing assets with custom built code (Postmus & Meijler, 2008). The custom code includes integration code that will glue the existing assets together and the new functionality code that is unique to the application. To determine the expenses in SOA based application development, I measure the effort spent on development of custom

code. Equation 3.2, which is adapted from COCOMO 2.0 model (Boehm, Clark, Horowitz, Westland, Madachy, & Selby, 1995) and Postmus and Meijler (2008) is used for this purpose, in which C_{dK} is the cost of developing a particular application K, s_{inK} and s_{newK} are the total amount of integration and functionality code for K developed in terms of thousands of lines of code (KSLOC) respectively, a_l is the average labor hour to develop one KSLOC, and β is the exponential factor to account for possible economies or diseconomies of scale in writing lines of code.

$$C_{dK} = \left(a_l * \left(s_{inK} + s_{newK}\right)^{\beta}\right) * e \tag{3.2}$$

The second impact of SOA reuse is on the maintenance costs of applications. A promise of SOA is to decrease software maintenance by reducing system complexity and function duplication (Beimborn & Joachim, 2009). A well accepted economic view on software maintenance is to model it as an economic production process, where maintenance effort (i.e. the input) is converted into corrected software (i.e. the output) (Banker, Datar, Kemerer, & Zweig, 2002). To model SOA application maintenance costs, I adopt from Banker, Datar, Kemerer and Zweig (1993) and Banker et al. (2002), where maintenance costs are defined as a function of overall maintenance effort, which is affected by two factors: (1) code complexity, (2) errors in the code.

Code complexity refers to the characteristics of the data structures and procedures within the code that make it difficult to understand and (Banker, Davis, & Slaughter, 1998). In the context of software applications, it can be measured by using module size (i.e. average size of application's modules), procedure size (i.e. average size of a module's procedures), and branching (i.e. density of call statements) (Banker, Datar,

Kemerer, & Zweig, 1993). For SOA applications' complexity measurement, I substitute module size with service size while keeping procedure size and branching metrics.

The second issue affecting the maintenance effort is errors in the application code. Variations in error rates are expected to be a function of either the software system itself, or factors in the maintenance environment. I propose to use complexity and system volatility as system factors and maintainer experience as the environment factor in a model of software maintenance effort. Here, system volatility refers to the frequency of changes made to the application software. Systems which undergo frequent modification have higher error rates, because each modification represents an opportunity for new errors to be generated (Banker, Datar, Kemerer, & Zweig, 2002). And maintainer experience is a measure of maintainers' familiarity with the application. If the programmers maintaining the system are relatively new, their inexperience will increase the chances of errors go undetected. Figure 11 is the representation of this maintenance effort model. In the context of SOA implementation, it is expected that as service reuse gets more common through the enterprise; the application complexity ratios will decrease, reducing the overall maintenance effort. However, error rates in the code are initially expected to go up, because of the programmers' unfamiliarity with the SOA development environment. This will affect maintenance efforts in a negative way.

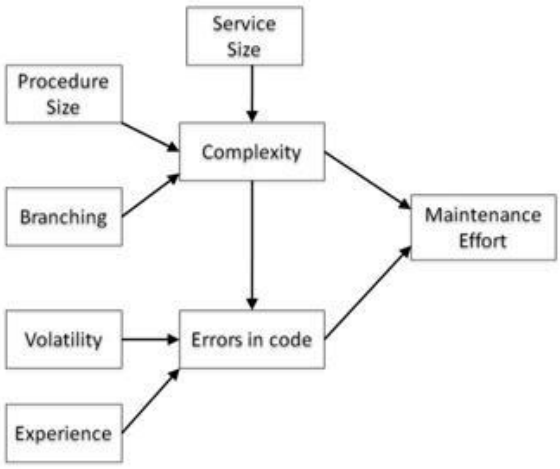

Figure 11. SOA maintenance effort model

3.2.2.2 Organizational dimension

SOA enablement is a major transformation for many organizations, requiring effective governance and organizational change. Organizational change impacts how individuals in an organization do their work and how they relate to one another (Bieberstein, Bose, Walker, & Lynch, 2005).

It is necessary to transform traditional software development roles and responsibilities under SOA based environments, mainly due to the increasing alignment between IT and business process management units (Kajko-Mattson, Lewis, & Smith, 2008). The resulting organizational structure is a one that streamlines cross-business operations, flattens management chains and puts business services at the core of system design (Maurizio, Sager, Jones, Corbitt, & Girolami, 2008; IBM Global Technology Services, 2008).

However, adoption to the new organizational structure can be costly. There are two main economic concerns that accompany an SOA based organizational change effort. First of all, SOA implementations introduce new responsibilities for IT professionals. The focus shifts from the implementation of the software system to the understanding of the purpose of the individual components, and their role and cooperation with other services within a combined business process (Kajko-Mattson, Lewis, & Smith, 2008). Acquiring such new skills and knowledge may require additional tools and training for employees.

Secondly, the organizational workforce can be affected by the changes driven by the SOA implementation. New roles and positions may be created to support SOA specific processes, whereas some existing positions can be eliminated. Proposed by Kajko-Mattson et al. (2008), Figure 12 is a schema for SOA-based system roles that includes new positions such as service developer and service designer and old ones such as traditional back-end support.

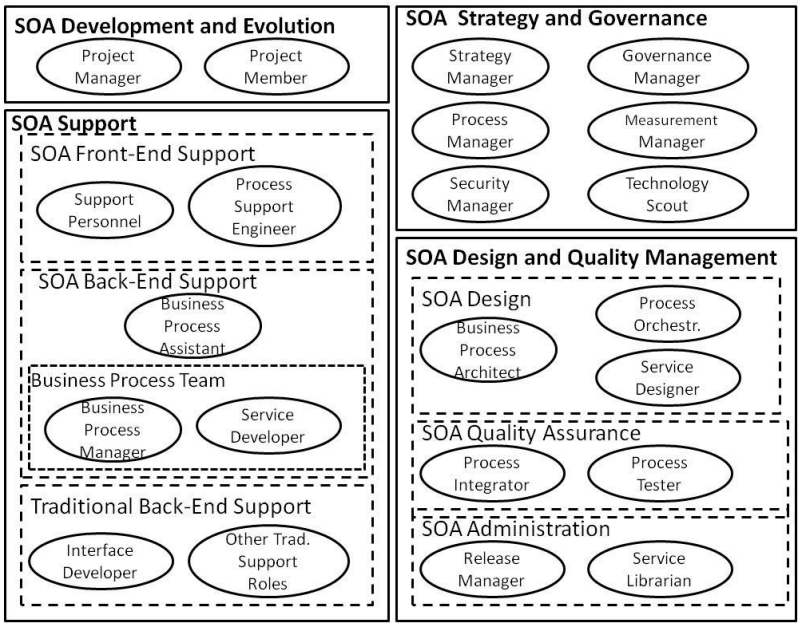

Figure 12. SOA based system roles (adapted from Kajko-Mattson et al., 2008)

3.2.2.3 Managerial dimension

IT service management (ITSM) is the third area that we can observe changes on with respect to SOA adoption. In broad terms, ITSM refers to the set of processes that deal with the control and monitoring of IT services (Sallé, 2004). The most common framework for implementing ITSM is Information Technology Infrastructure Library (ITIL), which can be described as a set of documents consisting of best practices of IT service management. ITIL focuses on two main areas: service support and service delivery (Hochstein, Zarnekow, & Brenner, 2005). Under service support, five main managerial concerns are addressed: incident management, problem management, change management, release management and configuration management. In addition, service delivery covers the issues related to: service level management, financial management, IT

service continuity management, capacity management and availability management (Hochstein, Zarnekow, & Brenner, 2005).

For many SOA characteristics, the impact on such areas can be both beneficial and costly. For example, while service reuse is a significant benefit for application development and maintenance, the fact that a service can be widely used in many applications throughout the enterprise may create capacity management problems (Bieberstein, Bose, Walker, & Lynch, 2005). For highly reused services, scarce infrastructure resources may cause system overload and affect response times. Eventually, availability of services may be compromised.

In another situation, service composition characteristic of SOA may positively or negatively affect application release times. While applications that are composed of existing services can be developed faster, coordination of such applications with existing IT assets may take more time, thereby delaying the actual release of applications.

Configuration and change management after application implementation may also be affected. Reuse of SOA services in many applications will result in increasing complexity of the configuration of IT assets. Changes to a reused SOA service for one application may also affect other applications using the same service, further complicating change and configuration management aspects under SOA.

In IBM Global Technology Services (2008), the impact of SOA on IT service management of an organization is discussed with respect to three topics: 1. problem identification, 2. service component relationships and 3. performance issues. Studying

this work under the context of ITIL framework, I identify the SOA characteristics that affect relevant ITIL sub-areas as shown in Figure 13.

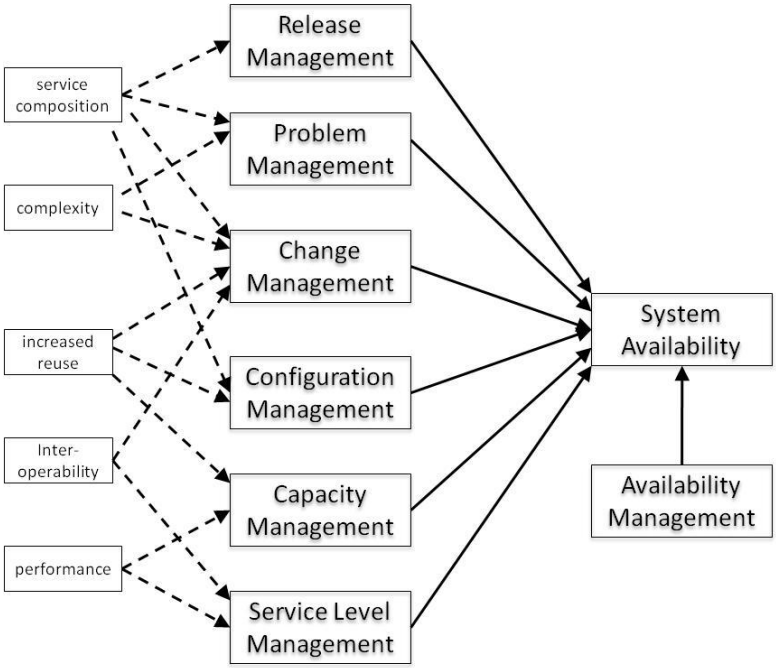

Figure 13. SOA impact on ITIL sub-areas

In this model, system availability (i.e. the degree of operational continuity of the system) is directly affected by the changes in the IT service management sub-areas. To measure the impact of SOA on IT service management, system availability values before and after the SOA implementation are compared.

3.2.3 Growth Options

As an enabling software technology, an SOA investment can be seen as a buy-in for development of new applications with a service-centric approach. Service-centric development refers to the process where reusable services are composed over an

orchestration engine and integrated into ESB to create composite business applications. These new applications are opportunities that can be brought into operation at certain implementation points when found beneficial (Taudes, 1998).

I define the composite business applications that are developed with SOA functionality as growth options of SOA investment. As shown in Figure 14, growth options provide the management with the flexibility regarding the implementation decision – meaning that management has the right but not the obligation to exercise the option (Dai, Kauffman, & March, 2007).

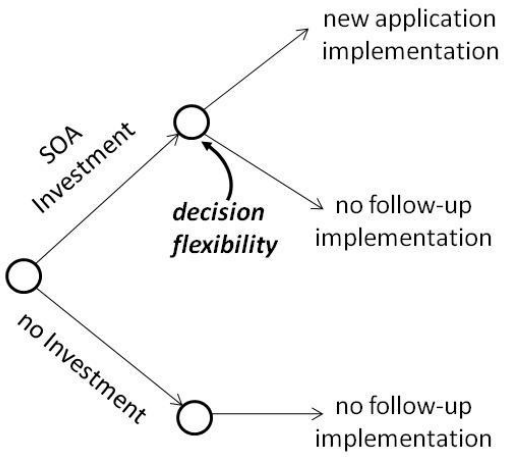

Figure 14. Growth option flexibility

The value of the option comes from the expected benefits of the new application, if the implementation is made. Since a rational decision maker wouldn't exercise an option that has a negative value known at a specific time point, we can be certain that values from implemented applications would be positive and add up to the overall SOA investment value. Figure 15 illustrates this idea, in which the SOA implementation is

completed at t_1 and a composite application i is implemented at t_2, bringing the overall investment value to v_2.

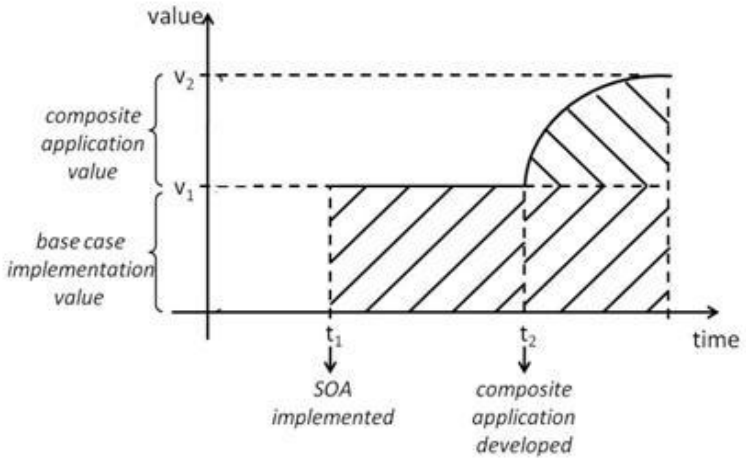

Figure 15. Effect of composite application on SOA value

However, estimating the expected value of a composite application (i.e. $v_2 - v_1$) may not be trivial, due to uncertainty involved in its future cash flows. According to Dai et al. (2007), stochastic market characteristics in which the option value is realized will significantly affect the value of an option.

To overcome this issue, different option pricing models from finance literature have been employed to value IT growth option opportunities, such as Margrabe's option pricing method, Carr's sequential exchange options model, and Geske's compound options approach (Singh, Shelor, Jiang, & Klein, 2004). In this study, I use one of the most common models, the Black-Scholes formulation (Black & Scholes, 1973) to compute the value of a composite business application. In short, the Black-Scholes model

is a closed formulation that computes the price of an option (e.g. composite business application *k*) as:

$$
\begin{aligned}
NPV_k &= V_0 N(d_1) - Ie^{-rT} N(d_2) \\
d_1 &= \frac{1}{\sigma\sqrt{t}}\left(\ln\left(\frac{V_0}{I}\right) + \left(r + \frac{1}{2}\sigma^2\right) \times T\right) \\
d_2 &= d_1 - \sigma\sqrt{T}
\end{aligned}
\tag{3.3}
$$

where,

NPV_k: Option value of composite business application *k*

V_0*:* current value of expected future benefits from application *k*

I: total cost of ownership for *k*

T: time period that *k* is implemented

r: risk free interest rate

σ: volatility of the expected future benefits from *k*

N(d): cumulative standard normal distribution function

This formula is derived assuming the payoffs of the option can be replicated through a continuously updated portfolio of two assets: a twin security that perfectly replicates value of the underlying asset and a risk free security such as a short-term bond (Erdogmus & Vandergraaf, 1999). In SOA investment analysis, I propose to calculate the growth option values of composite business applications using Equation 3 and add these values into the operational NPV value of SOA.

3.2.4 Deferral Option

An impact of service based reuse in SOA is reduced development time of applications. This impact has two significant outcomes. First of all, time-to-market delivery of applications is decreased, resulting organizations to quickly respond to new business demands, and thus achieving greater agility. Second, ability to choose optimal applications from an array of possible projects is increased; because the organization may now wait longer to see market uncertainties to resolve, and yet still develop an application on schedule.

From a real options perspective, reduced development time in SOA enables deferral options for developing and implementing composite business applications. After SOA, the management obtains the flexibility, as shown in Figure 16, to keep the option open (i.e. deferring the development kick-off decision) up to a time point t^*, when the value of the application will reach the highest value. As mentioned before, such flexibility is possible because the organization may now delay the implementation decision as the same amount of reduced development time.

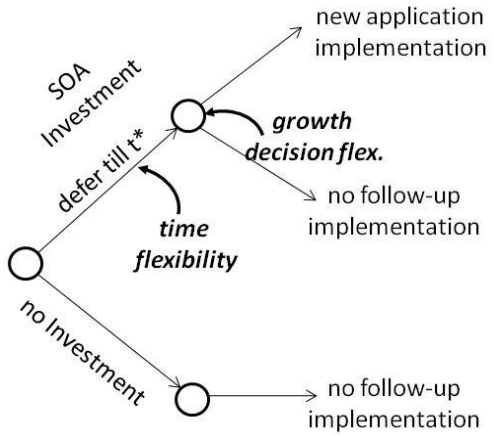

Figure 16. Time flexibility

An opportunity to flexibly defer implementation for some time period has a larger expected value than a now-or-never type of investment (Benaroch & Kauffman, 1999). In a case where there can be several possible application projects to pursue, but only a few of them could be picked because of development budget constraints, deferral option helps to resolve uncertainties by: (1) learning more about potential returns of each project, and (2) taking possible actions to lower market entry risks. On the other hand, by delaying the implementation, the organization may lose potential revenues from not implemented applications and also lose agility of quick time-to-market delivery.

Depending on the scale of these two competing effects, the option value of an application can increase or decrease. Therefore, the question becomes finding optimal time (i.e. *t**) for implementation, where the growth option value of the application is maximized, as shown in Figure 17.

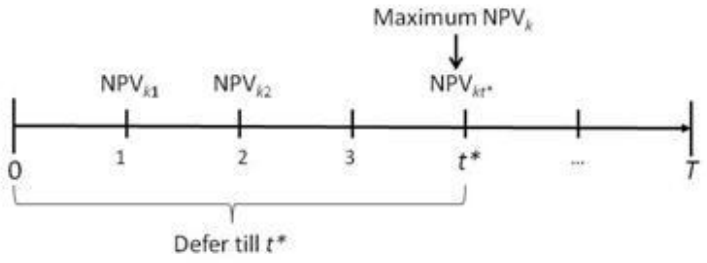

Figure 17. Maximum net present value

In Benaroch and Kauffman (2000), a decision rule to find the optimal implementation strategy is proposed as follows: Where the maximum deferral time is *T*, implement the application at time t^*, $0 \le t^* \le T$, for which the NPV_{kt^*} takes the max value.

$$NPV_{kt^*} = \max_{t=1,\dots T}\left(NPV_{kt}\right) \tag{3.4}$$

For each time period *t*, NPV_k can be computed using the Black-Scholes formula in Equation 3.4. However, decision makers should be careful about updating the formula variables and re-applying the decision rule every time new information arrives during the deferral period (Benaroch & Kauffman, 2000).

3.2.5 Switch-use option

In real options literature, the option to switch-use refers to the managerial flexibility of switching among alternate modes of operations such as projects, machines or technologies (Favaro, Favaro, & Favaro, 1998) (Trigeorgis, 2005). Two basic principles of SOA, *modularity* and *interoperability* are the features that can enable such managerial flexibility in IT infrastructures. As mentioned before, applications under SOA are composed of independent, autonomous and modular software components on an IT network. These modular components include services that encapsulate business functionality as well as IT resources that support application operations such as database systems. Through SOA interoperability, components and applications can integrate seamlessly along the enterprise IT network.

However, at some point in the future, an operational system application can be abandoned permanently, if conditions worsen severely. In that situation, an SOA implementation provides managers with the flexibility to re-utilize modular IT resources in other projects. Eventually, these salvageable resources could be sold or put to different uses, creating additional value for the organization (Benaroch, 2002). Figure 18 shows the strategic SOA options including the switch-use flexibility.

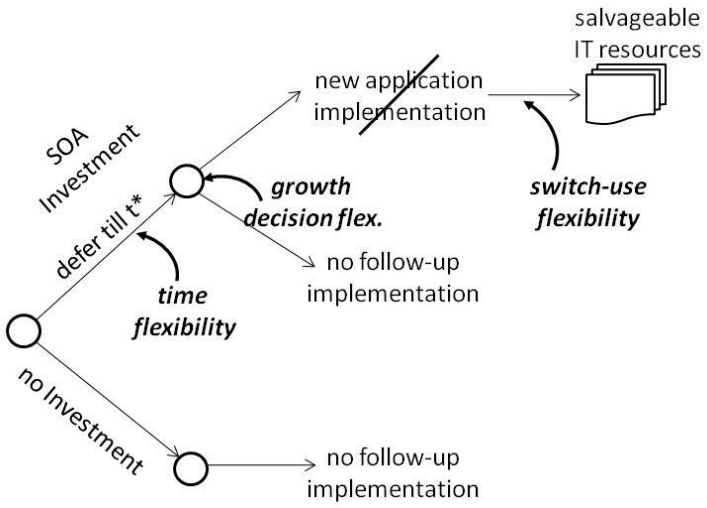

Figure 18. Switch-use option flexibility

Switch-use options extend the service reusability concept in SOA by also adding the IT resource (e.g. database systems, etc.) dimension into benefits calculation. To model the switch-use options, I use the formula in Equation 3.5, in which S_k refers to the salvage value for application k, c_m is the cost of IT resource m under application k's total cost of ownership, and β_m is a management estimate probability of re-utilizing resource m in any other project of the organization.

$$S_k = \sum_{m=1}^{M} c_m * \beta_m \tag{3.5}$$

For each application k that consists of IT resources m (m = 1,2,..,M), I correct the total cost of ownership I in Equation 3.3 as $I^* = I_k - S_k$. This correction corresponds to subtracting the expected salvage value of application k from its total cost of ownership. Applying the new I^* into Equation 3.3, I can then find NPV_k^* – the total option value of

composite business application *k* that considers both growth, deferral and switch-use flexibilities.

3.3 A Decision Support Model for SOA Investment Analysis

In total, I have identified five issues related to SOA investments in operational and strategic dimensions. Now, I combine these issues in a model to support managerial decision making regarding SOA investments. As shown in Figure 19, the model consists of four activities: (1) defining the up-front investment costs, (2) determining SOA led changes, (3) identifying strategic options, and (4) calculating the extended investment value. Topics to consider for each of these activities are given in the middle columns of the figure, while analysis tools and measures are given in the bottom two columns.

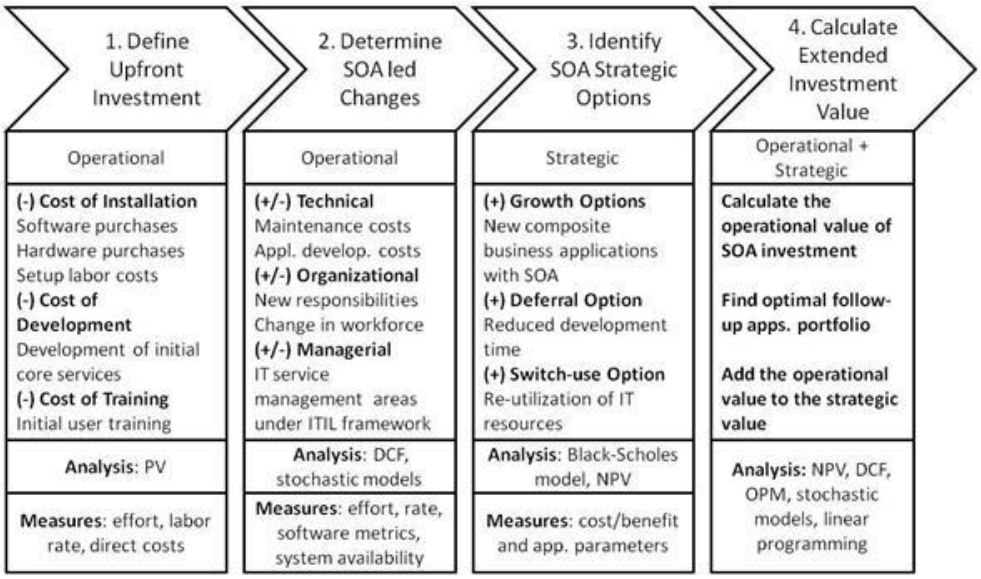

Figure 19. Decision support model for SOA investment analysis

In this section, I discuss on the last step – calculating the extended investment value in more detail. As mentioned before, the extended value (i.e. total value) of an SOA investment is composed of: (1) the operational value of SOA, and (2) the value of strategic options in SOA. Operational value can be calculated by adding the negative cash flows at time $t = 0$ (i.e. up-front investment) into the SOA led changes in IT operations (i.e. changes in technical, organizational and managerial dimensions).

On the other hand, to find the value of strategic options, the optimal follow-up composite business application portfolio must be identified, given budgetary constraints. This is an optimization problem that can be expressed as:

$$\begin{aligned} & max \sum_{k=1}^{K} x_k \times NPV_k^* \\ & s.t. \\ & \sum_{k=1}^{K} \left(x_k \times PV(I_k) \right) \leq Y \\ & x_k \in \{0,1\}; \quad k = 1,2,...,K \end{aligned} \tag{3.6}$$

where,

$k = 1, 2, \ldots, K$ is the set of all possible future composite applications that can be implemented with SOA

NPV_k^*: Corrected total option value (growth, deferral and switch-use options) of composite business application *k.*

x_k*:* the decision variable of whether to include k in the portfolio or not.

I: the total cost of ownership of k (non-corrected).

Y: the budget constraint of the organization regarding ownership costs for future applications.

Solution to this problem would give the value of strategic options in SOA. Strategic options value then can be added to the operational value to obtain the extended SOA investment value, as shown in Figure 20, where C_u is the total up-front expenditure and $C_{N(.)}$ and $C_{O(.)}$ correspond to the operational cash flows in technical, organizational and managerial dimensions after SOA implementation and before SOA implementation, respectively.

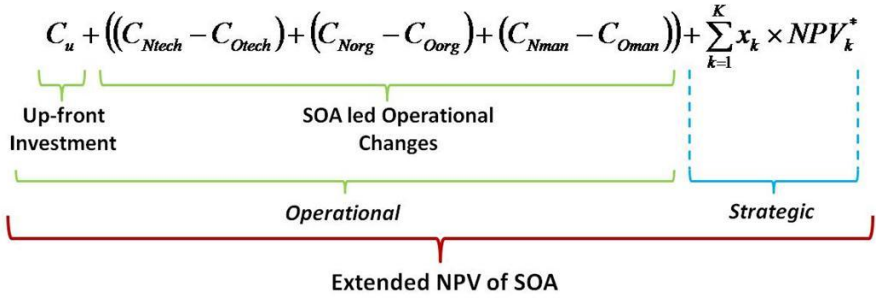

Figure 20. Extended SOA investment value formula

3.4 Validation of the Model

To illustrate the application of the model, I provide analysis of an SOA investment decision in a real-life organization. The organization in discussion is a Fortune 100 firm specializing in defense systems manufacturing. The decision problem involved whether to or not to employ SOA in one of its sub-divisions that develops complex software applications used in defense systems products. While the management of the organization was aware of the potential benefits of SOA, calculating a monetary value for the investment had been a concern. I applied the decision support framework proposed in the paper to estimate the extended NPV of the SOA implementation.

Interviews with the division executives revealed that there are four major application proposals (i.e. software development and upgrade projects) that are in pending process but unlikely to be initiated because of their negative return on investments under the existing development approach. Additionally, the base case SOA implementation was also found to be an infeasible investment based on the operational value, as shown in Table 7. In this table, the first row corresponds to the up-front investment required to purchase and install the SOA platform, whereas the remaining rows summarize the SOA led changes in IT operations after the implementation. Technical changes involve SOA-centric (i.e. service based) development and maintenance costs for new applications; and are covered under total costs of ownership (I) in Equation 3.3. Organizational and managerial changes are management estimates that include the cost of SOA based system roles and the impact of system availability change (no change is expected by management in this case) in monetary terms, respectively.

Table 7. Up-front investment and SOA led changes

Operational Measure	How to calculate	Present value
Up-front investment (C_u)	Equation 1	- $750,000
Technical changes ($C_{Ntech} - C_{Otech}$)	Included under total costs of ownership of new applications developed with SOA	Factor of I
Organizational changes ($C_{Norg} - C_{Oorg}$)	Management estimates	-$160,000
Managerial changes($C_{Nman} - C_{Oman}$)	Management estimates	$0
Current Operational Value		- $910,000

The next step in the framework involved identifying the strategic options. Four application proposals with negative return on investments under the existing development

approach were identified to be the growth option opportunities of the SOA investment. For each of these opportunities, parameter estimations were required to calculate the growth option value using Equation 3. Determining V_0, σ and T were rather effortless. The return on investment (ROI) analysis that was previously conducted by the division included the current values (V_0) and volatilities (σ) of expected future benefits from each application as well as the development schedule estimations (T). 7% was chosen to be the annual risk-free interest rate r. Determining the total cost of ownership (I) was more complicated. Through iterative discussions, estimates were made by studying the unique characteristic of each application and its applicability to the SOA platform. The findings for growth option parameter values and NPV computation using Equation 3.3 for each application were given in Table 8.

Table 8. SOA growth option parameters and NPVs

	V_0	I	*T (yrs)*	r	σ	*NPV from growth option*
Application 1	$275,000	$325,000	0.75	7%	25%	$11,900
Application 2	$297,000	$350,000	1.25	7%	30%	$30,350
Application 3	$792,000	$1,500,000	2.83	7%	65%	$231,000
Application 4	$43,000	$120,000	0.5	7%	20%	$0

Deferral and switch-use options are the other two components of SOA strategic options in the decision framework. For deferral option, reduced development time characteristic under SOA based development was considered to correct the development schedule estimations (T) previously identified in ROI analysis. A new parameter t^* is defined for this purpose, where $t^* = T +$ *development time reduced with SOA*. For switch-use option, the salvage value for each service based application was estimated using the

Equation 3.5 and the total cost of ownership was corrected as $I_k^* = I_k - S_k$. Values regarding these adjustments as well as the final corrected NPV for each application were given in Table 9.

Table 9. Adjustments for deferral and switch use options

	NPV from growth option	t*	NPV corrected for deferral option	S_k	I^*_k	Final corrected NPV
Application 1	$11,900	1.08	$18,400	$58,000	$267,000	$42,900
Application 2	$30,350	1.75	$41,650	$50,000	$300,000	$61,770
Application 3	$231,000	3.25	$261,200	$300,000	$1,200,000	$309,000
Application 4	$0	0.6	$0	$85,000	$35,000	$9,600

Given the budget constraints of $2,000,000 for application ownership costs (including application development, user training, additional hardware, etc.), the optimal application portfolio was found to include applications 2, 3 and 4, according to Equation 3.6. Consequently, the total strategic options value of SOA was calculated as $61,770 + $309,000 + $9,600 = $380,370.

Finally, strategic options value of $380,370 was added onto the current operational SOA value of −$910,000 to find the extended NPV of SOA investment as −$529,630.

There are two main implications of this result. Regarding the investment decision, it can be concluded that although strategic opportunities embedded in the SOA implementation mitigate the costs associated with the investment to some extent, currently it is not feasible to proceed with the investment. Secondly, it can be observed

that up-front investment costs constitute the most to the no-go investment decision. As SOA related software and hardware costs decrease with the acceptance and maturity of this new IT paradigm, the investment analysis should be re-visited to update the feasibility decision.

3.5 Conclusion

Capital budgeting scenarios such as SOA investments need to be evaluated analytically in order to make reliable investment decisions. In this study, I provided a managerial decision framework to analyze the monetary impact of SOA in an organization. Specifically, I discussed how to measure operational and strategic values from SOA investments. By using simple operational models as well as option pricing methods, the contribution of this study has been to expand the passive NPV of initial positioning investment with flexible strategic opportunities in the SOA domain.

There are certain limitations with this study. First of all, an assumption was that the composite business applications are opportunities unique to SOA implementations and their implementation values are added to SOA benefits directly. In a more realistic setting, these applications may also be instances of existing technologies and to value the SOA investment, a comparison between the service-centric development approach and the old development approach may be needed. A similar issue can also be observed for the case study problem, in which only a discrete number of new applications as potential strategic options of SOA were considered.

4 STUDY THREE – ASSIGNING LIVE-CHAT AGENTS TO HETEROGENEOUS E-CUSTOMERS UNDER IMPERFECT CLASSIFICATION

4.1 Introduction

E-commerce websites are increasingly adopting technology-enabled customer service systems, which were previously not available due to technological constraints (Froehle C. M., 2006). One such example is the live-chat system. Through this technology, companies seek to create an experience similar to physical store shopping, where customers can communicate with human agents in real-time to receive personalized sales promotions and service support (Andrews & Haworth, 2002). The use of real-time dialogue often proves to be a more effective way than other communication channels (such as e-mail and FAQ) to develop and sustain customer relationships (Qiu & Benbasat, 2005; Kalyanam & Zweben, 2005; Aberg & Shahmehri, 2000). Very often, the involvement of human agents also translates into opportunities to derive extra value through cross-selling (Demiriz, Kula, & Akbilek, 2009). Live-chat has also been shown to handle more complex transactions than FAQ or e-mail communication, and yet derive higher customer satisfaction ratings.

The synchronous nature of live-chat communication requires prompt response by the firm to attend customer communication requests. In fact, customers anticipating real-time communication are unlikely to wait long periods of time to connect with human

agents. Song and Zinkhan (2008) found that live-chat response time is a significant predictor of customer satisfaction and quality perceptions. However, with the increasing number of communication requests coming through live-chat channels (Froehle C. M., 2006), it may be very costly for a company to hire more service agents to retain low waiting times.

A way to manage service resources effectively without increasing resource capacity is to perform priority processing, where more valuable (i.e. high-value) customers of the firm receive preferential treatment (Zeithaml, Rust, & Lemon, 2001). Under this notion, heterogeneous customers are served according to their importance to the firm, rather than the order in which they arrive (Cobham, 1954). Within priority-processing, it may also be desirable to retain a reserve of resources exclusively for higher priority customers (Schaack & Larson, 1986). Reserve-based priority-processing policies are in common use in many real-life scenarios that involve multiple priority classes and limited service resources, such as machine maintenance, police vehicle dispatching, hospital bed management, bandwidth management and data packet routing in networks.

A common assumption made in such scenarios is that class attribute values are perfectly observable by the firm, meaning that upon arrival, the firm knows exactly which priority class a customer falls into. However, this assumption may not hold in the e-commerce context. A favorable characteristic of e-commerce transactions is the anonymity of users and the privacy of personal data. From the firm's perspective, this condition makes segmenting online customers more difficult and prone to classification errors. For instance, the firm may classify a high-value customer as being of low-value,

or a low-value customer as being of high-value. In this chapter, I study a reserve-based priority-processing policy for live-chat agent allocation at e-commerce firms that employ erroneous customer classification. I assume that the objective of the firm is to minimize customer total waiting costs by finding associated classification configuration and agent assignment rules. Specifically, two policy decisions are considered: 1) how many agents should be available to different customer types? 2) How sensitive should the firm be in classifying high-value customers? The first decision determines the number of agents exclusively reserved for high-value customers, while the second determines the number of customers in the system classified as high-value.

To answer these questions, I model a two-class live-chat queuing system as a continuous time Markov process and derive explicit expressions for average waiting times of both customer classes. I analyze the steady-state behavior of the system for three different business scenarios using numerical experiments. The experiments indicate results that have managerial implications. First, I find out that reserving agents for H-class customers may actually have negative consequences for H-class customers under imperfect classification. Second, I find the performance of the classifier to be a major determinant on making the best policy decisions. Finally, I observe that a change in one policy decision may have impact on the other decision if the goal of the firm is to minimize total waiting costs. Specifically, I find out that when the firm wants to increase its reserve capacity under good classification, it may also need to increase the classifier's sensitivity in order to keep the waiting costs at minimum.

The rest of this chapter is organized as follows. In the next section, I define the priority-processing model, derive average waiting times for customer classes and define a total waiting cost function. In Section 4.3, I illustrate and discuss the insights of the model with numerical results. Finally, in Section 4.4, I conclude the chapter.

4.2 Model

4.2.1 Definition and Notations

Consider an e-commerce service system with two classes (i.e. H and L) of customers. H-class customers arrive at the system according to a Poisson process with a rate of λ_H and are served at an exponential rate of μ_H. L-class customers arrive at the system according to a Poisson process with a rate of λ_L and are served at an exponential rate of μ_L. The two classes also differ in terms of service delay sensitivity. Having an s class customer wait in a queue before being admitted to service costs u_s for the firm per unit of time, where $s \in \{H, L\}$. Inherently, it can be assumed that $u_H * \mu_H > u_L * \mu_L$. This assumption implies that long run average costs of having H-class customers wait in queue will be greater than having L-class customers wait in queue. Therefore, H-class customers are defined as high-value and L-class customers are defined as low-value for the firm. There are N identical service agents ($N > 0$) that can serve both classes of customers. The system capacity is finite and an arriving customer is considered to be blocked if there are U customers already in the system (i.e. queue + service).

The firm's objective is to minimize the total long run average waiting costs, which are a function of priority class arrival rates, λ_s, delay sensitivities, u_s, and long run

average waiting times for each class in the queue. To control for waiting times, the firm employs a head-of-the-line priority-processing discipline, where H-class customers have admission priority over L-class customers in a non-preemptive service setting. A non-preemptive service implies that the service cannot be interrupted once a customer is admitted. It is also assumed that the service discipline may be non-work-conserving, i.e. there may be idle agents in the system even if there are customers waiting in the queues.

Reserve-based priority-processing policies can be used to control non-preemptive, non-work-conserving queues. In these policies, L-class customers are refused immediate assignment and placed on wait condition when the number of busy human agents in the system is above a cut-off value (denoted as C). The rationale behind this idea is to keep $N - C$ agents exclusively reserved for H-class customers, even if there are L-class customers waiting in the queue. This rationale is reasonable as long as $u_H * \mu_H$ is large enough to compensate for the wait of L-class customers (Atar, Mandelbaum, & Reiman, 2004). $(N - C)/N$ ratio is referred as the reserve capacity of the system.

Due to their complicated nature, reserve-based priority-processing policies have been the subject of relatively few studies in the literature (Pekoz, 2002). In Esogbue and Singh (1976), the problem of finding the optimum set of cut-off points for a reserve policy with a pre-defined cost structure has been studied in the context of patient bed scheduling. In Taylor and Templeton (1980), the authors derive probability distributions and the average L-class waiting time for two-priority class cases when $\mu_H = \mu_L$ and apply their results onto ambulance fleet management. Gerchak et al. (1996) question the optimality of these policies and show that an optimal policy is often not one of cut-off

number. Pekoz (2002) finds that a variation of reserve policy can perform arbitrarily close to the optimal for the objective of minimizing average H-class waiting times, while keeping the queue stable. Finally, Gurvich (2004) shows reserve-based admission control policies are asymptotically optimal in terms of steady state waiting costs and delay constraints for K number of classes ($K \geq 2$) with equal service times.

While analyzing a similar reserve policy, this study differs from existing literature in two aspects: 1.) I consider different service times and a finite system capacity and 2.) I introduce imperfection in identifying priority class members. The second point is especially important, since the true class (i.e. H-class or L-class) of a customer may not be directly observable in the e-commerce environment. In this respect, I only assume that the firm can make probabilistic estimations about a customer's class using an imperfect classification system (Argon & Ziya, 2009).

4.2.2 Classification System

To employ priority processing, a firm needs to categorize its customers into different classes. This step may be trivial, if there is an attribute, whose values can be used to determine the customer class. For example, for a priority scheme involving first time vs. existing customers, perfect classification can be made based on customer transaction data. A customer is a first time customer if he/she has no prior transactions; otherwise it is an existing customer.

On the other hand, classification may be harder to make and more imprecise, if there is no attribute that can directly differentiate between priority classes. For such cases, classification systems that combine various indirect class indicators through data

mining techniques (e.g. Logistic Regression, Naïve Bayes, etc.) have been developed. Commonly used classification systems in online environment analyze users' clickstream data as well as browsing history to differentiate between user profiles and to build priority classes (Liu, Sarkar, & Sriskandarajah, 2010).

For each of the three business scenarios that I study in this chapter (these scenarios are described in Section 4.3), I assume that the corresponding e-commerce website employs a classification system to differentiate between H-class and L-class customers. The classification system is not perfect, thus can classify a customer incorrectly. Denote β as the probability of an H-class customer classified as a L-class one (i.e. false negatives), and α as the probability of vice versa (i.e. false positives). The performance of the classification system is represented by a receiver operating characteristic (ROC) curve $1-\beta = \Omega(\alpha)$ that moves on α and $1-\beta$ axes. The function Ω is an increasing, concave curve with $\Omega(0) = 0$ and $\Omega(1) = 1$. Thus, depending on α and $1-\beta$ values, the firm will perceive different quantities of classified H-class and L-class customers. I use the sensitivity measure to represent the classifier configuration based on α and $1-\beta$ tradeoff, as shown in Figure 21. Sensitivity is the proportion of true H-class customers which are correctly classified. High sensitivity (i.e. high $1-\beta$ and high α) is preferred if the firm wants to make sure that H-class customers are classified correctly, at the expense of creating more false positives. Conversely, low sensitivity (i.e. low $1-\beta$ and low α) is preferred if the firm wants to make sure that customers in the classified H-class group are definitely H-class, at the expense of creating more false negatives.

Accounting for the imperfection of the classifier, I denote high-value customers after classification as being Class 1 and low-value customers after classification as being Class 2. The firm can observe the customers in the system only under these two new classes, as shown in Figure 22.

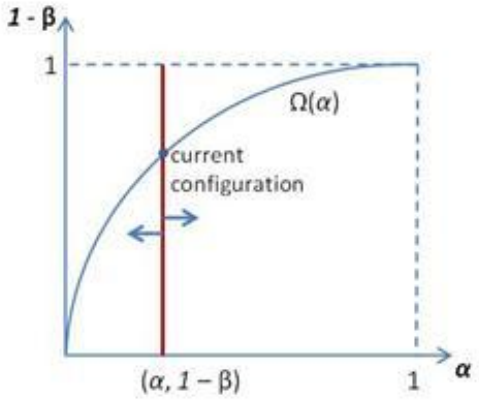

Figure 21. ROC curve and classifier configuration

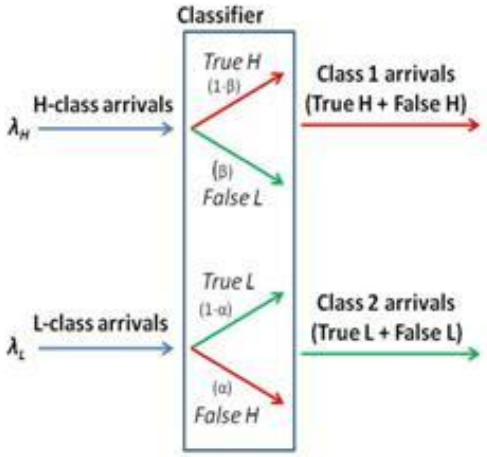

Figure 22. Customers in the system

Arrival and mean service rates for Class 1 and Class 2 can then be identified as:

$$\lambda_1 = \lambda_H(1-\beta) + \lambda_L(\alpha) \quad (4.1)$$

$$\mu_1 = [\mu_H\lambda_H(1-\beta) + \mu_L\lambda_L(\alpha)]/\lambda_1 \quad (4.2)$$

$$\lambda_2 = \lambda_L(1-\alpha) + \lambda_H(\beta) \quad (4.3)$$

$$\mu_2 = [\mu_L\lambda_L(1-\alpha) + \mu_H\lambda_H(\beta)]/\lambda_2 \quad (4.4)$$

In the next section, I study a stochastic process that incorporates the arrival and service rates given above to define the average waiting times for H and L customer classes.

4.2.3 Analysis of Average Waiting Times

Figure 23 depicts the queuing system for the priority model described above. Each arriving customer is directed to either Class 1 queue or Class 2 queue by the classifier based on classifier configuration and performance. Customers in the Class 1 queue have access to all available agents in the system, whereas customer in the Class 2 queue can only be served when the number of busy agents is below the reserve cut-off value C and the Class 1 queue is empty.

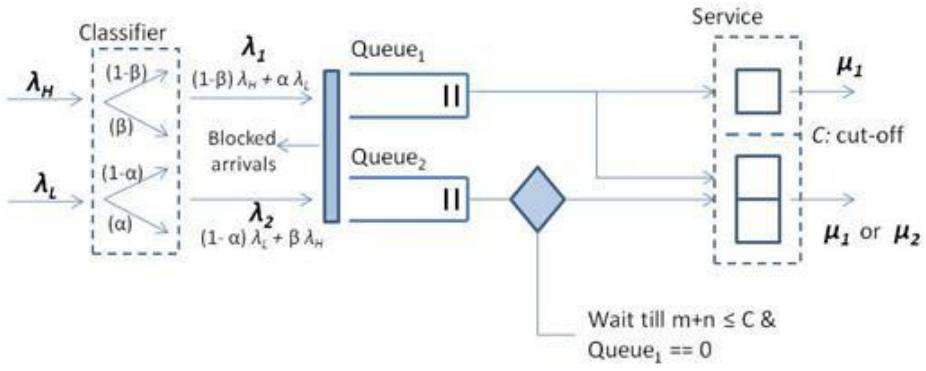

Figure 23. Queuing Model

This system can be described as a four-dimensional stochastic process $\{Z(t) = (k(t), l(t), m(t), n(t)), t \geq 0\}$, where k and l indicate the number of Class 1 and Class 2 customers in the system; m and n indicate the number of Class 1 and Class 2 customers in service at time *t*, respectively. Based on the assumptions of Poisson arrival processes, exponentially distributed service times and finite system capacity, $\{Z(t), t \geq 0\}$ is an ergodic homogenous continuous-time Markov Chain (CTMC). Let S be the set of all feasible states, i.e. $S = \{(k, l, m, n) \mid k, l, m, n \in \mathbb{N}^0,\ m + n \leq N,\ n \leq C,\ k + l \leq U,\ C \leq N \leq U\}$. Transition into a new state occurs when: (1) a customer (Class 1 or Class 2) arrives to the system and (2) a customer (Class 1 or Class 2) in service leaves the system. Table 10 depicts the rates for all possible transitions for the State (k, l, m, n). For the ease of representation, denote State (k, l, m, n) as State i, where $i \equiv (k, l, m, n)$.

Table 10. Transition rates for the CTMC

$q_{ij} =$	$\boldsymbol{\lambda_1}$	if $m+n<N$	and	if $j \equiv (k+1, l, m+1, n)$
	$\boldsymbol{\lambda_2}$	if $m+n<N$, $n<C$	and	if $j \equiv (k, l+1, m, n+1)$
	$\boldsymbol{\lambda_1}$	if $m+n=N$	and	if $j \equiv (k+1, l, m, n)$
	$\boldsymbol{\lambda_2}$	if $n=C$	and	if $j \equiv (k, l+1, m, n)$
	$\boldsymbol{\lambda_2}$	if $m+n=N$	and	if $j \equiv (k, l+1, m, n)$
	$\boldsymbol{m\mu_1}$			if $j \equiv (k-1, l, m-1, n)$
	$\boldsymbol{m\mu_1}$			if $j \equiv (k-1, l, m, n)$
	$\boldsymbol{m\mu_1}$	if $n<C$	and	if $j \equiv (k-1, l, m-1, n+1)$
	$\boldsymbol{n\mu_2}$			if $j \equiv (k, l-1, m, n-1)$
	$\boldsymbol{n\mu_2}$			if $j \equiv (k, l-1, m+1, n-1)$
	$\boldsymbol{n\mu_2}$			if $j \equiv (k, l-1, m, n)$
	$-\sum_{v \neq j} \boldsymbol{q_{iv}}$			if $j \equiv (k, l, m, n)$
	0			if j is any other state

Note that the steady-state condition for this chain will always exist, since the state space cannot grow beyond the system capacity. Denote $\vec{\pi} = (\pi_1, \pi_2, \ldots \pi_{|S|})$ as the steady-state distribution vector of this CTMC, where $\pi_i = \lim_{t\to\infty} \mathbb{P}\{Z(t) = i\}$. The vector $\vec{\pi}$ can be computed as the unique solution of the linear system $\vec{\pi}Q = 0$, where Q is the transition-rate matrix, i.e. $Q = \left(q_{ij}\right)$ (Kleinrock, 1975). Employing the normalization condition of $\sum_{i\in S} \pi_i = 1$, this system of linear equations can be solved by using Gaussian elimination.

Using the steady-state distribution vector $\vec{\pi}$, various performance characteristics of the system can be calculated. First, I define the blocking probability P_b (i.e. the probability that the system is full and incoming customers are rejected) as:

$$P_b = \sum_{k+l=C} \pi_i \tag{4.5}$$

In addition, I identify the mean numbers of Class 1 and Class 2 customers in queue as,

$$N_1^q = \sum_{i \in S} (k-m)\pi_i \tag{4.6}$$

$$N_2^q = \sum_{i \in S} (l-n)\pi_i \tag{4.7}$$

where $i \equiv (k, l, m, n)$.

Using (4.5), (4.6) and (4.7), Little's Law can be applied to define average waiting times, $E(W_o)$, after classification as follows:

$$E(W_o) = \frac{N_o^q}{\lambda_o(1-P_b)} \qquad o \in \{1,2\} \tag{4.8}$$

Here, $E(W_1)$ corresponds to the average waiting times of correctly classified H-class and wrongly classified L-class customers. Similarly, $E(W_2)$ corresponds to the average waiting times of correctly classified L-class and wrongly classified H-class customers. Using Equations 4.1 through 4.8, I can then identify the average waiting times for actual H-class and L-class customers as,

$$E(W_H) = (1-\beta)\frac{\sum_{i \in S}(k-m)\pi_i}{(\lambda_H(1-\beta)+\lambda_L\alpha)(1-\sum_{k+l=C}\pi_i)} + \beta\frac{\sum_{i \in S}(l-n)\pi_i}{(\lambda_L(1-\alpha)+\lambda_H\beta)(1-\sum_{k+l=C}\pi_i)} \tag{4.9}$$

$$E(W_L) = (1-\alpha)\frac{\sum_{i \in S}(l-n)\pi_i}{(\lambda_L(1-\alpha)+\lambda_H\beta)(1-\sum_{k+l=C}\pi_i)} + \alpha\frac{\sum_{i \in S}(k-m)\pi_i}{(\lambda_H(1-\beta)+\lambda_L\alpha)(1-\sum_{k+l=C}\pi_i)} \tag{4.10}$$

After defining the average waiting times, I can depict the total waiting costs (i.e. *TC*), as a function of priority class arrival rates, λ_s, delay sensitivities, u_s, and average waiting times $E(W_s)$, where $s \in \{H, L\}$ as follows:

$$TC = \lambda_H E(W_H) u_H + \lambda_L E(W_L) u_L \quad (4.11)$$

Given that λ_s and u_s are pre-determined parameters, the firm can only have control over $E(W_s)$ in order to minimize TC. Evidently, $E(W_s)$ can be controlled by two variables in the priority policy: 1.) C: The cut-off value for the reserve capacity of agents, and 2.) α: The classifier configuration. In the next section of the study, I run numerical experiments to observe the interaction between two policy decisions: C and α and note their respective impacts on average waiting times and the total waiting costs.

4.3 Validation of the Model

In this section, I present the insights of the study based on the numerical results from three sets of experiments. The first experiment considers the impact of reserve capacity on average waiting times and total waiting costs, when other parameters are held constant. The second experiment considers the impact of classifier configuration on average waiting times and total waiting costs, when the other parameters are held constant. The third experiment considers the interaction between reserve capacity and classifier configuration changes, when the firm's goal is to minimize total waiting costs.

In these experiments, I account for different e-commerce settings, where the firm could face different outcomes in classification. For instance, when customer membership is required for a transaction, firms are able to collect detailed customer information (such

as demographic profiles of the customers, purchasing histories, etc.) to allow for more accurate identification of valuable customers. In other cases, where customers engage in more browsing-oriented behaviors before making their purchasing decisions, the firm is faced with a more daunting task of accurately segregating different customer types from noisy data signals. To incorporate the differences in classification tasks into the model, I consider three different online scenarios involving live-chat communication. The first scenario assumes a perfect classifier, whereas scenarios 2 and 3 employ good and weak classifiers, respectively. Table 11 elaborates on the characteristics of these three scenarios.

Table 11. Descriptions of experiment scenarios

Scenario	Scenario Definition	Classifier Performance	Customer Type	Classification Attributes
1:Airline reservation system	Contact request to re-book a cancelled flight Priority given to higher membership status customers Membership information available	Perfect	Gold member vs. regular member	Membership ID, purchase information
2: Customer support system	Contact request for after-sale customer service Priority given to customers who can drop firm services Customer attrition likelihood predicted using metrics	Imperfect but good	Customer attrition vs. no customer attrition	Purchasing history, CLV, RFM

3: Online shopping system	Contact request for a general inquiry during shopping website visit Priority given to customers who are more likely to complete the transaction Session information available	Imperfect and weak	Purchasing-oriented customer vs. browsing-oriented customer	Real-time click-stream data

In all of the experiments, I set the base values as following: H-class arrival rate equals to L-class arrival rate: $\lambda_H = \lambda_L = 10$ customers per hour. Agent service rates are $\mu_H = 3$ customers per hour and $\mu_L = 4$ customers per hour. The number of total service agents in the system: $N = 5$ and the system capacity (i.e. U) is 15 customers. Long-run H-class delay cost per unit time is 15 times greater than that of L-class: $\mu_H * u_H = \$180$ per hour and $\mu_L * u_L = \$12$ per hour. The classifiers operate on ROC curves: $\Omega^1(\alpha) = 0^0$, $\Omega^2(\alpha) = \alpha^{0.1}$ and $\Omega^3(\alpha) = \alpha^{0.5}$ for business scenarios 1, 2 and 3, respectively. Average waiting times and total waiting costs were computed for reserve capacity $(N - C/N) \in S_{res}$ and $\alpha \in S_\alpha$, where $S_{res} = \{0, 0.2, 0.4, 0.6\}$ and $S_\alpha = \{0.1, 0.3, 0.5, 0.7, 0.9\}$. For Section 4.3.1, I set the base case α value as 0.5. For Section 4.3.2, I set the base case reserve capacity value as 0.4. Waiting times are in minutes and the total waiting costs are the daily averages.

4.3.1 Impact of Reserve Capacity on Average Waiting Times and Total Waiting Costs

As discussed before, a firm may opt to save reserve capacity for H-class customers, if keeping them waiting in queue would have significant negative

consequences. Intuitively, as more agents are reserved exclusively for H-class customers, I can expect H-class waiting times to decrease and L-class waiting times to increase. Figure 24 and Figure 25 depict the average waiting times for H-class and L-class customers, respectively, under different scenarios as reserve capacity is increased, when $\alpha = 0.5$.

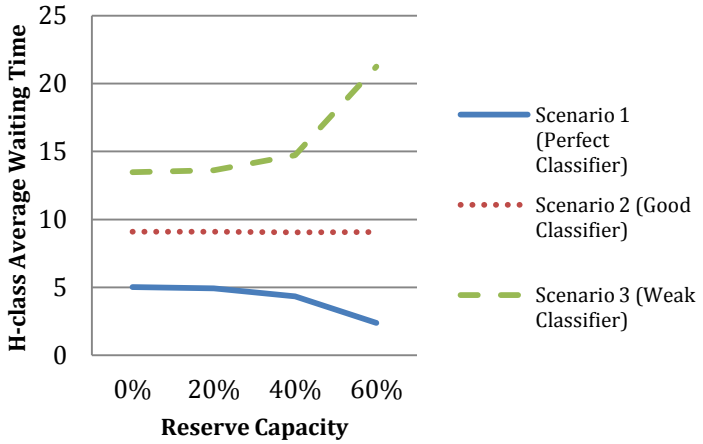

Figure 24. Impact of reserve capacity on H-class waiting times

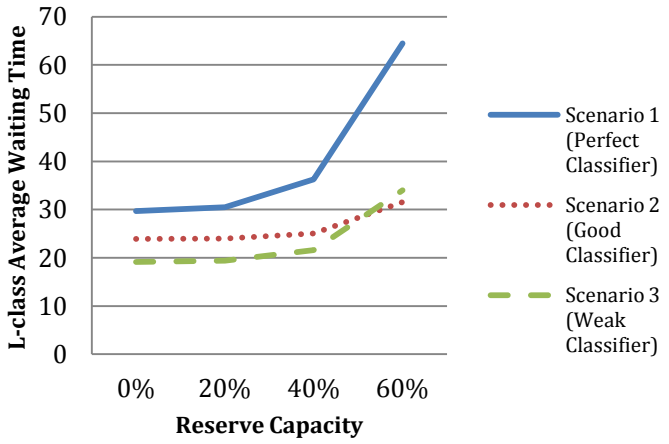

Figure 25. Impact of reserve capacity on L-class waiting time

Figure 24 points out that increasing the reserve capacity has a negative impact on average waiting time of H-class customers when the classifier performance is weak. This counter-intuitive result can be explained with the fact that under increasing reserve capacity, falsely classified H-class customers (i.e. false negatives) will endure increasing waiting times in the L-class queue. If the extent of this increase is large enough to overcome the waiting time benefits for correctly classified H-class customers, the average waiting time for overall H-class will increase as more agents are reserved. Inherently, this effect will be more evident when classification is weak and more false negatives exist in the system. Figure 26 shows the consequence of this result on total waiting costs, where costs decrease under perfect classification but increase under weak classification, when reserve capacity is increased. It can be seen that a seemingly good policy of reserving more capacity for H-class customers may be sub-optimal when the classifier performance is weak.

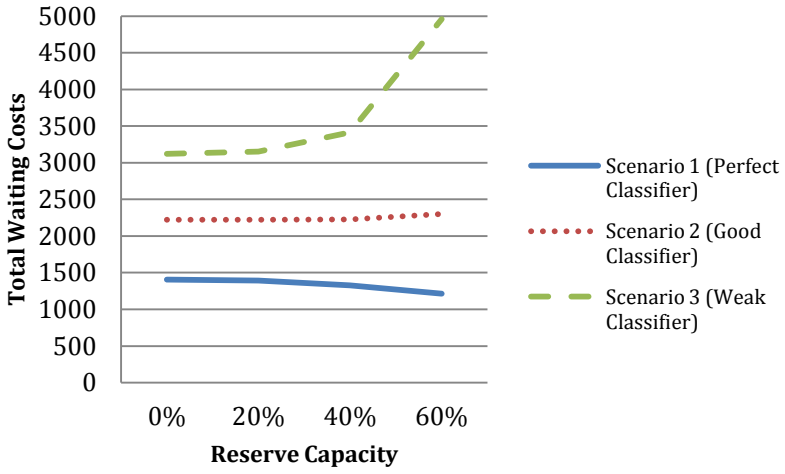

Figure 26. Impact of reserve capacity on total waiting costs

4.3.2 Impact of Classifier Configuration on Average Waiting Times and Costs

A second decision opportunity for the firm is to change the configuration of the classifier. As discussed in 4.2.2, by controlling α (and its consequent outcome, $1-\beta$), the firm can configure the classifier to behave more or less sensitive when classifying customers into Class 1. High sensitivity (i.e. high $1-\beta$ and high α) is preferred if the firm wants to make sure that H-class customers are classified correctly as Class 1, at the expense of creating more false positives. Conversely, low sensitivity (i.e. low $1-\beta$ and low α) is preferred if the firm wants to make sure that customers in Class 1 are definitely H-class, at the expense of creating more false negatives. Figures 27 and 28 illustrate the average waiting times for H-class and L-class customers under different scenarios as α is increased (i.e. sensitivity is increased) and when reserve capacity is at 40%.

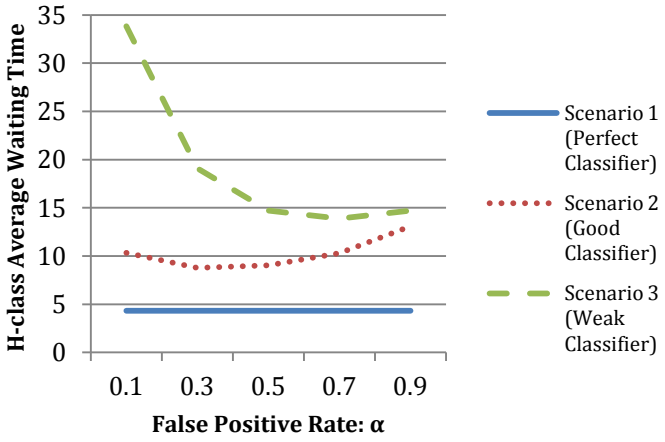

Figure 27. Impact of classifier configuration on H-class waiting time

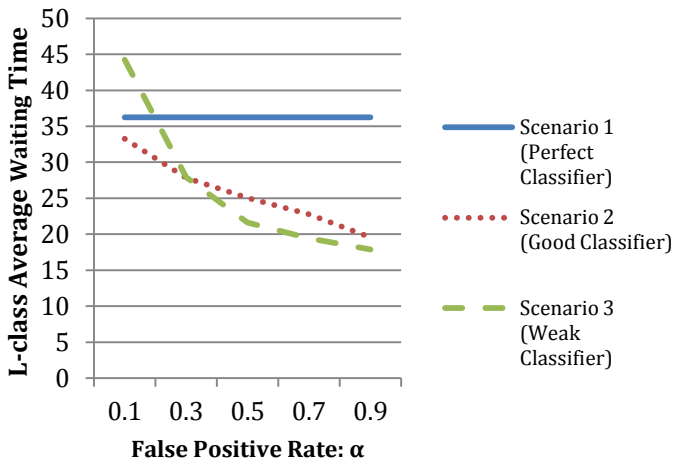

Figure 28. Impact of classifier configuration on L-class waiting times

In Figure 27, it can be seen that lower sensitivity configurations should be preferred for the good classifier (as opposed to the weak classifier) in order to minimize H-class average waiting times. This result can be explained by considering the number of misclassified H-class customers (i.e. false negatives) in the system. It is evident that when the classifier performance is good, there will be less misclassification in H-class, compared to the case when the classifier is weak. In this situation, the benefit of further reducing misclassification in H-class (by increasing α) does not justify the unintended outcome of increasing the number of misclassified L-class customers (i.e. false positives), in the Class 1 queue. This phenomenon can be observed in Figure 28, where there is a steady decrease in L-class average waiting times for the good classifier. On the other hand, when the classifier is weak, increasing α helps direct a higher number of misclassified H-class customers (i.e. false negatives) back to the Class 1 queue. In Figure

29, it can be seen that the good classifier incurs lower total waiting costs under less sensitive configuration, compared to the weak classifier.

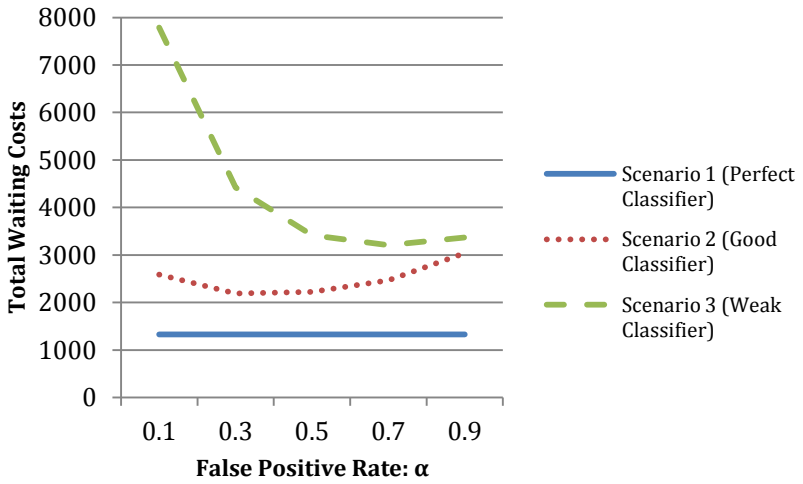

Figure 29. Impact of classifier configuration on total waiting costs

4.3.3 Interaction of Reserve Capacity and Classifier Configuration

In this section, I consider the joint decision on reserve capacity and classifier configuration to minimize total waiting costs and observe how a change in one variable affects a change in another. Tables 12 and 13 present the different reserve capacity and classifier configuration combinations that achieve a minimum in total waiting costs among the choices in S_{res} *x* S_{α}.

Table 12. Best α under increasing reserve capacity

Scenario 1 (Perfect Classifier)		Scenario 2 (Good Classifier)		Scenario 3 (Weak Classifier)	
Res.	Best $\boldsymbol{\alpha}$	Res.	Best $\boldsymbol{\alpha}$	Res.	Best $\boldsymbol{\alpha}$
0%	any	0%	0.3	0%	0.5
20%	any	20%	0.3	20%	0.5
40%	any	40%	0.3	40%	0.7
60%	any	60%	0.5	60%	0.9

Table 13. Best reserve capacity under increasing α

Scenario 1 (Perfect Classifier)		Scenario 2 (Good Classifier)		Scenario 3 (Weak Classifier)	
α	Best Res.	α	Best Res.	α	Best Res.
0.1	60%	0.1	0%	0.1	0%
0.3	60%	0.3	0%	0.3	0%
0.5	60%	0.5	20%	0.5	0%
0.7	60%	0.7	60%	0.7	0%
0.9	60%	0.9	60%	0.9	0%

Table 12 reveals that when the firm wants to increase its reserve capacity, it may also need to increase the classifier sensitivity in order to keep the waiting costs at minimum. In particular, it can be observed that the weak classifier's configuration to be more receptive to the changes in reserve capacity. There are two factors that may be attributed to this outcome. First, higher sensitivity configurations tend to offset the impact of excessive reservation by channeling L-class customers onto the H-class queue. By definition, higher reserve capacity means keeping more agents idle for H-class customer arrivals. When α is increased, these idle agents will be utilized to serve L-class customers who are falsely classified as H-class. Second, as mentioned in 4.3.2, increasing α helps direct a higher number of misclassified H-class customers (i.e. false negatives) to the Class 1 queue under weak classification.

Table 13 shows the best variable combinations from an increasing α perspective. For the perfect classifier, it can be seen that changing α has no effect on the reserve capacity decision. This is expected, since false positive and false negative rates will always be zero under perfect classification. For the good classifier, it can be observed that it is better to reserve more agents when α is increasing. This is because that by increasing

α, the majority of customers will be regarded as H-class and will wait in the Class 1 queue. This will ensure that those who are not in Class 1 will most likely to be true L-class. Therefore, reserving more capacity for Class 1 will surely benefit true H-class customers. However, the same result cannot be observed when the classifier is weak. This is due to the fact that under weak classification, there is still the possibility that customers who are not in the Class 1 queue may, in fact, be true H-class, even if α is increasing.

Overall, it can be seen that reserve capacity and classifier configuration decisions are interrelated. When the classification is good, a firm may increase its reserve capacity along with classifier sensitivity in order to keep the waiting costs at minimum. When the classification is weak, an increase in reserve capacity would most likely cause an increase in classifier sensitivity, however the opposite case may not hold due to the increased number of false negatives.

4.4 Conclusion

In this study, I considered the problem of allocating service resources to customers requesting live-chat communication at an e-commerce website. Resource allocation is a commonly studied problem in the literature, where priority-processing policies were proposed to maximize benefits when the availability of resources is fixed. I presented a reserve-based priority-processing policy for e-commerce systems that employ customer classification. A unique aspect of the investigation was the consideration of classification errors in different priority groups. This is a more realistic assumption due to difficulties associated with segmenting online customers.

To study the impact of the policy, I modeled a two-class live-chat queuing system as a continuous time Markov process and derived explicit expressions for average waiting times of both customer classes. Average waiting times were affected by two variables in the proposed policy: 1) the cut-off value for the reserve capacity of agents, and 2) classifier configuration. Using numerical experiments, I analyzed the steady-state behavior of the system for three business scenarios with different classifier performances.

The results indicated that reserving agents for H-class customers may generate negative impact on average waiting times of H-class customers, especially when the classifier is weak. I also found the performance of the classifier to be a major determinant in finding the best reserve capacity and classifier configuration combination. By studying the interaction of these two policy variables, I found out that a firm may increase its reserve capacity along with classifier sensitivity in order to keep the waiting costs at minimum, especially when the classification is good.

5 STUDY FOUR – ASSESSING THE VALUE OF CROSS-TRAINING FLEXIBILITY IN LIVE-CHAT CONTACT CENTERS

5.1 Introduction

Effective management and design of contact centers gain importance as the service industry grows in US and worldwide. Adoption of online channels such as live-chat, e-mail and web forums has lowered the communication barriers between customers and organizations, leading to a surge in the number of inquires that contact centers need to resolve every day. Incoming Call Management Institute (ICMI) estimated that by 2008, The United States would have over 47,000 contact centers and 2.7 million agents would be working in these centers (Aksin, Armory, & Mehrotra, 2007). 45% of contacts arriving to these contact centers happen via online channels (Froehle, 2006).

Under this situation, a vital problem for organizations is how to cope with the increasing demand for contact center services. Basing its roots on the notion of flexibility (Buzacott & Mandelbaum, 2008), cross-training has been a viable approach to address this problem. Cross-training service agents (i.e. resources) in diverse skills increases the number of customer classes that an agent can serve. In the presence of service demand variability among customer classes, cross-trained agents can be dynamically re-allocated to serve customers that are waiting in queues. This leads to higher agent utilization, prompter inquiry resolution and consequently increased customer satisfaction (Tekin, Hopp, & van Oyen, 2009).

Unfortunately, cross-training is costly and the benefits obtained from it can vary significantly between different cross-training structures. For example, training an agent on a secondary skill (i.e. increasing the agent flexibility) would be uneconomical if this agent is never idle due to a continuous stream of primary skill class customers. As a result, deciding on where to add the cross-training under a limited training budget remains an important question for contact centers. My aim is to address this question.

In this chapter, I study the impact of agent flexibility on the contact center performance. Specifically, I consider contact centers with live-chat communication channel and services. The type of the channel is important as it distinguishes this work based on the following property: live-chat channels enable agents to perform multi-tasking, i.e. processing multiple customers simultaneously. Under multi-tasking, a consequence of increasing agent flexibility is to process not just multiple customers but also multiple classes of customers together. Unfortunately, studies about human cognition have showed that processing different classes of tasks together (e.g. jobs, customers, etc.) may harm agent performance. Multi-tasking related issues such as increased task-switching and work interruption have been shown to produce resumption delays and increased stress on agents (Iqbal & Horvitz, 2007). Such problems are additional costs associated with increasing agent flexibility. As to our knowledge, neither the multi-tasking aspect nor additional flexibility related costs (besides training costs) have been considered in contact center literature before.

To account for these issues, I model a multi-class, multi-server queueing system with processor sharing discipline. The system is flexible in the sense that customers can

be processed by more than one server and servers can process more than one class of customers. Customer admission policy is dynamic and follows the longest queue first rule. Service times are state dependent, meaning that they depend on the cognitive load of the server that is affected by the number and class parity of customers in service. Processor sharing discipline divides the capacity of the server accordingly among all customers being processed by a particular server at a given time.

Based on this model, I study the impact of adding agent flexibility to the system under varying conditions. A particular point of interest is to observe the interaction between task switching costs and average waiting times. Further, I am also interested in seeing whether the relative location of the flexibility makes a difference. Specifically, I test the validity of a well-known flexibility structure called the chaining structure, if applied to contact centers.

There have been recent studies in literature that tackle similar questions. (Aksin & Karaesmen, Characterizing the performance of process flexibility structures, 2007) investigated the value of cross-training in the context of queueing systems such as contact centers. They applied network flow analysis to a stochastic dynamic network model to explore the relationship between different cross-training structures and their value. Of particular interest to my work is the study of Gurumurthi and Benjaafar (2004), who also studied the benefits of cross-training flexibility in queueing systems using a Markov chain model. Their model has been valuable for comparing static and dynamic admission control policies under varying flexibility structures for a non-shared server discipline. Finally, Jordan, Inman and Blumenfeld (2004) used queueing and simulation analysis to

study cross-training for labor workforce in a manufacturing plant. Their results demonstrated the robustness of the chaining strategy for making cross-training decisions.

My contribution to this literature has been twofold: 1) First, by using empirical data, I have demonstrated the existence of task switching delay in the context of live-chat contact centers. This had been an overlooked issue in contact-center research, in which switching costs were considered to be negligible. 2) Second, as to our knowledge, this has been the first study that considers both switching costs as well as multi-tasking ability of servers in an analytical model for flexible queueing systems. Numerical experiments based on this model have shown that increasing the flexibility may have negative consequences on the average waiting times of customers in the system. Further, I have also shown that the relative location of flexibility links makes a difference − even under chaining structures. This result is contradictory to the *efficient capacity shifting* assumption of the concept of chaining.

5.2 Switching Costs in Live-chat Systems

First, I aim to validate the assumption of agent performance being affected by task switching. Task switching refers to changing the functionality of a server from one customer to another. Consistent with the goal of cross-training, this mechanism allows service agents to be utilized more efficiently, thereby reducing the wait times in the system. However, excessive task switching may bring more harm than benefits, in case there are costs associated with switching. Schultz, McClain and Thomas (2003) point out that loss of server efficiency caused by frequent changes of tasks is a poorly understood negative aspect of flexibility (i.e. cross-training) that needs to be investigated further. As

a result, I am interested to see if there exists switching costs associated with changing the task of a live-chat agent.

Switching costs in cross training are caused by loss of work time due to equipment and/or server transition delays as well as loss of performance associated with adapting to the new customer class. It is generally assumed that contact centers can mitigate the impact of such problems through the use of technology. Hopp and van Oyen (2004) argue that automated call routing algorithms have nearly eliminated the delays in assigning different customers to available service agents. Aksin, Karaesmen and Ormeci (2006) state that computer telephony integration allows service agents to access customer data immediately through their computers regardless of the customer class, therefore a physical switching time is non-existent or minimal. They conclude that since the impact of switching inefficiencies are relatively small, other types of issues such as training costs should be given more attention during cross-training decisions.

Unfortunately, while such assumptions may work for traditional communication channels such as phone service, they are not applicable to a new type of contact center channel, which is live-chat based communication. A unique feature of live-chat services is to support multiple conversations simultaneously. A service agent using a live-chat system is expected to multi-task between several chat sessions concurrently, thereby increasing the overall productivity. Unfortunately, the degree of productivity benefits gained from multi-tasking is subject to degradation over time and increasing workload (Iqbal & Horvitz, 2007). Multi-tasking related events such as process-switching and work interruption have been shown to negatively affect the performance of a service agent.

Psychology literature points out that loss of context associated with the switch can be a major delay factor during the resumption (e.g. re-orientation) of the initial task (Czerwinski, Horvitz, & Wilhite, 2004). Further, it has been shown that the negative effects of task switching is greater between dissimilar tasks, as opposed to similar tasks (Arrington, Altmann, & Carr, 2003).

This brief literature survey hints that there may be performance related costs associated with multi- tasking different customer classes simultaneously in live-chat systems. To study this assumption, I construct and test the following hypothesis:

H1: A live-chat session that is multi-tasked with different classes of customers takes longer than a live-chat session that is multi-tasked with the same class of customers.

5.2.1 Dataset

To test this hypothesis, I used the transactional data collected from a large-scale live-chat contact center. The contact center specializes in providing tax preparation and filing support. Due to the intricate nature of United States income tax laws and important consequences of filing incorrect tax returns, the contact center receives significant amount of traffic, especially during the tax filing season. Again, due to the complexity in tax preparation activity, question types (i.e. customer classes) that arrive to the contact center vary considerably. Specifically, the contact center categorizes its customers into 30 skill based classes, each of which requires a unique skill to be processed. These skills are provided by live-chat agents, who are trained in 4 skills on average. Agents can perform multi-tasking, with two chat sessions being handled by the same agent simultaneously. These simultaneous chat sessions may or may not be of the same skill. The dataset

includes 1.9 million unique chat sessions that occurred between January 2011 and April 2012 with the following information recorded for each session:

- Agent ID
- Skill ID
- Customer ID
- Contact arrival and departure times
- Chat session begin and end times
- Contact's wait time in queue before being served by an agent
- Chat session service (i.e. handling) time

5.2.2 Data Processing / Transformation

First, I pre-processed the data to remove entries with null data values and outliers in terms of chat session handling time. Observing that a chat session on average took about twenty minutes, I removed chat sessions that have taken shorter than five minutes or longer than one hour. This is a reasonable assumption since a very brief or a very long chat session could indicate a technical problem (such as dropping out of the chat connection or not closing the chat session properly) rather than the actual length of the chat. Another pre-processing issue was related to the skill groups. Exploration of the skill definitions indicated that several skills were quite similar in nature, however had different IDs. Combining such skills, I reduced the skill set from 30 to 22.

Since the point of interest is the multi-tasking property of live-chat service, the goal was to identify the multi-tasked chat sessions in the dataset. Unfortunately, the existing data did not include such a feature by default. In fact, the very definition of

multi-tasking can be vague in nature because contacts arrive to and leave the system at random times while the agents are working on a continuous-time basis. For example, an agent can admit a contact into the service while having started to work on another contact already. In addition, since contact service times are also assumed to be independent of each other, it is unlikely that both contacts will leave the system at the same time. Figure 30 illustrates this situation, in which there are two contacts being served by the same agent while arriving to and leaving the system at different times. As a result, the agent performs multi-tasking only for *k* units of time, but not for the entire length of each service.

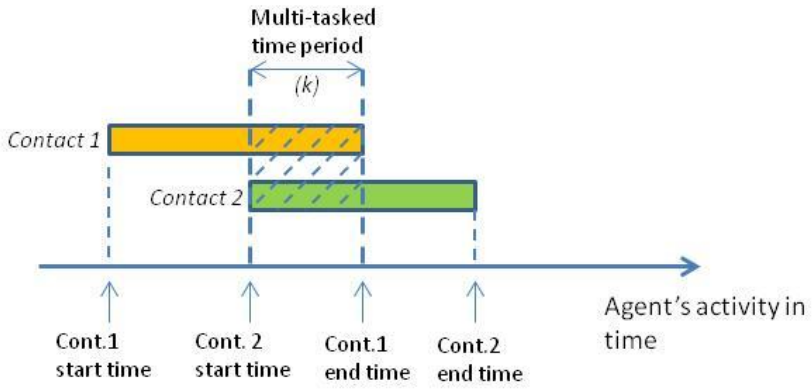

Figure 30. Multi-tasking definition

To annotate the dataset with a multi-tasking feature, I followed the following steps. First, I identified all chat session pairs, where $k > 0$. For each contact in each pair, I checked for the following property:

$$\frac{k}{(contact\ end\ time - contact\ start\ time)} \geq T \tag{5.1}$$

where *T* is an arbitrary threshold value. For contacts that satisfied this property, I classified them as multi-tasked contact. Further, if the contact had satisfied this property while the other contact in the pair having a different skill ID, I classified this contact as multi-tasked contact with different skills.

5.2.3 Results

I tested the hypothesis using the difference between average service times for multi-tasked contacts with different skills and multi-tasked contacts with the same skills. For multi-tasking identification, I used three different T scenarios: .25, .50 and .75. Table 14 shows the results including average service times, number of observations, t test values and p values for the two groups.

Table 14. Comparison of multi-tasking with same skills vs. with different skills

	T = .25		*T* = .50		*T* = .75	
	Same skills (Group 1)	Diff. skills (Group 2)	Same skills (Group 1)	Diff. skills (Group 2)	Same skills (Group 1)	Diff. skills (Group 2)
Average service time	1216.14	1261.39	1128.00	1147.40	996.34	1003.37
N	326143	255005	263434	211621	179768	151968
t test	-23.131		-9.705		-3.337	
p value	.000***		.000***		.001***	

To control for the effect of different skill frequencies in each group, I conducted a similar analysis on an independent skill level. Figure 31 provides the frequency distributions of skills for *T* = .50. It can be seen that the majority of chat sessions require only a handful of skills. I selected 5 skills out of 22 and run independent samples t-test analyses for each of these skills. The results were given in Table 15.

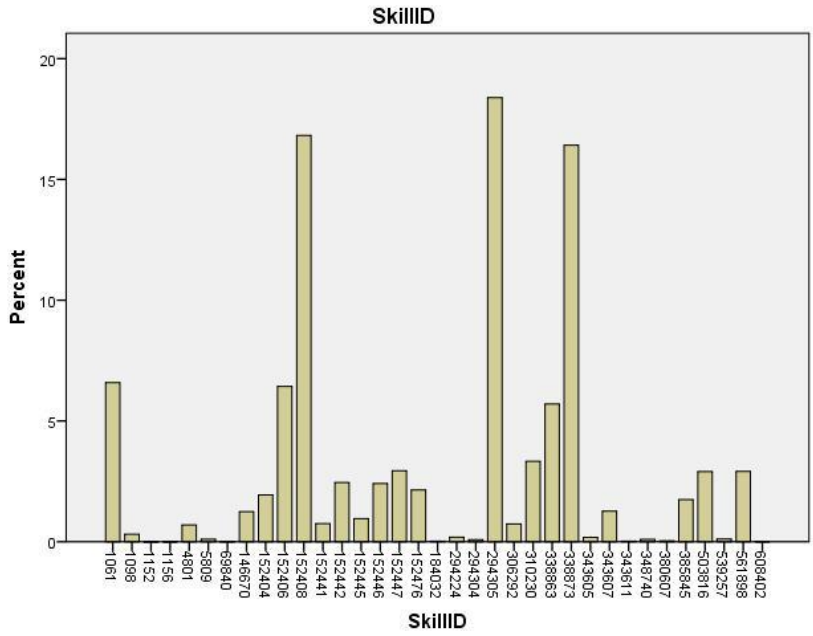

Figure 31. Frequency distributions

Table 15. Comparison by skill level

294305	*T* = .25		*T* = .50		*T* = .75	
	Same skills (Group 1)	Diff. skills (Group 2)	Same skills (Group 1)	Diff. skills (Group 2)	Same skills (Group 1)	Diff. skills (Group 2)
Average service time	1158.23	1221.66	1062.44	1111.22	922.01	958.56
N	100388	8923	80008	7319	54412	5383
t test	-7.613		-5.752		-4.254	
p value	.000***		.000***		.000***	
310230	T = .25		T = .50		T = .75	
	Same skills (Group 1)	Diff. skills (Group 2)	Same skills (Group 1)	Diff. skills (Group 2)	Same skills (Group 1)	Diff. skills (Group 2)
Average service time	1160.77	1250.66	1061.93	1143.13	917.34	988.21
N	11404	8614	8978	6851	6316	4774
t test	-8.464		-7.339		-6.133	
p value	.000***		.000***		.000***	

152441	T = .25		T = .50		T = .75	
	Same skills (Group 1)	Diff. skills (Group 2)	Same skills (Group 1)	Diff. skills (Group 2)	Same skills (Group 1)	Diff. skills (Group 2)
Average service time	992.18	1155.13	951.04	1070.93	823.84	941.20
N	586	3968	467	3111	295	2001
t test	-6.242		-4.378		-4.102	
p value	.000***		.000***		.000***	
561898	T = .25		T = .50		T = .75	
	Same skills (Group 1)	Diff. skills (Group 2)	Same skills (Group 1)	Diff. skills (Group 2)	Same skills (Group 1)	Diff. skills (Group 2)
Average service time	1178.36	1364.91	1087.10	1201.20	955.82	1059.15
N	17525	779	19277	575	8700	381
t test	-6.729		-3.953		-3.402	
p value	.000***		.000***		.001***	
385845	T = .25		T = .50		T = .75	
	Same skills (Group 1)	Diff. skills (Group 2)	Same skills (Group 1)	Diff. skills (Group 2)	Same skills (Group 1)	Diff. skills (Group 2)
Average service time	1316.11	1398.42	1211.49	1266.04	1055.81	1035.38
N	10098	637	7821	475	5249	296
t test	-2.553		-1.566		.559	
p value	.011**		0.118		0.559	

The majority of the results in Table 15 supports the argument that multi-tasking different skills simultaneously negatively affects the performance of a live-chat agent. The extent of this effect is largely dependent on the skill set and can be quite significant for certain skills.

5.3 The Model

I consider a live-chat queueing system with m customer classes and n agents. Classes of customers arrive to the system according to independent Poisson processes with rates λ_i, $i = \{1,2, \dots m\}$. The service requirement a class i customer is assumed to be exponentially distributed with mean $1/\mu_i$. Agents work according to a processor sharing discipline and with the speed of $\frac{v_j}{x(s)}$, where v_j is the inherent processing speed of agent j per customer ($j = \{1,2, \dots n\}$), $x(s)$ denotes the total number of customers the agent is currently serving and s is the current state of the system, which will be defined below. Further, assume that the cognitive performance of the agent is defined by $p(s)$, where is a value between 0 and 1. Values of $p(s) < 1$ indicate inefficiency in agent's cognitive performance due to excessive task switching. Combining these parameters, I define the service rate of a class i customer under agent j when the system is in state s as follows:

$$\mu_i^j(s) = \frac{\mu_i * v_j}{x(s)} * p(s) \tag{5.2}$$

Each agent can multi-task two customers maximum simultaneously and customers are admitted to the service according to following control policy: upon arrival, an arriving customer will get routed to an idle agent (who is trained on this customer's class) immediately, if such an agent is available. If two or more agents are available, the customer will get routed according to a static and arbitrary priority scheme. If no agent is available, the customer will enter the queue for his/her own class, which is emptied on a first come first serve (FCFS) basis. The system capacity is finite and an arriving customer is considered to be blocked if there are already C customers in the queue space. Upon

departure, the longest waiting customer whose queue is also the longest will be immediately picked up by the idling agent. If no customers are available, the agent will stay idle until the arrival of a new customer.

To incorporate flexibility structures into the model, let A = $[a_{ij}]$ be an $m \times n$ matrix that represents all possible customer-agent assignments as well as agent priorities of customer classes. If $a_{ij} = k, k > 0$, then class i customers can be processed by agent j with priority preference of k. Lower k values indicate higher preference. If $a_{ij} = 0$, then class i customers cannot be processed by agent j. To clarify this representation, consider the matrices below that accompany the flexibility structures in Figure 32 with relevant priority preferences.

$$A^{1a}{}_{3\times3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad A^{1b}{}_{3\times3} = \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \qquad A^{1c}{}_{3\times3} = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \\ 2 & 3 & 1 \end{bmatrix}$$

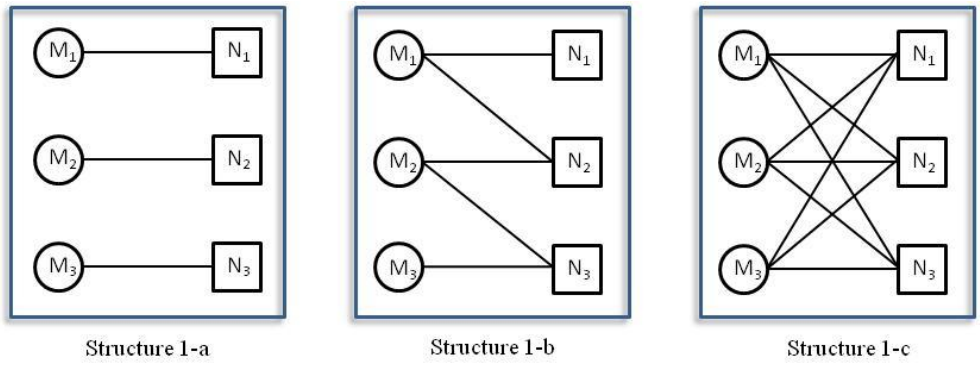

Figure 32. Sample flexibility structures

This system can be represented as a continuous time Markov chain (CTMC) with the state of the system being described by specifying: 1) the number of customers in queue for each customer class and 2) the number and class type of customers in service

for each agent. Let s represent the state of the system in $m \times (n+1)$ dimensions, i.e. $s \equiv \left(d_1, d_2, \dots, d_{m\times(n+1)}\right)$. First m dimensions (i.e. $d_1, d_2, \dots d_m$) represent the number of customers in queues, while dimensions d_{m+1} to d_{2m} represent the state of the first agent and d_{2m+1} to d_{3m} represent the state of the second agent and so on. Based on model assumptions, let S be the set of all feasible states and defined as:

$$S = \left\{ \left(d_1, d_2, \dots, d_{m\times(n+1)}\right) \mid \forall\, d_i \in \mathbb{N}^0, i = \{1,2, \dots m \times (n+1)\}, \sum_{i=1}^{m} d_i \leq C,\ \ \forall \sum_{k=1}^{m} d_{um+k} \leq 2, u = \{1,2, \dots n\} \right\} \tag{5.3}$$

Transition into a new state occurs when: (1) a customer arrives to the system and (2) a customer in service leaves the system. Let Q represent the transition-rate matrix, i.e. $Q = (q_{\mathrm{st}})$. State transitions with corresponding rates for this system can be defined by using the algorithm in Table 16.

Table 16. Transition-rate matrix generating algorithm

Input: Set *S*, Matrix *A*
BEGIN
Q = [ ]; // initialize empty transition-rate matrix
$p(s)$ = *f;* // initialize the *p(s)* function
for each $s \in S$

// Arrival transitions
  for each $i = \{1,2, \dots m\}$ // for each customer class arrival
    for each j, $A_{ij} = 1: A_{ij} = u, u > 0$ // check each possible agent assignment, start from highest preference
      if $\sum_{k=i*j+1}^{i*j+m} d_k < 2$ // if agent j is not fully occupied
        $t \equiv \left(d_1, d_2, \dots, \left(d_{\mathrm{m*j+i}} + 1\right), \dots, d_{m\times(n+1)}\right)$; // new state − *i* goes to j 's

service
$Q_{st} = \lambda_i$; // update transition-rate matrix
break
elseif $A_{ij} == u + 1$ // all agents are busy
$t \equiv (d_1, d_2 \ldots, (d_{\mathrm{i}} + 1), \ldots, d_{m\times(n+1)})$; // new state − i goes to the corresponding queue
$Q_{st} = \lambda_i$; // update transition-rate matrix
end if
end for
end for
// END Arrival transitions

// Departure transitions
for each $d_{\mathrm{k}} \in$ s, $k = m + 1 : m(n + 1)$
if $d_{\mathrm{k}} > 0$ // for each customer in service
aID = floor $((k - 1)/m)$; // find the ID of the agent that served the departing customer
cID = $k - (aID * m)$; // find the ID of the departing customer
dLQ = $max(d_u), for\ every\ A_{u,aID} > 0\ and\ u < m$; // find the length of the longest queue that can be served by the idling agent
if dLQ == 0 // if there is no customer in serviceable queues
$t \equiv (d_1, d_2, \ldots, (d_{\mathrm{k}} - 1), \ldots, d_{m\times(n+1)})$; // new state – agent becomes idle
$Q_{st} = Q_{st} + \frac{\mu_{cID}}{\left(\sum_{v=(aID*m)+1}^{aID*(m+1)} d_v\right)} * p(s)$; // update transition rate matrix
else
cLQ = u for $d_u ==$ dLQ; // find the ID of the customer waiting in the longest queue
$t \equiv (d_1, d_2, \ldots, (d_{\mathrm{cLQ}} - 1), \ldots (d_{\mathrm{aID*m+cLQ}} + 1), \ldots, d_{m\times(n+1)})$; // new state –customer cLQ gets admitted into agent aID's service
$Q_{st} = Q_{st} + \frac{\mu_{cID}}{\left(\sum_{v=(aID*m)+1}^{aID*(m+1)} d_v\right)} * p(s)$; // update transition rate matrix
end if
end if
end for
// END departure transitions

end for
END

Note that the steady-state condition for the Markov chain will always exist, since the state space cannot grow beyond the system capacity. Denote $\vec{\pi} = (\pi_1, \pi_2, \dots \pi_{|S|})$ as the steady-state distribution vector of this CTMC. The vector $\vec{\pi}$ can be computed as the unique solution of the linear system $\vec{\pi}Q = 0$, where Q can be generated by using the algorithm in Table 16. Employing the normalization condition of $\sum_{s \in S} \pi_s = 1$, this system of linear equations can be solved by using a general-purpose linear equation solver (Kleinrock, 1975).

Using the steady-state distribution vector $\vec{\pi}$, various performance characteristics of the system can be calculated. First, define the blocking probability P_b (i.e. the probability that the system is full and incoming customers are rejected) as follows (for the ease of representation, let s denote state $(d_1, d_2, \dots, d_{m\times(n+1)})$):

$$P_b = \sum_{\sum_{i=1}^{m} d_i = C} \pi_s \tag{5.4}$$

In addition, the expected number of customers in the system can be derived as:

$$N^{sy} = \sum_{s \in S} \pi_s \times \sum_{i=1}^{m\times(n+1)} d_i \tag{5.5}$$

Using (5.4) and (5.5), Little's Law can be applied to define the expected waiting time in the system as follows:

$$E(W) = \frac{N^{sy}}{\sum_{i=1}^{m} \lambda_i (1 - P_b)} \tag{5.5}$$

5.4 Numerical Experiments

In this section, I run two sets of numerical experiments to study the impact of agent flexibility on contact center performance. The first experiment investigates how increasing flexibility affects the expected waiting times in the system. The second experiment studies if the relative locations of flexibility links matter under chaining structures.

5.4.1 The Impact of Increasing Flexibility

An intuitive expectation for introducing flexibility into a system is to improve productivity by making a better use of service agents. Unfortunately, in real world settings, there are several factors that can offset the advantages associated with increased flexibility. In this sub-section, two scenarios are considered where flexibility may have a reverse effect on waiting times.

5.4.1.1. Scenario 1: when switching costs are considered:

When switching costs are considered, increasing the flexibility poses an interesting trade-off: it helps improve agent utilization, but at the same time agents' cognitive performances diminish due to multi-tasking of different customer classes. To see this effect, I run the model with five different flexibility structures given in Figure 33. As can be seen from the Figure, flexibility increases as the structure number increases (with Structure 1-1 having no flexibility and Structure 1-5 having full flexibility).

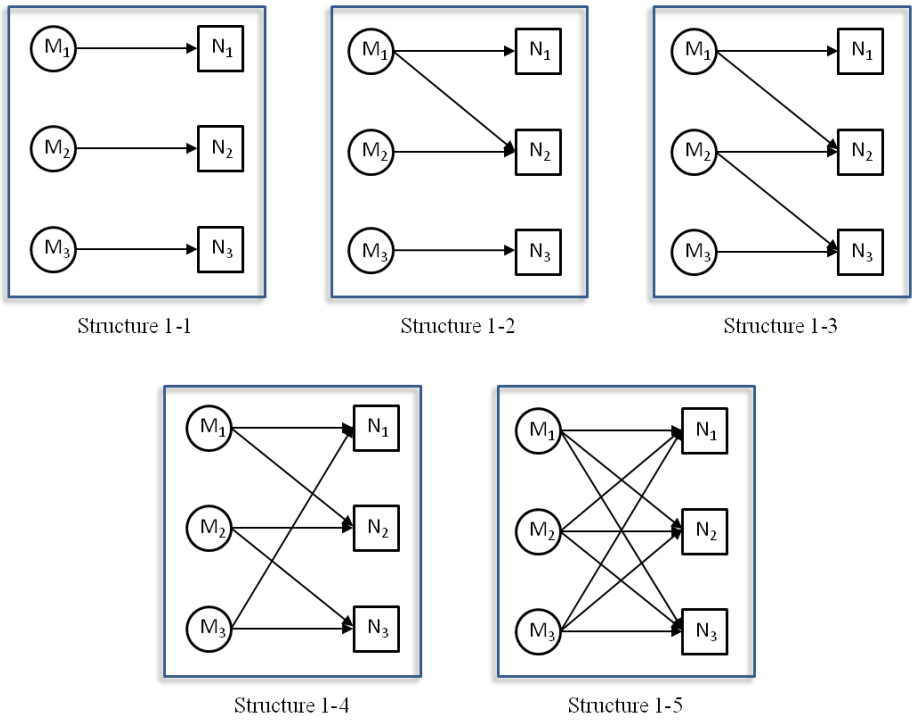

Figure 33. Flexibility structures for scenario 1

I keep the system symmetric (i.e. all customer classes have the same arrival rates and service time requirements and all agents have the same processing speed as well as the same cognitive performance) in order to exclude any external effects. In addition, I run the same experiment under two different utilization levels (i.e. system load) − high traffic (ρ = .9 - Structure 1) and low traffic (ρ = .4 - Structure 1) to control for utilization effects. The base case values are set as follows: $\mu_i = 10$ customers per hour, $i = \{1,2, \dots m\}$ and $v_j = 1, j = \{1,2, \dots n\}$. For high traffic, $\lambda_i = 9$ customers per hour and for low traffic, $\lambda_i = 4$ customers per hour. For computational considerations, I limit the queue buffer size to 5 customers the total system capacity to 11 customers.

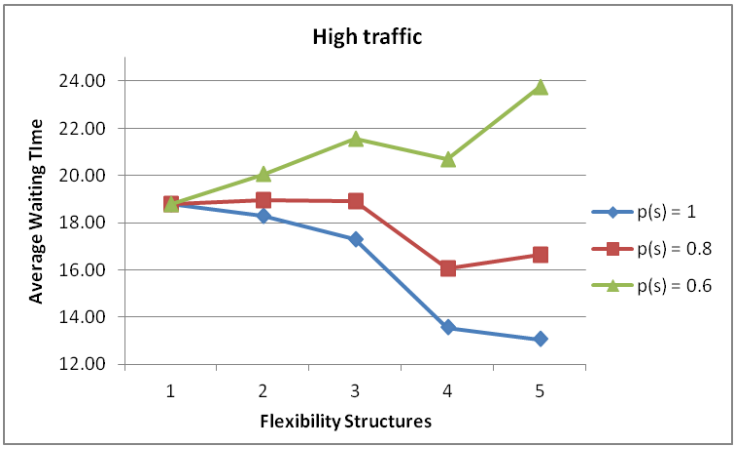

Figure 34. The impact of increasing flexibility under high traffic

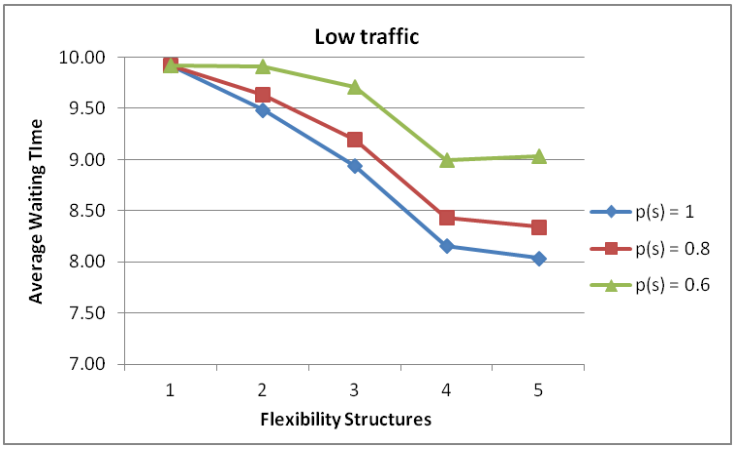

Figure 35. The impact of increasing flexibility under low traffic

Figure 34 depicts the effect of gradually increasing the flexibility on expected waiting times under different $p(s)$ levels for the high traffic case. Figure 35 shows the results for the low traffic case. In these Figures, it can be seen that average waiting times may increase as flexibility in the system is increased. Unsurprisingly, the extent of increase is more significant under high traffic regime. This result validates the concern

about flexibility: in realistic environments where switching costs exist, increasing the flexibility may do more harm to the system than its intended benefits.

5.4.1.2. Scenario 2: when the system is asymmetric:

The value of flexibility for symmetric systems is well established in literature. However, large-scale contact centers such as the one that motivated this work are rarely symmetric in nature. In fact, it would be naïve to expect that all customer classes would arrive to the system at the same rate and demand the same service time, when there are more than 20 different customer classes. In this situation, increasing the flexibility may not generate its intended consequences. To test this issue, I run a simple scenario, where the flexibility of one agent in an asymmetric system is increased gradually as shown in Figure 36. Assignment priorities for these structures are given above the Figure. The second agent (i.e. N_2) in Structure 2-3 is called the super agent, who can process all customer classes.

$$A^{2.1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad A^{2.2} = \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad A^{2.3} = \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \\ 0 & 2 & 1 \end{bmatrix}$$

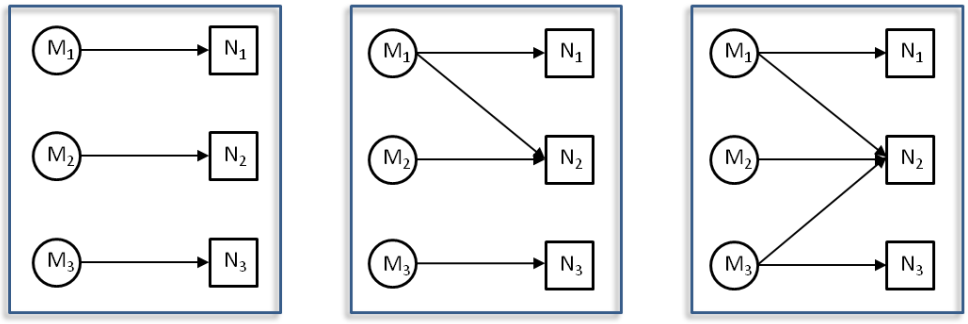

Figure 36. Flexibility structures for scenario 2

It is desirable to see the change in average waiting times as flexibility of the second agent is increased under different arrival rates. While keeping the total arrival rate constant, I tried the following asymmetric arrival rate configurations: 1) $\lambda_{M1} = 5,\ \lambda_{M2} = 10,\ \lambda_{M3} = 5$, 2) $\lambda_{M1} = 6,\ \lambda_{M2} = 8,\ \lambda_{M3} = 6$ and 3) $\lambda_{M1} = 7,\ \lambda_{M2} = 6,\ \lambda_{M3} = 7$ (all values are customers per hour).

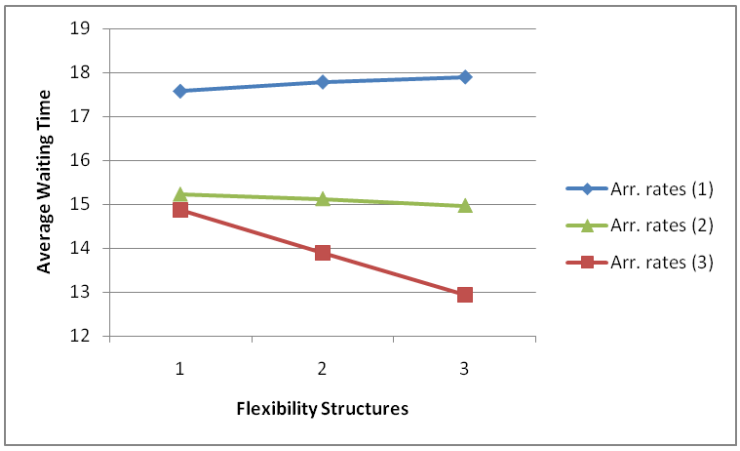

Figure 37. Scenario 2 results

The results are given in Figure 37. From these results, it can be seen that even a mild degree of asymmetry can offset the benefits of flexibility. Predictably, I observe that when an agent is already highly utilized, there wouldn't be much need for higher flexibility. A surprising result is that this can actually cause an increase (albeit slightly) in overall waiting times. It can be expected that as the asymmetry levels become more significant in the system, flexibility performance could get worse.

5.4.2 The Impact of Flexibility Location

In this experiment, I am interested to see if there exist capacity shifting inefficiencies that occur solely because of the relative location of the flexibility links. Capacity shifting refers to the contact center's ability to reallocate idle server capacity among different customer types in order to respond to fluctuations in demand. Clearly, the essential requirement to shift capacity between two servers is the existence of a cross-training path between them. In the presence of such a cross training structure, higher than expected demand that cannot be served immediately by a certain agent can then be assigned to a separate idle agent.

Note that for the general class of flexibility problems, it is not necessary for the two resources to be directly linked via a demand node in order to be able to share capacity. As long as there exists a path between them, it is assumed that extra capacity from one resource can be shifted to the other resource. This is a well-known property of a special flexibility strategy called chaining (Jordan & Graves, 1995). However, this property may not hold for queueing systems and there may be performance difference even between different structures of the same chaining configuration. Figure 38 shows one of those chaining configurations for a 3x3 system with 2 different structures.

$$A^{3.1} = \begin{bmatrix} 1 & 0 & 2 \\ 2 & 1 & 0 \\ 0 & 2 & 1 \end{bmatrix} \qquad A^{3.2} = \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 2 \\ 2 & 0 & 1 \end{bmatrix}$$

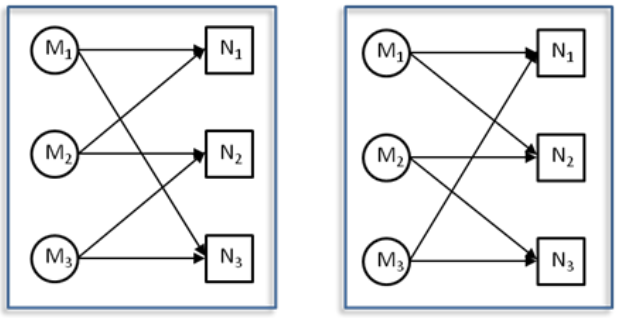

Figure 38. Different structures of the same chaining configuration

To see if different structures of the same chain would react to demand variability similarly, I run the model under different arrival rates while keeping the total arrival rate constant. All arrival rate configurations in the experiment simply represent an increase in one node, while the same amount of decrease in the other node and are given as follows: 1) $\lambda_{M1} = 1,\ \lambda_{M2} = 5,\ \lambda_{M3} = 9$, 2) $\lambda_{M1} = 1,\ \lambda_{M2} = 9,\ \lambda_{M3} = 5$, 3) $\lambda_{M1} = 9,\ \lambda_{M2} = 5,\ \lambda_{M3} = 1$, 4) $\lambda_{M1} = 9,\ \lambda_{M2} = 1,\ \lambda_{M3} = 5$, 5) $\lambda_{M1} = 5,\ \lambda_{M2} = 9,\ \lambda_{M3} = 1$ and 6) $\lambda_{M1} = 5,\ \lambda_{M2} = 1,\ \lambda_{M3} = 9$. Service rates are 10 customers per hour for each agent.

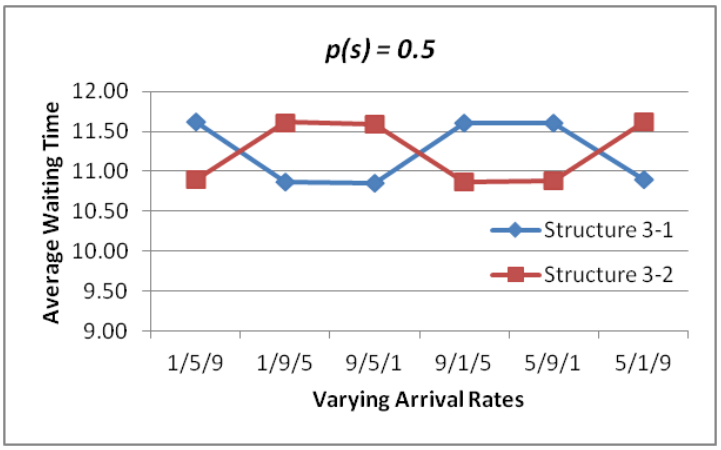

Figure 39. Results of the experiment under *p(s)* = 0.5

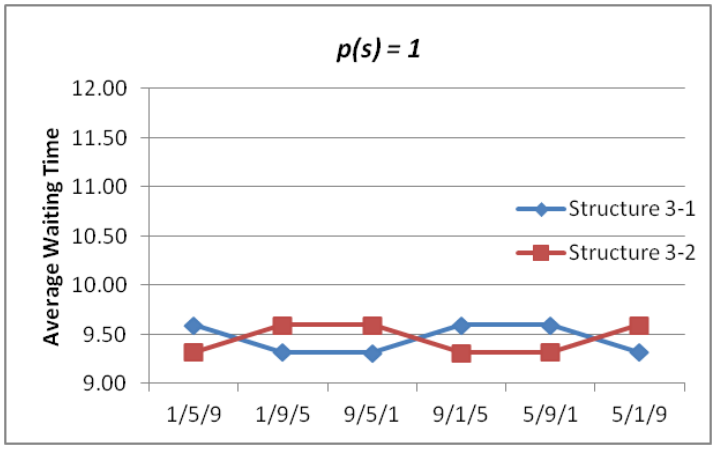

Figure 40. Results of the experiment under *p(s)* = 1

Figures 39 and 40 show the results of this experiment under $p(s)$ values: 0.5 and 1.0, respectively. From the Figures, it can be seen that not all structures of the same chain react similarly to the arrival rate variability. While there is no clear dominant structure, it can be seen that for certain arrival rates, one performs better than the other one. This difference is gratified as agent cognitive performance drops. This result implies that

effective capacity sharing assumption of the chaining strategy may not hold for live-chat contact centers and it must be considered cautiously.

5.5 Conclusions

In this section, I studied the impact of cross-training flexibility in live-chat contact centers. A unique aspect of this work was the consideration of switching costs. Through empirical analysis, I first showed that these costs exist in live-chat environments due to multi-tasking nature of chat communications. Then, I analytically modeled a multi-class, multi-server live-chat queueing system with processor sharing discipline. Solution of the model provided insights about the impact of adding agent flexibility to the system under varying conditions.

The results have indicated that increasing the flexibility can generate contrary effects in the system. While it may help agent utilization, it can also lead to cognitive overload in agent performance, thereby slowing down agent service productivity. The extent of these two contrary effects can distinguish how much flexibility would actually be valuable for the system. Further, the results have also shown that the relative location of flexibility links is another important performance consideration – even for chaining structures that were thought to not be affected by this issue.

6 CONCLUSION

Servitization of IT offers great opportunities for organizations to streamline their resources and to gain agility in competitive environments. Yet, many technical and managerial challenges lie ahead to leverage its full promise. In this dissertation, I have presented four studies that developed methods and policies to overcome some of these challenges and unlock the potential of service-orientation in IT.

In the first study, I aimed to address the issue of web service development out of legacy enterprise software. These web services represent repeatable business tasks that can be accessed on demand over a network. They can be assembled together to compose complex business functionality, enabling organizations to quickly adapt to changing conditions and requirements. An important condition to create such services is the existence of business semantics in the source code. Unfortunately, legacy enterprise software rarely includes such semantics or annotations inside the source code components. To overcome this problem, I developed an information retrieval based approach that makes use of data naming similarity to connect business definitions with source code components. Data names were extracted from both ends through parsing techniques. Then a similarity function was applied to identify equivalences between the naming sets. Based on identity matches, the source code model was annotated with service specific knowledge. The entire process can be automated via parsing tools and similarity function implementations. The approach was validated through a case study on SAP's Enterprise System Framework. The results indicated that the proposed approach is

useful for creating service-based knowledge in the source code components of an existing enterprise system.

In the second study, I developed a financial valuation model to analyze the monetary impact of service-oriented architecture investment in an organization. This is an important topic since SOA is a costly investment whose benefits have yet to be clearly defined in academia and the industry. By combining traditional NPV analysis with option pricing models, I expanded the basic value of SOA to account for strategic options and managerial flexibility. The specific model included three types of options: growth, deferral and switch-use. Methods on how to compute the value of each option as well as operational costs and benefits were provided. The model was validated through a case study in an organization that was going through SOA implementation. The results indicated that the model is valuable for supporting managerial decision making with respect to SOA investments.

In the third study, I considered the problem of allocating service resources to customers requesting live-chat communication services at an e-commerce web site. Specifically, I investigated the applicability of reserve type priority-processing policies in literature in the context of live-chat services. A unique aspect of my investigation was the consideration of classification errors in different priority groups. This was a more realistic assumption due to difficulties associated with segmenting online customers. To achieve this, I derived the explicit expressions for average waiting times in a two class queueing system with misclassification. Average waiting times were affected by two variables: 1) the cut-off value for reserve capacity, 2) classifier configuration. The results

indicated that reserving agents for high class customers may generate negative impact on the same class of customers, especially when the classifier is weak. I also found the performance of the classifier to be a major determinant in finding the best reserve capacity and classifier configuration. By studying the interaction of these two policy variables, I showed that a firm may increase its reserve capacity along with classifier sensitivity in order to keep the waiting costs at minimum.

In the fourth and the final study, I analyzed the impact of cross-training flexibility in live-chat contact centers. A unique aspect of this work was the consideration of switching costs. Through empirical analysis, I first showed that these costs exist in live-chat environments due to multi-tasking nature of chat communications. Then, I analytically modeled a multi-class, multi-server live-chat queueing system with processor sharing discipline. Solution of the model provided insights about the impact of adding agent flexibility to the system under varying conditions. The results indicated that increasing the flexibility can generate contrary effects in the system. While it may help agent utilization, it can also lead to cognitive overload in agent performance, thereby slowing down agent service productivity.

In conclusion, this dissertation contributes to the emerging service science literature by overcoming some of the problems that hinder the applicability of IT Servitization. As such, it is expected to have significant impact in both theory and practice.

For future research, I plan to continue investigating resource management in service-oriented systems in two directions. The first direction is quality-of-service (i.e.

QoS) driven resource allocation in SOA, in which customer requests and QoS constraints are identified at the business process level, but resources can only be allocated at the service level. Since a business process consists of multiple services and a service can be used in multiple processes, an optimal resource allocation policy for this problem would need to consider business process workflows along with QoS constraints and resource usage costs. The second direction I would like to pursue involves auction-based mechanisms for cloud services. As more service providers offer virtualized computing resources using e-services over the Internet, efficient allocation of these resources to utilize spare capacity becomes a challenge. I intend to study auctioning mechanisms that can help trade and distribute computing in benefits of both service providers and customers.

REFERENCES

Aberg, J., & Shahmehri, N. (2000). The Role of Human Web Assistants in e-Commerce: An Analysis and a Usability Study. *Internet Research : Electronic Networking Applications and Policy , 10* (2), 114-125.

Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2006). *Compilers: Principles, Techniques, and Tools.* (2nd, Ed.) Addison-Wesley Longman Publishing Co., Inc.

Aksin, Z., & Karaesmen, F. (2007). Characterizing the performance of process flexibility structures. *Operations Research Letters , 35* (4), 477-484.

Aksin, Z., Armory, M., & Mehrotra, V. (2007). The Modern Call Center: A Multi-Disciplinary Perspective on Operations Management Research. *Productions and Operations Management* , 665-688.

Aksin, Z., Karaesmen, F., & Ormeci, L. (2006). A review of Workforce Cross-Training in Call Centers From an Operations Management Perspective. *Working paper* .

AMR Research . (2008). *The SOA Spending Report, 2007-2008: Spending Reaches New Heights.* AMR Research, Inc. .

Andrews, D. C., & Haworth, K. N. (2002). Online Customer Service Chat: Usability and Sociability Issues. *Journal of Internet Marketing , 2* (1).

Argon, N. T., & Ziya, S. (2009). Priority Assignment under Imperfect Information on Customer Type Identities. *Manufacturing & Service Operations Management , 11* (4), 674-693.

Arrington, C. M., Altmann, E. M., & Carr, T. H. (2003). Tasks of a feather flock together: Similarity effects in task switching. *Memory & Cognition* , 781-789.

Atar, R., Mandelbaum, A., & Reiman, M. I. (2004). Scheduling a Multi Class Queue with Many Exponential Servers: Asymptotic Optimality in Heavy Traffic. *The Annals of Applied Probability , 14* (3), 1084-1134.

Baines, T. S., Lightfoot, H. W., Benedettini, O., & Kay, J. M. (2009). The servitization of manufacturing: A review of literature and reflection on future challenges. *Journal of Manufacturing Technology Management , 20* (5), 547-567.

Balasooriya, J., Padbye, M., Prasad, S. K., & Navathe, S. B. (2005). A System for Distributed Coordination of Workflows over Web Services. *In the Proceedings of 19th IEEE International Parallel and Distributed Processing Symposium.*

Banker, R. D., Datar, S. M., Kemerer, C. F., & Zweig, D. (1993). Software Complexity and Maintenance Costs. *Communications of the ACM , 36* (11), 81-94.

Banker, R. D., Datar, S. M., Kemerer, C. F., & Zweig, D. (2002). Software Errors and Software Maintenance Management. *Information Technology and Management* (3), 25-41.

Banker, R. D., Davis, G. B., & Slaughter, S. A. (1998). Software Development Practices, Software Complexity, and Software Maintenance Performance: A Field Study. *Management Science , 44* (4), 433-450.

Beimborn, D., & Joachim, N. (2009). Proposing the Relationship between IT Business Alignment and the Business Value of Service-Oriented Architectures in Financial Firms,. In *Enterprise Applications and Services in the Finance Industry* (pp. 115-122). Springer Berlin Heidelberg.

Benaroch, M. (2002). Managing Information Technology Investment Risk: A Real Options Perspective. *Journal of Management Information Systems , 19* (2), 43-84.

Benaroch, M., & Kauffman, R. J. (1999). A Case for Using Real Options Pricing Analysis to Evaluate Information Technology Project Investments. *Information Systems Research , 10* (1), 70-86.

Benaroch, M., & Kauffman, R. J. (2000). Justifying Electronic Banking Network Expansion Using Real Options Analysis. *MIS Quarterly , 24* (2), 197-225.

Bieberstein, N., Bose, S., Walker, L., & Lynch, A. (2005). Impact of service-oriented architecture on enterprise systems, organizational structures, and individuals. *IBM Systems Journal , 44* (4), 691-708.

Black, F., & Scholes, M. (1973). The Pricing of Options and Corporate Liabilities. *The Journal of Political Economy , 81* (3), 637-654.

Boehm, B., Clark, B., Horowitz, E., Westland, C., Madachy, R., & Selby, R. (1995). Cost Models for Future Software Life Cycle Processes: COCOMO 2.0. *Annals of Software Engineering* (1), 57-94.

Buzacott, J. A., & Mandelbaum, M. (2008). Flexibility in manufacturing and services: achievements, insights and challenges. *Flexible Services and Manufacturing Journal , 20* (1), 13-58.

Central Intelligence Agency. (2012). *The World Factbook.* Retrieved 2012 йил 24-July from https://www.cia.gov/library/publications/the-world-factbook/geos/us.html

Chapman, S. (2006). Retrieved from http://www.dcs.shef.ac.uk/~sam/stringmetrics.html

Chapman, S. (2007). Retrieved from http://sourceforge.net/projects/simmetrics/

Chesbrough, H. (2005). Toward a science of services. *Harvard Business Review , 83*, 16-17.

Chesbrough, H., & Spohrer, J. (2006). A Research Manifesto for Services Science. *Communications of the ACM , 49* (7), 35-40.

Chikofsky, E. J., & Cross, J. H. (1990). Reverse Engineering and Design Recovery: A Taxonomy Software. *IEEE , 7* (1), 13-17.

Cobham, A. (1954). Priority Assignment in Waiting Line Problems. *Journal of the Operations Research Society of America , 2* (1), 70-76.

Cohen, W. W., Rayikumar, P., & Fienberg, S. (2006). From http://sourceforge.net/projects/secondstring/

Cohen, W. W., Rayikumar, P., & Fienberg, S. E. (2003). A Comparison of String Distance Metrics for Name-Matching Tasks. *Proceedings of the IJCAI Workshop on Information Integration on the Web*, (pp. 73-78).

Czerwinski, M., Horvitz, E., & Wilhite, S. (2004). A Diary Study of Task Switching and Interruptions. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI).*

Dai, Q., Kauffman, R. J., & March, S. T. (2007). Valuing information technology infrastructures: a growth options approach . *Information Technology and Management , 8* (1), 1-17.

DeBaud, J. M., Moopen, B., & Rugaber, S. (1994). Domain analysis and reverse engineering. *Proceeding of International Conference on Software Maintenance*, (pp. 326-335).

Demiriz, A., Kula, U., & Akbilek, N. (2009). A Framework for Balanced Service and Cross-selling by Using Queuing Science. *Journal of Intelligent Manufacturing , 20*, 249-257.

Demirkan, H., Kauffman, R. J., Vayghan, J. A., Fill, H.-G., Karagiannis, D., & Maglio, P. P. (2008). Service-oriented technology and management: Perspectives on research and practice for the coming decade. *Electronic Commerce Research and Applications , 7* (4), 356-376.

Dietrich, B., & Harrison, T. (2006). Serving the services: the emerging science of service management opens opportunities for operations research and management science. *OR/MS Today , 33* (3), 42-49.

Erdogmus, H., & Vandergraaf, J. (1999). Quantitative Approaches for Assessing the Value of COTS-centric Development. *In Proceedings of the Sixth International Symposium on Software Metrics*, (pp. 279-290). West Palm Beach, Florida.

Esogbue, A. O., & Singh, A. J. (1976). A Stochastic Model for an Optimal Priority Bed Distribution Problem in a Hospital Ward. *Operations Research , 24* (5), 884-898.

Favaro, J. M., Favaro, K. R., & Favaro, P. F. (1998). Value based software reuse investment," Annals of Software Engineering. (5), 5-52.

Froehle, C. M. (2006). Service Personnel, Technology, and Their Interaction in Influencing Customer Satisfaction. *Decision Sciences , 37* (1), 5-38.

Gabriel, D. (2008). *SAP Network Wiki*. Retrieved from https://wiki.sdn.sap.com/wiki/display/ESpackages/Sales+Order+Processing

Gerchak, Y., Gupta, D., & Henig, M. (1996). Reservation Planning for Elective Surgery under Uncertain Demand for Emergency Surgery. *Management Science , 42* (3), 321-334.

Gravano, L., Ipeirotis, P. G., Jagadish, H. V., Koudas, N., Muthukrishnan, S., Piertarinen, L., et al. (2001). Using q-grams in a DBMS for approximate string processing. *IEEE Data Engineering , 24* (4), 28-34.

Guo, G. Y., Atlee, J. M., & Kazman, R. (1999). A Software Architecture Reconstruction Method. *In Proceedings of Working Conference on Software Architecture*, (pp. 15-33).

Gurumurthi, S., & Benjaafar, S. (2004). Modeling and analysis of flexible queueing systems. *Naval Research Logistics , 51* (5), 755-782.

Gurvich, I. (2004). Design and Control of the M/M/N Queue with Multi-Class Customers and Many Servers. M.Sc. Thesis. Technion.

Hochstein, A., Zarnekow, R., & Brenner, W. (2005). ITIL as Common Practice Reference Model for IT Service Management: Formal Assessment and Implications for Practice. *In Proceedings of the IEEE International Conference on E-Technology, E-Commerce and E-Service.* Hong Kong.

Hopp, W. J., & Oyen, M. P. (2004). Agile workforce evaluation: a framework for cross-training and coordination. *IIE Transactions , 36*, 919-940.

IBM Global Business Services. (2006). *Service-oriented architecture: A practical guide to measuring return on that investment.* Whitepaper, IBM Institute for Business Value.

IBM Global Technology Services. (2008). *How service-oriented architecture (SOA) impacts your IT infrastructure: Satisfying the demands of dynamic business processes.* Whitepaper, IBM Global Services.

Iqbal, S. T., & Horvitz, E. (2007). Disruption and Recovery of Computing Tasks: Field Study, Analysis and Directions. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI).*

Jaro, M. A. (1995). Probabilistic Linkage of Large Public Health Data Files. *Statisitics in Medicine , 14* (5), 491-498.

Jordan, W. C., & Graves, S. C. (1995). Principles on the benefits of manufacturing process flexibility. *Management Science , 41* (4), 577-594.

Jordan, W. C., Inman, R. R., & Blumenfeld, D. E. (2004). Chained cross-training of workers for robust performance. *IIE Transactions , 36*, 953-967.

Kajko-Mattson, M., Lewis, G. A., & Smith, D. B. (2008). Evolution and Maintenance of SOA-Based Systems at SAS. *In Proceedings of the 41st Hawaii International Conference on System Sciences*, (pp. 119-129). Waikoloa, Hawaii.

Kalyanam, K., & Zweben, M. (2005). The Perfect Message at the Perfect Moment. *Harvard Business Review , 83* (11), 112-120.

Kazman, R., & Carriére, S. J. (1999). Playing Detective: Restructuring Software Architecture from Available Evidence. *Automated Software Engineering , 6* (2), 107-138.

Kleinrock, L. (1975). *Queuing Systems; Theory* (Vol. 1). New York: John Wiley & Sons.

Kondrak, G. (2005). N-Gram Similiarity and Distance. *Proceedings of the Twelfth International Conference on String Processing and Information Retrieval*, (pp. 115-126).

Kuhn, A., Ducasse, S., & Girha, T. (2005). Enriching Reverse Engineering with Semantic Clustering. *In Proceedings of 12th Working Conference on Reverse Engineering*, (pp. 133-142).

Lawrie, D., Field, H., & Binkley, D. (2007). Extracting Meaning from Abbreviated Identifiers. *In Proceedings of the Seventh IEEE International Working Conference on Source Code Analysis and Manipulation.*

Lemahieu, W. (2001). *Web Service description, advertising and discovery: WSDL and beyond* (J. Vandenbulcke and M. Snoeck (eds.), New Directions in Software Engineering, ed.). Leuven University Press.

Liu, D., Sarkar, S., & Sriskandarajah, C. (2010). Resource Allocation Policies for Personalization in Content Delivery Sites. *Information Systems Research , 21* (2), 227-248.

Luthria, H., & Rabhi, F. (2008). Organizational Constraints to Realizing Business Value from Service Oriented Architectures: An Empirical Study of Financial Service Institutions. *In Proceedings of the 6th International Conference on Service-Oriented Computing, Springer Berlin Heidelberg,* (pp. 256-270).

Maurizio, A., Sager, J., Jones, P., Corbitt, G., & Girolami, L. (2008). Service Oriented Architecture: Challenges for Business and Academia. *In Proceedings of the 41st Hawaii International Conference on System Sciences,* (pp. 315-323). Big Island, Hawaii.

Montoya, M. M., Massey, A. P., & Khatri, V. (2010). Connecting IT Services Operations to Services Marketing Practices. *Journal of Management Information Systems , 26* (4), 65-85.

Murphy, G. C., & Notkin, D. (1996). Lightweight Lexical Model Extraction. *ACM Transactions on Software Engineering and Methodology , 5* (3), 262-292.

Papazoglou, M. P., & van den Heuvel, W.-J. (2007). Service-Oriented Architectures: Approaches, Technologies and Research Issues,. *The VLDB Journal , 16* (3), 389-415.

Patrick, P. (2005). Impact of SOA on Enterprise Information Architectures. *In Proceedings of the ACM SIGMOD International Conference on Management of Data*, (pp. 844-848).

Pekoz, E. A. (2002). Optimal Policies for Multi-server Non-preemptive Priority Queues. *Queuing Systems , 42,* 91-101.

Peltz, C. (2003). Web Services Orchestration and Choreography. *Computer , 36* (10), 46-52.

Pinzger, M., Fischer, M., Gall, H., & Jazayeri, M. (2002). Revealer: a lexical pattern matcher for architecture recovery. *In Proceedings of the Ninth Working Conference on Reverse Engineering.*

Postmus, D., & Meijler, T. D. (2008). Aligning the economic modeling of software reuse with reuse practices. *Information and Software Technology , 50,* 753-762.

Qiu, L., & Benbasat, I. (2005). An Investigation into the Effects of Text-to-Speech Voice and 3D Avatars on the Perception of Presence and Flow of Live Help in

Electronic Commerce. *ACM Transactions on Computer-Human Interaction , 12* (4), 329-355.

Sallé, M. (2004). *IT Service Management and IT Governance: Review, Comparative Analysis and their Impact on Utility Computing", HPL-2004-98.* Palo Alto, CA, : Hewlett-Packard Laboratories.

Schaack, C., & Larson, R. C. (1986). An N-Server Cutoff Priority Queue. *Operations Research , 34* (2), 257-266.

Schultz, K. L., McClain, J. O., & Thomas, L. J. (2003). Overcoming the dark side of worker flexibility. *Journal of Operations Management , 21*, 81-92.

Singh, C., Shelor, R., Jiang, J., & Klein, G. (2004). Rental software valuation in IT investment decisions. *Decision Support Systems , 38*, 115-130.

Singla, P., & Domingos, P. (2006). Entity Resolution with Markov Logic. *In Proceedings of the Sixth International Conference on Data Mining*, (pp. 572-582).

Song, J. H., & Zinkhan, G. M. (2008). Determinants of Perceived Web Site Interactivity. *Journal of Marketing , 72*, 99-113.

Spohrer, J., & Maglio, P. P. (2007). The emergence of service science: Toward systematic service innovations to accelerate co-creation of value. *Production and Operations Management , 17* (3), 1-9.

Stohr, E. A., & Zhao, J. L. (2001). Workflow automation: overview and research issues. *Information System Frontiers , 3* (3), 281-296.

Taudes, A. (1998). Software Growth Options. *Journal of Management Information Systems , 15* (1), 165-185.

Taylor, I. D., & Templeton, J. G. (1980). Waiting Time in a Multi-Server Cutoff-Priority Queue, and Its Application to an Urban Ambulance Service. *Operations Research , 28* (5), 1168-1188.

Tekin, E., Hopp, W. J., & van Oyen, M. P. (2009). Pooling strategies for call center agent cross-training. *IIE Transactions , 41*, 546-561.

Tiwana, A., & Konsynski, B. (2010). Complementarities Between Organizational IT Architecture and Governance Structure. *Information Systems Research , 21* (2), 288-304.

Trigeorgis, L. (2005). Making use of real options simple: An overview and applications in flexible/modular decision making. *The Engineering Economist , 50* (1), 25-53.

Tsourveloudis, N. C., & Valavanis, K. P. (2002). On the Measurement of Enterprise Agility. *Journal of Intelligent and Robotic Systems , 33*, 329-342.

van, d. B., Klint, P., & Verhoef, C. (1997). Re-engineering Needs Generic Programming Language Technology. *ACM SIGPLAN Notices , 32*, 54-61.

Vandermerwe, S., & Rada, J. (1988). Servitization of business: adding value by adding services. *European Management Journal , 6* (4), 314-324.

Vara, J. M., de Castro, V., & Marcos, E. (2005). WSDL automatic generation from UML models in a MDA framework. *International Journal of Web Services Practices , 1* (1), 1-12.

webMethods Inc. (2005). *The Business Case for SOA: Rationalizing the Benefits of Service-Oriented Architecture, Technical Report.* webMethods, Inc.

Wirth, N. (1996). *Compiler Construction.* Addison Wesley Longman Publishing Co., Inc.

World Wide Web Consortium. (2001). From http://www.w3.org/TR/wsdl

Yoon, T., & Carter, P. (2007). Investigating the Antecedents and Benefits of SOA Implementation: A Multi-Case Study Approach". *In Proceedings of the Thirteenth Americas Conf. on Inf. Sys.*, (pp. 1-11).

Zeithaml, V. A., Rust, R. T., & Lemon, K. N. (2001). The Customer Pyramid: Creating and Serving Profitable Customers. *California Management Review , 43* (4), 118-142.

Zhao, J. L., Tanniru, M., & Zhang, L.-J. (2007). Services computing as the foundation of enterprise agility: Overview of recent advances and introduction to the special issue. *Information Systems Frontiers , 9* (1), 1-8.