UTRECHT UNIVERSITY

Revisiting Legacy Software System Modernization

By

Ravi Khadka

A thesis submitted in partial fulfillment for the
degree of Doctor

in the
Faculty of Science
Department of Information and Computing Sciences

April 2016

SIKS Dissertation Series No. 2016-14
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Cover photo: Ravi Khadka@Fort Lunet III, Houtensepad 150, 3524 SB Utrecht
Cover design: Jayraj Bhatta/Arun Pratihast
ISBN/EAN: 978-90-393-6512-0

Revisiting Legacy Software System Modernization

Modernisering van Legacy Systemen Herbeschouwd

(met een samenvatting in het Nederlands)

Proefschrift

ter verkrijging van de graad van doctor aan de Universiteit Utrecht op gezag van de rector magnificus, prof.dr. G.J. van der Zwaan, ingevolge het besluit van het college voor promoties in het openbaar te verdedigen op woensdag 13 april 2016 des middags te 12.45 uur door

Ravi Khadka

geboren op 30 augustus 1984 te Jharuwarasi-2, Lalitpur, Nepal

Promotoren:	Prof.dr. S. Brinkkemper
	Prof.dr. J.T. Jeuring
Copromotoren:	Dr. R.L. Jansen
	Dr. J. Hage

This research was financially supported by the Dutch Joint Academic and Commercial Quality Research and Development (JACQUARD) program on Software Engineering Research under ServiciFi project.

Preface

This PhD endeavor started back when I was doing my masters. I doubt if I would have ever taken this research path had I not met BN Sapkota. Your guidance and numerous reviews of my papers that you've done over the years do deserve a sincere thanks.

Slinger, Jurriaan, Sjaak, and Johan- thank you for providing me this opportunity and supporting me over the years. We shared a lot of good memories- running out of travel budget, acceptance of papers- particularly ICSE paper, playing volleyball so on so forth. Slinger, thank you for your confidence in me. In particular at the last stage of my research, you believed on my decision and let me pursue my direction. Jurriaan, you've been the best reviewer of all our papers and my dissertation.

This dissertation is a result of close collaboration with industry. Heartily thanks to Rob Douwes, Geer P. Haas, and Edwin van Dis for supporting this research. The industrial insights that our ServiciFi research team received from collaborating with industry are of key values to the research community and without any doubt to my dissertation.

Michiel & Amir, due to you guys I count the four years as one of the cherished moments- be it those handful of paper acceptances and bunch of rejections or those all off-track discussions- 9gag & phdcomics. I am happy to count you guys as a part of this journey. Jaap, Kevin, Ivon, Wienand, Marijn, Cristobal, Erik, Garm, Jurriaan, Naser, Eko, Rogier, Oskari, Alexander, Sean, Anna, thanks for your company. Thanks to all the staff members of OenI group for your input, particularly, Fabiano & Remko- I will not forget your contributions in some of my papers.

I am grateful to my masters students whom I (in-)directly supervised. Gijs, Andrie, Belfrit, Bart, Thijs, and Prajan- thank you for your wonderful work and for intense discussions we had. Georgiana, Angeliki, and Dr. Maria- without your statistical expertise, I would never be an ISCE-ier. Tracy- I am grateful for your immediate availability to proof-read my papers and even this dissertation.

Thanks to Dai, Qianjie, Yu, Bing/Bo, and Quy. I treasure the moments that I've shared with you guys as housemates. Within this research period, I've lived with Michael & Lisa. Feb 10, 2014 was "the day the music died"- Michael, your songs and memories we've shared are everlasting. To all my Nepali friends here in the Netherlands- Ajit/Rubbi, Ashim, Arun/Puja, Barsha, Chandra/Sujata, Ekraj, Gaurab/Meng, Govinda, Khagendra/Bhagwati, Mohan, Prajwal/Ruby, Prajan/Priyanka, Rabindra/Renu, Rameshwor/Gehini, Reena, and Sunil- thanks for creating a home away from home. Yiouli, Nicole, Michalis- thanks for your Greek hospitality.

Finally, I wholeheartedly thank my family and friends back home for your unconditional support. Especially, to my mother and father- your continuous love and encouragement has finally paid off. I owe my thanks to my brothers/sister-in-laws, I can't image this dissertation would have been complete without your supports. Ananta/Bhaba, Sajjan/Srinkhala, and Him- thank you all for your (in-)direct contribution to this work.

As of now, when I think of this journey- what an adventure it is! Submission deadlines, paper accepts & rejects, conference visits & side trips, never ending Dutch course with Ricky (Kiwi), 2015 Nepal earthquake and your generous support. Simply, this dissertation is not just only about my research but a memorable snapshot of my life. Thank you all.

Contents

IV Finale

Chapter 1

Introduction

Enterprise information systems are indispensable backbones for enterprises. Enterprises have become dependent on information systems for the day to day running of their business, in the sense that they may go bankrupt because of prolonged system failures. Enterprise systems are defined as implementation and customization of software packages that enable the integration of transaction-oriented data and business processes throughout an organization [185]. Such systems enable an enterprise to integrate their data used throughout the entire organization by a seamless integration of the information flowing through the company such as financial and accounting information, human resource information, supply chain information, and customer information [73].

Many information systems have been operating within enterprises for decades, and consequently they are entrenched in the enterprises. The rich feature sets of information systems result in many different departments becoming dependent on them, and thereby making them irreplaceable. Furthermore, through many customizations, the information systems have been adapted to fit the needs of the enterprise, further strengthening the system's foothold in the organization. It is not surprising that information systems have been heavily adapted because of the needs of the enterprise and hence include significant organizational knowledge as business logic to perform daily operations [185, 196].

As information systems acquire a history of heavy customizations and adaptations according to the different demands and purposes of the enterprises over time, often the information systems become entrenched within the enterprise, and as a result conflict with the constant changes that enterprises undergo. Enterprises are challenged by business drivers such as adopting new business requirements, changes in legislation, escalating customer expectations and as well as IT drivers such as changes in the technology infrastructure and platform [280]. Changes due to new business requirements are driven by mergers and acquisitions, reorganizations, adopting new business opportunities, and cross enterprise collaborations. For instance, various banks including ING, ABN AMRO, Lloyds announced job cuts as a part of reorganization and planned to invest in digital banking to adopt new business opportunities [204]. Often, changes in new rules and legislations are key drivers of change within enterprises. A major recent example of such change is the adoption of Single Euro Payment Area (SEPA)– a payment-integration initiative of the European Union (EU) for the simplification of bank transfers denominated in euro. With SEPA adoption, banks within the EU were obliged to adhere to the SEPA payments mechanism by 1 February 2014 [82]. Escalating customer expectations are another cause of changes within enterprises. With the new advances in Internet technologies, business–to–customer (B2C) interaction has profoundly

changed. For example, the banking domain has drastically transformed its distribution of services from physical branches of the banks providing services to customer to Automated Teller Machines (ATM), and recently to online internet banking and mobile banking. These transitions are observed due to customer expectations of easy banking and to avoid staying in queues in ATMs or retail banking outlets as much as possible [266]. Such business drivers for change bring about changes in the information systems as most enterprises heavily rely on application of IT [282].

In addition to business drivers, IT drivers are key in bringing changes within enterprise systems. Consider a traditional bank that heavily relied on customers visiting the physical offices is now considering to adopt a digital strategy by replacing as many traditional services with online services. Similarly, new emerging technologies such as using business intelligence and big data to serve their customers and to better manage risk, and to leverage cloud computing technology to reduce IT costs have been key trends in the banking domain as identified by Gartner [100]. Adoption of service delivery via mobile channels within the financial domain is a good example of a change enabled by IT. A survey in 2013 indicated that more than 60% of the banks consider services via mobile channels as a top priority [37]. ING, one of the leading banks in the Netherlands, has witnessed considerable impact in providing mobile banking. Since introducing a mobile based online banking, over 1.2 million people have downloaded the app [126, 37]. Additionally, IT drivers such as adding new products and system features, and reducing IT costs, challenge the enterprises to evolve their IT landscape. These drivers of changes, be it business or IT, exert tremendous pressure on enterprises to evolve their enterprise systems. By addressing these frequent changes, the enterprises aim to achieve more flexibility and agility, and enable faster–time–to market [281]. Meanwhile, the changes are often addressed with ad–hoc modifications with frequent updates and alterations [22, 4] within the information systems thereby resulting in "ignorant surgery" [211]– a consequence of changing software without understanding the original design concept. Nevertheless, frequent changes in the information systems must at all time guarantee the desired functional and non-functional quality attributes, such as stability, availability, dependability and security.

Despite such "ignorant surgery" to adapt such changes, these systems are thoroughly tested, fine-tuned to optimize performance and represent a significant financial investment. As these systems mature, they reach a level of stasis: changes tend to be incremental, the architecture is frozen in time, and the system is managed with a "if it ain't broken, don't fix it" mentality. Ad–hoc modifications lead to unstructured code and incomplete or often lack of documentation regarding the changes [283, 196]. In the meantime, the systems become an integral part of the enterprise by adapting to the changes as per organizational needs. The day-to-day business is heavily dependent on the enterprise system such that a single failure can have a significant impact on business. For instance, an IT system failure at Royal Bank of Scotland (RBS) cost the bank £175m [225] and a failure at National Air Traffic Services (NATS) in the UK led to cancellation/delay of flights across all airports in the UK[1].

As these systems age, enterprises struggle with several challenges- knowledge about the system slowly disappears: the initial developers and maintainers retire or change jobs, subcontractors go bankrupt or change their business, and documentation ages while incremental changes are being implemented. Eventually, the existing information systems become too fragile to modify, too expensive to redevelop, and too risky to replace as they perform the core business operations of the enterprises. Managers

[1] http://www.bbc.com/news/uk-30460619

are reluctant to incur the cost and risk involved in replacing such invaluable information systems that are critical to day-to-day operations. The continuous aging of these information systems results in obsolescence of parts of the system. Hence, the systems resist modifications, are expensive to maintain, and eventually reach the end of their lifecycle.

These information systems– often referred to as legacy software systems– have been developed over the previous decades using programming languages such as COBOL, RPG, PL/I, C, C++, Java, and currently remain active within enterprises despite their well-known issues such as being inflexible and hard to maintain. The underlying reason includes legacy systems support complex core business processes and hence are still vitally important to the enterprises. They simply cannot be removed as they implement and store critical business logic, while the proper documentation, skilled manpower, and resources to evolve or maintain these legacy software systems are actually scarce.

Hence, legacy software systems present a dilemma. On the one hand the legacy software systems stay at the core of day-to-day business and are vitally important to the continuity of the business, while representing a massive, long-term business investment [32]. On the other hand, maintaining these systems is difficult and expensive due to a lack of resources such as documentation and skill-set. It is important to note that most organizations with legacy systems are spending up to 75%–90% of their development resources in maintenance [241, 179, 1], leaving only 25% for innovation [190]. Due to the issues associated with legacy software systems, momentum is growing to modernize those legacy software systems towards new technological environments, primarily aiming at reducing the maintenance costs and increasing flexibility [32, 170]. In academia there is significant interest in the modernization of legacy software systems, and a plethora of research has been reported on the issues of legacy software systems and possible modernization strategies [8, 32].

Recent research on legacy software system modernization indicates that the academic software modernization methods focus on solving the technical issues [174, 221, 143], whilst legacy modernization has to include the associated business issues. Because of such issues, a well–defined approach for modernization is advocated by researchers [202, 175, 174, 196]. This need for a method combining technical, and business issues has led us to define our first research objective as:

> **Research Objective 1 (RO1)**–"Develop a software modernization method that includes technical and business aspects."

Legacy system modernization has received much attention in industry as well. For instance, in the executive survey of Gartner 2013[2], legacy system modernization is placed at position 5 out of the top 10 technology priorities [98] and has been within the top 10 priorities for last the three consecutive years [97]. As identified by Gartner [239], legacy software modernization aims at achieving the following objectives:

- to retain legacy applications indefinitely due to their core position in the market, while coping with ever-changing requirements,
- to improve business process efficiency and agility by integrating monolithic legacy systems, and
- to move to new solutions such as Software–as–a–Service (SaaS) and cloud computing.

[2] `http://www.gartner.com/newsroom/id/2304615`

Similarly, in 2014 the National Association of State Chief Information Officers[3] (NASCIO) placed legacy application modernization (renovation) in the 6th position out of the top 10 technologies, applications and tools priority [199]. These reports indicate that enterprises still struggle with their legacy software systems and are planning to modernize them.

Although, the research community realized the need for legacy system modernization in the mid 1990s and since then a plethora of legacy system modernization approaches have been reported. It is therefore surprising to see that enterprises are still dependent on legacy software systems. As of 2008, a market research report [265] from the National Computing Center[4] (NCC) states:

> *".....it has been estimated that about 80% of IT systems are running on legacy platforms. International Data Corporation estimates that 200 billion lines of legacy code are in use today on more than ten-thousand large mainframe sites..."*

Additionally, a market research reported by Microfocus in 2013 states that 1.5 million new lines of COBOL code are written every day to support 90% of the Fortune 500 business systems everyday and COBOL still powers 85% of all daily business transactions processed [189]. Similarly, in 2014 Gartner reported that 92 out of 100 banks use mainframe as their back-end to process their high volume transactions [61].

The evidence indicates that despite a plethora of modernization approaches, the problems of legacy software systems and their modernization are still prevalent in industry. Research conducted by Razavian & Lago [223] suggests that legacy software modernization methods reported in academia are too abstract to be implemented in industry and 97% of the academic approaches do not fit the industrial purposes in the context of legacy to SOA modernization. Market research conducted by consulting firms indicates that legacy software modernization projects in industry often overrun budget and time. For instance, a survey published by Gartner in 2014 indicates that legacy software modernization projects in the insurance domain have high failure rates– only 42% of projects meet the original budget, and 82% take longer than expected [99]. Similarly, Forrester Consulting [69] reports that 31% of modernization projects fail to meet the planned due date and 33% of modernization project exceed the planned budget. This indicates that there is a knowledge gap among academia and industry in understanding legacy systems and their modernization, thereby inhibiting the adoption of knowledge flow [221]. This knowledge gap leads us to define the other research objective of this thesis that aims at exploring how legacy software systems and their modernization are perceived within the industry.

Research Objective 2 (RO2)–"Identify how software modernization is perceived and conducted in practice."

1.1 Research Context

In this section we provide an overview of the concepts that form the foundation of this thesis along with the relevant literature. We start with the concept of software evolution in subsection 1.1.1 in which we depict how software evolution is related with legacy software systems and software modernization. In subsection 1.1.2, we present an overview of legacy systems with respect to the available literature.

[3] http://www.nascio.org/
[4] http://www.ncc.co.uk

The subsection further explores the academic perception of legacy software systems, in particular, the characteristics of legacy systems. In subsection 1.1.3, we discuss legacy modernization approaches.

1.1.1 Software Evolution

The term evolution describes a phenomenon that refers to progressive changes in the properties and characteristics of classes of entities such as natural species, societies, artefact, theories. The changes are intended to maintain the status quo or to improve fitness in a changing environment. Such changes are also inevitable in the life cycle of software systems. The seminal work of Belady & Lehman is still relevant in understanding the foundation of software evolution. They performed empirical experiments on OS/360 to understand software evolution [19]. This pioneering work on software evolution stated the first three laws of software evolution and by 1996, a total eight laws of software evolution were formulated [168]. All these laws are defined in the context of E-type systems [167], i.e., software systems that solve a problem or implement a computer application in the real world and are thereby inherently more change prone.

The law of continuing change– an E-type program that is used must be continually adapted, else it becomes progressively less satisfactory [167]– provides an intrinsic need for evolution.

The law of increasing complexity– as a program evolves, its complexity increases unless work is done to maintain or reduce it [167]– identifies the need for maintenance. Despite the fact that some researchers and practitioners use software evolution as a preferable substitute for maintenance [23], this research distinguishes these two terms. Software maintenance refers to activities that take place at any time after the new development project is implemented, whereas software evolution is focused on examining the behavior of software systems so as to identify and implement changes for adaptation/improvements [140]. Furthermore, software evolution also focuses on methods and tools intended to facilitate software evolution after the intimal software development [169].

Software maintenance includes activities that are largely aimed at keeping systems operational. Swanson [267] categorized maintenance into three types as: (i) corrective maintenance– performed in response to the assessment of failures, (ii) adaptive maintenance– performed in anticipation of changes to the data and processing environments, and (iii) perfective maintenance– performed to eliminate inefficiencies, enhance performance, and improve maintainability. Corrective and adaptive maintenance are focused on keeping the systems up and running whereas perfective maintenance is aimed at keeping the software systems up and running at less expenses and to better serve the needs of the users. Later, Chapin et al. [54] introduce a fourth type of software maintenance as "Preventive maintenance"– perform to prevent problems in the future by taking pre-emptive actions. ISO/IEC 14764 uses these as four categories of software maintenance [127].

Definition of Software Evolution–"*the dynamic behavior of programming systems as they are maintained and enhanced over their life times, and all the activities including tools and methods intended to facilitate software evolution" [19, 169].*

Definition of Software Maintenance– "*the correction of errors, and the implementation of modifications needed to allow an existing system to perform new tasks, and to perform old ones under new conditions" [79].*

1.1.2 Legacy System

The law of continuing change states that a system must be continually adapted or it becomes progressively less satisfactory. The perfective maintenance indicates the need of modifications to eliminate inefficiencies, enhance performance, and improve maintainability. Additionally, the preventive maintenance refers to modifications performed for the purpose of preventing problems before they occur. The law of continuous change, perfective and preventive maintenance put emphasis on addressing (remedial) changes to make software systems more maintainable, otherwise software systems gradually turn to legacy software– software systems that significantly resist modification and are less maintainable. Several other characteristics are inherent to legacy systems such as inflexible, brittle, expensive to maintain, lacking documentation, difficult to extend and integrate with other systems, and lacking legacy experts. In general, the following problems are often associated with legacy software systems:

- Legacy systems are typically implemented using obsolete technology, possibly in older programming languages and hardware platforms.
- The lack of documentation and experts of the legacy systems lead to knowledge erosion and hence resulting in a slow and expensive maintenance process.
- Well-defined interfaces are often lacking in the legacy systems, thereby requiring significant efforts to extend and integrate with other systems.

Despite these drawbacks inherent to legacy systems, the importance of a legacy system is widely acknowledged. Legacy systems are the backbone of enterprises and often regarded as an organizational asset with a high economic value [285]. These systems are mission critical and embed a lot of business logic that represent many years of coding, developments, enhancements, modification and testing. Therefore, enterprises cannot simply discard their legacy systems, despite the maintenance nightmares and underlying problems of software erosion [214].

Definition of Legacy System– "*any system that cannot be modified to adapt to constantly changing business requirements and is still valuable to its stakeholder such that its failure can have a serious impact on business" [43].*

1.1.3 Software Modernization

Enterprises with legacy systems are confronted with a dilemma [22]. Despite unjustifiable maintenance costs, enterprises cannot simply get rid of legacy software systems as they are core systems for running day-to-day business. Therefore, momentum is growing to evolve and reuse those legacy systems within new technological environments via software modernization [32]. The primary aim of software modernization is to reduce maintenance cost and increase flexibility.

Definition of Software Modernization– We define legacy modernization as "*the process of evolving existing software systems by replacing, re-developing, reusing, or migrating the software components and platforms, when traditional maintenance practices can no longer achieve the desired system properties" [142].*

Due to the inherent issues with legacy software systems, various software modernization techniques have been proposed. Such modernization techniques can broadly be categorized into software modernization strategies.

1.1.3.1 Software Modernization Strategies

Software modernization can be categorized into four different strategies [8] that are briefly discussed below:

1. **Replacement strategy**– is a way of retiring a legacy system and replacing it with a commercial-off-the-self (COTS) package. Replacement is considered to be less risky but there is little or no possibility of reusing the existing business logic embedded within the legacy system. Normally, legacy software systems are modified and customized in the course of a life-cycle and are the sources of undocumented business logic or enterprise knowledge. An option does exist in which the COTS package is modified as per the need of the enterprise, but this incurs substantial costs. Additionally, there is no guarantee that the replaced new system will be as robust and functional as the original one [68, 8].
2. **Wrapping**– is one of the most widely used modernization strategies that allows the possibility of encapsulating existing legacy software for reuse in a new target architecture [255]. In general, wrapping provides a customized access to the legacy code with minimal changes to the code base itself such that the wrapped component can be used by other software components. Wrapping is a quick win strategy and can be used when the legacy system has a high business value. However, wrapping does not reduce maintenance cost, rather increases it as the enterprise has to maintain the interface (wrapper) layer as well.
3. **Redevelopment**– is a strategy to redevelop the legacy system functionalities. However, the risk of failure is usually too large for enterprises to seriously adopt a redevelopment approach [32] and management is not willing to spend a significant amount of investment to an approach having huge risk of failure. With respect to cost, redevelopment does incur significant development cost and limited reuse of existing legacy assets. Less reuse of existing assets is justified by the fact that the documentation of legacy software systems are in general not up-to-date.
4. **Migration**– concerns the transformation of legacy software systems to a new technological context by maximizing reuse [127]. The migration strategy tends to be costly and time consuming compared to other strategies. However, a migration strategy gradually allows to internally restructure, reuse and modify the legacy systems into a new target system, thereby potentially reducing maintenance costs associated with legacy systems in the long run [273].

Figure 1.1 depicts how these strategies compare with respect to cost and reuse of existing assets. Each modernization strategy has its own pros and cons, hence a variety of factors such as available budget, resources, time constraints play an important role in choosing a strategy. Often two or more modernization strategies are combined to conduct legacy modernization, as there is no silver bullet tackling the problem [8].

1.1.3.2 Software Modernization Methods

Within the last four decades, a plethora of legacy system modernization methods have been reported upon. Sneed [252, 256] presents a method to migrate legacy software systems from a mainframe to a client-server architecture using wrapping techniques. Sneed & Majnar [259] discuss the use of wrapping to migrate legacy software systems to a client-server architecture at different levels of encapsulation such as job, transaction, program, module, and procedure. Souder & Mancoridi [261] present a tool to

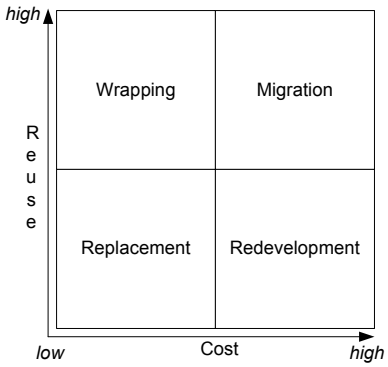

Figure 1.1: Comparison of modernization strategies with respect to cost and reuse [8]

integrate a legacy software system into a distributed environment using wrapping technology. Canfora et al. [44] describe an approach to decompose the legacy software systems using program slicing and then use wrappers to migrate to client–server architecture. For migrating legacy software systems to a client-server architecture, the wrapping technique is predominant [145].

With the advent of the object-oriented (OO) paradigm, numerous legacy to object-oriented programming language modernization methods are reported. Demeyer et al. [77] report several object-oriented re-engineering patterns to modernize object-oriented languages. Lucia et al. [74] present a six phase sequential legacy to object-oriented migration process that encompasses reverse engineering and re-engineering. Cimitile et al. [64] propose a method for decomposing legacy systems into objects using reverse engineering activities. Sneed [249, 253] reports on modernization methods to migrate a COBOL program into a functionally equivalent OO program. Newcomb [203] describes a re-engineering tool that automatically transforms a procedural program into a functionally comparable OO system. Zou & Kontogiannis [304] develop a re-engineering workbench that not only migrates a procedural language to OO, but also allows modeling quality requirements for the target migrant system.

In the last decade, the advancement of web-based technologies has fostered legacy software system modernization (e.g., [264, 164, 44, 75]). In particular, the service-oriented architecture (SOA) has been a popular target architecture for legacy software system modernization and various legacy to SOA modernization approaches have been proposed (e.g., see the systematic literature reviews by Razavian & Lago [222], and Almonaies et al. [8]). Lately, cloud computing has been seen as a new target platform for legacy software modernization. A systematic literature review (SLR) by Jamshidi et al. [129] reported that 23 different legacy to cloud modernization approaches have been reported from 2010 to 2013. The popularity of legacy to cloud computing modernization is reflected in a number of research projects such as REMICS [192][5], ARTIST [26][6], MODAClouds [10][7].

The techniques used for modernization are diverse and largely dominated by wrapping technology. Other frequently used techniques include program slicing [143, 16, 58, 184, 300], feature modeling [187, 181], model transformation techniques [57, 92, 124], wrappers [268, 289, 229, 247, 248], aspects [193], and architectural pattern languages [106, 116].

The legacy system modernization research identified in academia are largely technology–oriented.

[5]http://remics.eu/
[6]http://www.artist-project.eu/
[7]http://www.modaclouds.eu/

They provide different techniques/methods to facilitate legacy system modernization and point out various challenges faced in the course of applying those techniques/methods. It has been observed that insufficient attention is provided to the business issues of the legacy software modernization [196, 202].

1.1.3.3 Service-Oriented Architecture

Service-oriented architecture (SOA) [83] is an architectural paradigm that represents an open, extensible and composable software architecture built from reusable software components known as services. SOA focuses on the reusability of the components by separating the interface from the internal implementation. The underlying principles that promote SOA include loose coupling, abstraction of underlying logic, agility, flexibility, reusability, autonomy, statelessness, and discoverability [83, 206]. From a software modernization perspective, SOA promises to reuse the pre-existing legacy assets by encapsulating them as added-value services [207, 221]. Channabasavaiah & Holley [52] argue that enterprises will realize the following benefits by modernizing legacy software systems to SOA:

- *Leverage existing assets*: One of the key, and the most significant, benefits of SOA is the reuse of the existing legacy assets by encapsulating the legacy features as added-value services. The reusability of the legacy assets enables an enterprise to preserve the business value of the investment made in developing the legacy software systems over the years. Additionally, the services hide the implementation and platform complexity of the legacy application and provide a uniform mechanism of access via service interfaces.
- *Faster time-to-market*: SOA facilitates the creation of a service inventory that contains both business and technical details of business services. Enterprises can use the service inventory to identify services that meet their requirements and reuse whenever possible. Additionally, due to reuse of existing legacy assets, the time needed for design, development, testing and deployment is significantly reduced, thereby shortening the time–to–market for any new products.
- *Cost Reduction*: In a SOA environment, several existing services can be composed to deliver an added-value service via service composition [208]. As new requirements emerge for new products, enterprises can leverage the service inventory to identify matching services and compose them to deliver a new service. At the same time, the reuse of existing assets significantly contributes to cost reduction as compared to the new development of products.
- *Flexibility*: The computing and the development platform can vary significantly within an enterprise leading to interoperability issues [281]. SOA can significantly reduce interoperablity issues by hiding the platform complexity via the definition of standard service interfaces. By encapsulating the computing and platform complexity, SOA opens up a wide possibility of integrating silo–applications within enterprises.

In this dissertation, SOA is considered as the target architecture of the legacy modernization. In particular, Part I of the dissertation is focused on modernizing legacy software systems towards a SOA environment.

1.2 Scientific Relevance

Legacy software systems, their characteristics, and possible solutions for modernizing these systems are well–known and long researched domains within the software engineering scientific community. The re-

search work reported in the late 70's and early 80's on software evolution and software maintenance are pioneering works that are still valid for legacy software systems and their modernization. In particular, Lehman's laws of software evolution [168] and the contribution made by Swanson in software maintenance [267] are still relevant to understand the foundation of software evolution. Based on the concept of software evolution and software maintenance, a lot of research has been reported within the domain of legacy software system and their modernization. One of the key empirical contributions within legacy software modernization goes to Sneed for his work from 1984 on software renewal [250]. In the mid 90's the term legacy systems and their characteristics are widely discussed, highlighting in particular the high cost to maintain those systems. Rough estimates of the cost allocated for software maintenance could go as high as 70-80% of the total life cycle cost of a system [180]. The importance of software evolution and maintenance was clearly visible during the "Year 2000 (Y2K)" problem [84]. Smith et al. [245] reported that one of the reasons of the severity of the Y2K problem is the existence of legacy software systems and their huge code base.

Since then legacy software systems and their modernization gained steadily in importance and moved into the center of attention of software engineers [23, 218]. Over the years, the software engineering community has been closely following the developments within software evolution. As early as 1996, client–server platforms were used as targets to modernize legacy software systems (e.g., Sneed [252] and Canfora et al. [44]). Later, with the advancement in Internet technology, the trend of legacy software modernization gained momentum to modernize to web applications (e.g., Stroulia et al. [264], Lavery et al. [164], and De Lucia et al. [75]). With the promised benefits of SOA, legacy software modernization became largely focused on modernizing towards SOA (e.g., the SLR by Razavian & Lago [222], Almonaies et al. [8]). Lately, with the potential offered by the clouds, now legacy software modernization has turned to modernizing towards clouds (e.g., legacy to clouds SLR by Jamshidi et al. [129]).

Changes to software systems during their life-cycle is a continuous process. Such changes are triggered by evolving requirements, technologies, and market demands. Over the years, the reverse engineering has played a key role in legacy software system modernization. Muller et al. [195] argue that reverse engineering techniques have been key in assisting legacy software system modernization. Chikofsky & Cross [60] define reverse engineering as *"the process of analyzing a software system to identify the systems components and their inter-relationships and create representations of the system in another form or at a higher level of abstraction"*. Several reverse engineering techniques are widely used to assist modernizing legacy software systems. For instance, techniques for system understanding such as feature location, program visualization, source code analysis, program slicing, concept analysis, software architecture recovery are being extensively used to understand and assist modernizing legacy software systems.

Legacy software and its modernization are regularly reported upon at the top tier software conferences such as the International Conference on Software Engineering[8] (ICSE) and Foundations of Software Engineering[9] (FSE) under the headings of software evolution and software maintenance, thereby indicating an active research field. In addition to that there are two dedicated and leading conferences– the International Conference on Software Maintenance and Evolution (ICSME– formerly known as the International Conference on Software Maintenance[10] (ICSM)) and the International Conference on Software Analysis, Evolution, and Re-engineering (SANER– merger of the European Conference on Software Maintenance

[8] `http://www.informatik.uni-trier.de/~Ley/db/conf/icse/index.html`
[9] `http://www.informatik.uni-trier.de/~Ley/db/conf/sigsoft/index.html`
[10] `http://www.informatik.uni-trier.de/~ley/db/conf/icsm/index.html`

and Re-engineering[11] (CMSR) and the Working Conference on Reverse Engineering[12] (WCRE)– where researchers publish their research on legacy software system and their modernization. Workshop venues such as the symposium on the Maintenance and Evolution of Service-Oriented Systems and Cloud-based Environments[13] (MESOCA), the International Workshop on Principles on Software Evolution[14] (IWPSE) regularly publish new research ideas and experience reports from industry related to the legacy software systems domain.

Legacy software systems and their modernization has regularly been a topic of significant interest in industry as well. Market research firms such as Gartner, Forrester Research have regularly indicated that legacy software modernization is one of the top priorities within industry. Additionally, a majority of financial institutions is still using their legacy software systems in the back office to run their day–to–day operations.

The research reported in this dissertation is relevant to both the scientific community and industry. To the scientific community, this research adds to the body of knowledge of software evolution and software maintenance by delivering a structured legacy software system modernization method. This research also provides an insight into how industry values their legacy systems and what challenges industry faces while modernizing. These challenges can be viewed as future research direction for the scientific community. Finally, we believe that this research will facilitate technology and knowledge transfer between academia and industry in the domain of legacy software system modernization.

1.3 Research Approach

In this section, we discuss the scientific research approach and the research methods used in this dissertation. We use the concept of practical problem and knowledge problem in the context of design science [291] (cf. Fig 1.2) to justify the research objectives.

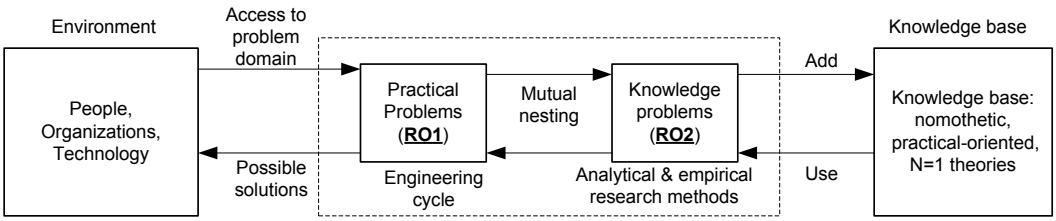

Figure 1.2: Design science framework (Wieringa [291])

Wieringa [291] defines a practical problem as a difference between the way the world is experienced by stakeholder and the way they would like it to be. Practical problems require solution(s) that bring changes within the world of the stakeholder to meet the goals. In contrary, a knowledge problem is a difference between the current knowledge of stakeholder about the world and what they would like to know. Unlike practical problems, knowledge problems do not call for a change to the world but for a change in the knowledge about the world. RO1 is a practical problem whereas RO2 is a knowledge problem. It is important to note that a practical problem, when decomposed into subproblems, can include knowledge questions– to facilitate the understanding of the current state of the key problem or

[11] http://www.informatik.uni-trier.de/~Ley/db/conf/csmr/index.html
[12] http://www.informatik.uni-trier.de/~Ley/db/conf/wcre/index.html
[13] http://www.informatik.uni-trier.de/~ley/db/conf/mesoca/index.html
[14] http://www.informatik.uni-trier.de/~ley/db/conf/iwpse/index.html

to identify if the artifacts meet the objectives. An example of a decomposition of a practical problem is depicted in Figure 1.3.

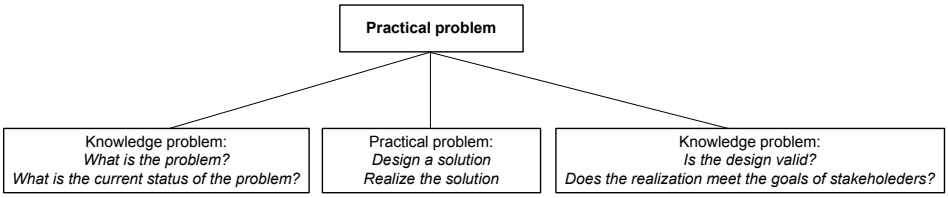

Figure 1.3: Decomposition of practical problem (adapted from Wieringa [291])

Research Objective 1 (RO1)–"Develop a software modernization method that includes technical, and business aspects."

In RO1, we explore existing academic legacy software system modernization methods in the context of modernizing towards service-oriented architecture (SOA) with an aim to develop a legacy to SOA modernization method that addresses technical, and business issues of modernization. Within RO1, the method is a solution that makes a difference in the world of the concerned stakeholder.

A logical structure to solve a practical problem is an engineering cycle– an approach that starts with an investigation to understand the problem, then specifies and implements the solution design with its validation [291, 292]. To investigate RO1, we design a legacy to SOA modernization method and validate the method.

We have formulated the following research question to investigate the RO1.

RQ 1 How can a modernization process be designed that facilitates enterprises in modernizing software systems?

This research question represents a practical problem within the legacy software modernization domain. Like Wieringa [291], we use an engineering cycle to investigate the problem, and then develop and validate a structured method. The method is focused on legacy to SOA modernization in which we consolidate the technical and business issues associated with legacy to SOA modernization. This research question is further sub–divided into three sub–questions:

RQ 1.1 What are the (essential) steps to combine business aspects and technical aspects in software modernization?

Several legacy software system modernization approaches have been reported in literature to modernize legacy systems to a new technology. A significant number of such approaches focuses on the development of the supporting technology to address the technical modernization perspective (i.e., implementation techniques to reuse legacy software system). Other approaches focus on developing a modernization strategy to determine the migration feasibility. Furthermore, not enough attention is given to business aspects of the modernization

in spite of the fact that researchers [175, 174, 202] argue a need of consolidated method that combines all the aforementioned aspects.

To provide a solution to this practical problem and address the research question, we identify steps necessary to combine business and technical aspects of the software modernization. To consolidate the method, we use method engineering [39]– an information system development method to construct advanced development methods by reusing parts of existing methods. The consolidated legacy to SOA method is validated with modernization experts and is further validated with two case studies.

RQ 1.2 What is the state of the art of software modernization in academia?

After we have shown the feasibility of combining different aspects in one legacy modernization method, we systematically investigate what techniques and methods are reported in academia regarding legacy to SOA software modernization. We use a systematic literature review (SLR) approach to answer this knowledge problem. The SLR is used to create an inventory of methods and techniques used in various phases of legacy to SOA software modernization. To this end, 121 research papers were identified and evaluated so as to create an inventory of current research approaches, methods, tools and techniques To minimize the researcher bias, we have carefully followed the guidelines outlined by Kitchenham et al. [154] to perform an SLR. In every step of the SLR process, we have identified potential threats to validity and have taken appropriate measures to mitigate those validity threats. For instance, to minimize the researcher bias, 121 papers were distributed over 5 researchers to identify the methods and later the results were cross-validated by a researcher other than the one who categorized the methods initially. Additionally, inclusion and exclusion criteria for the papers were clearly documented to minimize missing relevant studies. used in legacy software system modernization.

RQ 1.3 How can a structured legacy to SOA software modernization process be developed from existing methods and techniques?

This is a practical problem with the aim of developing a legacy to SOA modernization process from existing legacy modernization methods. This research question extends the steps of the RQ 1.1 and aims at developing a phase-wise structured method for legacy to SOA modernization. For each phase, we present a rationale to justify its need, current practices, and challenges that require further attention. This research is based on the rationale that there is a need for a structured legacy to SOA modernization method that incorporates not only the technical issues but also the business issues [175, 174, 202].

The proposed structured process is then evaluated by migrating features of two simple yet representative applications to SOA. To further validate the structured process, we selected 17 academic papers reporting legacy to SOA modernization from 2000 to 2011 and mapped the activities described therein to the phases of the structured process.

Research Objective 2 (RO2)–"Identify how software modernization is perceived and conducted in practice."

In RO2, we investigate how legacy software systems and legacy software modernization are perceived in industry. This knowledge problem aims at providing answers for a change or update of the current knowledge about legacy systems and (legacy) software modernization. We consider this knowledge to be of significant value, due to the fact that there is a knowledge gap between academia and industry in understanding legacy systems and their modernization, thereby inhibiting the adoption of knowledge transfer. For example, Razavian & Lago [223] indicate that 97% of the academic legacy to SOA modernization approaches do not fit the industrial purposes in the context of legacy to SOA modernization [223]. We believe that the knowledge gained from RO2 can be used to leverage the adoption of academic modernization methods in industry. To investigate this research objective, we use different empirical research methods: (i) case studies to investigate how legacy system modernization is performed on an industrial scale, (ii) a grounded theory method to identify how practitioners view legacy systems and what challenges they face during modernization, and (iii) mixed methods such as combining case studies and interviews to understand the impact of legacy modernization.

We have formulated the following research question to investigate RO2.

RQ 2 What are the perceptions of practitioners about software modernization?

In RQ 2, we explore the industrial perception of legacy systems and legacy software modernization. In particular, we investigate what characteristics of legacy software systems still keep them operational, what are the key drivers for modernization, what key challenges are faced in the modernization process and what business objectives are met after conducting legacy software modernization. To gain knowledge on such questions, we further divide the research question into three sub-questions:

RQ 2.1 How is large scale software modernization performed in practice?

To establish a context for legacy system modernization in industry, this research question investigates how legacy software systems are modernized in practice. It further explores what techniques are used for modernization, and what challenges are faced during modernization. It is evident that there are relatively few case studies of industrial legacy software modernization reported in academia thereby limiting the knowledge on how modernization is conducted on an industrial scale. Hence, this research question presents a large scale legacy software to SOA modernization method in the financial domain and details the modernization process.

A single case study research method [297] is used to conduct this research. One of the limitations of a single case study is the possible bias in data collection and interpretation. This potential bias in this research is minimized by including multiple data collection methods (documentation, interviews, workshops) and involving two researchers within the project to regularly cross-validate the findings.

RQ 2.2 What are the discrepancies between the perception of legacy software and their modernization in academia and industry?

The preliminary findings of the research question RQ 2.1 and the extensive use of legacy software systems in the financial domain lead us to investigate this research question in which we aim at identifying how legacy systems and their modernization are perceived in industry. Particularly, what characteristics of legacy software systems keep them operational in industry, what drivers lead to modernization and what are the challenges faced during modernization? We use the grounded theory research method–an explorative research method that aims at discovering new perspectives and insights, rather than confirming existing ones to investigate this research question [103], and analyze the interviews of 26 practitioners.

In order to validate the findings of the grounded theory research, we use survey as a data triangulation method– a validation process that uses more than one data source to increase (decrease) confidence in a finding by providing confirming (contradictory) evidence. The findings of the interviews are in–line with the survey.

RQ 2.3 How often are pre-modernization business goals achieved after a "technically" successful software modernization?

In this research question, we investigate what it means for a legacy system modernization to be "successful" from a business perspective. As of now, legacy software modernization is claimed to be successful when the technical modernization is completed. However, there has been limited research on investigating the post–modernization results. With this research question, we aim at identifying what business goals are met by enterprises upon modernizing their legacy software systems. We investigate five case companies that have completed software modernization and explore the business goals behind modernization. We further identify which of those business goals are met by software modernization.

We use multiple case studies [297] to investigate this research question and utilize multiple data sources to minimize the research bias.

1.3.1 Research Methods

In this thesis, we have used a number of research methods that are dominantly used in software engineering and information system research. We discuss the research methods in the following subsections:

1.3.1.1 Design Science

Design science research [121] aims at creating and evaluating IT artifacts intended to solve identified organizational problems. Such artifacts are then developed and validated in coordination with a knowledge base. Figure 1.4 depicts the design science research based on Hevner et al. [121]. In Part I of this dissertation we develop a legacy to SOA modernization method and subsequently validate the method. This context of developing and evaluating the legacy to SOA modernization method is suitably covered by the design science research method. Design science allows to combine the knowledge from existing theories and methods (e.g., existing modernization methods, reported academic literatures) with the new

data (e.g., lesson learned from case studies) to derive new results and theories. Such results are then evaluated using other data sources such as interviews, reference cases or surveys [272].

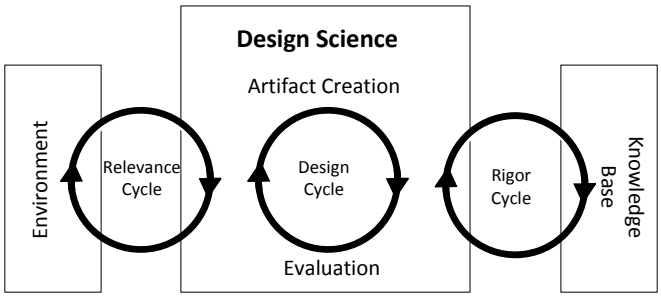

Figure 1.4: Design Science Research Method [121]

IS Research

Information science (IS) research is typically a combination of behavioral science and design science [121]. Behavioral science seeks to develop and justify theories around information systems while design science aims at creating innovations around information systems to enhance effectiveness and efficiency [78, 121]. This research has combined both (design science and behavioral science) and has resulted in the development and evaluation of several design artifacts, such as a legacy software system modernization method to enhance the modernization process. The evaluation phase aims to justify the suitability of the method by validating it with experts or case studies.

Environment

The environment defines the problem space in which the phenomena of interest reside, and in which the research is conducted. The space is composed of practitioners, business organizations and their existing and planned technologies and practices. Also, the environment incorporates goals, tasks, problems and opportunities that define business needs as they are perceived by people within the organization [121]. With respect to the environment, this research is particularly focused on, but is not limited to, legacy software system modernization researchers, industrial practitioners, and various existing modernization processes, methods, tools and techniques to support modernization.

Knowledge Base

The knowledge base provides the scientific foundation (e.g., theory and literature) and methodologies (e.g., the systems of methods used in a particular area of information systems research) from and through which information science research is performed [121]. This research is related to legacy software systems and the modernization process, which finally contributes to the existing knowledge base of the software evolution and maintenance community. The results and findings of this research are significant to the practitioners. Additionally, the contributions of behavioral science and design science in information systems research are applied to a business need in a particular environment, and add to the content of the knowledge base for further research and practice [121].

The evaluation of artifacts developed within IS research is a continuing process which is referred to as the design cycle. In this continuous process, there is a constant feedback loop with the Environment and the

Knowledge base. These feedback loops are known as Relevance cycle and Rigor cycle respectively [120, 291].

The design science research method is used in this dissertation to develop several design artifacts such as modernization methods (Chapter 2 and Chapter 4), theory building (Chapter 6) and to evaluate these.

1.3.1.2 Case Studies

Substantial research within this dissertation relies upon the observation of real–world industrial legacy software system modernization. Hence, it is important to choose an appropriate research method in software engineering that allows us to study contemporary phenomena in its natural context [233]. Case study research method is a suitable research method to observe these real–world legacy system modernization processes.

Case study research method is used in many situations to contribute to our knowledge of individual or organizational phenomena [297]. Case studies strive to portray what it is like to be in a particular situation, by looking at a case or phenomenon in its real-life context, usually employing many types of data. Case study research involves the close examination of people, topics, issues, or programs, for purposes of understanding, and theory building and testing [297].

Figure 1.5 depicts different types of case study design, as illustrated by Yin [297]. Figure 1.5 (A) represents a holistic single case design that involves an intensive description and analysis of a single case. Chapter 5 is based on a holistic single case design in which we explore a large scale legacy to SOA modernization process in a large financial institution. Figure 1.5 (B) illustrates a multiple case design in which multiple sources/cases are analyzed. In Chapter 7 we use multiple case studies to study the post–modernization effects within enterprises. Figure 1.5 (C) and (D) are embedded case design in which different units/cases are analyzed in the same context.

Within the field of information systems, successful completion of a case study research requires initiative, pragmatism, the ability to take advantage of unexpected opportunities, and optimism and persistence in the face of difficulties and unexpected events, particularly during data collection activities. Additionally, research based on case studies is often reported for being too specific and directed towards hypothesis generation, thereby leading to validity threats and research bias [90]. In this research, we mitigate these challenges by identifying representative cases and by designing and following a case study protocol. Furthermore, in our case studies, we have rigorously followed guidelines to create case study protocols as prescribed by Jansen & Brinkkemper [131] and Yin [297].

1.3.1.3 Systematic Literature Review

Over the last four decades, a plethora of research methods have been reported in the context of legacy software system modernization, SOA being the target architecture. However, there is no systematic overview of this research, and in particular the techniques, methods and approaches used to evolve legacy systems to a SOA environment. In the systematic review conducted by Razavian & Lago [222], a classification of SOA migration into eight families is discussed. However, this review does not provide an inventory of methods and techniques used for legacy to SOA modernization. Hence, we have adopted a Systematic Literature Review (SLR) method to systematically gather and analyze existing literature and

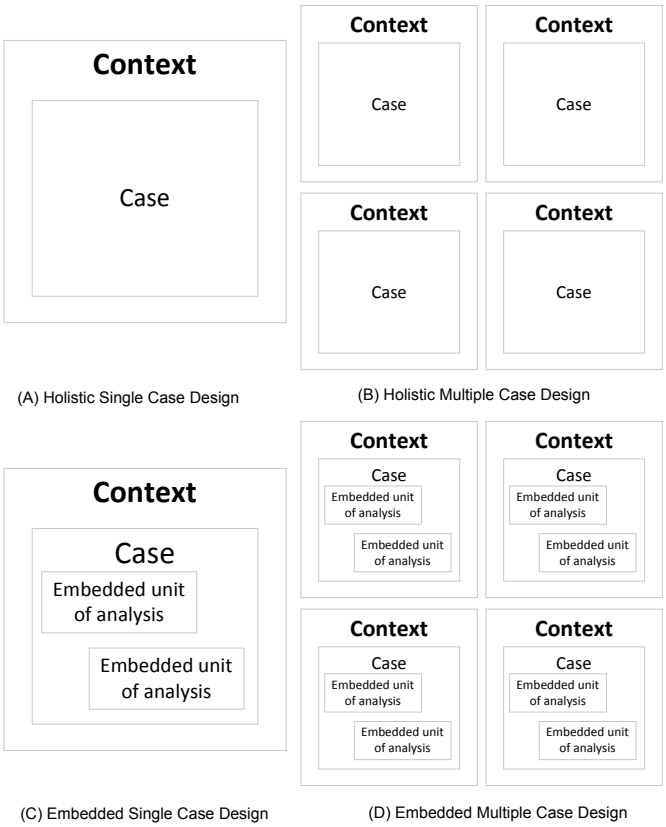

Figure 1.5: Types of case study designs adapted from Yin [297]

provide an extensive inventory of methods and techniques used. An SLR is an evidence-based approach that aims at providing answers to some research questions by documenting an exhaustive summary of current literature, analyzing and synthesizing findings [154]. SLR is a useful and powerful research method in collecting and analyzing existing work, which is a common task in establishing background knowledge for any research.

However, SLR is not the only research method to systematically gather and analyze existing literature. A systematic mapping study (SMS) [154] is also a potential approach. However, Kitchenham [153] argues that SMS is suitable when few papers exist on a topic, or the topic is too broad or scattered. SMS in particular is more directed towards uncovering research trends, rather than providing answers to specific research questions [215]. In our case, SLR is deemed more appropriate due to the fact that there is abundant research published and the aim is to provide a complete overview of a research domain, based on all papers published on legacy to SOA modernization.

A typical SLR consists of three phases as shown in Figure 1.6 with the details of activities that are performed within each phase. Initially, the Plan Review Phase provides the context of the review by identifying the research questions and developing a review protocol. The Conduct Review Phase represents the operationalization of the review process by selecting data sources, defining search queries, selecting primary studies (often based on scanning topic and abstract) and analysis of the data. Finally, the results of the review are documented in a report.

The SLR method is used in Chapter 3. The SLR constitutes an inventory of current research

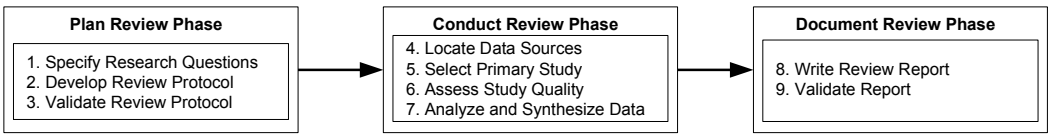

Figure 1.6: An SLR Process [154]

approaches, methods and techniques used in legacy to SOA evolution. The result of the SLR also identifies current research issues in legacy to SOA modernization and provides future research directions to address those research issues.

In this thesis, we have also used literature study as a research method. The literature study method closely resembles SLR as both research methods are used to gain certain knowledge or understanding of a certain research topic. However, a literature study might not be as complete and valid as that of a SLR because a literature study is much less formal in the sense that it allows more freedom in collecting relevant studies and analyzing their content. Nonetheless, literature study is an effective and efficient method to get an overview of a research topic. In Chapter 4, we have used a literature study to validate a structured process and in Chapter 6, the literature study method is used to identify the current academic perspective of legacy systems and legacy software system modernization.

1.3.1.4 Snowballing Based Literature Search

Snowballing based literature search is a method for identifying additional relevant articles through the reference lists of a set of identified articles [293]. Webster and Watson [287] use snowballing to find relevant literature and propose two types: backward snowballing– using the reference list of a paper to identify new papers to include, and forward snowballing– identifying new papers based on those papers citing the paper being examined. In this dissertation, we use a backward snowballing approach in Chapter 7 to identify benefits claimed due to software modernization. Backward snowballing approach is used due to the fact that the time and effort required to conduct a literature search is relatively less. Jalali & Wohlin [128] compare SLR with backward snowballing approach and argue that backward snowballing approach is easy to use and requires less time and effort.

1.3.1.5 Grounded Theory Method

Grounded Theory (GT) is a systematic, inductive and comparative approach to develop a theory iteratively from data. In contrast to the hypothetico-deductive method, where the researcher has a predefined hypothesis at the beginning of the investigation, GT is explorative, aimed at discovering new perspectives and insights. Adolph et al. [3] argue that GT is an excellent method for studying software engineering and generating theories that are relevant to the practitioner, and is increasingly popular in software engineering research [3]. For instance, Coleman et al. [65, 66] use GT to understand software process improvement in Irish software product companies; Hoda et al. [123, 122] adopt GT to study the human aspects of software engineering; Dedrick et al. [76] use GT to study adoption of open source platforms within industry; Angela et al. [186] use GT to understand customer-focused practices in eXtreme Programming (XP); Balasubramaniam et al. [220] use GT to identify factors that influence Internet software development processes; Hutchinson et al. [125] use GT method to document technical, organizational and social factors that influence organizational responses to Model–Driven Engineering (MDE) in industry;

and Greiler et al. [111] use GT to identify how developers and testers perceive testing plug–in based systems.

In this dissertation, one of the objectives is to discover new perspectives and insights about legacy software systems and (legacy) software modernization from the practitioners perspective. We do not have any explicit hypothesis about the practitioners perception of legacy software systems and their modernization, rather intend to generate theory. Additionally, there is little empirical evidence documented in academia regarding the practitioners perception of legacy software systems. Adolph et al. [3] emphasize that GT is useful for research in areas that have not been previously studied or where a new perspective might be beneficial. Hence, we exploit GT in Chapter 6 to identify the industrial perspective of legacy software system and legacy software modernization. Using the illustrative diagram shown in Figure 1.7

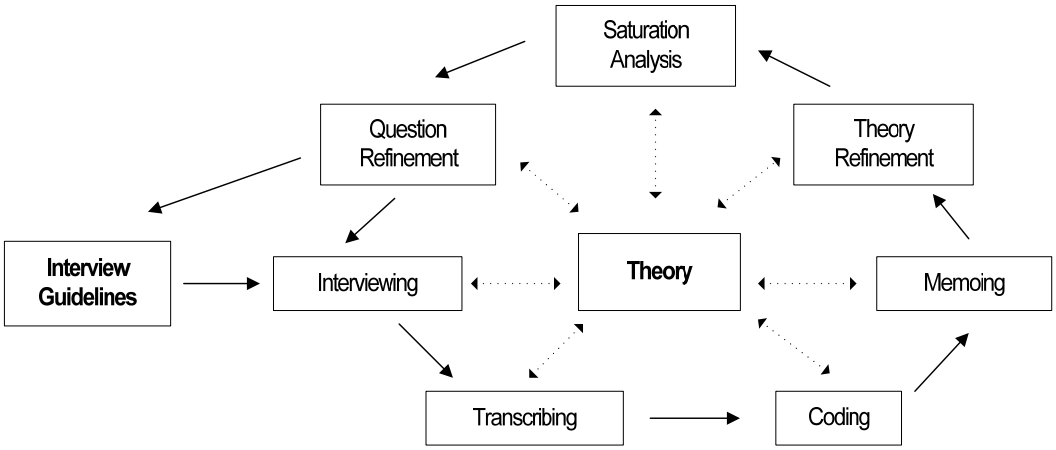

Figure 1.7: Grounded Theory Method [111]

from Greiler et al. [111], we briefly summarize the GT process. Following the interview guidelines, we start collecting data by interviewing practitioners that have experience in legacy software systems and transcribe the interviews. The transcription is then analyzed to find patterns and accordingly codes are assigned to indicate concepts, which in turn are aggregated into categories. The resulting concepts and categories are connected via memos, leading to theory refinement. The process continues until the categories are "saturated"– no new codes/categories are added even with new interviews. The generated theory/finding is then compared with literature and/or validated, if required, covering new perspectives and insights, rather than confirming existing ones.

1.3.1.6 Survey Method

A survey is a comprehensive system for collecting information to describe, compare or explain knowledge, attitudes and behavior. In this research, survey is used for a data triangulation [233] to validate the findings of the interview results of the GT method in Chapter 6. A data triangulation process– a method that uses more than one data source, or collects the same data at different occasions– is typically used to provide confirming (contradictory) evidence and further helps to improve validity of the findings of an empirical study. Data triangulation process is being increasingly used in software engineering research. Some recent examples such as Greiler et al. [111] and Beller et al. [20] use survey as a secondary data source to improve validatory of the findings from another primary source.

1.3.2 Validation

A crucial aspect of conducting any research is to rigorously operationalize the entire research process and validate it, whenever possible [109]. Validation of research and data have become increasingly an area of concern within the scientific community. An article published in Nature in October 2011 indicates that misconduct (i.e., falsification or fabrication, (self-)plagiarism) accounts for 44%. Honest error (i.e., honest differences in the design, execution, interpretation or judgement in evaluating research methods or results) accounts for 28% of the total number of retractions within the scientific community [279].

The research methods used in this dissertation are predominantly empirical, and therefore require a rigorous validation and evaluation measure. For example, Yin [297] and Eisenhardt [81] argue that validation, in particular research method validation and data validation, is an important aspect of the case study research method. In this dissertation, we have paid careful attention to operationalizing the selected research methods, validating the data and reducing the research bias throughout the research.

In case of operationalizing the selected research methods, we have diligently followed available research guidelines. For instance, in the case studies we have followed guidelines from Runeson & Host [233] by creating case study protocols and clearly documenting the data sources, if possible. In the SLR-based research method, the guidelines to conduct SLR by Kitchenham et al. [153] have been strictly followed. In the case of interviews (Chapter 2, 5, 6, and 7), the participants are given an interview protocol to prepare a common understanding of what is going to be discussed. Such protocols (research protocol and interview protocols) have been proven to be highly important as they provide a common ground for understanding the terms and techniques used (e.g., the concept of legacy systems was understood differently for different enterprises). Whenever possible, the developed artifacts (e.g., Chapter 2) are validated with experts and subsequently enhanced.

In case of data validation, we have focused on documenting the data sources, whenever possible. For example, the majority of the interviews conducted within the context of this research has been recorded and transcribed with the permission of the interviewee. In case of misunderstanding and to further validate, the interviewees were consulted via email to clarify the issues– a method known as cooperative inquiry [226]. Artifacts collected in the course of data gathering such as forms, SLR classification forms, disputes among the researchers while classifying primary studies, documents from the case study are documented.

To mitigate research bias in the research analysis phase, different types of validity are discussed throughout the dissertation. Goodwin & Leech [109] describe validation as a measure to ensure the well–foundedness of the measurements with the real world. As per Yin [297], the following four different kinds of validity are identified:

1. Construct validity: reflects to what extent the operational measures that are studied really represent what it was meant to be measured.
2. Internal validity: reflects the extent to which a causal conclusion based on a study is warranted.
3. External validity: concerns the generalizability of the findings and measures the extent to which it is possible to generalize the findings.
4. Reliability: concerns with the extent to which the data and the analysis are dependent on the specific researchers.

Furthermore, we have also used data triangulation to validate the findings of some of the research. For instance, we used survey method in Chapter 6 to ensure the reliability of the findings. Similarly, a focused workshop was conducted to validate the findings of Chapter 5. Some of the artifacts such as the serviciFi method of Chapter 2 and the legacy to SOA migration method of Chapter 4 are evaluated using controlled experiments.

To promote reproducibility within software engineering, various artifacts produced within this research are freely available. Such resources include the anonymized interview transcripts (Chapter 6 and 7), anonymized survey data (Chapter 6) and the complete data form used in the SLR (Chapter 3).

Table 1.1 lists the research questions, the research methods used and the measures undertaken to ensure the validity of the research.

Table 1.1: Summary of mapping of research questions with research method and validity

RQ No.	**Chapter**	**Research Method**	**Validation Measure**
RQ 1.1	2	Design Science	Controlled experiments, Expert evaluation
RQ 1.2	3	SLR	Cross validation among co–authors
RQ 1.3	4	Design Science, Case Study	Controlled experiment
RQ 2.1	5	Case Study	Expert Interviews
RQ 2.2	6	Grounded Theory	Survey
RQ 2.3	7	Case Study	Cooperative Inquiry

1.4 Outline of Thesis and Publications

The research presented in this thesis is previously published in peer–reviewed venues. The dissertation is divided into two parts based on the two research objectives. Part I of the thesis focused on developing a structured software modernization method that combines technical and business issues. The modernization method is developed in the context of modernization of legacy software systems towards a SOA system. Part I of the dissertation consists of three chapters.

Chapter 2: In this chapter, we identify necessary steps to combine technical and business aspects of a legacy to SOA modernization method. These steps are identified using method engineering approach. As a research contribution, this chapter provides a starting point towards developing a structured method by identifying the necessary steps. This chapter addresses research question RQ 1.1.

R. Khadka, G. Reijnders, A. Saeidi, S. Jansen, and J. Hage. A method engineering based legacy to SOA migration method. In *the 26th IEEE International Conference on Software Maintenance (ICSM 2011)*, pages 163–172. IEEE, 2011.

Contributions: R.K. and S.J. designed the research. R.K. and G.R. conducted the research. R.K. wrote the manuscript. R.K. and A.S. performed additional case studies. R.K. conducted additional research to enhance validation. A.S., S.J., and J.H. participated in discussion and editing of the manuscript.

Chapter 3: To document current state of art of legacy software system to SOA modernization, we perform a systematic literature review (SLR). This chapter addresses research question RQ 1.2 and presents an inventory of methods and techniques that are currently available for legacy to SOA modernization based on the research articles published within the period of 2000 to 2011.

> R. Khadka, A. Saeidi, A. Idu, J. Hage, and S. Jansen. Legacy to SOA evolution: a systematic literature review. In A. D. Ionita, M. Litoiu, and G. Lewis, editors, *Migrating Legacy Applications: Challenges in Service Oriented Architecture and Cloud Computing Environments*, pages 40–71. IGI Global, 2012.
>
> *Contributions*: R.K. designed and conducted the research. R.K. and A.I. collected the articles. R.K., A.I., A.S., S.J., and J.H. reviewed the articles and cross-validated with the other team members. R.K. wrote the manuscript. A.S., S.J., and J.H. edited the manuscript.

Chapter 4: Based on chapter 2 and chapter 3, we develop a structured legacy to SOA modernization method consisting of six phases. In chapter 2, we detail the necessary steps and demonstrate the usability of the proposed method to combine business and technical aspect of modernization. In this chapter, we present rationale to justify the need of each phase, current practices within each phase, and challenges that require further attention. In terms of research contributions, this chapter identifies relevant phases to develop a structured legacy to SOA modernization method, whereas in Chapter 2, we identify necessary steps to combine business and technical aspects of legacy to SOA modernization. This chapter corresponds to RQ 1.3.

> R. Khadka, A. Saeidi, S. Jansen, and J. Hage. A structured legacy to SOA migration process and its evaluation in practice. In the IEEE 7th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA 2013), pages 2–11. IEEE, 2013.
>
> *Contributions*: R.K. designed and conducted the research, and wrote the manuscript. A.S., S.J., and J.H. participated in discussion and edited the manuscript.

Whilst Part I of this dissertation is aimed at answering a practical problem, Part II is focused on exploring a knowledge problem. While conducting research for Part I, the author participated in discussions with experts from industry and identified that they still highly value legacy software systems. Even though experts from industry acknowledge the problems of legacy software systems, they consider legacy software systems to be performant and fit for purpose. With respect to legacy software system modernization, experts from industry indicate that legacy modernization methods reported in academia are too abstract to apply in practice. Some of the views expressed by experts from industry are to some extent new to academia, thereby resulting in a potential knowledge gap between perception of legacy software systems and their modernization within industry and academia. Part II of this dissertation investigates the knowledge gap by conducting empirical research focusing on industry. Hence, the research performed in Part II is largely empirical in nature. The empirical research is based on observed and measured phenomena from which knowledge is derived and documented about industrial perception of legacy software systems and their modernization. Because of its empirical nature, Part II of this

dissertation is not a natural continuation of Part I. The structured method legacy modernization method developed in Part I of this research is not used in the chapters of Part II rather the research questions of Part II are based on observations to gain new perspectives of industry around legacy software systems and software modernization. Part II consists of three chapters.

Chapter 5: In this chapter, we present the findings of a case study of a large scale legacy to SOA modernization process in the payments domain of a Dutch bank. In particular, we discuss the challenges faced and lesson learned during the modernization process. This chapter addresses research question RQ 2.1.

> R. Khadka, A. Saeidi, S. Jansen, J. Hage, and G. Haas. Migrating a large scale legacy application to SOA: Challenges and lessons learned. In *the 20th Working Conference on Reverse Engineering (WCRE 2013)*, pages 425–432. IEEE, 2013.
>
> *Contributions*: R.K. designed and conducted the research and wrote the manuscript. A.S., S.J., J.H., and G.H. participated in discussion and edited the manuscript.

Chapter 6: In this chapter, we describe the outcome of an exploratory study in which 26 industrial practitioners were interviewed on what makes a software system a legacy system, what the main drivers are that led to the modernization of such systems, and what challenges are faced during the modernization process. The findings of the exploratory study is further validated with a survey. This chapter addresses research question RQ 2.2.

> R. Khadka, B. V. Batlajery, A. Saeidi, S. Jansen, and J. Hage. How do professionals perceive legacy systems and software modernization? In *the 36th International Conference on Software Engineering (ICSE 2014)*, pages 36–47. ACM, 2014.
>
> *Contributions*: R.K. designed the research and wrote the manuscript. R.K. and B.V.B. conducted the research. B.V.B. collected the data and R.K. interpreted the data. A.S., S.J., and J.H. participated in discussion and edited the manuscript.

Chapter 7: In this chapter, we report on five software modernization case studies to document the pre-modernization business goals, and to decide whether those goals have been achieved after modernization. Software modernization is claimed to be successful when the modernization is completed using those technical solutions. We use an explanatory case study approach to document the pre-modernization business goals, and to decide whether those goals have been achieved. This chapter addresses research question RQ 2.3.

> R. Khadka, P. Shrestha, B. Klein, A. Saeidi, S. Jansen, J. Hage, E. van Dis, and M. Bruntink. Does software modernization deliver what it aimed for? A post modernization analysis of five software modernization case studies. In *the 31st International Conference on Software Maintenance and Evolution (ICSME 2015)*, pages 477–486. IEEE, 2015.
>
> *Contributions*: R.K. designed the research and wrote the manuscript. R.K., P.S., and B.K.

conducted the research. P.S. and B.K. collected the data and R.K. interpreted the data. A.S., S.J., J.H., E. van D., and M.B. participated in discussion and edited the manuscript.

Chapter 8: In this chapter, we provide an overview of the answers to all research questions, discuss the contributions and implications of this thesis. Finally, we identify potential future research directions.

Part I

Developing a Legacy System Modernization Process

Chapter 2

A Method Engineering based Legacy to SOA Migration Process

Abstract

Legacy systems are vitally important for the continuation of business in an enterprise as they support complex core business processes. However, legacy systems have several well-known disadvantages such as being inflexible and hard to maintain, so momentum is growing to evolve those systems into new technology environments. Recently, service-oriented architecture has emerged as a promising architectural style that enables existing legacy systems to expose their functionality as services, without making significant changes to the legacy systems themselves. A significant number of the legacy to service migration approaches addresses the technical perspective (i.e., supporting technology) to expose the legacy code as services. The other approaches focus on determining the feasibility of the migration that includes economical and technical feasibility, based on the characteristics of the existing legacy system and the requirements of the target SOA system. In this chapter, a legacy to SOA migration method that does not single out the migration feasibility and technical perspectives, but combines these two perspectives of migration, is proposed. Method engineering is used to develop the migration method by reusing method fragments from existing service-oriented development methods. Then, concept slicing is used to develop the service by extracting the relevant parts of the legacy code. The method is evaluated and enhanced by interviewing experts and further validated with two case studies. The method is found to be appropriate and effective in extracting services from legacy code with the aim of reusing these services in new configurations.

2.1 Introduction

A large number of enterprises depend on business-critical systems for consolidating business information that have been developed over the last three decades or more using 3GL programming languages such as COBOL, RPG, C, C++ [22]. These systems are called *legacy systems*. It is estimated that more than 80% of the world's business runs on COBOL, and 50-70% of the total IT costs are devoted in the maintenance of these systems [162]. Legacy systems are now a roadblock for the evolution of the IT infrastructures in an enterprise due to their well-known disadvantages such as being inflexible and hard to maintain [32].

However, enterprises still rely on these legacy systems as they usually implement complex core business processes, and the high risk associated with necessary changes [137]. Since legacy systems are vitally important for the continuation of business in the enterprises, momentum is growing to evolve those systems into new technology environments [32]. Recently, Service-Oriented Architecture (SOA) has emerged as a promising architectural style that enables existing legacy systems to expose their functionality as services, without making significant changes to the legacy systems themselves [170]. The migration from legacy systems to SOA can be beneficial from an economical and technical perspective. From the economical perspective, enterprises are constantly challenged by an accelerating pace of changes, such as intra-organizational changes, changes in market demands and opportunities, and, consequently, changes in enterprise collaboration. The migration of legacy to SOA enables legacy systems to adapt to such changes [151] and aims at reducing the maintenance costs [207]. From the technical perspective, seamless enterprise collaboration through service composition and heterogeneous application integration within/outside the enterprises [148] are claimed.

Several approaches have been reported in literature to migrate legacy systems to SOA and web service technology. A significant number of such approaches focus on the development of the supporting technology to address the technical migration perspective (i.e., implementation techniques to expose the legacy code as service) [162, 300, 257, 176, 182, 63, 8, 165, 72]. Other approaches focus on developing a migration strategy to determine the migration feasibility. Such feasibility is determined based on the characteristics of the existing legacy systems for their potential to be exposed as services and the requirements of the target SOA system [170, 53, 273, 227]. However, a legacy migration method requires the consolidation of both the aforementioned perspectives (i.e., migration feasibility and supporting technology) [75], which, as per our knowledge, is still missing.

In this chapter, a legacy to SOA migration method, hereafter called *ServiciFi method*, is developed that combines the migration feasibility and development of supporting technology of the legacy to SOA migration. The serviciFi method is developed by assembling the fragments of existing service-oriented development methods using method engineering [39]. For the development of the supporting technology, concept slicing [108] is used to facilitate the extraction of the services from the legacy code.

The rest of the chapter proceeds as following: Section 2.2 introduces the research design that has been followed to develop the serviciFi method. Section 2.3 explains the serviciFi method followed by the evaluation using experts review and two case studies in Section 2.4. Finally, Section 2.5 concludes the chapter with an outlook to future research directions.

2.2 Background

The serviciFi method is designed following the design science in information systems research, suggested by Henver et al. [121]. The serviciFi method is first designed and evaluated with experts review followed by two case studies. Method engineering is used to design the serviciFi method by assembling the activities of existing service-oriented development methods. In the following subsections, the method engineering approach and it's related concepts used while designing the servicFi method are detailed. Further, concept slicing is also explained as a supporting technology for the migration.

Method engineering for information system development is an approach to construct advanced development methods by reusing parts of existing methods [39]. In the work of Brinkkemper [39], a *"method"* is defined as an approach to perform a system development method consisting of directions and rules, structured in a systematic way in the development activities with corresponding products. The development activities and corresponding products are called *"method fragments"*. The method engineering approach is used to develop the serviciFi method due to the fact that reusing the existing and proven method fragments from existing service-oriented development methods saves time and reduces the adoption problem (i.e., easy to adapt to the existing standards/methods). A specific strategy of method engineering is assembly-based situational method engineering [40] [274], which includes a step to create a method base in case, if it does not exist. A method base is a repository where the method fragments can be stored and retrieved [39]. To create the serviciFi method, there is no existing method base from which the method fragments can be reused. So, the assembly-based situational method engineering is used to create the method base for the serviciFi method. The assembly-based situational method engineering has the following steps that are followed to create the serviciFi method.

2.2.1 Analyze Situation and Identify Needs

The need for the serviciFi method has been described in Section 2.1. The serviciFi method should combine the migration feasibility and development of supporting technology to extract legacy code and expose them as services. Also, the extracted services should adhere to the current standards of SOA to facilitate the heterogenous application integration across/within the enterprises.

2.2.2 Select Candidate Methods

In this step, the service-oriented development methods containing relevant method fragments are selected. Based on higher number of citations (popularity), availability of documentation and completeness of the method, the following service-oriented development methods are selected: Service-Oriented Design and Development Methodology (SODDM) [208], Web Service Implementation Methodology (WSIM) [166], and Service-Oriented Modeling and Architecture (SOMA) [12]. Hereafter, these methods are called *"candidate methods"*.

2.2.3 Analyze Candidate Methods and Store Relevant Method Fragments in Method Base

In this step, the method base is filled with the relevant method fragments derived from the three candidate methods. For each candidate method, a Process Deliverable Diagram (PDD) is created, by applying the metamodeling technique as described by Weerd et al. [274]. The PDD of the candidate methods

depicts every activity and deliverable of each phase. The details of the metamodelling technique to develope a PDD is describe in Reijnders et al. [228]. To give an impression of how a PDD is represented, the PDD of the serviciFi method (cf Figure 2.4) is presented as an example. A PDD consists of two integrated diagrams: the process view on the left-hand side of the diagram is based on a UML activity diagram, and the deliverable view on the right-hand side of the diagram is based on a UML class diagram. The process/deliverable view diagram constitutes different types of activities/concepts. These activities/concepts are depicted in Figure 2.1 and explained as follows:

- *Standard activity/concept:* A standard activity/concept contains no further activities/concepts.
- *Open activity/concept:* An open activity/concept contains further activities/concepts.
- *Closed activity/concept:* An activity/concept whose activites/concepts are not elaborated since it is not known or not relevant in the specific context.

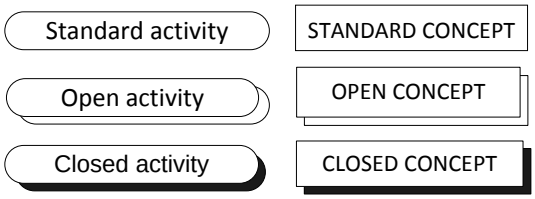

Figure 2.1: Activity and Concept types [274]

The PDD of the candidate methods depict every activity and deliverable of each phase, which are stored in method base as method fragments.

2.2.4 Select Useful Method Fragments and Assemble a New Method

In this step, useful method fragments from the method base are selected and assembled to form the serviciFi method. This step is divided into three sub-activities.

First, the general phases of the serviciFi method need to be identified, which is done by comparing and analyzing the phases of the three candidate methods. The phase comparison is depicted in Table 2.1. As a result of this comparison, five phases of the serviciFi method are identified, resembling the phases

Table 2.1: Phase Comparison

#	**WSIM**	**SODDM**	**SOMA**
1	Requirements	Planning	Business Modeling & Transformation
			Solution Management
2	Analysis	Analysis & Design	Identification
3	Design		Specifications
4	Coding	Construction & Testing	Realization
	Testing		Implementation
5	Deployment	Provisioning	Deployment, Monitoring & Management
		Deployment	
		Execution & Monitoring	

of the three candidate methods. To make these phases reflect their intent, they are renamed as follows:

- Project initiation
- Candidate service identification
- Service specification
- Service construction and testing
- Deployment, monitoring and management

Second, a method comparison among the method fragments, stored in the method base, is done using the method comparison matrix [275] to create a so-called "*super-method*". The super-method contains the activities that are considered reusable for the serviciFi method. Creating the super-method involves the step-by-step comparison of all the activities and deliverables among the three candidate methods of the method base. Each activity, in the same phase, was evaluated based on their description to find if the activity was:

- Out of scope such that the activity is discarded.
- Equal to another activity such that only one of the activities is included in the super-method.
- Fully contained within another activity such that scoping decision is made.
- Not relevant in the current phase such that the activity is discarded.

Table 2.2: Excerpt of the project initiation phase of the method comparison matrix

WSIM	**SODDM**	**SOMA**
Determining the need of web service	**Analyzing the business needs**	-
-	**Review current technological landscape**	-
Elicit web service requirement	**Conceptualize the new requirements**	-
Manage Web service requirements		
Model usage scenario		
-	Manage project deliverables and resources	**Initiate project management**
Prepare test cases for user acceptance test and system test	-	-
-	-	*Define business architecture and models*
-	-	*Select solution templates and patterns*
-	-	*Conduct method adoption workshop*

Table 2.2 depicts an excerpt of the method comparison matrix of the project initiation phase. The activities within **bold** are the assembled method fragments to form the super-method. The activities within *italics* are out of scope. The activities within * * are not relevant in the current phase. The activities that are similar or if combined are similar to higher-grained activities, are presented in same row of the method comparison matrix. The corresponding excerpt of the super-method representing the excerpt of the method comparison matrix (Table 2.2) is depicted in Figure 2.2. Among *"Analyzing the business need"* of SODDM and *"Determine the need for web service"* of WSIM, which are functionally similar activities, the earlier one is chosen as the naming is more meaningful. The *"Conceptualize the new requirements"* activity of SODDM is chosen since it represents the higher-grained activity as compared to the three similar activities of WSIM (i.e., *"Elicit Web service requirements"*, *"Manage web service requirements"*, *"Model usage scenario"*). Finally, the *"Initiate project management"* activity of SOMA is chosen as compared to *"Manage project deliverables and resources"* activity of SODDM as naming of the

activity is more meaningful. By comparing the activities and the deliverables of each phases, the super-method of the serviciFi method is created. Third, the super-method by now consists of method fragments

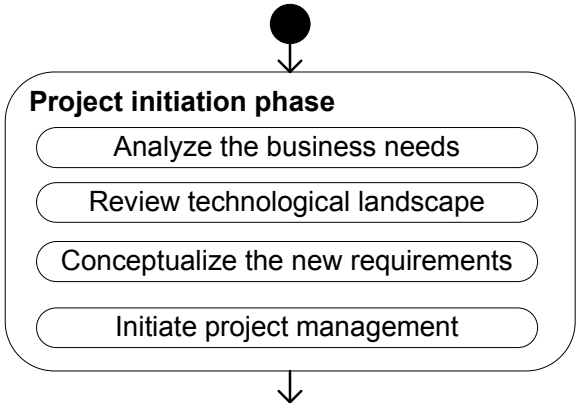

Figure 2.2: Excerpt of the project initiation phase of the super-method

assembled from the three candidate methods. But, to adapt the super-method for migration, several other activities and deliverables such as cost-benefit analysis [246], identifying third party services that have similar functionality to the to-be-extracted services, priority techniques to determine the priority of extraction of identified services are added. Also, the assembled method fragments are renamed to adapt with the migration context. For instance, the *"Analyze the business needs"* of the super-method (see Figure 2.2) is renamed to *"Define project goals"*, *"Review technological landscapes"* to *"Analyze technological landscape"*, and so on. Based on these modifications in the super-method, the serviciFi method is finalized and is explained in Section 2.3.

Figure 2.2 and Table 2.2 are only excerpts of the super-method and the method comparison matrix, respectively. The details of the super-method and the method comparison matrix are not included in this chapter due to the reasons of brevity. Their details have been reported in the work of Reijnders et al. [228].

The serviciFi method aims at reusing the existing legacy code to extract services. The *"Service construction and testing"* phase of the serviciFi method is distinct with the corresponding *"Implementation"* phase of the three candidate methods. The *"Service construction and testing"* phase should include activities to facilitate the legacy source code extraction, whereas, the corresponding *"Implementation"* phase of the three candidate methods aims at creating new services from scratch. Concept slicing is used as an implementation technique (i.e., supporting technology) to extract the legacy codes and expose them as services in the *"Service construction and testing"* phase.

Concept slicing [108] combines two techniques from software (re)engineering and maintenance domain: *program slicing* and *concept assignment* to generate an "*Executable Concept Slice*" (ECS). Program slicing [290] is a well known code analysis technique that is used to identify and abstract the smallest possible subset of a program that can perform an expected functionality. Concept assignment [30] is a technique that assigns individual human-oriented concepts to portions of source code. Both techniques have been used as source code extraction techniques that take a criterion and program source code as input and yield parts of the program as output. However, the extraction criterion of program slicing is expressed at a very low level to construct a slicing criterion such as using program variables, which makes

slicing difficult to apply. In concept assignment, the extraction criterion is expressed at the domain level, making it more practical to apply, but unlike program slicing, the extracted code is not executable as a separate (sub)program. To achieve the combined advantages, while overcoming the individual weaknesses, Gold et al. [108] combined these two techniques as concept slicing and successfully extracted executable source code from legacy code.

2.3 The ServiciFi Method

The serviciFi method is depicted in Figure 2.4 as a PDD. In the following subsections, each phase and its constituent activities are detailed.

2.3.1 Project Initiation

The *"Project initiation"* phase performs the assessment of the viability of the legacy to SOA migration by analyzing the technical and economical feasibility. The phase starts with the "*Define project goals*" activity that identifies what functionalities of the legacy system need to be exposed as services. The second activity, "*Analyze technological landscape*", analyzes the technical aspects of the existing legacy systems such as the programming language used to build the legacy system, availability of the documentation or resources and also the requirements of the target SOA system. In the *"Analyze technological landscape"* activity, the *"portfolio analysis"* [246] of the identified functionalities that are to be exposed as services, is performed. The portfolio analysis assesses both the technical information and business values of the identified functionalities. The technical information includes the overall system functioning of the legacy system and the functionalities present in the legacy system from which the identified functionalities are to be migrated. The business value of the identified functionalities includes the preliminary benefits that is achievable by exposing the identified functionalities as services. The *"Portfolio analysis"* gives the high level overview of the legacy system, its functionalities and the economic benefits. The portfolio analysis is performed by interviewing the developers, if there are any, and/or consulting the available documentation, and/or the current users of the legacy systems. The business value indicates the viable business investment and the preliminary return of investment, which is calculated in terms of maintenance cost (if possible). The business value of the migration is analyzed by interviewing the business managers and by investigating the market needs. The project goals and analysis of technological landscape provide new requirements for the project, which are documented as requirements in "*Elicit new requirements*" activity. Finally, the *"Project management plan"* is stated. The outcomes of the project initiation phase are project management plan and the (dis)approval of the migration project.

2.3.2 Candidate Service Identification

This phase focuses on identifying candidate services to satisfy the requirements detailed in the "*project initiation*" phase. The first activity, "*Analyze as-is situation*", is an open-activity consisting of three sub-activities as shown in Figure 2.3.

The "*Analyze development history*" sub-activity investigates the artifacts of the legacy system, such as requirements documents, UML diagrams, data diagrams, source code, class diagrams, system dependence graphs and even comments in the code in detail as compared to the "*Analyze technical landscape*" activity of the "*Project initiation*" phase. All this information provides better understanding about the

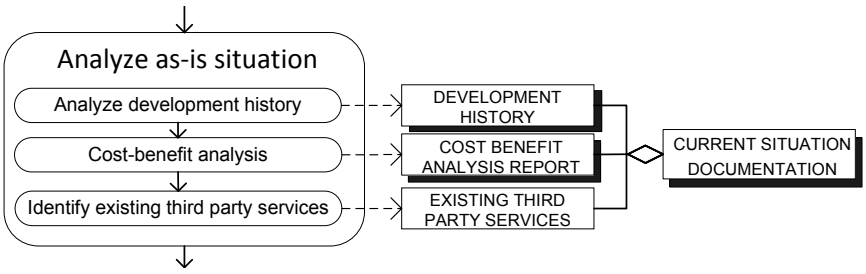

Figure 2.3: Analyze as-is situation

development history as well as functionalities contained within the legacy system. The *"Cost-benefit analysis"*, as suggested by Sneed [246], is performed to estimate the cost of the migration. The cost-benefit analysis is carried out to compare the migration costs with expected benefits. Typically, in this step a comparison between the benefits of migration, redeveloping and doing nothing is performed. The *"Cost-benefit analysis"* sub-activity determines if the migration project is economically viable. The "*Identify existing third party services*" activity investigates if the services that the migration method aims to extract is already available in the market. Availability of such existing services can either provide opportunities to create composite functionalities or already decide to reuse those existing services rather than extracting from the legacy system, if possible. For instance, if one of the functionalities to be extracted as a service is the "*Validation of credit card*" functionality, then reusing the available free web services such as *ValidateCreditNumber*[15] can be economical.

Once the "*Analyze as-is situation*" is documented, the next activity is the "*Identify candidate services*". As for now, this activity is carried out manually based on the functionalities identified in the legacy code against the requirements identified in the "*project initiation*" phase. It is possible that there could be mismatches between the identified functionalities in the legacy code and the requirements of the services to be exposed. Such mismatches are documented in the "*Goal comparison*" activity, which can be used to determine if the identified candidate service can satisfy the business requirements. Depending on the goal comparison, the project might be canceled if the identified candidate services do not fulfil the requirements. For each of the identified candidate services, the granularity has to be determined, which is performed in the "*Determine service granularity*" activity. The service granularity determines if the identified candidate services need to be extracted as an atomic service or as a composite service, representing the composition of the atomic services. Both the granularities have their own advantages such as extracting the atomic services allows composing the new functionalities in future and hence, increases reusability of the extracted services. Whereas, extracting the composite services allows maintaining few services after deploying in service infrastructure and hence, reduces the maintenance cost. The final activity is "*Set priority of services*" in which the identified candidate services are prioritized based on the requirements and business needs. Priority technique MoSCoW [194] is used to determine and create the priority list of the identified candidate services. Based on the priority list the development iterations are planned. The first iteration is started with the highest priority functionality. After every iteration, the priority list is re-evaluated. The iteration fosters the incremental development of the migration.

[15] `http://www.webservicex.net/ws/WSDetails.aspx?WSID=14&CATID=2`, Last accessed on: 30 March 2011

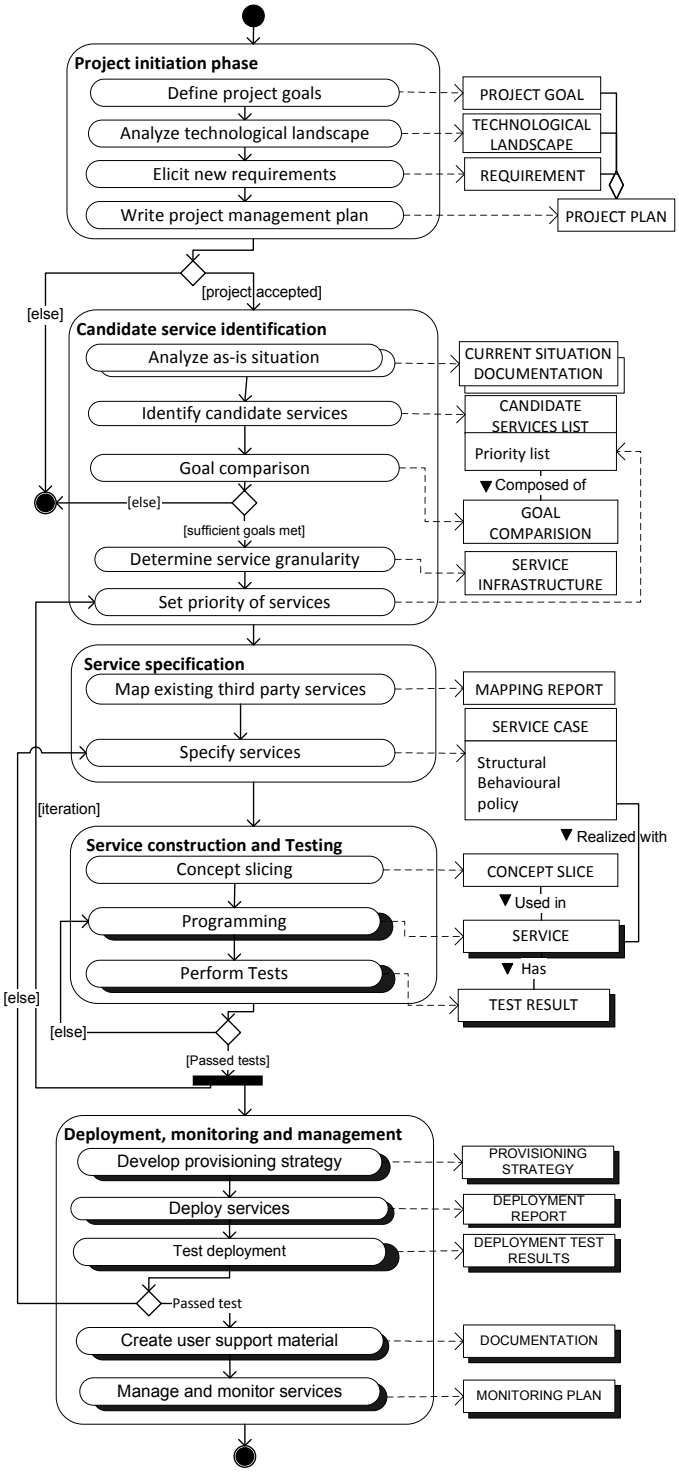

Figure 2.4: PDD of the serviciFi method

2.3.3 Service Specification

The "*Service specification*" phase further details the identified candidate services for the current iteration as well as redesigning any existing third party services, if any third party services are identified from the

"Identify existing third party services" sub-activity of the *"Analyze as-is situation"* activity. Such third party services involved in the current iteration need to be mapped against the existing data types and variable names of the legacy code. This activity provides an overview of the input and output messages required by these candidate services. Once the mapping between the existing services and candidate services is done, the candidate services are detailed on a technical level. Based on the work of Johnston [135], the following three steps are performed to complete the service specification in the technical level: (i) *Structural specification* that determines the necessary operations that represent the functionality of the candidate services and messages that are communicated via the operations by observing the legacy code, (ii) *Behavioral specification* that defines service interfaces that a client will use along with the necessary input, output messages and operations, and (iii) *Policy specification* that denotes policy assertions and constraints for each specific service.

2.3.4 Service Construction and Testing

The "*Service construction and testing*" phase is concerned with the extraction of the source code from the legacy system. The "*Concept slicing*" technique is used to extract the legacy code as an executable concept slice (ECS). This technique is successfully used by Harman et al. [114] to extract the executable (sub)programs from the COBOL-based financial application and by Gold et al. [108] from C-based accounting software. The concept slicing activity is followed by "*Programming*" activity to facilitate the execution of the ECS, if required. For instance, the ECS might need to be deployed in different environments so the ECS might need to be refactored and/or the system calls present in the ECS might need to be replaced or re-programmed. Such modifications are done in the *"Programming"* activity. Finally, each ECS is tested to determine if it is functioning in accordance with the legacy code. In "*Perform tests*" activity, the following tests are performed: unit tests, functional tests, and system tests depending on how the ECS is developed. In case of automatic extraction of an ECS using concept slicing tools, only functional tests are carried out. In case, if the ECS is developed using manual extraction, then all the specified tests need to be executed. When the ECS passes these tests, it is safe to assume the extracted code does in fact satisfy the required functionality and is ready to be deployed as a service in a service infrastructure. At the same time, the iteration for the next candidate service in the priority list is initiated.

2.3.5 Deployment, Monitoring and Management

The "*Deployment, monitoring and management*" phase concerns the deployment of the ECS as a service. The initial activity "*Develop provisioning strategy*" facilitates the usage of the services from both technical and business aspects by a consumer [216]. Service provisioning typically includes activities such as publishing services into a catalog, versioning of services, and metering & billing of the usage of the services [146]. The services are then deployed in the service infrastructures and tested. Based on the outcome of the "*Test deployment*" activity, either the extraction of the service has to be performed again if the specified functionality is not met or the service is ready for usage. Subsequently, the user support materials such as documentation are created. Furthermore, to ensure the proper functioning of the deployed services, the support for management and monitoring is provided in the "*Manage and monitor services*" activity.

2.4 Evaluation

In order to evaluate the serviciFi method, initially eight semi-structured interviews have been conducted with experts. Later, the serviciFi method was applied to two case studies to evaluate it in practice. In the following subsections, each evaluation method is detailed.

2.4.1 Experts Review

The method is evaluated with several experts from industry and academia by conducting semi-structured interviews. This method was chosen because it includes a mixture of open-ended and specific questions, designed to elicit not only the information foreseen, but also unexpected types of information [242]. Table 2.3 depicts the details of the experts. The names are kept anonymous and the experience (in years) in the related field was explicitly asked before the interview was conducted. The variation in the expertise of the experts enabled the assessment of the serviciFi method from different perspectives of software practitioners. Prior to the interview, the experts were provided the PDD of the serviciFi method

Table 2.3: Details of the experts

Name	**Expertise**	**Experience**	**Sector**
A	Software product manager	5	Industry
B	Application manager	5	Industry
C	Software engineering researcher	5	Academia
D	Migration from legacy to SOA/cloud	5	Industry
E	Migration from legacy to SOA	3	Industry
F	Software migration	3	Industry
G	SOA researcher	7	Academia
H	Requirement engineering researcher	6	Academia

and the corresponding documentation. The method was first explained to each expert before conducting the actual interview. Later, the experts were asked to give feedback or remarks on the method. The interviews were conducted in English and each of them took between 80 to 120 minutes. Every interview was recorded and then transcribed. The interviews were conducted to evaluate the proposed method on the basis of five quality measures, suggested by Brinkkemper et al. [40] while designing methods using method engineering. The five quality measures are described as following:

- *Completeness:* The serviciFi method is the assembly of the method fragments of existing service-oriented development methods, so the experts were asked if the serviciFi method captures the complete activities/phases of a migration scenario.
- *Applicability:* The experts were asked if any activities/phases in the serviciFi method were in contradiction with each other in a way that hinders the applicability in any migration project.
- *Efficiency:* The experts were asked if the serviciFi method is efficient and does not contain any repetition of the phases/activities that would increase cost and effort.
- *Consistency:* The experts were asked if the phases/activities were consistent (i.e., semantically correct and meaningful) with each other.
- *Reliability:* The experts were asked if the serviciFi method is reliable to apply in any real world legacy to SOA migration project.

2.4.2 Discussion of the Experts Review

After all experts were interviewed, Constant Comparative Analysis (CCA) [263, 36] was used to analyze the results. The CCA method is used to create knowledge from the data source by avoiding subjective interpretation [269] (i.e., interpretation of the data in accordance with the research objectives). The outcome of the analysis was categorized into two: (i) *minor change requests* that included changes such as renaming the activity names, renaming the deliverables, and (ii) *major change requests* that included changes such as adding/deleting activities, adding iterations among the phases, adding/deleting branchings. The PDD depicted in Figure 2.4 already includes the adjustments made after CCA.

In response to the *completeness* and *consistency* quality measures, 75% (6 out of 8 experts) agreed that the phases of the serviciFi method are complete and consistent. However, the addition of *"Cost-benefit analysis"* sub-activity in *"As-is situation"* activity of *"Candidate service identification"* phase was emphasized. Also, the analysis resulted in the emphasis on using the iterative development method. One of the experts described the need for the iterative development method as "*the migration process is time consuming so there needs to be iterative development cycles to include the changes occurred within the project life time.*"

In response to the *applicability* quality measure, the analysis indicated that almost all experts agree on the applicability of the method as the serviciFi method contains all the typical phases and activities required for a migration project.

In response to the *efficiency* and *reliability* quality measures, half of the experts (4 out of 8 experts) were not confident about the efficiency and reliability of the method. The analysis showed that efficiency and reliability might be influenced by the tools and techniques used in the method along with the various factors of the legacy systems such as availability of the documentation, support from the current users to understand the system, quality of the source code, redundancies in the source code, and understandability and maintainability of the code. One of the experts explained this situation as "*determining the efficiency is contextual because the candidate service identification phase and service construction phase highly depend on the status of the legacy system such as how well the legacy code is written and maintained so far.*"

The analysis also emphasized the need of migration for the current users. The migration of such users is indeed an important aspect, which should not be underestimated [183, 200]. It is recommended to carry our proper user migration planning by conducting training programs [231], however, user migration is out of scope of this research.

Overall the analysis of the expert reviews revealed that most of the experts agreed on *completeness*, *consistency*, and *applicability* quality measures but were not confident on the *efficiency* and *reliability*. In order to evaluate the *efficiency* and *reliability* quality measures, two case studies were conducted in which the serviciFi method was used to extract services from legacy system. The details of those case studies are described in 2.4.3 and 2.4.4.

2.4.3 COBOL Case Study

The initial case study was conducted in a Dutch product company that uses a COBOL-based legacy system for wage and personnel administration. For the reason of confidentiality, information such as

company name, product name, and available functionalities of the legacy systems are kept anonymous. The company wanted to reuse one of the functionalities present in the legacy system as a web service. For the migration of the functionality, the serviciFi method was used. The project goal was to extract the specific functionality and expose it as a web service. The portfolio analysis of the *"Analyze technological landscape"* revealed that the system was developed about 10 years ago in COBOL. Various other functionalities of the whole legacy system were also dependent on this particular functionality. The aim of this migration was to attract more customers to use their software as a service. Also, the company aimed at reducing the maintenance cost by exposing the functionality as a service. The *"Elicit new requirements"* activity of the *"Project initiation"* phase emphasized the use of web service standards to be followed while migrating. The project plan was documented and the *"Candidate service identification"* phase was started. In the *"Candidate service identification"* phase, the *"As-is situation"* sub-activity indicated that the legacy code consists of 1588 lines of code with a lot of exceptions. The availability of the original programmer of the source code helped us to understand the specific functionality. The *"Cost-benefit analysis"* was done by the company which showed that the migration is economically viable and profitable. The *"Identifying existing third party services"* was not relevant in this case because the company wanted to have it's own service and also the *"Goal comparison"* activity was not relevant. The service was to be extracted as an atomic service such that it can be reused in future for any other service composition. Due to the extraction of a single service, no priority list was created. In the *"Service specification"* phase, the initial activity *"Map existing third party services"* was not relevant for the current functionality. The structure, behavior and policy of the service were specified by consulting the original programmer and the product manager of the system.

"Concept slicing" was performed manually due to the unavailability of a slicing tool for COBOL programs. Using the concept assignment, 14 out of 132 variables were identified that were related to the concept (i.e., the functionality to be extracted). Based on those 14 variables, program slicing was applied using which we extracted 426 lines of code from the 1588 lines of code. The extracted code was successfully compiled and in total 240 test cases were run to validate the functionality of the extracted code. All the results of the test cases were compared with the results of the test cases that were run on the original code and a 100% successful result was obtained.

2.4.4 C++ Case Study

In a second case study, a C++ program called SrnaCalc[16] has been used to demonstrate and evaluate the proposed approach. It is an open-source calculator issued under the GNU General Public License ver.3.0 (GPLv3). SrnaCalc is a simple command-line calculator with essential mathematical functions, memory and scripting capability. The *"Analyze technological landscape"* activity indicated that the following functionalities were present: *operators* to display the operators used, *eval* to evaluate the expression, *getPrecision* and *setPrecision* to manage precision, *memory* to list the contents of memory, *add*,*set*, *isset*, *get*, *remove* and *read* to add, change, find, append, remove, and read a variable of the memory, respectively.

The goal was to extract the *eval* functionality, which calculates expressions, and to expose it as a service. The *"Elicit new requirement"* and the *"Write project management plan"* activities were not relevant in this case study. The *"Analysis as-is situation"* activity resulted in program metrics as depicted

[16] `http://sourceforge.net/projects/srnacalc/`

in Table 2.4 and program dependency graph as depicted in Figure 4.5. The program dependency graph was generated using the Understand tool [240] and was manually edited to enrich it visually by filling with colors and differentiating the edges. The *SC\eval.cpp*, represented with the dark background, is the main code which implements the *eval* functionality. The dotted edges represent the non-relevant dependencies of the main code with other program files after the concept slicing. The comments in the program were written in Czech so Google translator was used to translate the comments for a better understanding of the program.

Table 2.4: Program metrics

Attributes	**Count**
#Classes	12
#Class & header Files	21
#Methods	84
#Library Units	114
#Lines of Code	2196
#Blank Lines	236
#Comment Lines	223
#Ratio Comment/Code	0.13

The *"Goal comparison"*, *"Determine service granularity"*, *"Set priority of services"*, and *"Map existing third party"* activities were not relevant in this case study. By manually investigating the code, the service specifications were documented by identifying the function that evaluates the expression and the required parameters.

To facilitate the *"Service construction and testing"* phase, CodeSurfer [110], a C/C++ slicing tool, has been used to extract the legacy code. The extracted code has been initially tested in Visual Studio 2010 Ultimate edition and the shared library file, *evaluate.dll*, is created. To deploy the service and make it available to clients, a service descriptor file, *service.xml*, is created. Listing 2.1 depicts the *evaluate* interface of the *evaluate* web service. Finally, the *service.xml* and the *evaluate.dll* are deployed in the WSO2/C++ web service framework [294]. The service is successfully tested by developing a client application.

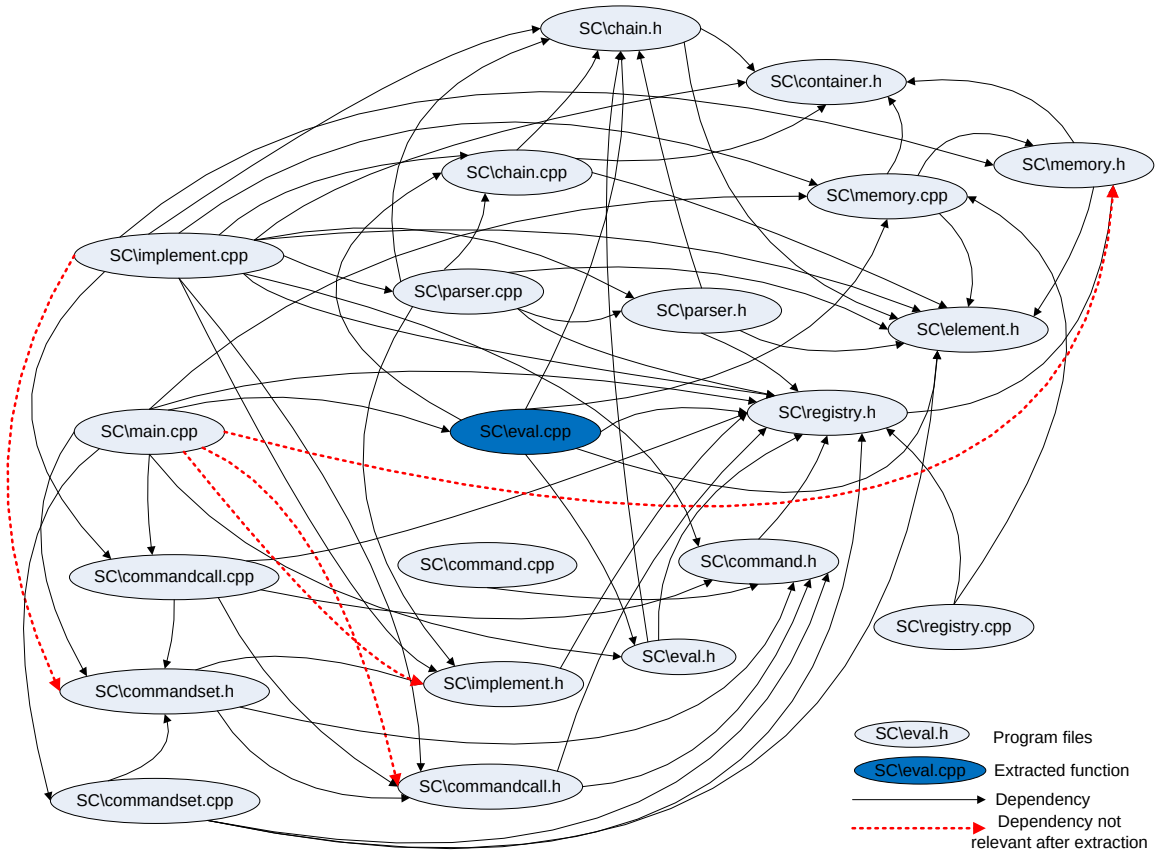

Figure 2.5: Dependency Graph

Listing 2.1: Service description file (service.xml)

```
<service name="evaluate">
  <parameter name="ServiceClass"
        locked="xsd:false">evaluate</parameter>
  <description>
    The Calculator evaluation function presented as Evaluate service
  </description>
  <operation name="evaluate">
  <messageReceiver class="wsf_cpp_msg_recv"/>
  </operation>
</service>
```

2.4.5 Discussion of the Case Studies

The serviciFi method has been successfully assessed and proven to be feasible and practical in the two case studies. Moreover, the result of these case studies also complemented some of the quality measures and the findings of the expert reviews. The successful extraction of services in both the case studies supported the applicability, efficiency and reliability of the serviciFi method. However, the completeness and consistency quality measures are yet to be determined as some of the activities in different phases of the serviciFi method were skipped. Also, the extraction of services as an iterative process was not applicable as both the case studies involved only a single service extraction. However, the experience

of service extraction in the case studies complemented the experts review regarding the efficiency and reliability quality measures. As per the analysis of expert reviews, the efficiency and reliability of the method are influenced by the availability of tools and techniques along with the characteristics of the legacy system, which was reflected while conducting the case studies. Considerable time has been invested to conduct the manual slicing in the COBOL case study while it was easy in the C++ case study due to the availability of slicing tool. Also, the availability of the original programmer and his support enhanced the service extraction process in COBOL case study.

2.4.6 Threats to Validity

The followings are the main threats to the validity of the results of a case study [297]: (i) reliability validity, (ii) internal validity, and (iii) external validity. The reliability validity concerns with the repeatability (i.e., if the same case study is performed by another researcher later following the same procedures as conducted by the earlier researcher, the result of the latter should also arrive at same findings and conclusions). With respect to reliability, the repetition of the COBOL-based industrial case study was not possible. But, the procedures followed in the initial case study are well documented and the same procedures were followed in the C++ case study, which was conducted by a different researcher. The potential threat to the internal validity was the direct involvement of the authors in both case studies, which can introduce bias in the result. To minimize the threat to internal validity, the two case studies were conducted by two different researchers and the result was analyzed by a third researcher. With respect to the external validity, more case studies will have to be performed to extend the results of the current case studies. The different execution context of the two case studies (i.e., one being industrial and the other being experimental) has supplemented the generalizability (i.e., external validity).

2.5 Conclusion

In this chapter, the serviciFi method for legacy to SOA migration has been presented that was developed using method engineering and concept slicing. Unlike other approaches reported in the literature, the serviciFi method successfully combined the migration feasibility with supporting technology to expose the legacy code as services. The serviciFi method has been evaluated, enhanced with expert reviews and has been proven to be feasible to migrate legacy systems to SOA with two case studies. The core of the method is method engineering that reuses the method fragments from existing service-oriented development methods and enhanced by the concept slicing method to develop services by extracting the legacy code.

As a part of future work, the serviciFi method needs to be evaluated with realistic industrial case studies. Also, the serviciFi method can be improved in several ways. The first one concerns the possibility of identifying the service-rich areas in legacy code and automating the identification of the candidate services. Currently, the candidate service identification activity is carried out manually, which can be time consuming and difficult in realistic legacy to SOA migration projects. Pattern identification techniques such as architectural and structural patterns [115], process mining techniques [137], and feature location techniques [101] are possible directions for future work to automate the candidate service identification phase. The next possible direction of the future work concerns the enhancement of service construction phase of the serviciFi method by implementing service extraction activities using *code query technologies* [270, 87]. Code query technologies support the so-called extract-abstract-present paradigm.

Extraction maps source code to relations, the query language then provides the means to query these relations, in order to build new relations and, finally, the results can be presented.

Chapter 3

Legacy to SOA Evolution: A Systematic Literature Review

Abstract

Enterprises depend on business-critical systems that have been developed over the last three decades or more, also known as legacy systems. They have several well-known disadvantages (e.g., inflexible, domain unspecific, and hard to maintain), and this is recognized by both vendors and customers of these software systems. Both vendors and customers of these systems are well aware that better and more flexible customer specific solutions can be created following the service-oriented paradigm. Hence, momentum is growing within enterprises to evolve legacy systems towards Service-Oriented Architecture (SOA). The evolution to SOA is favored because of various advantages including well established sets of open standards, platform and language independent interfaces, clear separation of service interface and implementation, and loose-coupling among services. In the last decade, there have been significant developments in legacy to SOA evolution, and that has resulted in a large research body of which there exists no comprehensive overview. This chapter provides a historic overview, focusing on the methods and techniques used in legacy to SOA evolution. The authors conducted a systematic literature review to collect legacy to SOA evolution approaches reported from 2000 to August 2011. To this end, 121 primary studies were found and evaluated using an evaluation framework, which was developed from three evolution and modernization methods widely used in the software re-engineering domain. The evaluation constitutes the inventory of current research approaches and methods and techniques used in legacy to SOA evolution. The result of the SLR also identifies current research issues in legacy to SOA evolution and provides future research directions to address those research issues.

3.1 Introduction

Recently, many enterprises have focused on increasing their business flexibility and achieving cross-enterprise collaboration to remain competitive in the market, and to meet their business objectives. Enterprises are especially challenged by constant changes in the business environment and changes in the supporting information technology (IT) infrastructures that hinder the overall success of enterprises [280]. Furthermore, most enterprises still rely on so called legacy system- software developed over the previous decades using 3GL programming languages like COBOL, RPG, PL/I, C, C++. Despite the well-known disadvantages, such as being inflexible and hard to maintain, legacy systems are still vitally important to the enterprises as they support complex core business processes; they cannot simply be removed as they implement and store critical business logic. Unsurprisingly, the knowledge contained in these systems is of high value to an enterprise. On the other hand, proper documentation, skilled manpower and resources to evolve these legacy systems are scarce. Therefore, momentum is growing to evolve and reuse those legacy systems within new technological environments Service-Oriented Architecture (SOA) being the most promising one [32, 170].

SOA has emerged as an architectural style that enables the reuse of existing legacy assets within a new paradigm that facilitates loose coupling, abstraction of underlying logic, flexibility, reusability and discoverability [206]. The evolution from legacy to SOA can be beneficial from both economical and technical perspectives. From an economical perspective, legacy to SOA evolution fosters change management including intra-organizational changes, and changes in enterprises [149, 150, 210]. From a technical perspective, seamless enterprise collaboration through service composition [148] and reduction in maintenance cost are claimed as long term benefits [210, 238]. Motivated by these benefits, there has been significant research in legacy to SOA evolution. However, there is no systematic overview of legacy to SOA evolution, particularly focusing on the techniques, methods and approaches used to evolve legacy systems to a SOA environment. In the systematic literature review conducted by Razavian & Lago [222], an overview of SOA migration families is reported. It focuses on classifying the SOA migration approaches into eight distinct families. The classification is inspired by the reengineering horseshoe method [25] rather than giving a historical overview of SOA migration methods. Also, a brief overview of legacy to SOA evolution is reported by Almonaies et al. [8] that divides the legacy to SOA evolution approaches into four categories: replacement, redevelopment, wrapping and migration. The legacy to SOA evolution approaches reported in this research were not based on any systematic literature review process, so a complete, historical overview of the legacy to SOA evolution approaches is still lacking.

In this chapter, we provide a systematic literature review (SLR) of the existing literature of legacy to SOA evolution. We provide a historical overview of the legacy to SOA evolution approaches reported in academia. We focus on identifying techniques, methods and approaches that are relevant to legacy to SOA evolution or that facilitate the legacy to SOA evolution process. In order to provide such a historical overview, we have developed an evaluation framework inspired by three software evolution frameworks reported in literature. The evaluation framework consists of six distinct phases and each phase has its own evaluation criteria to evaluate any legacy to SOA evolution approach reported in academia. The main contributions of this research are as following:

- A historical overview of legacy to SOA evolution.
- A legacy to SOA evolution process framework.

- An inventory of methods and techniques used in various phases of legacy to SOA evolution.
- A series of research issues and recommendations for future research directions.

We argue that our evaluation framework enables a more comprehensive understanding of legacy to SOA evolution allowing us to recognize the contributions made so far, opportunities for combining approaches and identifying open issues and research challenges that still exist in legacy to SOA evolution. We believe that such an overview will benefit academic researchers and industrial practitioners. The academic researchers can follow up on identified research issues to foster the legacy to SOA evolution, whereas the industrial practitioners can adopt various methods and techniques that are reported in research in real world industrial practices.

The chapter is structured as follows: Section 3.2 provides the details of our research method; Section 3.3 presents the evaluation framework; Section 3.4 discusses the overview of the primary studies; Section 3.5 elaborates the findings of our SLR, Section 3.6 discusses the findings & best practices in legacy to SOA evolution and describes the threats to validity. Finally, Section 3.7 we present the conclusions of our research and possible future research directions.

3.2 Research Method

We have adopted the procedures of conducting a systematic review process based on the guidelines proposed by [154]. A systematic review consists of a review protocol that details the rationale of the survey, research objectives, search strategy, selection criteria, data extraction, synthesis and analysis of the extracted data and interpretation of the findings. Such a review process is typically appropriate in our research since it summarizes the existing contributions, identifies the gaps in the current research and avenues for further research, and provides a background to position new research activities in a research framework.

3.2.1 Review Protocol

A review protocol is a plan that specifies the procedures to be undertaken prior to the execution of a systematic review. Such a review protocol describes how to conduct the search, select relevant studies and selection criteria, and the analysis of the extracted data. A review protocol is composed of the following: research question, data sources, search strategy, study selection strategy, data extraction, and data synthesis. The first four define the scope and motivation of the research while the last two describe how the results are concluded from the data.

3.2.1.1 Research Question

In order to achieve our objective of creating an overview of legacy to SOA evolution approaches, we have formulated the following research questions:

1. How can a legacy to SOA evolution method be systematically defined?
2. What methods and techniques are used to facilitate such a systematic legacy to SOA evolution method?
3. What are the existing research issues and what should be the future research agenda in legacy to SOA evolution?

3.2.1.2 Data Sources

For our research, we have included the following eight electronic libraries/indexing sources as data sources: ACM Digital Library, CiteseerX, IEEE Xplore, ISI Web of Knowledge, ScienceDirect, Scopus, SpringerLink, and Wiley Inter Science Journal Finder.

3.2.1.3 Search Strategy

We have constructed a search string using SOA, legacy, and migration as main keywords, and have included synonyms and related terms. The search string is then constructed using Boolean "AND" to connect the three keywords and Boolean "OR" to allow synonyms and word class variants of each keyword. The resulting search string is depicted in Listing 3.1.

Listing 3.1: Search string

```
(SOA OR ''Service−Oriented'' OR ''Service−Based'' OR ''Service−Centric'' OR ''Service−Engineering'' OR ''SOSE''
    OR ''web service'' OR ''service−oriented computing'') AND(Monolith OR ''legacy code'' OR ''Legacy system''
    OR ''existing system'' OR ''legacy component'' OR ''legacy software'' OR ''monolithic system'' OR ''existing
    software'' OR ''pre−existing software'' OR ''legacy information system'' OR ''legacy program'' OR ''pre−existing
     assets'') AND(migration OR evolution OR modernisation OR reengineering OR re−engineering OR reuse OR ''
    service identification'' OR ''candidate service identification'' OR ''service extraction'' OR bridging OR
    reconstruction OR modernization OR decomposing OR ''incubating services'' OR integrating OR redesigning OR
     ''Service mining'' OR migrating OR transformation)
```

The search string was executed in the digital libraries/indexing services to titles, abstracts and metadata– assuming that these provide a concise summary of the work. Besides the search string, the range of study dates also has to be defined in the search strategy. We decided to choose 2000 as the starting year for the search strategy because SOAP [38] was first submitted to W3C in 2000.

3.2.1.4 Study Selection

It is likely that some of the results (study data) of a search might contain the keywords but are irrelevant to our research. For instance, a study data with the title "An evaluation of legacy systems and grid service systems of health-care domain: An initial step towards transformation to cloud-based system" is included in the result of the initial selection. In order to exclude such irrelevant studies, study selection is performed such that the study data is assessed to determine the actual relevance. A set of inclusion and exclusion criteria based on the scope of research and the quality of the studies were determined by us. The inclusion and exclusion criteria are given in Table 3.1.

The study selection not only eliminates irrelevant studies, but also ensures the quality of the study and the scoping of the research. For instance, I1 inclusion criterion and E4 exclusion criterion ensure that the study data meet the standards of peer-reviewed scientific papers. Inclusion criteria I2, I3 and exclusion criteria E1, E2 and E3 scope the research in accordance with the research objective/motivation.

Table 3.1: Inclusion and Exclusion Criteria for study selection

Inclusion Criteria	**Exclusion criteria**
I1. A study in the form of a scientific peer-reviewed paper. **Motivation:** A scientific paper guarantees a certain level of quality through a peer review process and contains a substantial amount of content.	E1. A study that is not about legacy to SOA evolution. **Rationale:** Our objective is to study legacy to SOA evolution, so we exclude any other legacy modernization. For example, legacy modernization to object-orientation, cloud computing or grid services will be excluded.
I2. A study that is focused on legacy to SOA evolution. **Motivation:** We are interested in legacy to SOA evolution, which implies that any study targeting legacy to SOA evolution should be included.	E2. A study that is related to challenges and issues while modernizing legacy systems to SOA. **Rationale:** We focus on a specific solution to legacy to SOA evolution. We exclude papers with an objective of presenting challenges, issues, and future directions to legacy to SOA evolution.
I3. The objective of the study is to present/propose a solution(s) to legacy to SOA evolution. **Motivation:** We are interested in a specific solution to legacy to SOA modernization. A solution could be a complete evolution process/method or solution enabling legacy to SOA evolution.	E3. A study that has other objective(s) than providing a solution(s) to legacy to SOA evolution. **Rationale:** We exclude papers with a main objective other than proposing a solution to legacy to SOA evolution. For instance, we exclude papers with an objective of presenting challenges, issues, future directions to legacy to SOA evolution and comparing the modernization techniques of legacy to SOA evolution.
	E4. The Study is reported in another language than English. **Rationale:** We exclude the papers that are written in languages other than English, since English is the common language for reporting in most of the international venues of computer science

3.2.1.5 Data Extraction

We extracted the study selection in a spreadsheet including the following details: title, authors, publication year, publication form (journal/conference/workshop/book chapter), name and abstract. We conducted the first selection round based on the "title and abstract" of the study. The study was categorized as follows: (i) relevant (study inside the scope of the research), (ii) irrelevant (study outside the scope of the research), and (iii) moderate (unable to decide the relevancy of the paper). For each irrelevant and moderate study, explicit reasons were provided in the spreadsheet. The moderate category was decided by repeating the review by a reviewer other than the initial reviewer and by discussing the paper with the team. The final outcome is the collection of relevant studies, which we refer to as the primary studies.

3.2.1.6 Data Synthesis

The primary studies were evaluated against the evaluation framework and the findings are reported the following sections.

We conducted a review process adhering to the review protocol. Initially, we had 8493 hits when we ran the search query over the electronic libraries/indexing sources. Those 8493 articles were analyzed by five researchers to determine the relevancy based on title & abstract, which left 269 articles. These articles were then evaluated based on inclusion and exclusion criteria, which resulted in 121 primary studies. The details of the review process can be found in Idu et al. [144]. Figure 3.1 depicts the review process.

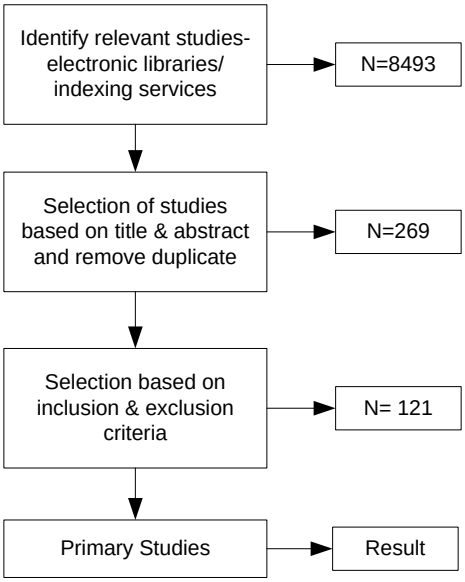

Figure 3.1: The review process with number of studies

3.3 Evaluation Framework for Legacy to SOA Evolution

To develop an evaluation framework for legacy to SOA evolution, we needed to identify the phases that are typically related to evolution/modernization of legacy systems. Based on a high number of citations (popularity), availability of documentation, and completeness of the legacy evolution/modernization process, the following methods from software re-engineering domain were used to identify the phases for our evaluation framework: the butterfly method [295], the Renaissance method [286], and the Architecture-Driven Modernization (ADM) [152]. The main reason for using these evolution/modernization methods is that the software re-engineering domain has been extensively researched and widely practiced in industries, as compared to SOA evolution methods. In particular, we want to reuse the concepts from those methods in the development of a new method for legacy to SOA evolution. Method engineering [39] allows us to reuse existing concepts from existing methods to construct new methods. Hence, we use method engineering and reuse the concepts from the three above-mentioned legacy evolution/modernization methods. We argue that reusing the methods and practices from existing standards/methods saves time and reduces the adoption problem (i.e., it is easier to adapt to the existing methods/practices than learning new methods). The details of the construction of the evaluation framework are reported in Idu et al. [144]. One can argue that there are sufficient relevant legacy to SOA evolution methods that could have been used to develop the evaluation framework. Most of the legacy to SOA evolution methods reported in literature, either focus on developing supporting technology (i.e., implementation techniques to expose legacy systems in SOA) or planning the legacy to SOA evolution (i.e., determining the feasibility of evolution) [222]. However, a legacy to SOA evolution requires the consolidation of both, developing supporting technology and planning the legacy to SOA evolution [75, 143]. In our approach, we aim at developing such a framework that combines both aspects (i.e., planning legacy to SOA evolution and implementation). Furthermore, we aim at assessing those existing legacy to SOA evolution methods using our developed evaluation method rather than using them to develop a new method. From the three methods, we have identified phases that are common to all of them. For instance, legacy system understanding, target system understanding, evolution feasibility determination

and implementation of evolution are common phases in the above-mentioned methods. To make our evaluation framework more relevant to the SOA domain and to reflect the intent of legacy to SOA evolution, we further analyzed and identified some phases from the following service-oriented development methods: Service-Oriented Design and Development Methodology (SODDM) [208], Web Service Implementation Methodology (WSIM) [166], and Service-Oriented Modelling and Architecture (SOMA) [12]. The details of the identification of the phases are detailed by Reijnders et al. [228] using the method engineering approach. From these service-oriented development methods, we have added candidate service identification and deployment & provisioning phases to our evaluation framework. Finally, our evaluation framework includes six phases divided over two generic stages. The evaluation framework and the phases are depicted in Figure 3.2. The evolution planning addresses the question "what to do?" and "is evolution feasible in the given context?" The evolution implementation & management addresses the question "how to do it?" and "what techniques can be used to perform the evolution?" In the following subsections, we explain the phases of our evaluation framework.

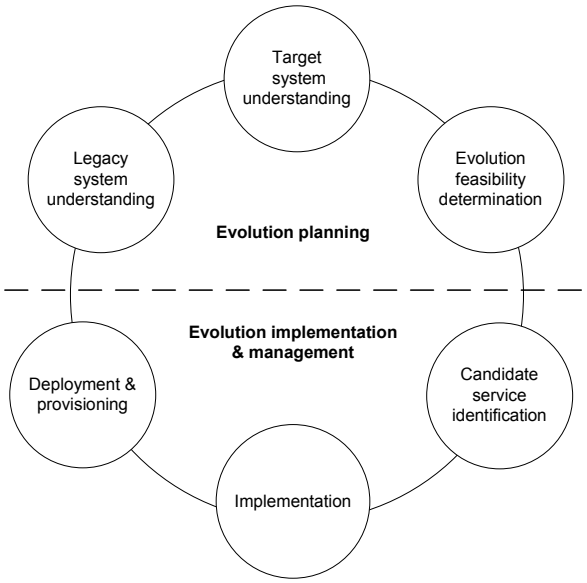

Figure 3.2: The evaluation framework

3.3.1 Legacy System Understanding

Understanding the legacy system and its as-is situation are crucial to the success of any evolution [241]. This includes a detailed analysis of the legacy system and various techniques can be used. For instance, reverse engineering, program understanding, architectural recovery can be used, often with tool support to generate system artifacts. Legacy system understanding often includes analyzing the development history, interviewing the developers (if any) and current users to come to an understanding of the architecture of the legacy system. In our evaluation framework, we have defined evaluation criteria to investigate if any legacy to SOA evolution method includes legacy system understanding and to what extent this phase is discussed.

3.3.2 Target System Understanding

The target system understanding phase facilitates the representation of the desired architecture of the to-be SOA. This phase describes the target SOA environment, which includes activities like defining major

components/functionalities of the SOA environment, specific technologies and standards to be used, the state of the targeted SOA, and availability of existing similar services to reuse. In our evaluation framework, we have defined evaluation criteria to determine whether a legacy to SOA evolution method includes target system understanding for the desired SOA system and to what extent this phase is discussed.

3.3.3 Evolution Feasibility Determination

The legacy system understanding and the target system understanding phases provide better understanding of the as-is and to-be situations, respectively. Based on this understanding, the feasibility of the evolution has to be determined and is done in the evolution feasibility determination phase. The feasibility assessments are carried out at a technical, economical and organizational level. The technical assessment includes measuring the code complexity of the legacy system in terms of cohesion, coupling, reusability and abstraction [227]. Economical assessment includes determining economic feasibility of the evolution, for instance by using the cost-benefit analysis, as suggested by Sneed [246]. Upon analyzing the technical and economical feasibility, the organization approves the evolution project by also considering whether its business goals are met by the intended SOA system. In our evaluation framework, we have defined evaluation criteria to determine whether a legacy to SOA evolution method includes evolution feasibility and if so, how is it performed.

3.3.4 Candidate Service Identification

Legacy systems are subjected to evolutionary development and bug fixing in the source code often by people who did not develop it. This typically leads to much redundancy in the code. Furthermore, poor documentation and lack of appropriate resources (e.g., developers, architects) make the understanding of source code a hard task. In such a scenario, identifying the potential services and service-rich areas in legacy code is definitely a challenging task [141]. The candidate service identification phase aims at locating service-rich areas. Various techniques can be used for this purpose. For instance, architectural reconstruction, feature location, design pattern recovery, cluster analysis techniques, concept analysis, and source code visualization can be used to identify service-rich areas in a large body of legacy code. In our evaluation framework, we have defined evaluation criteria to investigate if any legacy to SOA evolution method includes techniques to identify potential candidate services.

3.3.5 Implementation

The implementation phase is concerned with the technical evolution of the whole legacy system to the target system using various techniques, often supported by the tools. For instance, wrapping, program slicing, concept slicing, graph transformation, code translation, model-driven program transformation, screen scraping, code query technology, and graph transformation can be used to extract/leverage the legacy code as services. In our evaluation framework, we have defined evaluation criteria to investigate if a legacy to SOA evolution method includes any techniques to extract/leverage legacy code as services.

3.3.6 Deployment & provisioning

The deployment & provisioning phase is concerned with deployment and management of the services after extraction of the legacy code. Upon extraction, services are deployed in the service infrastructure. Service provisioning typically includes the after-deployment activities such as publishing, versioning of services, metering and billing of the usage of the services [146]. In our evaluation framework, we have defined evaluation criteria to determine whether a legacy to SOA evolution method includes deployment & provisioning.

Based on the identified phases, we have derived the list of evaluation criteria given in Table 3.2: the first column presents the stages within an evolution, the second column lists the identified phases of our evaluation framework, the third column presents the evaluation question as evaluation criterion for each phase, and the final column gives possible answers for each evaluation question. The answers can be of three types: Yes/No– to indicate whether the given criterion is met, narrative– to answer an open question and scale– to quantify the degree of support for any criterion. The judgement of scale is presented in Table 3.3.

Table 3.2: The evaluation criteria based on the evaluation framework

Stage	**Phase**	**Evaluation Criteria**	**Answer**
Evolution Planning	Legacy System Understanding	Does the solution include legacy system understanding?	Yes/No
		Which technique(s) is used for legacy system understanding?	Narrative
		To what extent are those techniques used?	Scale
		Is there any tool support for legacy system understanding?	Yes/No
	Target System Understanding	Does the solution include target system understanding?	Yes/No
		What criteria/factors are included for target system understanding?	Narrative
		To what extent are those criteria/factors used?	Scale
	Evolution Feasibility Determination	Does the solution include evolution feasibility assessment?	Yes/No
		What technique(s) is used for evolution feasibility assessment?	Narrative
Evolution Implementation & Management	Candidate Service Identification	Does the solution include candidate service identification?	Yes/No
		What technique(s) is used for identifying candidate services?	Narrative
		Is there tool support for candidate service identification?	Yes/No
	Implementation	Does the solution provide any implementation technique for evolution?	Yes/No
		What technique(s) is used for implementation?	Narrative
		Is there tool support for the implementation?	Yes/No
	Deployment & Provisioning	Does the solution provide deployment & provisioning of the services?	Yes/No
Case Study		What empirical evidence (industrial/experiment) is provided?	Narrative
		In which language is the legacy system developed?	Narrative

Table 3.3: The judgement scale to assess the support of techniques and method used

Scale point	Scale Definition	Representation
No support	The specified technique is not mentioned.	-
Implicitly discussed	The specified technique is mentioned.	+
Explicitly discussed	The specified technique is mentioned and discussed but no detailed information is given.	++
Explicitly discussed with evidence of use	The specified technique is mentioned, discussed and there is empirical evidence of its usability.	+++

3.4 Overview of the Primary Studies

In total, we found 121 publications as our primary studies after evaluating against the inclusion and exclusion criteria. Figure 3.3 shows the distribution of primary studies published per year along with the trend-line. The positive slope of the trend–line not only indicates an increasing amount of research being carried out in legacy to SOA evolution domain, but also reflects the increase of legacy to SOA evolution approaches along with the maturity of SOA paradigm– SOA being used as architectural style after SOAP [38] was first submitted to W3C in 2000. We cannot be certain that we have covered all studies with a publication date in 2011, since studies may not have been indexed yet at the time. This is one of the possible reasons for the sharp decrease in publications in 2011.

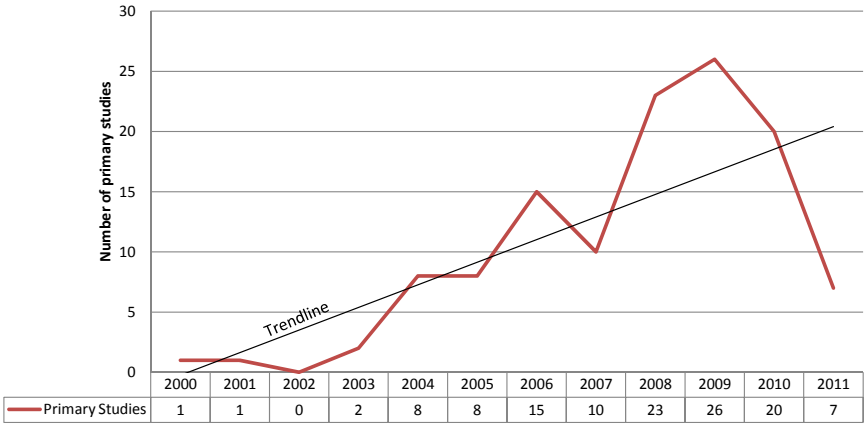

Figure 3.3: Distribution of the primary studies published per year

Figure 3.4 presents the distribution of the primary studies across venues from which at least two articles were selected. It is very interesting to notice that the largest amount of research is reported at venues related to system maintenance, evolution and re-engineering such as CSMR, ICSM, WSE rather than core service-oriented computing venues such as SCC, ECOWS, ICSOC. This implies legacy to SOA evolution is often seen as a solution to maintenance/evolution problems of (legacy) software systems. Also, the frequency of publication in journals is relatively low as compared to conferences or workshops, which is not surprising in such a young field. Note that we have not included the venues with less than two occurrences.

Table 3.4 presents the distribution of the primary studies according to the kind of source. The result shows that conferences are the most widely used method of dissemination for legacy to SOA evolution approaches. The number of journal articles for legacy to SOA evolution approaches is less as compared to the conference papers.

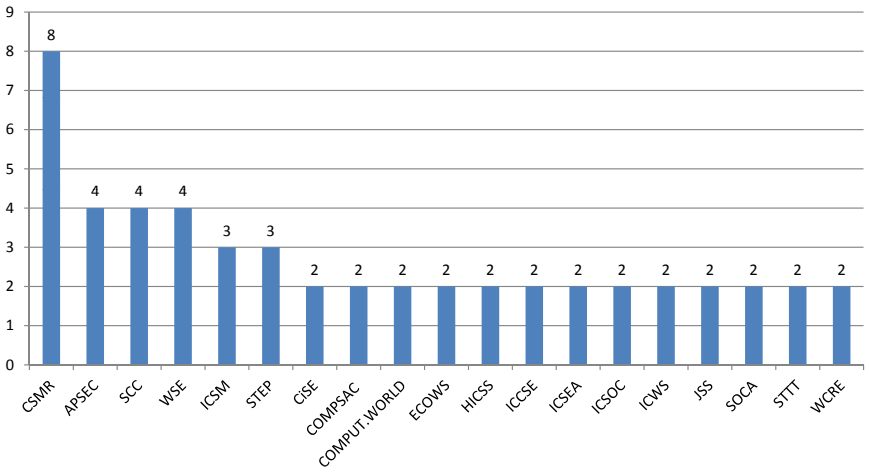

Figure 3.4: Summary of primary studies across different venues

Table 3.4: Summary of primary studies according to the sources

Source/Year	**00**	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**10**	**11**	**Total**
Book	0	0	0	0	0	0	0	0	1	2	1	1	4
Conference	1	1	0	1	6	4	13	9	12	17	17	4	85
Journal	0	0	0	0	1	0	2	1	7	5	2	2	20
Workshop	0	0	0	1	1	4	0	0	3	2	1	0	12
Total	1	1	0	2	8	8	15	10	23	26	20	7	121

3.5 Result

The result of our SLR is based on the evaluation criteria described in Table 3.2. Using our evaluation criteria, we evaluated 121 publications. Due to limitations of space, we have not included the full result of our complete evaluation in this chapter. **Appendix A** depicts an evaluation of a small number of articles. For the complete evaluation result, we refer to Khadka et al. [144]. The result is primarily focused on whether the publication supports the phases of our evaluation framework, what methods and technologies are used (if supported), and whether any tool support for methods and techniques is discussed. Furthermore, the details of empirical evidence (case study) reported in each publication are also presented. In our evaluation, we created an inventory of the methods and techniques as mentioned in the publication. We did not conduct any subjective assumption for categorization. For instance, in many publications "architectural recovery" and "architectural reconstruction" of the legacy system understanding phase are considered to be identical; however, we did not combine them into one. Since we do not conduct any subjective assumption, we believe that this will reduce bias of our findings. We present our findings with two aspects: (i) *degree of coverage*– indicates what stages/phases are supported by the primary studies and (ii) *methods and techniques used*– inventory of what methods and techniques are generally in practice in each phase.

3.5.1 Degree of Coverage

Out of 121 publications, 12 publications have full coverage of the evolution planning stage, i.e., 12 publications support the legacy system understanding, target system understanding and evolution feasibility determination phases. Individually, under the evolution planning stage, 66 publications support legacy

system understanding, 43 publications support target system understanding and 20 publications support an evolution feasibility determination phase.

Similarly, 15 publications out of 121 have full coverage of the evolution implementation & management stage, i.e., 15 publications support the candidate service identification, implementation, and deployment & provisioning phases. Individually, 59 publications support the candidate service identification phase, 97 support the implementation phase and 22 support the deployment & provisioning phase. Interestingly, only 2 publications [143, 303] support the overall phases of our legacy to SOA evolution framework. Table 3.5 presents the distribution of primary studies per phase.

Table 3.5: Distribution of primary studies per phase

Legacy to SOA Evolution 2					
Evolution Planning 12			**Evolution Implementation & Management** 15		
Legacy System Understanding	Target System Understanding	Evolution Feasibility Determination	Candidate Service Identification	Implementation	Deployment & Provisioning
66	43	20	59	97	22

3.5.2 Methods and Techniques

We have inventoried the methods and techniques reported in the primary studies and depict them as in bar chart accordingly, one for each of the phase of our evaluation framework. Note that in most of the phases the information was Not Available (N/A) and that the results presented in the bar charts do not include N/A.

Figure 3.5 depicts the methods and techniques that are used for the legacy system understanding phase. Reverse engineering technique is by far the most widely used technique. Documentation and Interviewing are the second and third most used techniques followed by mostly source code analysis or architectural reconstruction techniques. Based on the Scale criteria (-, +, ++, +++), 22 papers extensively discussed legacy system understanding with +++, 18 papers with ++, and 20 papers with +. In most of the cases, multiple methods and techniques were used for legacy system understanding. An interesting observation is that most of the methods and techniques used for legacy system understanding are technical in nature such as reverse engineering, architectural recovery, program understanding. Manual techniques like documentation and interviewing are less common than in-depth descriptions of technical methods. One of the reasons for using methods and techniques of such technical nature is that legacy resources like documentation and developers are scarce– a widely identified problem in legacy evolution [22, 32]. Furthermore, only 26/121 papers discuss tool support for legacy system understanding. In most of the papers, multiple techniques are combined for legacy system understanding.

Figure 3.6 depicts the methods and techniques that are used for target system understanding. Here selection of a specific architecture is most widely used. It is interesting to note that almost all of the instances in the chart are techniques that actually represent the technological aspects of the target system, while only Interviewing refers to the process (i.e., organizational) perspective. Only 13/121 papers extensively discusses the target system understanding with +++, 12/121 papers with ++, and 12/121 papers with +.

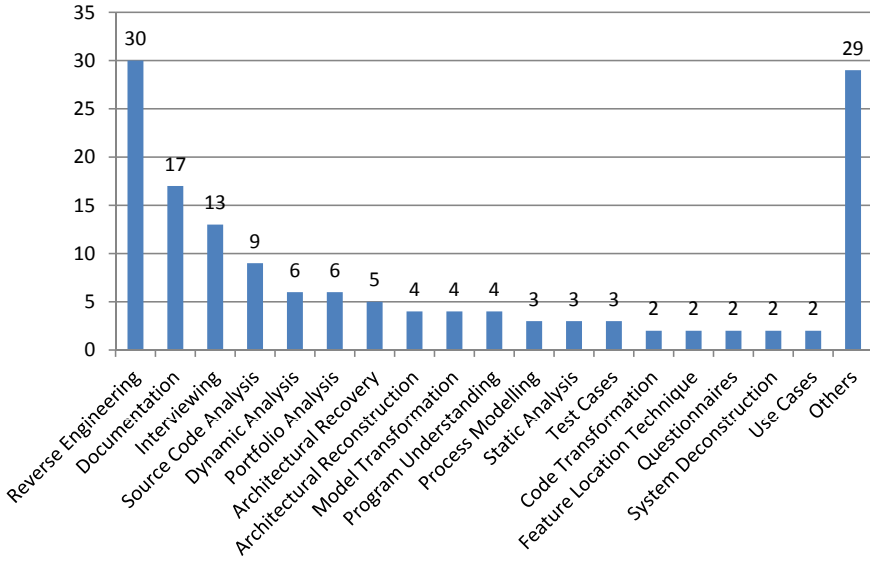

Figure 3.5: Distribution of methods and techniques used for legacy system understanding

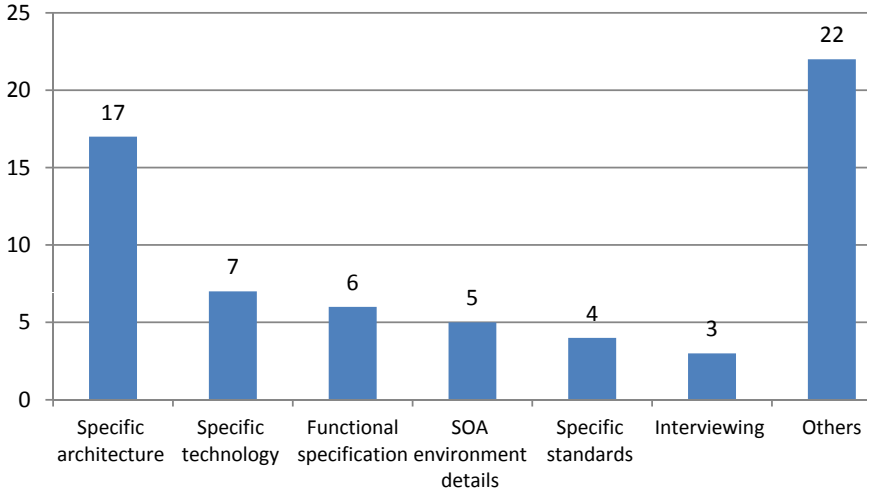

Figure 3.6: Distribution of methods and techniques used for target system understanding

The methods and techniques that are used for the evolution feasibility determination phase are shown in Figure 3.7. Here Cost-Benefit Analysis (CBA) is widely used, followed by Code complexity. While the CBA technique is primarily an economically oriented analysis, the other most used techniques, Code complexity and Reusability assessment, refer to a technical analysis. The details of CBA are presented by Sneed [246] and Umar et al. [273] and the details of code complexity are explained by Sneed [248]. The concept of option analysis for re-engineering (OAR) is proposed by Bergey [24]; it has been used in SMART [15, 172, 173].

Figure 3.8 depicts the methods and techniques that are used for the candidate service identification phase. Manual identification is most commonly used. It is also noteworthy that none of the other techniques are widely used, leading to 51 distinct techniques encountered in the primary studies other than Manual. It is interesting to note that candidate service identification has also been separately researched to foster legacy to SOA evolution.

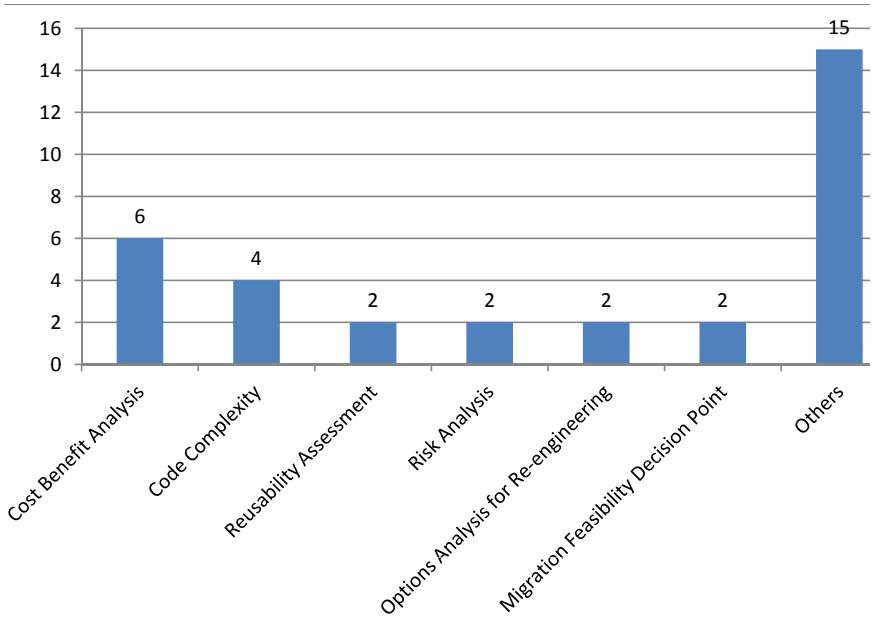

Figure 3.7: Distribution of methods and techniques used for evolution feasibility determination

Alahmari et al. [5] propose model-driven architecture based service identification using a SOA meta-model to identify services in legacy code. Aversano et al. [13] combined information retrieval techniques with a similarity based metric to identify potential services in legacy systems. In [58], the authors propose an ontology based approach in which an ontology stores knowledge of both the application domain and the legacy code. Later, formal concept analysis and relational concept analysis are used to identify the candidate services. Nakamura et al. [197] generate data flow diagrams from the legacy code using reverse engineering techniques to aid candidate service identification. Zhang et al. [298] use clustering technique to analyze the architectural information and identify the related modules as potential candidate services.

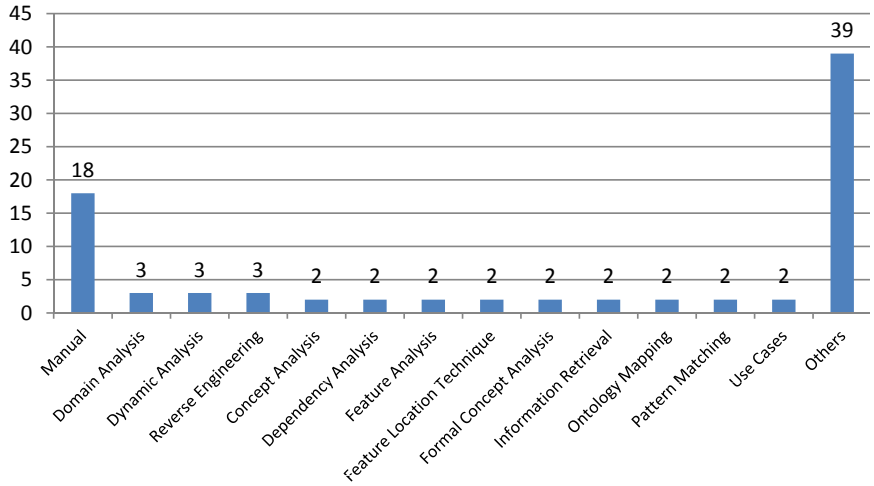

Figure 3.8: Distribution of methods and techniques used for candidate service identification

The methods and techniques that are used in the implementation phase are presented in Figure 3.9. Wrapping is by far the most widely used. Considering the big difference between Wrapping and the other techniques used, we believe that most of the legacy to SOA evolution techniques do not focus on

altering existing legacy code bases. Also, wrapping is a fast, less risky, economical and easy solution although the legacy system remains monolithic. The result of our evaluation shows that techniques like model transformation, program slicing, and code transformations are much less frequently used. From Table 3.5, we find that 97 out of 121 papers support the Implementation phase. In our evaluation, we also found out that 74 papers out of these 97 papers (i.e., papers supporting the implementation phase) also have tool support for implementation. Furthermore, 22 publications support the deployment & provisioning phase.

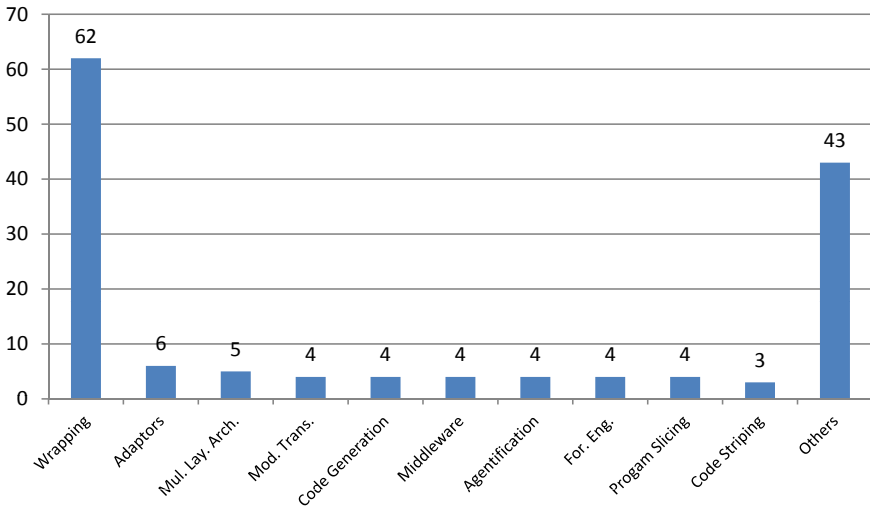

Figure 3.9: Distribution of methods and techniques used for implementation

Figure 3.10 depicts the distribution of empirical studies conducted to validate the proposed legacy to SOA evolution in the primary studies. The majority of primary studies presented case studies which were performed at an Industrial level. Interesting to note is the fact that there was a small number of studies that presented both Experimental and Industrial case studies, thus covering a wider applicability of validation. Among the industrial case studies, C++ and COBOL based legacy systems are most common: four cases for each. In the experimental case studies, Java-based systems were widely used (sixteen in all), followed by COBOL (four systems).

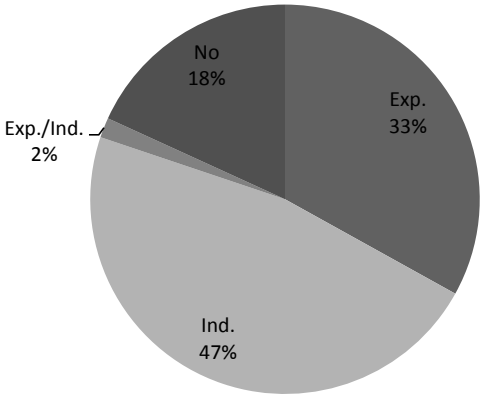

Figure 3.10: Distribution of case study performed

3.6 Discussion

Based on the SLR, we present our findings & best practices, and open research issues & agenda in the following paragraphs.

3.6.1 Findings & Best Practices

The evolution planning stage of the evaluation framework (cf., 3.2) addresses the feasibility of evolution from business and technical perspectives. The evolution planning focuses on justifying whether the legacy system is economically and technically suitable for evolution. To a large extent, the success and failure of an evolution project depends on proper planning [246]. In the context of legacy to SOA evolution, evolution planning becomes more complicated as various technical factors of the legacy systems should be well understood. Such technical factors include complexity metrics [251] and coupling and cohesion metrics for reusability [113, 212]. In the case of legacy systems, obtaining such information is a challenging task, particularly due to the unavailability of resources and documentation. Other important factors include cost estimation for evolution and economic feasibility to determine the profitability of evolution. The economic feasibility should take into account the current expense of maintaining the legacy system and the costs predicted for maintaining the target system after evolution. Hence, evolution planning should also consider the architecture and standards of the target system.

Within evolution planning, legacy system understanding has been extensively investigated, primarily with reverse engineering techniques. The two major categories under reverse engineering in legacy system understanding are program understanding and architectural reconstruction. Program understanding [70] is defined as the process of acquiring knowledge about a computer program and is extensively used for software maintenance, software evolution and software re-engineering. Corbi [70] identifies three actions that can be used to understand a program: read about the program (e.g., documentation), read the program itself (e.g., read source code) and run the program (e.g., watch execution, get trace data etc.). All these three actions have been used in legacy to SOA evolution under various related topics, such as documentation, source code analysis, static analysis and dynamic analysis. Most of the articles state that program understanding or source code analysis techniques are used to understand the legacy codes. However, only few articles explain such program understanding techniques in detail. Source code analysis techniques have been presented well by Zhang et al. [300]; static analysis in [303] using Flow Graph Manipulator (FGM) and dynamic analysis using JGrabLab/GReQL in [303] and TGraph by Fuhr er al. [92]. Architectural reconstruction is a process in which the architecture representations of a software system are obtained from the existing source code [139] and is widely used in the software reengineering domain. Similarly, the use of architectural reconstruction has been also been reported in legacy to SOA evolution approaches. Cuadrado et al. [72] used the QUE-es Architecture Recovery (QAR) workflow to reconstruct the architecture of legacy systems using Jude, Omondo UML studio and Eclipse Test and Performance Tools Platform (TPTP) tool. Lewis et al. [171] and O'Brien et al. [205] used the ARMIN tool to reconstruct the architecture of the DoD command and control (C2) legacy application such that various undocumented dependencies in the source code were identified. Li and Tahvildari [177] used the Extracting Business Services (E-BUS) toolkit to reconstruct the architecture of various Java-based systems. Similarly, Zhang et al. [300] use architecture recovery to obtain design and architectural information that are used as input for service identification. Our evaluation indicates that architectural reconstruction has been used more often with tool support than program understanding. Also, in most

of the cases both program understanding and architectural reconstruction have been employed. Feature location techniques ([56, 284]) have also been reported in understanding legacy systems.

The target system understanding phase intends to choose the architecture and related SOA technologies of the future system, which eventually plays an important role in the quality of the future SOA system. Lewis et al. [172] argues that the characteristics of the target system will temper decisions about whether legacy components can be reused. Basically, target system understanding can be viewed from two perspectives: functional characteristics and technical characteristics of the target system. The functional characteristics include the potential functionalities to-be evolved from the legacy code. This process is referred to as service design. It also defines to what level of granularity the services are to be defined and, accordingly, the orchestration of the services has to be managed to support business processes. Various functional and non-functional properties should also be considered, such as maintainability, interoperability, responsiveness, performance, security, and availability. The technical characteristics of the target environment include service technology (SOAP or REST-based), messaging technologies, communication protocols, service description languages, and service discovery mechanisms. Despite the importance, target system understanding is not described in detail in most of the articles. Rather, the articles just state that the target architecture or target system is an important aspect. However, the functional characteristics of target system understanding has been well explored in SOAMIG [92] and SMART [172, 171]. The SOAMIG method describes the importance of service design, which is the result of forward engineering (design of the target architecture and the orchestration of services) and reverse engineering (potential functionalities as services from legacy system understanding). The SMART method focuses on designing the target system based on the potential functionalities as services and to assess them with the stakeholders by taking into account of various functional and non-functional characteristics of the target system. From the technical characteristics perspective, Cuadrado et al. [72] provide a clear explanation of using the OSGi specification and service platform. The authors consider maintainability and interoperability as important criteria of the target system and accordingly use OSGi specifications to support those non-functional characteristics.

One of the important phases from the organizational perspective is evolution feasibility determination that determines the go or no-go of the evolution project. Evolution feasibility determination focuses on an economical and technical assessment of the legacy system and the target system along with the business goals that the organization wants to achieve through evolution. The evolution feasibility determination phase uses the finding of the legacy system understanding (e.g., code complexity, cohesion and coupling metrics, etc) and the findings of the target system understanding (e.g., non-functional characteristics, selection of service technology, orchestration design, etc) to determine the technical and economical feasibility. The best practices in the evolution feasibility determination phase include the cost-benefit analysis proposed by Sneed [246] for re-engineering projects and serves as a good starting point. This CBA model has been widely followed in legacy to SOA evolution [143, 247, 248]. Umar and Zordan [273] extended the CBA model to include the integration costs which facilitates the strategic decision making in legacy to SOA evolution. The SMART method uses options analysis for re-engineering (OAR) [244] to determine the so called migration feasibility decision point. Based on the SMART method and a decision framework by Erradi et al. [85], Salama and Aly [237] present a decision making tool for the selection of the legacy to SOA modernization strategies, which also considers evolution feasibility.

Based on the outcome of the evolution planning, the next step is to decide how to implement the

evolution and what implementation techniques are preferred. It has been widely recognized that legacy evolution is not purely a technical problem, but involves business engineering as well [302]. The main challenges are how to identify business functionality as a potential service, how to evolve such business functionality as a service and finally, how to maintain and monitor the service once it is deployed. Based on these three requirements, we have identified three phases under evolution implementation & deployment stage.

Identifying service-rich areas in a huge chunk of legacy code has been a challenging task in legacy to SOA evolution. Our survey has revealed that techniques applied to locate service-rich areas can be broadly classified into two: modeling the business requirements (top-down) approach and legacy code to business functionalities (bottom-up) approach. In modeling the business requirement approach, the core business process is designed from the functionalities identified from the legacy system understanding and then the process is subdivided until it can be mapped to functionalities in legacy system. In most of such approaches, BPMN is used to model the business process [5, 92, 178, 229, 303]. The legacy code to business functionalities approach utilizes legacy code as starting point to discover existing business knowledge within legacy systems. Various techniques have been used, such as information retrieval [13], concept analysis [300], cluster analysis [299], business rule recovery [184] and pattern matching and discovery [96, 301, 134].

Based on the findings of the Implementation phase, the legacy to SOA evolution can be either categorized as legacy system integration or legacy system migration. Legacy system integration is an approach in which the legacy code is not substantially modified and is used from within a new environment. The legacy systems typically remain in their original environment. Generally, techniques like wrapping, adaptors and middleware based approaches fall into the integration category, which is the predominant implementation technique as far as we have seen in legacy to SOA migration. Integration is claimed to be a fast, less risky, economical and easy solution but the legacy system remains as it is [8, 273]. Wrapping–based legacy to SOA evolution are reported by Sneed [247, 248] in which the author has developed various tools to support the evolution; Ricca and Marchetto [229] used wrapping to evolve ATM functionality to SOA. On the other hand, the legacy system migration approach is one in which the legacy code is transformed, internally modified, or reused in a new environment. Umar and Zordan [273] define migration as "an internal restructuring and modification of legacy systems into target systems". The migration technique is claimed to be costly, time consuming but in the long run the organization can gradually replace the existing legacy system. Various techniques have been used to migration legacy systems to SOA (program slicing [16]; Chen, et al. [58]; Khadka et al. [143]; Marchetto and Ricca [184]; Zhang, et al. [300]; model transformation techniques ([57, 92, 124]). The distinction between integration and migration is discussed by Umar et al. [273] in detail with respective benefits and drawbacks.

In the deployment & provisioning phase, the evolved services have to be deployed and activities are required to manage and control the behavior of services during usage. In the context of legacy to SOA evolution, activities such as testing, versioning and monitoring are important. Service testing has been a research challenge in the SOA domain due to the dynamic binding [46] and the fact that the source code of services might not reside within a single organization [175]. Service testing in the context of legacy to SOA evolution is even more complicated because the exposed service after evolution should perform correctly when compared to the legacy system. Some legacy to SOA evolution approaches also address the testing of exposed services [92, 143, 184, 247, 303]. Due to changing business requirements, services

need to evolve and this leads to multiple versions of an exposed service [86, 143]. Service versioning is inevitable in legacy to SOA evolution as well, particularly, in legacy system integration approaches. In legacy system integration, the legacy code is exposed through interfaces, without making any changes to the original code. Later, changes made to legacy code after evolution have to be reflected in the service interfaces as well and this creates multiple versions of the original service. Also, service monitoring for non-functional attributes becomes important while the exposed services are in use. Service versioning and service monitoring has not received much attention in legacy to SOA evolution.

An increasing number of articles from 2000 to 2011 on legacy to SOA evolution suggests that the hype is gaining momentum in academia and is still in maturing stage. It is also interesting to see that almost half of the results of the research are evaluated in an industrial context. Some good examples of such research include: SMART [172], which has been evaluated in migrating the Department of Defense Mission Status System and Command and Control system, the SOAMIG process model [92, 303] used in Amadeus Germany's RAIL-system, the wrapping method [247, 248] for a COBOL-based insurance system, the migration of Java-based legacy application to SOA [27], the feature analysis method for migrating a COBOL-based telecommunication systems [191], and two case studies of adopting SOA in the transportation sector and public service sector [202].

3.6.2 Research Issues & Agenda

Several research issues still persist in legacy to SOA evolution. In the following subsection, we present research topics based on the results of our evaluation.

3.6.2.1 Legacy to SOA Evolution as a Process

Legacy to SOA evolution is a complex process, which is influenced by technical, economical and organizational factors. So, any legacy to SOA evolution requires a structured process model that can address these technical, economical and organizational factors. The need of such a structured process model has been also argued by various researchers [159, 175]. Such a structured evolution process should include a legacy system assessment to recover knowledge, the standards and architecture of the target system, technical and economical feasibility, a risk analysis, candidate service identification, and the implementation and maintenance of the system after evolution. Our evaluation framework (cf., Fig 3.2) addresses these requirements as it covers all the aspects necessary to support any legacy to SOA evolution project. One interesting finding of our SLR is that only two articles [143, 303], cover all aspects of legacy to SOA evolution as identified by our evaluation framework.

3.6.2.2 Automation of the legacy to SOA Evolution Process

Upon establishing a legacy to SOA evolution process model, the next challenge is the automation of such legacy to SOA evolution process through the development of tools and techniques. As identified by various researchers, e.g., ([159, 175, 202]), one of the major issues of legacy to SOA evolution is tool support for the various phases. In fact such automation would be expensive and needs a huge effort due to variation in legacy systems. As can be seen from our SLR finding, various tools and techniques have already been successfully developed and used in legacy to SOA evolution. Establishing the suitability of those tools and techniques following a legacy system assessment (technical qualities of legacy code) in

the various phases is an interesting and challenging future research topic. Another issue that is worth investigating is "Can legacy to SOA evolution be carried out in language independent manner?" A potential research direction to address this issue could be model-driven legacy to SOA evolution. We are currently involved in an ongoing research project[17] that aims at generating a model of the legacy code and identifying patterns to locate service-rich areas. Such patterns are then employed in tandem with a code-query based program slicer after which the sliced out functionality can be exposed as a service. There have been other initiatives in the model-driven legacy to SOA evolution as well [2, 89, 93].

3.6.2.3 Post Evolution Experience Reporting

Legacy to SOA evolution is not just about the successful technical transformation of an existing state to a new state. Most reports about the legacy to SOA evolution claim successful evolution because it was technically and economically feasible and the desired target state of SOA has been achieved [202]. However, this "successful" evolution does not really indicate that the enterprise has achieved its business goals. Answers to various questions still remain unclear after such a "successful" evolution. Did the legacy to SOA evolution deliver the promised benefits such as increased flexibility, enhanced maintainability, and reduced costs? In many legacy to SOA evolution projects, there were explicit requirements of the enterprise (e.g., Cuadrado et al. [72]) aimed at increased usability and interoperability. Does the evolution to SOA successfully meet such requirements? As identified by Sneed [257] one of the issues after evolution is performance. Such issues are still to be investigated in sufficient detail through experimental analysis.

3.6.2.4 Determining the Decomposability of Legacy Systems

One of the fundamental issues, pointed out by Brodie & Stonebraker [43], is that the evolution of legacy system depends on its decomposability. The less decomposable a system is, the more difficult evolution will be. However, there are still no explicit factors that determine the decomposability of a legacy system. Sneed [257, 248] provides requirements in terms of code properties for determining the suitability of legacy code for wrapping. Similar requirements should also be investigated to determine the decomposability of a legacy system based on the legacy code and complexity. Determining the decomposability of the legacy code facilitates the evolution feasibility process and thus enables choosing the right evolution strategy (i.e., wrapping, replacement, redevelopment, migration) [8].

3.6.2.5 Evolution from Organizational Perspective

The SLR reveals that legacy to SOA evolution is primarily seen as a technical challenge, focused on finding an efficient solution for evolution. However, legacy to SOA evolution also introduces various organizational challenges such as ownership of services, responsibility of maintaining and monitoring of services and resistance from the current IT staff to change. One of the peculiar challenges includes the adoption problem [143, 183] in which the existing users of legacy systems may fear that their expertise may become redundant due to the introduction of SOA. Such organizational issues should also be properly investigated and considered in legacy to SOA evolution.

[17] `http://www.servicifi.org/`

3.6.3 Threats to Validity

Construct validity concerns with "to what extent the inferences can be made correctly". In our research, construct validity refers to the consistent understanding between the study designers and executors. In our review, the review process was designed by one researcher and executed by a group of researchers. Since the review process was designed by a single researcher there is a chance of misinterpretation of the theoretical concepts by other executors. One potential area of such misinterpretation is the selection of the search keywords. In order to avoid such misinterpretation, we have included possible synonyms and even related terms for each keyword and had them reviewed by all five researchers. Further, we have followed specific guidelines to conduct the systematic literature review, which also enhances the consistent understanding among the researchers. The other potential area of subjective misinterpretation is the scale measurement in the evaluation framework. For such subjective interpretation, we provide a clear explanation of the judgement scale (cf. Table 3.3).

Internal validity refers to the extent to which the design and execution of the study are likely to prevent systematic errors. In our research, internal validity refers to the elimination of bias. In our review, the involvement of five researchers in the study selection and evaluation process minimizes the threats to internal validity. Furthermore, in each round of study selection the distribution of the studies was done in such a way that each researcher obtains a different set of studies. We have introduced three categories of studies "relevant", "irrelevant" and "moderate". For each moderate study, the next categorization is done by a researcher other than the one who categorized the study as "moderate". Another potential area of bias is the categorization of the studies into "relevant", "irrelevant" and "moderate". Such a threat is mitigated by clearly specifying the inclusion and exclusion criteria (cf 3.2). Furthermore, the data selection (initial selection, secondary selection and primary study) process was distributed among five researchers rather than one researcher. This step also reduces the possibility of bias.

External validity refers to the generalizability of the results of the study. The scope of our study is restricted purely to the academic domain and in particular peer–reviewed scientific papers. We are aware of the fact that legacy to SOA evolution approaches also originate in industry, and may not have been reported upon academically. Due to feasibility issues and to maintain the quality of the research, we did not include such industry based legacy to SOA evolution approaches.

3.7 Conclusion and Future Research

In this chapter, we have reported on a systematic literature review on legacy to SOA evolution. We have collected 121 relevant papers, published in between 2000 and August 2010, and evaluated them. In order to evaluate those relevant papers, we have described an evaluation framework for legacy to SOA evolution consisting of six phases, categorized over two stages. The proposed evaluation framework is designed by analyzing common phases from three major frameworks related to evolution/modernization of legacy systems, taken from the domain of software engineering. Based on our legacy to SOA evolution framework, we defined evaluation criteria against which all 121 papers were evaluated.

The resulting overview of the evaluation has created an inventory of historical contributions to the evolution of legacy to SOA, and a list of methods and techniques that are widely practiced. Due to limitations of space, only a snapshot of the result of evaluation can be presented in Appendix A.

Particularly, the methods and techniques according to the phases of our evaluation framework have provided insights into existing practices in the legacy to SOA evolution process. In summary, the work described in this chapter offers the following contributions:

- A historical overview of legacy to SOA evolution approaches.
- A systematic evaluation framework for legacy to SOA evolution.
- An inventory of methods and techniques used in legacy to SOA evolution.
- An overview of research issues and future research directions.

We believe that the contributions of this work will benefit researchers on addressing the identified research issues. On the other hand, the inventory of methods and techniques successfully used in academic research can be used by legacy to SOA evolution practitioners in real world industrial practices.

We have identified several possible improvements of our research as well. One of the enhancements of the current evaluation process includes double checking the evaluation result. In the presented evaluation, the primary articles were divided among five researchers and then evaluated. As an enhancement, we aim at double checking each evaluation result by at least one other researcher. This will surely reduce bias (i.e., no subjective categorization are made) and lead to more accurate findings. In our evaluation, we documented what was reported in the article. For instance, "architectural recovery" and "architectural reconstruction" techniques can be considered to be the same and both of them again can be considered to fall under the heading of "reverse engineering". In our evaluation we have not made use of such subjective categorizations. In the future, we aim at refining the results of our evaluation with attribute generalization [71]– a way to generalize the values of the finding into common and related category. Furthermore, we also aim at evaluating the proposed evaluation framework with case studies and enhance it accordingly. Currently, our research is only focused on the legacy to SOA evolution reported in academia. In future, we aim to also provide similar insights into the legacy to SOA evolution approaches practiced in industry.

Appendix A

Reference	Evolution Planning									Evolution Implementation and Management						Deploy & Provisioning	Case Study	
	Legacy System Understanding				Target System Understanding			Evolution Feasibility Determination		Candidate Service Identification			Implementation					
	Y / N	Technique	Scale	Tool supp.	Y / N	Technique	Scale	Y / N	Technique	Y / N	Technique	Tool supp.	Y / N	Technique	Tool supp.		Exp. / Ind.	Language
(Salama & Aly, 2008) [151]	Y	Source Code Analysis	+	Y	N	N/A	-	Y	Filtration, Organizational Assessment, Cost Benefit Analysis	N	N/A	N	N	N/A	N	N	N/A	N/A
(Khadka, Reijnders, et al., 2011) [87]	Y	Technical Analysis, Functional Analysis, Documentation, Interviewing	+++	N	Y	Service Requirements Identification	-	Y	Cost Benefit Analysis	Y	Concept Analysis	N	Y	Concept Slicing	Y	Y	Ind., Exp.	COBOL, C++
(Sneed, 2009) [160]	N	N/A	-	N	N	N/A	-	Y	Code Complexity, Reusability Assessment	Y	Manual	N	Y	Code Stripping, Wrapping	Y	N	Exp.	Cobol
(Zhang, Yang, Zhou, & Zhong, 2010) [194]	Y	Reverse Engineering, Domain Analysis	++	N	Y	Domain Business Logical Model	++	N	N/A	Y	Matching Algorithm	N	Y	Wrapping, Code Modification, Redevelopment	N	N	Exp.	Java
(Canfora, Fasolino, Frattolillo, & Tramontana, 2008) [40]	Y	Reverse Engineering, Static Analysis, Dynamic Analysis	++	N	N	N/A	-	N	N/A	Y	Business Value, Technical Quality Assessment, Use Cases	N	Y	Wrapping, Finite State Automaton Specification	N	Y	Ind.	N/A
(Lewis, et al., 2006) [108]	Y	Documentation , Interviewing, Interviewing, Source Code Analysis, Architectural Reconstruction	+++	Y	Y	SOA Environment Details, Functional Specification	+++	Y	Code Complexity, Dependency Analysis, Risk Analysis, Cost Benefit Analysis	Y	Manual	N	N	N/A	N	N	Ind.	C++

(Sneed, 2008) [159]	N	N/A	-	N	N	N/A	-	Y	Code Complexity, Reusability Assessment	Y	Manual	N	Y	Code Stripping, Wrapping	Y	N	Ind.	COBOL
(Erradi, et al., 2006) [57]	Y	Portfolio Analysis, Interviewing, Source Code Analysis	+++	Y	Y	Decision Making Criteria	+++	Y	Multi Criteria Decision Making	N	N/A	N	N	N/A	N	N	Ind.	N/A
(Zhang, et al., 2006) [193]	Y	Source Code Analysis	+++	N	N	N/A	-	Y	Options Analysis for Re-engineering	Y	Formal Concept Analysis	Y	Y	Program Slicing, Wrapping	N	N	Exp.	Java
(Vemuri, 2008) [182]	Y	Test Cases, Feature Analysis	+++	N	N	N/A	-	Y	Return of Investment	Y	Feature Analysis	N	Y	Wrapping, Rewriting, Refactoring, COTS, Declarative Rule Engine	N	N	Ind.	N/A
(Sneed, 2006) [164]	N	N/A	-	N	N	N/A	-	N	N/A	Y	Business Rule and Value Analysis	N	Y	Code Stripping, Wrapping	Y	N	Ind.	COBOL
(Fuhr, et al., 2011) [61]	Y	Code Parsing, Model Transformation	++	Y	N	N/A	-	N	N/A	Y	Graph Query	Y	Y	Graph Transformation	Y	N	Exp.	Java
(Umar & Zordan, 2009) [175]	Y	Strategic Analysis	++	N	N	N/A	-	Y	Strategic Analysis, Architecture Analysis, Cost Benefit Analysis	N	N/A	N	N	N/A	N	N	Ind.	N/A
(Lewis, Morris, Smith, et al., 2005) [111]	Y	Interviewing, Documentation	++	N	Y	Interviewing, Reference Models, SOA Environment Details	++	Y	Cost Benefit Analysis, Code Complexity	Y	Manual	N	N	N/A	N	N	Ind.	C++
(O'Brien, et al., 2005) [132]	Y	Architectural Reconstruction, Program Understanding	+++	Y	Y	Interviewing, Documentation, Requirements	+++	Y	Options Analysis for Re-engineering	N	N/A	N	N	N/A	N	N	Ind.	C++
(Zillmann, et al., 2011) [196]	Y	Reverse Engineering, Documentation, Test Cases, Static Analysis, Dynamic Analysis	+++	N	Y	Specific Architecture Selection	-	Y	Technical Feasibility	Y	Business Process Mapping	N	Y	Code Transformation	Y	Y	Ind.	Java

Chapter 4

A structured legacy to SOA migration process and its evaluation in practice

Abstract

Legacy to Service-Oriented Architecture migration approaches have been extensively researched over the last decade, primarily to reuse the valuable business logic that resides within legacy applications. Interestingly, most of the proposed approaches fail to cover the complete process from the technological and business perspectives. This chapter presents a structured six-phase process that covers both migration planning and execution, and does so by considering the aforementioned perspectives. Furthermore, within each of the six phases of the process, we present a rationale to justify the need of each phase, current practices within each phase, and challenges that require further attention. The proposed structured process is then evaluated by (i) migrating features of two simple yet representative applications to SOA, and (ii) by mapping activities reported in literature. Based on our findings, we believe that the proposed structured process is successfully fitting to capture the essence of the activities that are performed within the legacy to SOA migration domain by combining various perspectives.

4.1 Introduction

One of the IT challenges faced by many enterprises is the maintenance of their legacy applications and migration of those applications to modern and flexible platforms. Legacy applications inherit various problems such as lack of up-to-date documentation, skilled manpower, resources of the legacy applications, and high maintenance costs [22]. Despite such problems, enterprises cannot simply remove/replace those applications as they are mission critical, implement the core business logic, and their failure can have a significant impact on business[18]. Thus, legacy applications present a dilemma: they are vitally important to the business, however maintaining them incurs unjustifiable expenses [32]. A viable solution to this dilemma is to migrate those systems into new technological environments in which the legacy features can be re-used. Service-Oriented Architecture (SOA) has gained significant attention from academic and industry as a promising architectural style enabling legacy applications to expose and reuse their functionalities [170]. A legacy to SOA migration not only aims at reducing maintenance costs [210], but also enables a higher Return on Investment (ROI) [202] and promotes flexibility to changing business needs [158]. With all these aforementioned benefits of SOA, several approaches to migrate legacy applications to SOA have been reported in academia [145, 222] and in industry [223, 224]. These approaches can be basically categorized into two aspects: (i) migration planning: to determine the migration feasibility based on technological and economical assessments, and (ii) migration execution: to develop a supporting technology so as to expose legacy applications as a service and to provide service provisioning upon exposing the service. However, a legacy to SOA migration approach requires the consolidation of both the aforementioned aspects (i.e., migration planning and migration execution) [75]. Furthermore, legacy to SOA migration is not only a complex technical endeavor, but it also involves various organizational and business perspectives [202]. So, any legacy to SOA migration requires a structured process that can address these technical, organizational and business issues. The need of such a structured process has been argued by various researchers such as Kontogiannis et al. [158] and Lewis et al. [175, 173].

In this chapter, we present a structured process that combines the migration planning and migration execution aspects of a legacy to SOA migration. The structured process is divided into two aspects (i.e., migration planning and migration implementation & management), each consisting of three phases. For each phase, we present a rationale to justify the need for each phase, current practices for each phase, and challenges that require further attention. The proposed structured process is then evaluated by migrating features of two simple yet representative applications to SOA. To further validate the structured process, we selected 17 academic papers reporting legacy to SOA migration from 2000 to 2011 and mapped the activities described therein to the phases of the structured process. The chapter makes the following contributions:

- It presents a structured legacy to SOA migration process that consolidates migration planning and migration execution aspects.
- It identifies rationale, current practices and challenges for each phase of the proposed structured process.

The rest of the chapter is structured as follows: Section 4.2 describes the structured process with its phases and elaborates rationale, best practices and challenges of each phase. Section 4.3 presents two case studies to evaluate the proposed process and provides mapping of activities reported in the literature.

[18]RBS IT Failure Cost Hits £175m: `http://goo.gl/xpDjy`

In Section 4.4, we analyze and discuss our findings and finally, Section 4.5 concludes our research with an outlook to future work.

4.2 The Structured Process

Fig. 4.1 depicts the proposed structured legacy to SOA migration process that includes six phases divided over two aspects: *migration planning* and *implementation & management*. The *migration planning* aims at answering the questions "*how to plan the migration?*" and "*is migration feasible in the given context?* while *Implementation & Management* addresses the question "*how to execute & manage the migration process?* In the following subsections, we explain each phase.

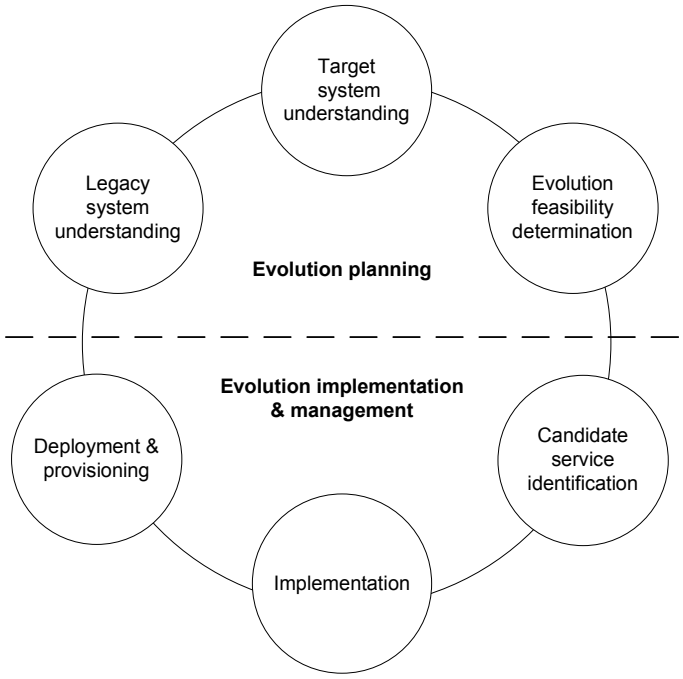

Figure 4.1: Legacy to SOA Migration Process

4.2.1 Legacy System Understanding

Legacy system understanding (LSU) is a deductive process of acquiring knowledge about the "as-is" situation of legacy applications. The goal of this activity is to find the answers to questions such as "*What does the legacy application do?*", "*How does the legacy application do it?*" *LSU* aims at acquiring information including source code characteristics, identifying dependencies, recovering "as-is" legacy system architecture. Techniques to obtain the legacy information range from manual inspection of development history, interviewing developers (if any) and current users to automated re-engineering techniques [45].

Rationale: In a legacy application, factors such as scarcity of knowledge, lack of resources and up-to-date documentation make the migration process expensive, complex and error prone. In such a context, understanding the existing capabilities of the legacy application is essential. The *LSU* phase does not only assist at creating an inventory of the existing features within the legacy applications, but also

facilitates the decomposition of the legacy applications with the aim to maximize reusability. Hence, LSU is essential to the success of legacy to SOA migration [241, 205].

Current Practices: LSU has been extensively investigated, primarily using reverse engineering techniques. Nevertheless, the soft knowledge [196] (i.e., knowledge in the form of skills and experiences within technical staff) is one of the main sources of understanding the legacy applications because these systems are developed and/or maintained by staff familiar with existing legacy systems [202]. Fig. 4.2 depicts the techniques that are used (not intending to be exhaustive) for *LSU*.

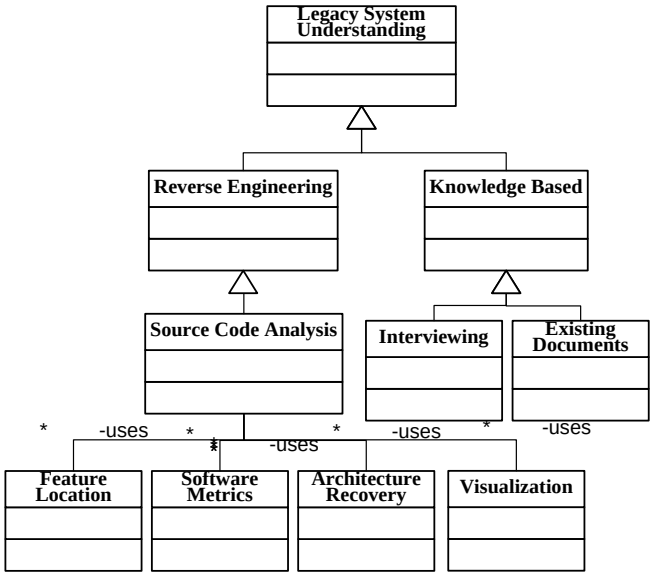

Figure 4.2: Legacy system understanding techniques

We have categorized the techniques into two: (i) reverse engineering using source code analysis and (ii) knowledge-based. By knowledge-based techniques, we mean the knowledge and experience of initial developers and/or maintainer, the end-user experiences obtained via interviewing, and using existing documentation to understand the legacy application. Murer et al. [196] refer to such knowledge as soft knowledge and several authors argue that soft knowledge is of utmost importance to perform a successful legacy to SOA migration [202, 170, 143, 171]. Nevertheless, resources of the legacy applications are scarce and hence, reverse engineering techniques are widely used to further understand legacy application. In particular, researchers have used Source Code Analysis (SCA) to extract information from the legacy applications. Binkley [31] defines source code analysis as "*as a process of extracting information about a program from its source code or artifacts (e.g., from Java byte code or execution traces) generated from the source code using automatic tools*". In legacy to SOA migration, SCA is used to locate and extract features, extract various software metrics, to recover the architecture of the legacy application and to visualize the dynamic behavior of the legacy application. Feature location, used in identifying functional units in a source code, has been used to understand the legacy application [56, 191, 284]. Similarly, software metrics have been extensively used; Sneed [248, 247] measured the size, complexity and quality of legacy programs in terms of modularity, reusability, maintainability metrics; Perepletchikov et al. [212] used coupling metrics to determine maintainability. Several authors such as Lewis et al. [171, 170], O'Brien et al. [205], Cuadrado et al. [72], Zhang et al., [298] have used architectural recovery- a technique to extract architectural information/views of a software system from the lower-level artifacts such as

source code- to facilitate legacy system understanding. Source code visualization, a technique to visualize static and animated forms of software artifacts such as source code and their dependencies, is also used to understand the legacy application [277, 72, 303].

Challenges: Despite the extensive use of techniques to understand the legacy application, various challenges still persist. Being core to the business of the enterprises, legacy applications tend to stay around much longer than the IT staff who built or maintained them, so the soft knowledge about those legacy applications, their inter-relationships with other components, and their design decisions will eventually become scarce if not carefully preserved. This process is known as the knowledge erosion [130] problem, one of the main issues of enterprises with legacy applications. A viable solution to this problem is to keep the documentation up-to-date and also to initiate technology transfer & training programs where the experienced skilled IT staffs transfer their knowledge. Knowledge erosion is an organizational challenge. One of the technical challenges often encountered in the *LSU* is the heterogeneity of the legacy application landscape with multiple programming languages spanning over different hardware platforms and operating systems. Developing generic tools and techniques for understanding a heterogeneous IT landscape results in high cost. Furthermore, most of the reverse engineering techniques are semi-automatic and require human expertise to complement or correct the information extracted from the legacy application. Maximizing the process of automated reverse engineering techniques to understand the legacy applications is also a challenge.

4.2.2 Target System Understanding

The target system understanding (TSU) phase facilitates the representation of the desired architecture of the "*to-be*" SOA. This phase enables the design of a target architecture with major components of the SOA environment, standards to be used, quality of service (QoS) expectations, and interaction patterns between services. In general, the *TSU* represents two aspects of the target architecture: (i) the functional aspect, and (ii) the technical aspect. From the functional aspect, the target architecture not only represents the *"to-be"* functionalities, but also focuses on various non-functional characteristics such as performance, security, availability. From the technical aspect, decisions are made regarding the selection of the technology to be used (SOAP or REST), messaging and communication protocols, service description languages, and service registry.

Rationale: One of the key benefits of legacy to SOA migration is leveraging existing assets [171] and the specification of the target architecture has impact on the level of reusability. Lewis et al. [170] argue that the target architecture largely determines the reusability of the existing legacy components. The other crucial factor that indicates the importance of the TSU phase is the fact that legacy applications have undergone numerous bug fixes and over the years they have been efficient, reliable and responsive to the daily business of the enterprise [22]. Hence, while migrating to SOA, considerable attention should be given to preserve those non-functional characteristics and the target architecture should facilitate to preserve such non-functional characteristics.

Current Practices: Despite its importance, the *TSU* phase is not given sufficient attention. The authors of the SMART method [171, 170] provide guidelines for developing the target architecture based on the legacy components and to assess them with the stakeholder by taking into account various functional and non-functional characteristics of the target system. The SOAMIG method [303] describes the importance of service design as a part of target system understanding, which is the result of forward engineering

(design of the target architecture and the orchestration of services) and reverse engineering (potential features from the *LSU*). Cuadrado et al. [72] explain the selection of using the OSGi specification and service platform to preserve maintainability and interoperability non-functional characteristics.

Challenges: For a successful legacy to SOA migration, various business issues have to be considered. From a business perspective, a legacy to SOA migration is generally triggered by new business needs and goals. To fulfill those new business needs and goals, a target architecture needs to ensure a balanced support to the business and to the IT, often termed business-IT alignment. One of the challenges of legacy to SOA migration is to define an appropriate scope for "business-IT alignment" [196]. Key to overcome this challenge is componentization- a process to deconstruct, analyze and identify business components contributing towards business goals of the enterprise [59]. From a technical perspective, achieving and maintaining the non-functional characteristics of the legacy applications is a key challenge while developing a target architecture.

4.2.3 Migration Feasibility Determination (MFD)

LSU and *TSU* provide better understanding of the "*existing capabilities (as-is)*" and "*target requirement (to-be)*" situations, respectively. Following these phases, the feasibility of the migration is determined from technical, economical and organizational perspectives. From a technical perspective, the findings of the *LSU* are used to decide the viability of the migration. From an economical perspective, the tentative cost of the migration project (economic feasibility) is compared with the allocated budget. From a business perspective, the management team will also assess whether the business goals are met by the legacy to SOA migration.

Rationale: A legacy to SOA migration is a multifaceted complex process that involves technical, organizational and business perspectives. Predicting the feasibility of such a complex process to mitigate the risk of failure can contribute to the success. Needless to say, any failure can threaten the success and fortune of an enterprise [55]. With such associated risk of failure, determining the migration feasibility becomes inevitable from all three perspectives (i.e., technical, organizational and business perspectives).

Current Practices: One of the widely used techniques for determining migration feasibility is the Cost-Benefit Analysis (CBA), proposed by Sneed [246]. The CBA technique is used by Khadka et al. [143] and Sneed [248, 247]. Umar & Zordan [273] extended the CBA model to include the migration costs, which facilitates decision making in choosing a migration strategy. The SMART [170] method uses Options Analysis for Re-engineering (OAR) to determine the so called migration feasibility decision point.

Challenges: A key challenge in the *MFD* phase is to develop tool sets that automate the decision making process by including technical, economical and business value of the legacy application.

4.2.4 Candidate Service Identification (CSI)

Legacy applications are subjected to evolutionary development and bug fixing. These activities leads to so-called "spaghetti code" [117]. Furthermore, lack of up-to-date documentation and resources make the understanding of the code inevitably hard. In such a scenario, identifying candidate services is a challenging task [158], [303].

Rationale: Identifying candidate services is an important activity in the context of legacy to SOA migration as this activity enables reusability and leveraging the existing legacy features [170]. A plethora of methods are reported (Gu & Lago [112], Arsanjani et al. [12]) to identify potential services.

Current Practices: Current practices of CSI are broadly categorized into two: (i) *top-down* and (ii) *bottom-up* approaches. In a *top-down* approach, initially a business process is modeled based on the requirements and then the process is subdivided into sub-processes until these can be mapped to legacy functions. The *top-down* approach is used by Alahmari et al. [6], Fuhr et al. [92], Ricca & Marchetto [184], and Zillmann et al. [303]. In contrast, a *bottom-up* approach utilizes the legacy code to identify services using various techniques such as information retrieval [13], concept analysis [300], business rule recovery [184], source code visualization [277].

Challenges: Several researchers argue that locating and identifying service-rich areas in legacy applications is not only a challenging task, but also an open problem [112, 159]. The functionalities are embedded in legacy applications in such a way that it is difficult to isolate the business functionality from the complex user interfaces and data access logic. Equally challenging is the determination of the optimal granularity of the candidate services such that they contribute to the business goals. Zhang et al. [298] argue that (i) service rationalization- an analysis process to identify the least frequently accessed component as a candidate; and (ii) service consolidation- an iterative process for redefining all the similar service instances into a consolidated version that supports a superset of all the functions exposed by the individual functions, can improve the candidate service identification.

4.2.5 Implementation

The *implementation* phase is related to the execution of the migration of the legacy applications to SOA. Needless to say that the implementation depends on factors such as proper migration strategies, assessments of available tools and techniques while executing the migration process. In our approach, we classify migration strategies into four categories: (i) *replacement* in which a legacy application is replaced entirely with a commercial off-the-shelf (COTS) product; (ii) *integration* in which the existing legacy application is accessible via an interface; (iii) *redevelopment* in which the entire legacy application is re-developed into SOA, and (iv) *migration* in which a legacy application is gradually moved to SOA with reusing the legacy components. Fig. 4.3 depicts the four strategies in a quadrant against "*business value & cost*" versus "*technical value*". Details of these strategies, their merits and demerits are presented by Almonaies et al. [8].

Rationale: This phase is one of the crucial phases of the process in which the migration is technically realized. The legacy application domain is often heterogeneous in terms of programming languages, hardware and operating systems. Furthermore, these heterogeneous applications are highly dependent on each other. In such a scenario, relying on a single implementation strategy is not preferred. Hence, multiple strategies should be used. Various factors such as business value, business priority and the technical qualities of the legacy applications can be used to decide upon the selection of proper strategies.

Current Practices: The *implementation* techniques used in legacy to SOA migration can be broadly grouped into *code level* and *architecture level*. Fig. 4.4 depicts various implementation techniques (non-exhaustive) that are used in legacy to SOA migration. The *code level* group is further divided into various techniques that have been used in legacy to SOA migration such as slicing [143, 300, 184, 58], wrapping [247, 248, 184], refactoring [72], code transformation [303]. In general, wrapping is by far the

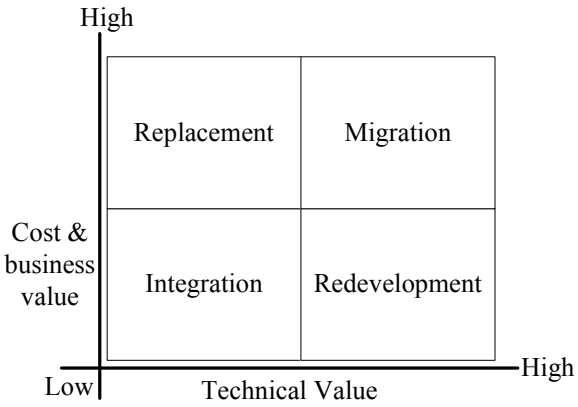

Figure 4.3: Realization strategy

most extensively used technique. A plausible explanation of the predominant use of wrapping is due to the fact that it is fast, less risky, economical and easy. At the *architecture level*, graph transformation techniques are used by Heckel et al. [117] and Fuhr et al. [92]. Some of the other techniques being used in legacy to SOA migration are inspired by model-driven engineering [92, 6].

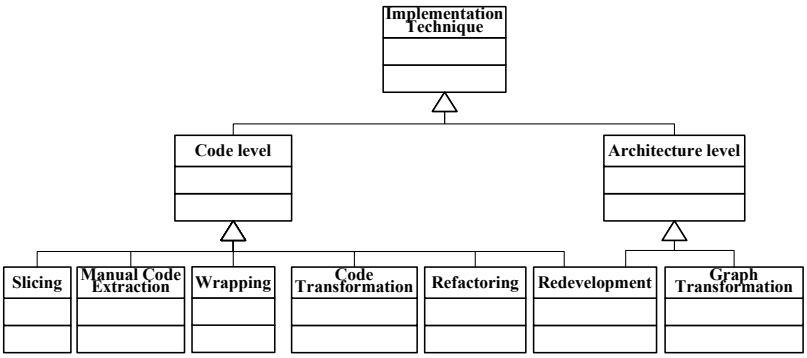

Figure 4.4: Implementation Techniques

Challenges: One of the important challenges in the "*implementation*" phase is the selection of an appropriate migration strategy. There are no proper guidelines for deciding which factors (e.g., business priority, technical qualities, business value, non-functional characteristics) have to be considered while selecting a strategy. Furthermore, legacy applications are in a heterogeneous IT landscape and developing tools for each variation of the legacy languages tends to be very expensive. This leads to the following research challenge: can legacy to SOA migration be realized in a language independent manner?

4.2.6 Deployment & Provisioning (D&P)

This phase is related with the deployment and management of the services after exposing the legacy application as a service. The exposed service is then deployed in the service infrastructure and tested to determine if the expected functionality is indeed exposed correctly as a service. A successful deployment then requires service provisioning that includes activities such as publishing and discovering services in a catalog, maintaining Quality of Services (QoS), versioning, testing, and evolution of services [146]. Furthermore, the user support materials such as documentation are created.

Rationale: This phase includes post migration activities that are crucial to the SOA environment. Services are loosely coupled computation entities [207] and proper management of these entities throughout their life cycle is an absolute requirement [210]. Activities such as service discovery, maintaining QoS of services, testing and evolution of services that lead to the proper functioning of the services ensure that the SOA environment operates reliably and efficiently.

Current Practices: A plethora of research work is reported on service discovery domain [219] in which the authors present categories of service discovery approaches and compare those approaches. Whilst service testing is a relatively new research domain, various traditional testing approaches are used to test services. A survey of these approaches has been reported by Canfora & Di Penta [46]. With respect to service evolution, various approaches have been reported for managing the evolution of services, e.g., Papazoglou [209] present a theoretical approach for addressing the service evolution problem; Andrikopoulos et al. [9] presents a service evolution management framework to identify changes and introduce version control mechanism for services; and Fang et al. [86] describe a service versioning mechanism to assist service evolution.

Challenges: One of the challenges of the *D&P* phase is automated service discovery with minimal user involvement. Most of the approaches, reported in the literature, are semi-automated [219]. The use of semantics markup languages is a step towards automated service discovery [210]. With respect to service testing, various traditional testing approaches have been used. Nevertheless, a key challenge in service testing is to develop a testing approach combined with run-time verification [47]. In terms of service evolution, service versioning plays an important role. One of the challenges in service versioning is to determine the service compatibility between a new service version and the old one [86]. Apart from service compatibility, service versioning potentially introduces overlapping between the service functionalities. Hence, determining the proper level of service commonality in terms of overlapping functionality is also a challenge in service evolution [209].

4.3 Evaluation

The proposed structured process has been evaluated with two simple yet representative mathematical based calculator case studies: one in C++ and one in Java. The details of the case studies are presented below.

4.3.1 SrnaCalc application

SrnaCalc[18] is an open-source and simple command-line calculator with basic mathematical functions and scripting capabilities. The goal of this migration project is to extract the function that evaluates any expression. As a part of the *LSU*, we used the reverse engineering tool Understand[19] to explore the features, functional dependencies and compute various metrics of the calculator program. This activity resulted in identifying the following features: "eval" to evaluate the expression, "operator" to display the operators used, "getPrecision" and "setPrecision" to manage precision, "memory" to list the contents of memory and functions to add, find, change, delete, and append a variable. Furthermore, the dependency graph, as depicted in Fig. 4.5, helped us to understand the structure of the program. As

[18] `http://sourceforge.net/projects/srnacalc/`
[19] `http://www.scitools.com/`

to *TSU*, a SOAP based web service is used in addition to WSO2/C++[20] web service framework as it supports C++. Due to the experimental case, determining economical feasibility of the migration was not relevant. Nevertheless, various software metrics such as coupling, cohesion and Mc-Cabe complexity were derived to determine the technical feasibility of migration. Furthermore, the *CSI* phase was also not relevant due to the small code base. As for *Implementation* phase, we used CodeSurfer[21] tool as a program slicing tool to extract the "eval" feature. Finally, the extracted service was deployed and was successfully tested.

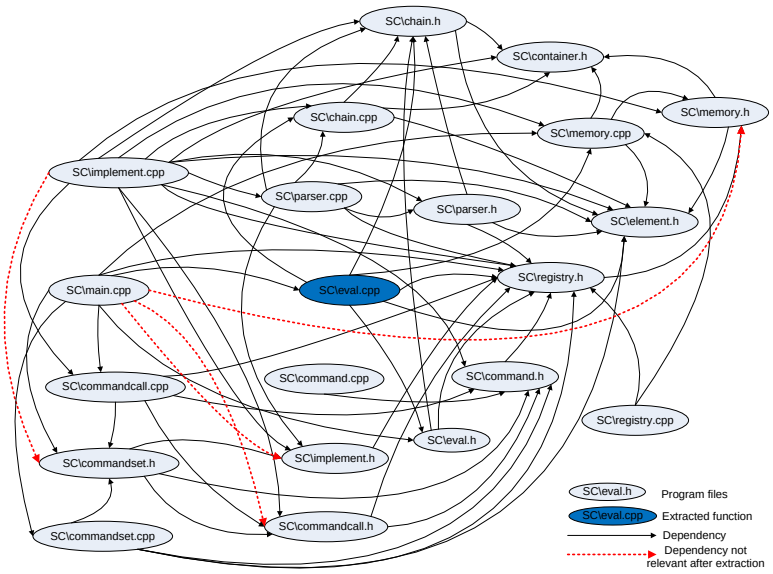

Figure 4.5: Dependency Graph

4.3.2 Java calculator suite

The second case study is a Java-based mathematical calculator, called Java Calculator Suite[22]. It is an open-source and has a Graphical User Interface (GUI) with basic mathematical functions. The motivation of this case study is to migrate and expose the evaluation function that calculates any given expression. Unlike SrnaCalc, the Java calculator does not have scripting and memory related operations. To perform *LSU*, we used STAN[23] to generate the dependency graph, as shown in Fig. 4.6. The dependency graph shows how the GUI is related with the calculator feature. Regarding the *TSU* phase, a SOAP based web service is used, and Apache Axis2 web service container[24] is used as infrastructure. The Java Calculator suite is relatively small in terms of size and is an experimental case study. Because of this, the *MFD* and *CSI* are not relevant. For "Implementation" phase, again program slicing was used. Indus[25], a program slicer for Java programs, was used to extract the evaluate function and then the service was deployed and tested.

With the two experimental case studies we have identified the importance of various phases of the structured process such as the role of *LSU* phase was important to understand the features of the

[20] http://wso2.com/products/web-services-framework/cpp/
[21] http://www.grammatech.com/products/codesurfer/academic.html
[22] http://sourceforge.net/projects/bfegler/
[23] http://stan4j.com/
[24] http://axis.apache.org/axis2/java/core/
[25] http://indus.projects.cis.ksu.edu/

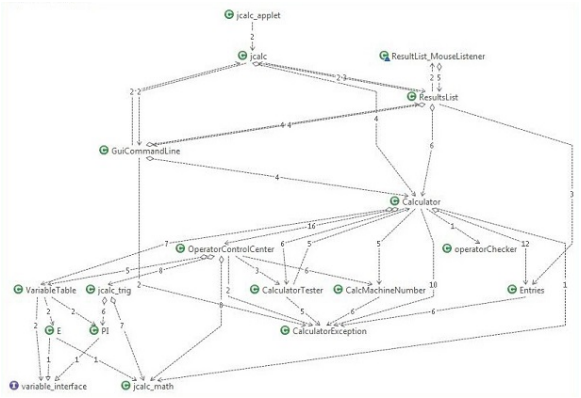

Figure 4.6: Dependency Graph

program. The derived program metrics from the two programs using reverse engineering tools enabled us to understand the structure, core features of and dependencies within the program. Equally important was the use of SOAP and selection of the service infrastructures as a part of *TSU* phase. The program slicing technique used in the "Implementation" phase helped to extract the relevant source code and eventually to expose that as services. One interesting observation is that in the case of Java-based calculator, the GUI code was interwoven with "evaluation" code so it was not so easy to slice. Finally, the services were deployed, tested and exposed using WSDL as activities of the *D&P* phase of the structured process.

Both case studies that we performed to evaluate the applicability of our proposed structured process were relatively small in terms of LoC and were of experimental nature. Because of this, some of the phases were not completely relevant such as in both of the cases we did not perform the *MFD* phase and the *CSI* was done manually. Thus, to further evaluate the applicability of the structured process, we selected academic papers reporting legacy to SOA migration from 2000 to 2011 and mapped the phases/activities from this literature to the phases of the structured process. The papers are selected from our previous work [145] in which we conducted a systematic literature review of legacy to SOA migration. Out of 121 papers, we selected 17 papers from 2000 to 2011 based on the following inclusion criteria:

1. papers reporting a legacy to SOA migration method that is evaluated with an industrial or preliminary case study.
2. papers with high citation count (citation count was recorded on 17-05-2013).

Table 4.1 depicts the selected papers and the mapping between the activities reported in the papers with the phases of our structured process. The full citation report is available here[26]. Due to the two inclusion criteria, there were no papers selected from 2000-2002. From Table 4.1, we can observe that all the phases of our proposed structured process can be mapped to the activities found in the literature. Given this observation, we believe that the proposed structured process is extensive enough to capture the essence of the activities that are performed within the legacy to SOA migration domain. It is interesting to observe that most of the literature (50%) from Table 5.3 have the *LSU*, *TSU*, *CSI*

[26] `http://legacyreengineering.googlecode.com/files/Citation_Report.xlsx`

and *Imp.* as common phases. This indicates that legacy to SOA migration is perceived as a technical endeavor. Out of 17 publications, 15 papers address the implementation aspect of the migration followed by 15 papers addressing the *LSU* phase. Furthermore, much less attention is given to the *MFD* phase, which is a crucial phase for deciding if a migration process is to be performed. The *MFD* phase is an organizational perspective in which the concerned stakeholder decides on whether the migration project is to be executed based on economical, technical and business values. The low frequency of the occurrence of the *MFD* phase further confirms the claim that legacy to SOA migration is largely perceived as a technical endeavor [201, 196]. In the mapping process, Zillmann et al. [303] is the only paper in Table 5.3 that addresses all the phases of our structured process.

Table 4.1: Activity mapping between the selected papers and the structured process

Paper	**Year**	**Evaluation**	**LSU**	**TSU**	**MFD**	**CSI**	**Imp.**	**D&P**
Sneed et al. [260]	2003	Industrial	X	X		X	X	
Zhang & Yang [299]	2004	Industrial	X	X		X	X	X
Jiang & Stroulia [134]	2004	Industrial				X	X	
Chen et al. [56]	2005	Preliminary	X	X		X	X	
Zhang et al. [298]	2005	Industrial	X	X		X	X	X
Sneed [257]	2006	Industrial				X	X	
Canfora [49]	2006	Preliminary	X				X	X
Cetin et al. [51]	2007	Industrial	X	X		X	X	X
Chung et al. [62]	2007	Industrial	X	X			X	
Canfora et al. [50]	2008	Industrial	X			X	X	X
Nakamura et al. [197]	2008	Industrial		X		X	X	
Umar & Zordan [273]	2009	Industrial	X		X			
Chen et al. [58]	2009	Industrial	X	X	X	X	X	
Alahmari et al. [6]	2010	Preliminary	X			X		
Bissyandé et al. [35]	2010	Preliminary	X				X	
Fuhr et al. [92]	2011	Industrial	X			X	X	
Zillmann et al. [303]	2011	Industrial	X	X	X	X	X	X

Based on two different types of evaluation (i.e., simple yet representative case studies and activities mapping), it can be observed that our proposed structured migration framework covers most of the important phases of a legacy to SOA process. From the initial two case studies, two of the phases of the structured process (i.e., *MFD* and *CSI* phases) were not relevant. However, with the mapping of activities, we can conclude that those phases are important to any legacy to SOA migration process as these phases are reported in literature. From these two evaluations, we believe that the proposed structured process has included all the phases that are crucial to a legacy to SOA migration.

4.4 Analysis and Discussion

We acknowledge the findings and challenges put forward by the following literature [175, 158, 202, 210]. In this section we analyze and discuss the findings of our research. Initially, we present and explain the findings focusing on each of the phase of the structured process. We then discuss the findings of the structured process in terms of research issues.

Table 4.2: Overview of the current practices, challenges and the possible solutions

Phases	**Current Practices**	**Challenges**	**Possible Solutions**
Legacy System Understanding	Feature location	Preventing knowledge erosion	Knowledge transfer programs
	Software Metrics	Developing generic tooling for heterogeneous legacy understanding	Model-Driven engineering
	Architecture recovery	Maximizing automation in reverse engineering process	Utilizing the human feedback
Target System Understanding	Specific standards	Identifying optimal business-IT alignment	Componentization
	Specific technology	Maintaining non-functional characteristics	Use of proper standards & technologies
	Functional specification		
Migration Feasibility Determ.	Cost-Benefit Analysis	Automating migration feasibility determining toolset	Technical, economical & business value information based toolset
	OAR		
Candidate Service Ident.	Modeling legacy process	Identifying functional areas in source code	Feature location
	Information retrieval		Trace visualization
	Concept analysis		Source code search
Implementation	Slicing	Selecting appropriate migration strategies	
	Code extraction	Tooling for developing generic toolset	Model-Driven engineering
	Code transformation		
	Refactoring		
	Graph transformation		
Deployment & Provisioning	Discovery	Automated service discovery	Use of semantic markup languages
	Testing	Testing with run-time verification	Techniques to combine testing with run-time verification
	Evolution	Addressing service versioning	Usage of service compatibility
	Publication	Addressing service commonality	Self-adaptive services [209]

Table 4.2 summarizes the findings of current practices, challenges and possible solutions (future research directions) of each phase of the structured process. The *LSU* phase has a crucial role in the legacy to SOA migration because of the fact that knowledge about the legacy applications is scarce, particularly, documentation and resources are limited. Also, the original vendors of the programming languages or the hardware on which the legacy applications run may not be supported anymore. In such a context, discovering the existing capabilities of the legacy applications is inevitably hard. Due to this, understanding the legacy applications and its existing capabilities is crucial for a legacy to SOA migration. As seen from Table 4.2 and Fig. 4.2, the *LSU* phase leverages reverse engineering techniques to understand the legacy applications and its features and the knowledge residing within the legacy applications. Various reverse engineering techniques are employed to understand legacy applications such as feature location, using software metrics to determine the technical qualities, architecture recovery to extract the high level diagrammatic representation and software visualization to identify dependencies among the legacy applications. The reverse engineering techniques that are presented are not-exhaustive. Equally important as the reverse engineering techniques is the soft knowledge within the original developers, maintainers and users of the legacy applications because the systems are developed and/or maintained by such staff familiar with the existing legacy applications. Over the years, such knowledge and skills become scarce resulting in knowledge erosion due to factors such as ageing and retirements of the technical staff. Despite the wide use of reverse engineering techniques, several challenges still persist in the *LSU* phase. One of the important challenges is the development of generic toolsets for understanding the legacy application as these applications are heterogeneous in terms of programming languages, hardware and operating system on which they run. A viable approach to this challenge is leveraging Model-Driven Engineering (MDE) based techniques as they facilitate computation at a language independent level. Currently, the reverse engineering techniques are semi-automated and require human expertise to complement or correct the extracted information. One of the solutions to this challenge is to integrate the knowledge from human expertise as feedback to improve the reverse engineering process [48]. Finally, prevention of the (soft-)knowledge erosion is also another challenge which can be mitigated by conducting knowledge transfer program within the enterprises.

The *TSU* phase aims at developing a future "to-be" SOA environment not only based on standards and technologies but also considering the non-functional characteristics of the legacy applications. This phase also provides a blueprint for service design [303] that enables the reusability of the existing legacy features as services and orchestration of those services. To develop a future target architecture, various techniques are employed: using specific standards such as messaging and communication protocols for SOA based development; specific technology such as SOAP or REST-based, service discovery mechanism, and functional specifications to specify and preserve the existing functional and non-functional characteristics of the legacy applications in the future SOA environment. One of the challenges of the *TSU* is finding an optimal business-IT alignment of the future SOA with the business goals of the enterprise as the *TSU* phase intends to enable service design [196]. A solution to this challenge is to use a componentization process: a process to deconstruct, analyze and identify business component contributing to the business goals of the enterprise [59]. Equally important in the *TSU* phase is to maintain the existing functional and non-functional characteristics of the legacy applications in the future SOA environment. Lewis et al. [170] and Cuadrado et al. [72] argue to use SOA-based standards and technologies as countermeasures to this challenge.

The *MFD* phase involves business and organizational perspectives of the migration by allowing the

organization to decide if migration is necessary and feasible. The need for migration is determined based on whether the future SOA environment fulfills or contributes to the business goals expected from the migration, and the migration feasibility is determined by assessing the technical characteristics and the cost of the migration. Furthermore, this phase can assist with determining which implementation strategy is to be considered. Some of the widely used techniques in the *MFD* phase are: Cost-Benefit Analysis to determine the economic value of the migration; and, Option Analysis for Re-engineering, reusability assessment and code complexity techniques aim at assisting on determining migration feasibility in terms of technical characteristics such as mining existing components for reusability, calculating legacy code complexity to decide on which implementation strategy to follow. One of the challenges of the *MFD* phase is to automate the migration feasibility with a toolset that allows the stakeholders to provide information about the business value of a legacy component and the tool then extracts information about technical characteristics of the legacy application and the economic feasibility of the migration. Finally, the migration feasibility is determined based on this information. A representative framework is developed by Salama et al. [237] that facilitates the decision making process of selecting an appropriate migration strategy for SOA migration considering the migration feasibility determination.

Identifying service-rich areas in a huge chunk of legacy code has been a challenging task in legacy to SOA migration. Various techniques are currently used such as modeling business process and mapping these to the features within legacy code, information retrieval, concept analysis, business rule recovery, code visualization and so on. Despite the availability of many techniques still the candidate service identification remains a challenge. A potential research area to assist with locating candidate services can be feature location, source code searching and trace visualization techniques. Feature location and trace visualization have been already used in legacy to SOA migration. Source code searching [14], a technique to search for relevant code within source code, can be an interesting area to investigate.

In general, legacy to SOA migration is perceived as a technical endeavor. This has resulted in a plethora of techniques that are used in the "Implementation" phase. Such techniques include but are not limited to slicing, manual code extraction, wrapping, code transformation, refactoring and graph transformation. In addition, selection of proper migration strategies is equally important. The challenges encountered in the "Implementation" phase include determining an appropriate migration strategy based on technical, organizational and business perspectives, and developing a language independent generic toolset.

The *D&P* phase includes various post migration activities that are vitally important and ensures that the future SOA environment operates reliably and efficiently. Such activities include service specification publication and discovery, service testing, service evolution and Service Level Agreement (SLA) management. Some of the key challenges of the *D&P* phase are automated service discovery, service testing combined with run-time verification, service versioning, and service commonality for evolution.

Based on the findings of the Evaluation (Section 4.3), the role of each phase and the structured process itself is applicable in any legacy to SOA migration method. However, there are some challenges remaining within the structured process. One of the challenges is the automation of the structured process with a toolset in which the tools and techniques for each phase can be suitably integrated. Such an integrated toolset can include tools and techniques ranging from reverse engineering tools, fact extractors, transformation tools, and code generators tools as per requirement. The need for such automation through the development of tools and techniques to assist phases of legacy to SOA migration

is advocated by various researchers (e.g., Lewis et al. [175], Nasr et al. [202]). Additionally, legacy to SOA migration is not only about a successful technical transformation from a legacy application to a SOA, but also to determine whether an enterprise benefits from the claimed benefits of SOA and achieves its business goals. Currently, few case studies are reported in the literature about the post migration experiences. Hence, we urge that more case studies in collaboration with industries are conducted and that they report on these experiences.

4.5 Conclusion

In this chapter we present a six-phase structured process that combines migration planning and migration execution aspects of a legacy to SOA migration. The structured process is divided into two aspects (i.e., migration planning and migration implementation & management), and each perspective consists of three phases. For each phase, we presented a rationale to justify the need of each activity, current practices for each activity, and challenges that require further attention. The structured process is then evaluated by migrating features of two simple yet representative applications to SOA. Due to the experimental nature and small size of those applications, determining the applicability of two of the phases (*MFD* and *CSI*) were not possible. Hence, we further validated the structured process by selecting 17 academic papers reporting legacy to SOA migration from 2000 to 2011 and mapping the activities of those papers to the phases of the structured process. Based on our evaluation, we believe that our proposed structured process is not only successfully fitting to capture the essence of the activities that are performed within the legacy to SOA migration domain, but also has combined the migration planning and migration execution aspects. Based on our findings, we make the following contributions:

- a structured legacy to SOA migration process that consolidates migration planning and migration execution aspects.
- identification of rationale, current practices and challenges for each phase of the proposed structured process.

As to future work, we have identified several possible directions. One of the future works is to evaluate the structured process in industrial case studies. Evaluation with industrial case studies will point out various challenges while executing the phases of the structured process. Furthermore, the structured process and the activities within each phase can be validated by migration experts from academia and industry, which can be expected to further enrich the structured process.

Part II

Legacy System Modernization in Practice

Chapter 5

Migrating a large scale legacy application to SOA: Challenges and Lessons Learned

Abstract

This chapter presents the findings of a case study of a large scale legacy to service-oriented architecture migration process in the payments domain of a Dutch bank. The chapter presents the business drivers that initiated the migration, and describes a 4-phase migration process. For each phase, the chapter details benefits of using the techniques, best practices that contribute to the success, and possible challenges that are faced during migration. Based on these observations, the findings are discussed as lessons learned, including the implications of using reverse engineering techniques to facilitate the migration process, adopting a pragmatic migration realization approach, emphasizing the business perspectives, and harvesting knowledge of the system throughout the system's life cycle.

5.1 Introduction

In the current business environment, enterprises are pressured to respond to changes in the market, laws and regulations, and to remain efficient and innovative to reap benefits from on-demand and new business opportunities. In order to manage these changes and remain competitive, flexibility is required within the enterprise, supported by technology [280]. Technology support itself is constantly evolving with the advancement of new computing paradigms and improvements in hardware infrastructures. Enterprise systems should therefore be designed to enable continuous evolution and to remain responsive to new business opportunities, realizing better re-use and maintainability, and to improve business-IT alignment to achieve business goals [280]. One of the obstacles to adapt to such changes is the presence of legacy systems [22]. Despite their well-known disadvantages, such as being inflexible and hard to maintain, legacy systems are still vitally important to enterprises as they support complex core business processes; they cannot simply be removed as they implement and execute critical business logic effectively and accurately. Unsurprisingly, the knowledge contained in these systems is of significant value to an enterprise. On the other hand, proper documentation, skilled manpower, and resources to evolve these legacy systems are scarce. Therefore, momentum is growing to evolve those legacy systems within new technological environments such as Service-Oriented Architecture (SOA) as SOA facilitates the reuse of existing assets [210]. The SOA paradigm is favored by loose-coupling, flexible composition of business services, re-usability, and abstraction from the underlying technology platforms. Hence, migration from legacy systems to SOA enables enterprises to achieve flexibility for collaboration, agility within a constantly changing environment [210] and thus enabling business-IT alignment. With these claimed benefits, there has been an increasing interest in academia to investigate approaches for migrating legacy systems to SOA [145].

This chapter presents the findings of a case study of the migration of a large scale legacy system from a Dutch bank to a SOA. For reasons of confidentiality, hereinafter the bank is referred to as "NedBank". The chapter describes a 4-phase migration process that is used in NedBank. For each phase, the chapter identifies the benefits of using particular techniques/methods within that phase, best practices that helped to achieve success, and possible challenges that were faced during migration. Based on these observations, the chapter presents the lessons learned from the case study. The findings of the chapter not only emphasize the benefits of using reverse engineering techniques to facilitate the migration process, but also urges academia to pay attention to business and organizational aspects. The business and organizational aspects include governance of the migration process, early involvement of the existing technical staff, and knowledge harvesting of the system.

The chapter is structured as follows: Section 5.2 discusses related work; Section 5.3 explains the research approach and current technological landscape of the payments domain of NedBank; Section 5.4 presents the migration process and discusses benefits, best practices and challenges faced during the migration; Section 5.5 analyzes and presents the findings as lessons learned. In Section 5.6, the chapter concludes with some potential future work.

5.2 Related work

De Lucia et al. [75] describe an approach and tools to migrate legacy applications to web applications. Sneed [248, 258] has contributed several wrapping techniques to migrate COBOL applications to SOA.

A plethora of research has been reported on migrating legacy applications to SOA. They are reflected in the survey [145]. However, considerably fewer real world case studies of legacy to SOA migration are reported. Nasr et al. [202] describe two large scale industrial case studies of legacy to SOA migration; Colosimo et al. [67] present an empirical study of legacy migration in Italian companies; Kokko et al. [157] report on SOA adoption process in nine Finnish organizations.

The use of reverse engineering techniques in software evolution has been extensively researched. Various research roadmaps and surveys (e.g., Bennett et al. [23], Muller et al. [195], Canfora et al. [45]) have been presented. As per the interest of this research, extracting program quality metrics has been reported in [118, 138] and details of use of software visualization in reverse engineering & re-engineering has been reported in a survey by Koschke [160]. Data-intensive legacy system migration has been reported by Henrard et al. [119].

5.3 Research Background

The current research has adopted an exploratory case study method [233], primarily reporting on how the migration is carried out and seeking new insights about which activities are performed during migration. Data collection in this case study is performed based on the participant observation method [233], wherein two of the researchers were directly involved in the migration project. The data collection included the following: (i) *consulting documentation* to identify the need for and goals of the migration process; (ii) *workshops & informal discussions* to discuss the progress of the migration in real time, and (iii) *semi-structured interviews* to understand various aspects of the migration process. In total six semi-structured interviews were conducted. The interviews were conducted in English and lasted between 60-120 minutes. Prior to the interviews, each expert was introduced to an interview protocol, a document detailing the objectives of the interview with some sample questions, and a glossary of the technical terms to attain a common understanding.

Research Context: NedBank is one of the largest banks in the Netherlands with more than 900 branches worldwide. Triggered by the Single Euro Payment Area (SEPA) initiative of the European Union, NedBank started the migration of its core banking systems under a project that started in 2010 and is expected to end in 2018. The main objective of this project is to renovate and migrate its legacy systems to SOA. The estimated cost of the project is 600M Euro. The project is subdivided into 6 different portfolios: channel support, payments, current accounts, customer reporting, counter, and sales & product agreements. After an initial investigation, this research is scoped to the migration of the payments domain because of the following two reasons: (i) the subsystems within the payments domain are diverse with respective to programming languages, hardware and operating systems in use, and (ii) the payments domain is of prime importance in the day-to-day operation of the banking business.

The payments domain is responsible for the overall management of banking transactions including foreign transactions, and interest & cost calculation per transaction of NedBank customers. The subsystems within the payments domain are considered to have high impact on the business of NedBank and have a high priority within the banking system. The payments domain was one of the first domains to adopt automatization in NedBank. Over the years, the subsystems of the payments domain have been subjected to frequent changes which have resulted in a "spaghetti architecture" [117] posing long-term problems such as increased complexity, inflexibile to changes and evolution, and increasing maintenance and running costs. Currently, the payments domain consists of five major legacy subsystems as detailed

in Table 5.1.

Table 5.1: Details of the subsystems in the payments domain

Subsystem	**Language**	**Platform**	**LOC**
CalculateInterest	COBOL	IBM Z/OS	401,761
ForeignAccount	COBOL	HP Tandem	2,193,570
BalanceCheck	COBOL	HP Tandem	817,882
AccountAgreement	COBOL	IBM Z/OS	529,055
ReportCustomer	COBOL	HP Tandem	587,519

To better understand the basic working principles of and dependencies between these systems, a use case is described in which a customer is created and (s)he withdraws money from an Automated Teller Machine (ATM). Figure 5.1 depicts a high level sequence diagram, in which every directed edge between two subsystems implies a coupling between the two.

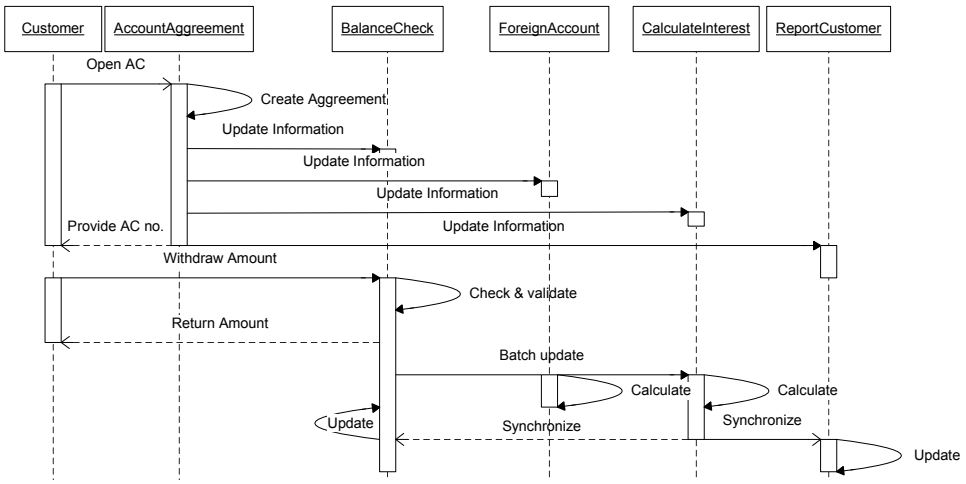

Figure 5.1: Sequence diagram depicting coupling within the payments domain

A new contract for opening an account is created in a Siebel-based sales environment in one of the local branches. During the account opening process, Siebel requests several *AccountAgreement* services in order to get an account number and to create agreements for the customer. The account and agreement creation are processed in real-time. Upon opening an account for a customer, the other four subsystems (*BalanceCheck*, *ReportCustomer*, *ForeignAccount* and *CalculateInterest*) are updated accordingly. When the customer withdraws money from an ATM, initially the request is validated with the agreements stored in the *BalanceCheck* subsystem and the withdrawn amount is reserved from the customer's account. Such individual banking transactions are stored in a flat file and at the end of the day, the flat file is updated with the transactional information from the *ForeignAccount* subsystem: a subsystem responsible for recording the foreign transactions. Afterwards, the flat file is processed by the *CalculateInterest* subsystem that is responsible for interest, commission and cost calculations, and synchronizing the updated records to the other subsystems.

From the information in Table 5.1, it is obvious that the subsystems of the payments domain are combined with heterogeneous IT infrastructures with variations in the COBOL dialects used, and the hardware platforms on which they operate. The subsystems range from internally developed subsystems like *CalculateInterest* to third party built-in packaged subsystems such as *ForeignAccount*. The subsys-

tems are efficient in terms of performance, capable of effectively analyzing, processing and synchronizing millions of records. Nevertheless, to achieve such performance, various features within the subsystems are duplicated and/or updated in ad-hoc manner, increasing system complexity. The increase in complexity has now become a bottleneck to the changeability of the subsystems within the payments domain.

Additionally, based on our observations (interview, documentation, workshops and informal discussion), the main business goals of the NedBank upon migrating to a SOA are (i) accelerating time-to-market, (ii) reducing costs in the payments domain, (iii) transparency in ownership & governance of the products, and (iv) preventing knowledge erosion.

5.4 The Migration Process

In this section, we explain the legacy to SOA migration process of the payments domain. Due to the complexity and tight coupling between the subsystems of the payments domain, the activities within the migration process are performed in a phased, controlled manner based on the business priorities. The migration approach consists of the following four phases, being: (i) Forming a migration program management committee, (ii) Developing a logical target-architecture, (iii) Analyzing the gap, and (iv) Realizing the migration.

5.4.1 Forming a Migration Program Management Committee

The legacy to SOA migration process needs to be capable of addressing various kinds of issues including business, organizational and technical issues [202]. The migration process involves a long term investment of resources and is aimed at conducting a large-scale migration by performing activities with minimal dependencies and maximal parallelization. Thus, to establish a suitable planning and management, a governing body, the *Program Management Committee*, was created. It includes various stakeholders representing senior management officials, business architects representing the different business units, software architects and technical managers, external consultants and application developers. The committee is divided into the following teams with specific responsibilities: a *Steering Committee* to develop a strategic policy for the migration; a *Core Team* to develop the business-IT alignment strategy; a *Program Management* team to manage the payment portfolio; a *Business Change Management* team to ensure proper alignment of business goals with the IT architecture; and an *Architecture Board* to develop an architectural governance within the payments domain. The role of the *Business Change Management* and *Architecture Board* is crucial, in particular, in developing and executing the migration process, aligning the business goals with the architectural requirements, and coordinating architectural priorities inline with the business goals.

Benefits, Challenges and Best Practices: A legacy to SOA migration is a multifaceted process that involves technical, organizational and business issues [196]. To manage such a multifaceted process, a central governing body with suitable governance of the entire migration process is indispensable. Needless to say, a legacy to SOA migration is a complex and challenging process and any failure can threaten the success and fortune of an enterprise [55]. In particular, software failures in the financial domain not only cost millions but also decrease customer confidence[27]. In this migration process, the formation of the *Program Management Committee* has suitably fulfilled the need of such a governing body and hence,

[27]IT failure of Royal Bank of Scotland (RBS): `http://goo.gl/xpDjy`

contributes towards a successful migration. The teams within the *Program Management Committee* have clear responsibilities such that any unpredicted changes were systematically resolved. For instance, any Request For Change (RFC) is primarily resolved by the *Business Change Management* and *Architecture Board* unless the RFC has high business priority and higher estimated cost than a chosen threshold value. Then, the RFC is forwarded with recommendations from the *Business Change Management* and *Architecture Board* to the *Core Team* and to the *Steering Committee* for further considerations.

Realizing that a large scale migration to SOA is not only a technical endeavour, the existing knowledge within the technical staff need to be utilized. The involvement of technical staff (legacy system developers and maintainers) in the committees facilitated the knowledge transfer to the migration team. It is a recurring phenomenon that the technical staff is hesitant to share knowledge due to the fear that their expertise may become redundant due to migration. This phenomenon was countered here by involving the technical staff to actively participate in the migration process.

5.4.2 Developing a Logical Target-Architecture

Initially, a logical target-architecture conforming to the business goals was developed. A logical target-architecture forms the organizing logic for business processes and IT infrastructure, in which the business components are contained. Developing a logical target-architecture that conforms to the business goal was not an easy task. To start with, a group of members from the migration project initially participated in a workshop to define a functional architecture: an architectural model that identifies features that contribute to achieving the business goals. The team members included business process analysts and business architects from the *Business Change Management* team along with software architects and application developers from the *Architectural Board*. Various other external consultants and experts from financial software vendors also participated in the workshop. Together they provided the initial blueprint of the functional architecture. Following the first workshop, three more workshops were conducted that resulted in identifying various business components to realize the initial functional architecture as shown in Figure 5.2. The identification of the business components, referred to as "componentization", was one of the initial activities to realize potential candidate services. The notion of componentization is a way to construct a business component, which corresponds to a feature contributing towards a business goal.

Previously, such features were scattered over various subsystems. For instance, the "calculate interest" feature was previously found in two subsystems: (i) *CalculateInterest* and (ii) *BalanceCheck*. Earlier, "product agreements" feature was also distributed over various subsystems. In the logical target-architecture, related features are gathered within one logical unit to ensure that architecture governance is easy, and to minimize the product gaps within the payments domain.

Benefits, Challenges and Best Practices: Identifying business components representing potential candidate services in legacy to SOA migration is a challenging task. In this migration process, a goal-service modeling approach, proposed by Arsanjani et al. [12], is followed. A goal-service modeling approach is used to componentize the business component because it ties services to the business goals. Each identified candidate service was prioritized based on its business value. Additionally, a catalogue for each business component was created, indicating the degree of reusability by other components and possible functional dependencies (coupling) with other components in the logical architecture. Such a catalogue provides an overview of components whose migration can be performed independently, preferably in parallel with other relatively independent components and hence, maximizing parallelization of

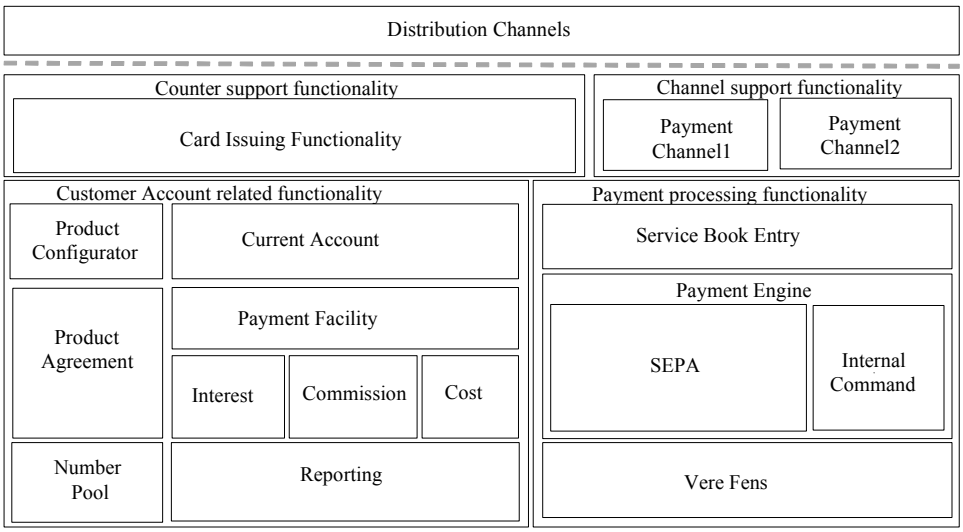

Figure 5.2: Logical Target Architecture

the migration process. In total, 44 different high level features were identified.

5.4.3 Analyzing the Gap

The third phase of the migration process is to gather and determine the information about the legacy system features that can contribute to the realization of the logical target-architecture. The payments domain consists of a mix of many systems ranging from in-built COBOL system such as *CalculateInterest* to a third party packaged application such as *ForeignAccount*. There are not only variations in the COBOL dialect, but also in the running platform such as IBM Z/OS, HP Tandem Nonstop. Also, the documentation of most subsystems was outdated. An investigation of the documentation quality of the *CalculateInterest* system identified missing technical documentation (TD), limited finalized/approved documentation, and fairly good functional documentation (FD). However, the details of the TD and FD for features are still not complete. Furthermore, the technical quality characteristics such as coupling, maintainability, and duplication within the subsystems were still unknown.

As a starting point, all the subsystems of the payments domain were analyzed using source code analyzers to determine the program quality in terms of quality metrics including maintainability, module coupling, duplication and changeability. These quality metrics were derived using reverse engineering tools. Such quality metrics provided a better understanding of the technical qualities of the subsystems within the payments domain. Table 5.2 depicts an excerpt of the assessment results of the subsystems in the payments domain in which Maint. represents maintainability; Coup. represents coupling; Dup. represents duplication; Change. represents changeability and Test. represents testability metrics. Refer to [118, 138] for the details and explanations of these metrics.

Furthermore, to have an in-depth understanding of the technical qualities of each COBOL program, a detailed analysis was carried out for each subsystem using proprietary automated source code analyzers. Such a detailed analysis provided insights into individual COBOL programs within each subsystem. For instance, individual programs were categorized into good, bad and average based on their complexity. Figure 5.3 depicts an excerpt of a detailed program analysis derived from source code analyzers of the

Table 5.2: Excerpt of legacy assessment result

Name	#prog	Maint.	Coup.	Dup.	Change.	Test.
CalculateInterest	913	2.10	2.14	1.32	2.06	2.13
ForeignAccount	9249	1.07	2.47	1.24	1.95	1.80
BalanceCheck	2902	2.03	2.70	1.32	2.05	1.72
AccountAgreement	1364	2.45	3.69	1.34	2.33	1.77
ReportCustomer	918	2.25	3.04	1.23	2.24	1.82

CalculateInterest COBOL programs. Due to reasons of brevity, the detailed analysis is not presented in this chapter, but anonymized reports of the *CalculateInterest* and the *ForeignAccount* are available[28].

				Max Norm Good	30	0	1000	15%	90	5000
				Max Norm Average	0	-30	2000	10%	120	10000
			McCabe							
#	Cobol Program	Lines of Code (LoC)	Complexity v(G)	Maintainability Index (MI)	Check Maintainability Index	Check Miwoc	Check Volume NCLoC	Check Comments	Check McCabe	Check Metrics
1	RT00000	464	20	53.4951227	Good	Good	Good	Average	Good	Good
2	RT00100	1120	84	6.5556751	Average	Average	Good	Bad	Good	Bad
3	RT00200	1273	85	2.7461408	Average	Average	Average	Bad	Good	Bad
4	RT00400	559	26	38.7093703	Good	Good	Good	Bad	Good	Good
5	RT00800	505	38	39.2838751	Good	Good	Good	Bad	Good	Good
6	RT00900	1137	79	6.2149798	Average	Average	Good	Bad	Good	Bad
7	RT01100	467	20	45.5457803	Good	Good	Good	Bad	Good	Good
8	RT01200	542	28	40.7082519	Good	Good	Good	Bad	Good	Good
9	RT01400	1011	67	20.4825089	Average	Average	Good	Bad	Good	Average
10	RT01500	882	39	31.7499408	Good	Good	Good	Average	Good	Average
11	RT01600	1853	115	-7.6646816	Bad	Average	Average	Bad	Bad	Bad
12	RT01700	248	6	84.4787602	Good	Good	Good	Good	Good	Good
13	RT01800	360	11	60.5492274	Good	Good	Good	Bad	Good	Good

Figure 5.3: Excerpt of a detailed program analysis of the *CalculateInterest* COBOL programs

As a part of the legacy assessment, a call dependency diagram of the subsystems was generated and analyzed based upon number of incoming call (NIC) and number of outgoing calls (NOC). As a result, numerous computationally intensive COBOL programs (with high NIC and high NOC) and core libraries (with high NIC) within each subsystems were identified, following the work of Van Geet & Demeyer [277]. Such programs were later manually investigated to locate features within the subsystems. Figure 5.4 depicts an excerpt of the generated call dependency diagram of the *CalculateInterest* subsystem in which the red-circled programs (RT23N, RT23M, RT20K and RT009), for instance, could potentially be core libraries of interest.

After the legacy assessment, an inventory of high level features available within the subsystems of the payments domain was created. The inventory of the features was created by consulting the available documentation and interviewing the technical staff of the subsystems. The latter method proved to be very useful and confirms that the knowledge residing within the organization is of the utmost importance. The inventory of the high level features was then analyzed by the *Business Change Management* and *Architecture Board* to determine the priority and business value. The features were then mapped to the logical components within the logical target-architecture via the gap analysis method [172]. The mapping was performed by focused workshops conducted for each subsystem in which business analysts, application analysts, consultants, developers and lead architects discuss and finalize the mapping of each

[28] `http://goo.gl/bwqnq`

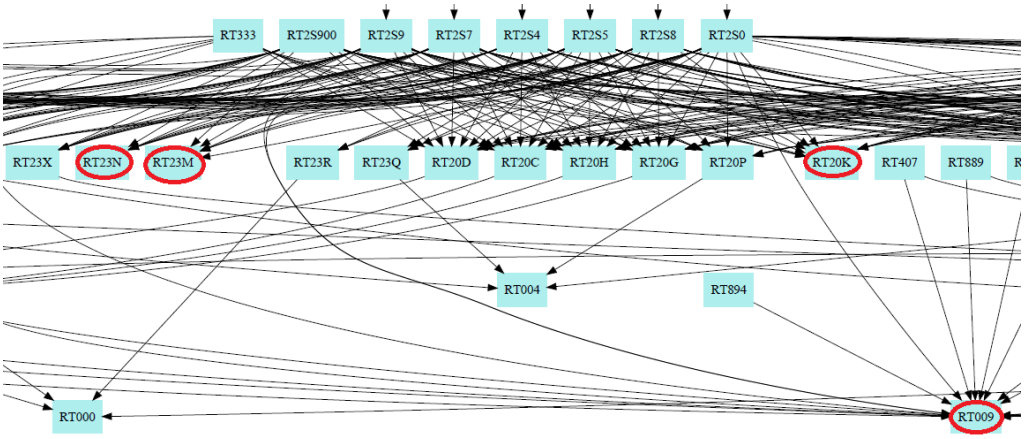

Figure 5.4: Excerpt of a call dependency diagram of the *CalculateInterest* COBOL programs

subsystem. This mapping approach was effective such that the migration team not only identified the mappings, but also the dependencies within the high level features of the subsystems. Table 5.3 depicts an excerpt of the feature mapping of the *CalculateInterest* and the *AccountAgreement* to the logical target-architecture.

Table 5.3: Excerpt of feature mapping to the logical target-architecture

High Level feature	**Target Arch. Component**	**Priority**	**Remarks**
CalculateInterest			
Register data	Bank Administration	High	–
Calculate interest	Interest	High	Merge international interest
Bank guarantee commission	Fees	Medium	–
Checkout coupon	Interest	Medium	To be included in the Interest logical component
AccountAgreement			
Opening accounts/contracts	Product Agreement	High	Merge current agreements in current account
Account management	Number Pool	High	–
Managing data rate	Product Configurator	Low	Include tariff data from other components

Benefits, Challenges and Best Practices: The "analyzing the gap" phase enabled the migration team to catalogue the existing features with the aim to maximize reuse features. In particular, the use of reverse engineering tools/techniques has facilitated understanding the current legacy assets, their technical qualities, and identifying the potential features based on call dependency diagrams. The "analyzing the gap" phase has not only been effective in identifying, prioritizing and determining the granularity of existing features, but also in determining which feature is to be reused. The legacy assessment activity contributed to identifying the technical program quality in terms of software metrics such as maintainability, module coupling, duplication, changeability. Such software metrics have been extensively used in the software evolution domain, for instance, to determine the reusability factor [248]. To better understand the individual programs within each subsystem, the program level quality metrics along with the program visualization in the form of a call dependency diagrams were generated using

reverse engineering tools. Furthermore, the interview sessions with the technical staff of the payments domain proved to be extremely important. Needless to say, intimate knowledge of the existing resources is essential to a successful migration, and necessary steps should be taken to harvest and preserve such existing knowledge.

5.4.4 Realizing the Migration

The payment domain of the bank has a heterogeneous IT infrastructure with some of the features being efficient and robust with respect to performance while others being rigid commercial off-the-self (COTS) applications. In such a scenario, relying on a single approach to realize migration is not a viable solution. Thus, the migration process made the following four explicit choices for realization:

5.4.4.1 Reuse and/or Upgrade

One of the key performance indicators of a bank is accuracy and efficient processing of voluminous financial transactions. In the payments domain, some of the features are highly robust in terms of accurate and efficient processing of transactions. Such features are either reused or upgraded based on their business value and the program quality characteristics derived in the "legacy assessment" of the "analyzing the gap" phase. For example, the *"calculate interest"* feature is reused. With regards to the *CalculateInterest* subsystem, one of the business analysts says "*The clear separation in the features of the CalculateInterest subsystem has eased our maintenance. Also, if we consider rebuilding or splitting the features then the impact will be very high– technically and economically and we are not sure if we can achieve the current performance. Thus, for the time being we decided to reuse the features of the CalculateInterest*".

5.4.4.2 Package Replacement

Numerous logical components within the logical target-architecture cannot be directly mapped to existing features of the legacy applications. Thus, some of the components in the target-architecture are being replaced by a packaged solution. The decision to replace is reached by assessing the technical program qualities and economical feasibility of the feature. One of the examples of such a replacement is within the features of the *ForeignAccount* subsystem, which in itself is a third party packaged subsystem responsible for international payments. Thus, the features of the *ForeignAccount* subsystem are replaced by a packaged solution. An application architect says "*ForeignAccount is a package software with very limited documentation and reusing its features will lead to long-term maintenance problems in the future. So we decided to replace it with a packaged solution*".

5.4.4.3 Customized Replacement

With the introduction of the Euro currency, various laws and regulations within the payment domain have changed in the European Union. The bank has to comply with such changes. One of the business consultants emphasizes the importance of the SEPA stating that "*SEPA is one of the triggers for the renewal of the whole payments infrastructure. It is also one of the main business drivers for lowering cost. Such crucial features have to be custom–built so that its maintenance and upgrade in the future will be easy for us*".

5.4.4.4 Outsourcing

The final option is to outsource an entire feature to an external party for development. This option is chosen only if outsourcing contributes to the strategic objectives of Payments (such as cost reduction) and must fulfill the requirements as formulated in the outsourcing strategy- guidelines to ensure that outsourcing is done via strategic partners and only when no other option is viable. A lead architect explains the need of outsourcing as "*Features that are of low business value and can be developed cheaply are outsourced such as card authorization and card payments. This helps us to focus on the high priority features.*"

The selection of an appropriate realization option is based on various factors such as business value of the logical component, technical quality of the legacy assets, cost of implementation and the importance of ease of upgrading/updating.

Benefits, Challenges and Best Practices: Realization of the migration is the starting point of implementing the logical components. Realization is not only about deciding which programming language is to be used, but also the associated environment such as hardware and operating system. The other factor that contributes to the success of realization is the determination of the suitable granularity of the potential candidate services. In this migration project, there were predefined guidelines provided by the program management committee on deciding which realization option to use. The guidelines were developed by considering "external vs internal development" and "adoption of existing vs new technology". Figure 5.5 depicts the realization options based on development and technology adoption criteria.

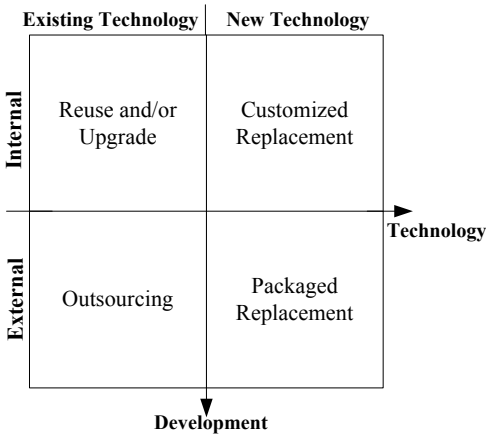

Figure 5.5: Realization Choices

A crucial criterion to determine the realization option was the business value and priority of the logical component. For instance, central to NedBank's business are the calculation of interest, commission and cost calculation features. The logical components encompassing these features are reused and/or upgraded from the existing ones. Upon deciding to reuse and/or upgrade, the technical qualities of the features are examined to estimate the migration time. It was not always simple to reuse or upgrade, particularly for third party packaged solutions that are not updated or supported by the vendor anymore. For instance, the vendor who developed COBOL running in HP Tandem Nonstop went through various mergers and acquisitions such that the infrastructure is hardly updated and maintained. In such cases,

a suitable packaged replacement is preferred. Any new logical component with high business value falls under the "Customized Replacement" option. A low priority logical component is outsourced to a third party upon strictly fulfilling the "outsourcing" criteria.

5.5 Lesson Learned

5.5.1 Implications of Reverse Engineering Techniques

The reverse engineering techniques that are used in this migration project had a significant role in finding the facts of the current legacy programs and subsystems. In particular, the use of such techniques in the "analyzing the gap" phase to obtain various metrics and call dependency graphs not only facilitated the creation of an inventory of the current assets, but also identified computationally intensive COBOL programs. In addition to creating an inventory of the current assets, using reverse engineering techniques has the following two implications:

(i) *Assisting in Selecting the Realization Approach*: The use of reverse engineering techniques has strongly facilitated the selection of the realization approaches in the migration process. The migration team used the program quality metrics generated by the reverse engineering tools to estimate the complexity of the programs. For example, in Figure 5.3, the COBOL program RT01600 has high McCabe complexity and a negative maintainability index (MI). Hence, *RT01600* is a potential candidate of replacement unless it has a low business priority. Similarly, programs with *good* and *average* index have potential for reuse. In case of identifying candidate COBOL programs for services, the generation of call dependency graphs of the subsystems was helpful. For instance, based on the work of Van Geet et al. [277], the red-circled programs (RT23N, RT23M, RT20K and RT009) of Figure 5.4, were COBOL programs of interest as they have high number of incoming calls (NIC). Thus, upon generating and analyzing the call dependency graph those programs were investigated by the existing programmers to identify their functionalities.

(ii) *Knowledge Harvesting*: Most of the documentation of the subsystems was either outdated or incomplete. With the results of the reverse engineering techniques, a considerable amount of new information about the programs was identified that was even unknown to the current maintainers of the subsystems. For instance, 599 out of 21085 copybooks are not used by the *"CalculateInterest"* subsystem. This finding was a surprise to the current maintenance team of the subsystem. Additionally, the flow graphs were generated to understand the overall flow of the programs within the subsystem. These artifacts helped to update the documentation.

5.5.2 Adopting a Pragmatic Realization Approach

In the realization phase of the migration process, a pragmatic approach to executing the migration is adopted in which the choices are based on various factors such as business value, business priority, and the technical qualities of the features. The initial choice of reusing the existing functionalities is one of the notable benefits claimed by the proponents of SOA for leveraging existing assets. However, in a large scale legacy application, reuse is not always feasible. Therefore, the migration process of NedBank suitably adapted other possible realization methods for a successful migration. Additionally, for large scale legacy applications that include heterogeneous IT infrastructures (diverse programming languages,

various hardware and operating systems) there is no silver bullet solution to realize the migration process. For a successful migration process, any approach can contribute to the success of the migration, provided that the approach is well defined, and suited for the enterprise.

5.5.3 Emphasizing Organizational and Business Perspectives

Migration from legacy to a SOA environment is not only a technical endeavor, but also involves significant issues from the organizational and business perspective. Particularly in the case of legacy systems having no or outdated documentation, early involvement of existing technical staff in the migration process is proven to be useful. The involvement of the technical staff facilitates the knowledge transfer to the migration team. The synergy between the technical staff and the migration team that was observed in this migration process was one of the key factors contributing towards the success of the migration. Equally important is the focus of the *"Program Management Committee"* in the business-IT alignment that facilitated the migration to achieve the business goals. An important lesson learned is that technical staff tends to resist change because they fear that their expertise and professional experience with legacy systems may become redundant due to the introduction of SOA. Therefore, it is important to involve the technical staff from the start of the migration process and provide necessary training to adapt to new technology. In the current migration process, the formation of various teams under the "*Program Management Committee*" has actively involved the technical staff whose knowledge about the legacy systems have proven to be of significant importance.

5.5.4 Harvesting Knowledge to Prevent Knowledge Erosion

Apart from available documentation, existing knowledge in the form of skills and experience within the technical staff is one of the most important assets. Over the years, such knowledge and skills become scarce resulting in knowledge erosion due to factors such as ageing, and staff changing jobs. Hence, suitable strategies such as conducting and archiving interviews, and initiating knowledge transfer programs via training should be undertaken to harvest and preserve such existing knowledge. In the migration process of NedBank, the involvement of the technical staff in the migration process has significantly helped in knowledge harvesting and preservation while creating inventories of the high level functionalities of the subsystems. Some of technical staff were interviewed and focused training programs were organized to facilitate knowledge transfer. Additionally, the existing documentation was updated or created in case no documentation existed.

5.6 Conclusion

In this chapter, we presented the findings of a case study of a large scale legacy to SOA migration process within the payments domain of a Dutch financial institution. The chapter presented the business drivers that initiated the migration, and describes a 4-phase migration process. The migration process equally focuses on the technical and the business & organizational aspects of the migration. The migration process starts by forming a "*Program Management Committee"* for proper governance. Then, a logical-target architecture is developed based on the concept of componentization. The business components within the logical target-architecture are aligned to support the business goals. The logical target-architecture is then mapped to the existing legacy features using a gap analysis method. Such a gap analysis suitably supports the potential of reusing the legacy features without significant changes to the

legacy systems itself– one of the key promises of the SOA. The migration is then realized following the pragmatic realization options of the migration process.

The chapter presented an industrial report detailing how reverse engineering techniques were employed to facilitate a large scale legacy to SOA migration process. It illustrated the pain of the practicalities and under-emphasized aspects of a large scale migration project, and should be considered a call-to-action for computer scientists to study the project environment, both from a business and organizational point of view, as much as they study the technical aspects of migration.

As to future research, we aim to exploit model-driven modernization approaches to extract features from the subsystems. Another interesting research area is to investigate how to automatically translate legacy applications to a modern language.

Chapter 6

How Do Professionals Perceive Legacy Systems and Software Modernization?

Abstract

Existing research in legacy system modernization has traditionally focused on technical challenges, and takes the standpoint that legacy systems are obsolete, yet crucial for an organization's operation. Nonetheless, it remains unclear whether practitioners in the industry also share this perception. This chapter describes the outcome of an exploratory study in which 26 industrial practitioners were interviewed on what makes a software system a legacy system, what the main drivers are that lead to the modernization of such systems, and what challenges are faced during the modernization process. The findings of the interviews have been validated by means of a survey with 198 respondents. The results show that practitioners value their legacy systems highly, the challenges they face are not just technical, but also include business aspects.

6.1 Introduction

A legacy system is any business critical software system that significantly resists modification and their failure can have a serious impact on the business [22, 32]. Software modernization is the process of evolving existing software systems by replacing, re-developing, reusing, or migrating the software components and platforms, when traditional maintenance practices can no longer achieve the desired system properties. The primary aim of software modernization is to reduce maintenance cost and increase flexibility. After three decades of legacy system modernization research, it may come as a surprise that many legacy systems are still in daily operation. Most of these systems were developed years ago, and have continued to evolve. New requirements have led to frequent modifications of these legacy systems resulting in unstructured source code that is difficult to maintain. Furthermore, knowledge about the legacy systems is scarce as original programmers leave the company or retire, and documentation is usually lacking [283].

These issues have been recognized by the software engineering community and a plethora of legacy system modernization approaches have been proposed (cf. Section 6.5). Despite the problems introduced by legacy systems, and the acclaimed benefits of legacy system modernization, technology consulting firms estimate that 180-200 billion lines of legacy code are still in active use [11, 265, 283]. This fact has triggered us to investigate how legacy systems and their modernization are perceived in industry. Specifically, we aim to identify the perceived benefits of legacy systems, the main drivers for legacy system modernization, and the challenges professionals face with the modernization of legacy systems.

We have set up our research as an exploratory study, aiming to discover new perspectives and insights about legacy systems in the industry. Accordingly, we adopted the grounded theory approach [103], an increasingly popular method to conduct empirical software engineering research [3]. We designed, conducted, and analyzed semi-structured interviews with 26 industrial practitioners. The findings of the interviews were then validated through a separate structured survey, in which 198 professionals expressed (dis)agreement with the results of the interview.

Our study makes the following contributions:

- We document the industrial perception of legacy systems and their modernization.
- We identify the perceived benefits of the legacy systems, drivers of modernization, and challenges that the industry faces during modernization.
- We report the differences in perception of legacy systems between the industry and academia.

The chapter is structured as follows. In Section 6.2, we describe our study design. In Section 6.3 we detail the key findings. Subsequently, in Section 6.4 we present the confirmation/contradiction of the findings and address threats to the validity. In Section 6.5 we discuss related work. Finally, in Section 6.6 we conclude our research and propose future research directions.

6.2 Study Design

We employed two different research techniques to conduct our study. We started with semi-structured in-depth interviews, a qualitative technique, to identify how legacy systems and their modernization are perceived in industry. We followed a Grounded Theory (GT) approach [103] to analyze 26 in-depth semi-structured interviews. To further validate the findings of the GT, we used a separate structured survey– a quantitative technique. In an empirical study such as this, the use of multiple research

techniques (qualitative and quantitative techniques in our case) increases the confidence that the results are reliable.

GT is an explorative research method that aims at discovering new perspectives and insights, rather than confirming existing ones [103]. We started with a series of interviews conducted with 26 practitioners (identified as P1-P26 in this chapter), each lasting 1-2 hours. The informants were selected based on the two criteria that they have experience with legacy systems, and with legacy system modernization projects. The informants were identified opportunistically via industrial collaborators, followed by snowball sampling [155], in which the first generation informants helped us to identify other informants who fulfilled the criteria. In total, 23 interview sessions were performed. In three of the interview sessions, the interview was conducted with two informants from the same organization. Furthermore, two practitioners were from the same organization and this reduced the sample size of the participating organizations to 22. The sample is arguably broad enough to well represent the software engineering professionals. Moreover, the sample shows diversity in the industry domain, and the roles and experiences of the participants. Table 6.1 provides the details regarding the domain of each informant's company. With respect to size, the companies range from small consulting firms to global corporations such as IBM, Deloitte, and Capgemini. The variation among the informants' roles is also broad, ranging from software developers to system analysts, consultants, software architects, business architects, research and development managers, and Chief Information Officers (CIOs). The experiences of the informants range from 5 years to 43 years, with 19 years as an average experience of the sample and cumulatively, the informants have 490 years of experience in information technology. Additionally, the interview data totaled more than 25 hours of recorded data. We conducted semi-structured face-to-face interviews in

Table 6.1: Details of the informants

Participants	**Domain**
P2, P11, P12, P20, P21	Information Technology Services
P1, P15, P17, P22	Financial Service Providers
P4, P5, P25, P26	Government Organizations
P7, P8, P18, P19	Software Development Company
P6, P10, P24	Consulting Company
P3	Aviation Industry
P9	Manpower (Security) Company
P16	Flower Auction Company
P13	Food & Dairy
P23	Machinery Production
P14	Poultry

English, which were recorded. Prior to the interview session, informants were provided with an interview protocol that consisted of sample questions to be discussed during the interview sessions. The interview session has three categories of questions: about characteristics of legacy systems, drivers for legacy system modernization, and challenges faced during modernization. Afterwards, the recorded interviews were transcribed and each interview transcript was analyzed through *coding*: a process of breaking up the interviews into smaller coherent units, and adding codes to these units. Subsequently, a process of writing down narratives that explain the ideas of the evolving theory, known as *memoing*, was used to develop the coding. These coherent units represent key characteristics of the interview being analyzed. Later, the codes were organized into *concepts*, which in turn were grouped into *categories*. As the interviews progressively provided answers similar to the earlier ones, a *saturation stage* [3] was observed. To confirm the saturation stage, we conducted two more interviews and found that the analysis resulted

in similar responses to the earlier ones. We used Nvivo 10[29] as an instrumentation tool to facilitate the interview analysis process.

In the second and final phase of this research, we adopted a structured survey as a data triangulation process in order to validate the findings of the interview results. A data triangulation process– a method that uses more than one data source, or collects the same data at different occasions– is typically used to increase (decrease) confidence in a finding by providing confirming (contradictory) evidence [80, 233] and helps to improve validity of the findings of an empirical study such as this [107]. The survey was public and announced via mailing lists, social media such as Twitter, Linkedin, Facebook, and via personal referrals. In the end, 198 participants responded to the survey, originating from more than 30 different countries. We performed sampling to exclude responses having no experience with legacy systems. In total, 22 out of 198 were excluded, leaving 176 valid responses. The respondents have an average experience of 13.5 years with legacy systems. Developers formed the largest group of participants (22%), followed by IT Managers (14%) and researchers (12%); they come from various domains such as software development companies (28%), consulting companies (21%), service providers (11%), and financial institutions (9%).

In the subsequent sections, we present the results of our research, categorized over the four themes: legacy systems, perceived benefits of legacy systems, drivers of legacy system modernization, and challenges of legacy system modernization. For each theme, we provide relevant "*quotes*" from the practitioners and the results of the survey.

For further details of the research, we refer to our technical report [17], in which we provide additional data to support our analysis. In the technical report, we provide the coding process, consisting of 44 different codes within 19 categories. For each code, we give a short working description. In addition, we detail key quotes of all the informants (P1–P26) to illustrate how the codes are derived. Furthermore, the technical report also presents the survey with final results along with response counts and percentages. The report also describes statistical analysis, particularly, multiple regression analysis to identify the relationships among the drivers of legacy system modernization and use the results in this chapter to support our findings. To increase transparency of the data analysis process, we publicly provide anonymous interview transcripts, the Nvivo 10 project file of interview analysis, and the survey data in excel format[30].

6.3 Findings

In the following subsections, we present the results of the analysis of the interview sessions and the results of the survey. For each interview, we started asking questions regarding their personal information and experiences with legacy systems. Subsequently, the informant was asked to give a definition for a legacy system based on his/her opinion and the findings are discussed in subsection 6.3.1. The discussion then proceeded by asking questions about perceived benefits of legacy systems; these findings are presented in subsection 6.3.2. Furthermore, the informants were asked about the issues associated with legacy systems, which are the drivers for legacy system modernization and such drivers are discussed in subsection 6.3.3. Moreover, the discussion proceeded to investigate the challenges faced during legacy system modernization; these findings are presented in subsection 6.3.4.

[29] `www.qsrinternational.com/`
[30] `http://www.servicifi.org`

6.3.1 Legacy Systems

The interview sessions started by asking a definition of a legacy system from the informants. Most of the informants agreed that legacy systems are "old" systems. Additionally, the informants pointed out that legacy systems are "core" systems that have been proven to work correctly in a production environment for decades. P1 said: *"Most of the legacy systems are older than 20-30 years..[] Most of the systems of the legacy systems are the core system".* P11 agreed with P1 by stating: *"It [Legacy system] is an old system; ... a lot of legacy system is the core system".* Interestingly, most of the informants related legacy systems as systems that do not fit with the future IT strategy of the organization. P19 expressed this as *"My definition of a legacy system is systems and technologies that do not belong to your strategic technology goals".*

To investigate further, the informants were asked if programming language is a determining factor for a system being legacy, we obtained a mixed opinion. More than half of the informants do not agree that the programming language is a determining factor for a system to be legacy, while the rest were in agreement. Such a mixed opinion is also observed from the results of the survey. Around 50% of the respondents agreed that the programming languages do determine if a system is legacy. The top five languages that these informants indicated as legacy are depicted in Figure 6.1.

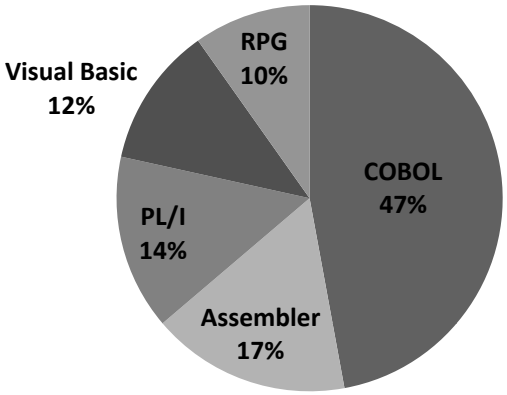

Figure 6.1: Legacy languages by as per the informants

6.3.2 Perceived Benefits of Legacy Systems

6.3.2.1 Business Critical

The interviews reveal that the practitioners view legacy systems as business critical. As per the informants, legacy systems are the core systems of the organizations and their failure can result in serious consequences for daily business. P11 argued that legacy systems are significant to business, taking an example of a financial organization. He expressed his opinion as *"It is very useful and has a business impact still and generates a lot of revenue for banking and their clients".* P14, a 20 years experienced IT director of a poultry company, simply stated that *"Legacy for me is let's say business critical".* P24 explained that legacy systems are old and business critical as: *"Because they [Legacy systems] have been there for 30 years, so they really are the foundation for the survival of the organization".* These opinions clearly indicate that legacy systems support core business processes of a business and their failure can have significant impact on an organization.

The results of the survey also strongly support that legacy systems are business critical. 76.7% (cf. Figure 6.2) of the respondents indicated that legacy systems are typically business critical. One of the respondents with 28 years of experience gives his strong opinion through an open question of the survey as: "*By definition a legacy system is business critical. A system that is old and obsolete and is not business critical would never reach the status of legacy".*

6.3.2.2 Proven Technology

Informants have frequently expressed that legacy systems are old and have been developed, tested and have been in production environment for years. Hence, it is an indication that legacy systems are of a proven technology that still remain as the core systems of many organizations. P17 explained the proven technology characteristics with an example of AS400 as: *"Most of the time it's [a] proven technology. AS400 is stable, it always works [24/7] and is quite good. So, it's proven technology and normally it's stable, [which] is a good thing"..* In our discussion with P4, he expressed: *"Proven technology is often the reason why they are still in use".* P11 associated "Availability" with the "Proven technology" and said: *"They are available, and they are more less 24/7 up and running".*

In the survey, 52.8% (cf. Figure 6.2) of the respondents indicated that legacy systems are proven technology.

6.3.2.3 Reliable Systems

The definition of reliability is adopted from ISO/IEC 25010 standard [262] as "degree to which a system, product or component performs specified functions under specified conditions for a specified period of time". This definition was provided to the participants of interview and survey. Based on the interviews, the practitioners indicated that legacy systems are reliable systems, primarily, because they are running in a production environment for decades. P18 supports this statement and said: *"It is reliable...people know how to use it. All the problems have been fixed over the years from it. So technical problems are usually not there".* Other informants also share P18's opinion as P1 stated: *"The system has been around for a long time and has been tuned to stability, robustness, availability and so on. So they're well performing and stable. Functionalities [Quality attributes] that count are stability, robustness, reliability and availability of this system".* In general, legacy systems are perceived as reliable systems in industry because of the fact that they have been in production for years and possible bugs and errors are already fixed in the past. P12's opinion is also in-line with the others and he said: *"They have been around for many years and during this period they have been stabilized".*

The findings of the survey are in agreement with the opinion of the practitioners regarding legacy systems being reliable systems. 52.3% (cf. Figure 6.2) of the respondents indicated that legacy systems are reliable systems.

6.3.2.4 Performance

With regards to performance, the interview revealed a mixed opinion. Some of the informants (e.g., P3, P10, P24) strongly emphasize on the high performance characteristics of legacy systems. P3 expressed: *"It's [Performance] enormous. Enormous quick. So the old system itself didn't have [performance issues]...I don't think the performance is a problem".* P24 agreed with the opinion of P3 and further added:

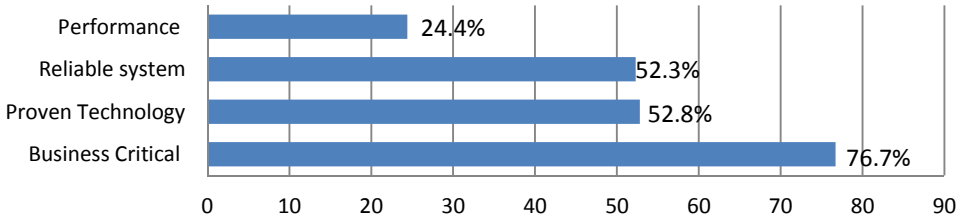

Figure 6.2: Survey responses for perceived benefits of the legacy systems

"I think ten thousand people are doing airlines booking with this little processing power. So performance is never an issue in legacy system, at least I've never seen it". In contrast, P9 and P18 perceive that legacy systems do not have high performance. P9, with reference to a real time system that is being used in his Manpower (Security) Company, referred: *"It [performance] is very poor".* P18, a software developer, agreed with P9 *"The performance is not really good".* There are informants (P13, P14, P9, P16, P26) who consider that performance of legacy systems as *"Ok or enough".*

In the survey, 24.4% (cf. Figure 6.2) of the respondents indicated that legacy systems are of (high) performance, which is comparatively low with respect to other characteristics. One of the plausible reasons could be the fact that legacy systems get their job done and operate at good/enough (satisfactory) speeds in many cases.

6.3.3 Drivers for Legacy System Modernization

The informants not only expressed their opinion about perceived benefits of the legacy systems, but also explained about issues related to the legacy systems.

6.3.3.1 To Remain Agile to Change

In current dynamic business environment, organizations have to quickly adapt to various changes, including intra-organizational changes, changes in laws and regulations, changes in business collaboration (mergers and acquisitions), and faster time-to-market [280]. Despite the fact that legacy systems are business critical and reliable systems, the informants expressed that legacy systems are inflexible to support changing business requirements. P20 explained how the inflexibility to adapt to new changes has enabled him to modernize legacy systems. He said: *"Other point is that my costumer wants flexibility, and a short time-to-market, then you have to get rid of your legacy. Because legacy is rigid, and it is not flexible".* P25, an IT architect at the tax office, agreed with P20 and said: *"We had a lot of systems before, and they were built in CICS, COBOL, DB2 and were not flexible. So they needed to be modernized to get more flexibility".* The informants identified faster time-to-market as one of the drivers for legacy system modernization because legacy systems are rigid, which increases the difficulty in promptly addressing the market demands. P22 expressed this as: *"We need a faster time-to-market, and we are not able to do that in COBOL environment".*

The findings of the survey are in-line with the opinions expressed by the interview respondents, in which respondents have strongly expressed that the need of flexibility to comply with the changing business requirements and rapidly evolving future technologies do cause organizations to initiate legacy system modernization projects. For the "Flexibility to change" driver, we depict the contributing drivers

(i.e., "Become flexible to change", "Create business opportunities via mergers/acquisitions", and "Faster time-to-market") in Figure 6.3, as presented in the online survey. It is interesting to observe that 85% (38.1% for "Strong" and 46.9% for "Very Strong") of the respondents of the survey have indicated "Become flexible to change" a major driver. Similar observation can be noticed with "Faster time-to-market", for which 71.1% of the respondents indicated as a driver of legacy system modernization (39.6% for "Strong" and 31.5% for "Very Strong"). But in contrast, the "Create business opportunities via mergers/acquisitions" driver is indicated as a medium driver by 43.5% of the respondents.

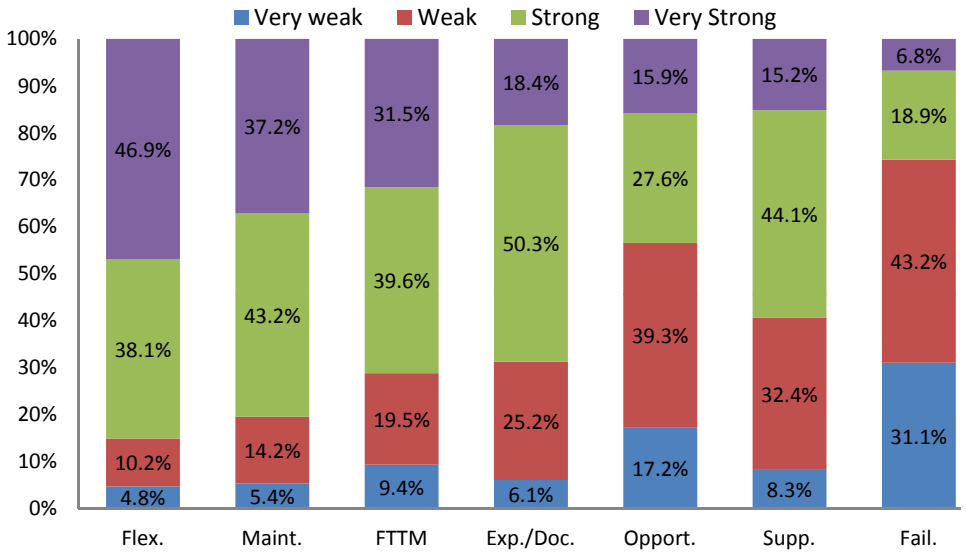

Figure 6.3: Drivers for Legacy System Modernization (*Legends: Flex.:–Become flexible to change; Maint.:–High cost of maintenance; FTTM:–Faster time-to-market; Exp./Doc.:–Lack of experts/documentation; Opport.:–Create business opportunities via mergers/acquisitions; Supp.:–Lack of suppliers/vendors; Fail.:–Prone to failure*)

6.3.3.2 High Maintenance Cost

The interviews revealed that one of the significant drivers of legacy system modernization is high maintenance cost. More than half of the informants strongly argued that the cost involved in maintaining the legacy systems is high. One of the most common motivations to modernize the legacy systems is to reduce maintenance cost. P17 explained: *"[With legacy systems] the cost is getting higher because maintenance is getting more expensive, [then] maybe you should think of modernization"*. Often participants such as P12, P13, P14 argued that maintenance cost could be lowered if the legacy systems were modernized to standard software products. For instance, P12 mentioned: *"But if you can move to standard product, then it could be usually an advantage because the maintenance cost for standard product is usually lower"*. P14 expressed: *"if you look at the [maintenance] cost, I'm quite sure I can run a similar environment against lower cost, if I would use a standardized product. Let's say state of the art ERP environment...because I don't need somebody to maintain"*.

From the survey, we observed that respondents (80.4%) strongly indicated (43.2% for "Strong" and 37.2% for "Very Strong") that the high maintenance cost of legacy systems is one of the major drivers behind legacy system modernization (cf. Figure 6.3).

6.3.3.3 Lack of Knowledge

As per the participants, one of the most significant drivers of legacy system modernization is a lack of knowledge, particulary scarcity of experts, unavailability of (up-to-date) documentation and limited number of suppliers/vendors of the legacy system. More than 90% (24/26) of the informants pointed out that lack of resources of the legacy systems causes organizations to modernize their legacy systems. P10 argued that: *"I think the big problem is that you can't find people to understand them [legacy systems] and understand the technology".* P11 not only indicated lack of experts, but also pointed out the outdated documentation and explained: *"The issue is that there is less knowledgeable people [experts] are available in the organization because the [knowledgeable] people of the system are already gone [left job]. The [other] characteristic of legacy systems is the lack of documentation".* Some informants (P1, P2) expressed a fear that in future the scarcity of the experts of legacy systems will be a severe problem. As to the limited number of suppliers/vendors, 12/26 informants indicated it as a problem. P17 expressed: *"There's no patches. If suppliers stop their product, organization needs to find another way to keep supporting their system".*

In Figure 6.3, we depict the "Lack of knowledge" driver comprising of "Lack of experts/documentation", and "Limited suppliers/vendors", as presented in the online survey. Almost 60% (44.1% for "Strong" and 15.2% for "Very Strong") of the respondents indicated that "Lack of suppliers/vendors" is a strong driver for modernization. Furthermore, 68.7% of the respondents indicated "Lack of experts/documentation" as a "Strong" (50.3%) and "Very Strong" (18.4%) driver for modernization (cf. Figure 6.3).

6.3.3.4 Prone to Failures

Although "Reliable system" is one of the perceived benefits of the legacy systems, there is a fear shared by most informants that the legacy system might fail due to lack of experts and suppliers/vendors. Informants have identified that "Prone to failure" as a driver of legacy system modernization. Legacy systems are "business critical" and organizations cannot afford their legacy systems to fail. P13 expressed this as: *"We have an old ERP system, old almost 10 years old. And it drives the production in the plant and also the logistic and warehouse and also the order towards the customers. If that system stops, the plant stops, the warehouse stops".* Often, the informants indicated that risk of failure can be the result of other drivers such as limited suppliers/vendors, and lack of experts. Such opinion is shared by P2 as: *"So when your environment [legacy systems environment] runs out of support then it is really dying and if that's true then you are already late".*

In comparison with other drivers, "Prone to failures" is indicated as the weakest driver by 25.7% of the respondents (cf. Figure 6.3). One of the plausible reasons for this could be the fact that the "Prone to failure" issue is countered by the "Reliable system" perceived benefit of the legacy systems.

6.3.4 Challenges of Legacy System Modernization

Upon identifying the drivers of legacy system modernization, the study focused on identifying what challenges are faced by practitioners while modernizing legacy systems. Based on the interviews, the following challenges were identified.

6.3.4.1 Time Constraints to Finish Modernization

The interviews revealed that finishing any legacy system modernization on time is the biggest challenge. The timing constraint is influenced by other existing issues of legacy systems such as scarcity of resources such as experts and documentation. P1 expressed the challenge as *"They run out of budget.. they run slightly out the time. [...] that's mainly to do with scarcity of people on the legacy system"*. P13 also agreed with P1 and stated that the time constraint to finish modernization project is influenced by lack of resources. He expressed this as: *"Your biggest problem is an availability of resources [documentation and experts] and availability of money and [to some] extent availability of time"*. Due to the long duration of the legacy system modernization projects, many earlier plans regarding the modernization change and this further delays the projects. P7 shared his experience as: *"Legacy modernization project lasts too long. We plan modernization for 3 years, and after 5 years we stop the whole modernization, and start it over"*. Some of the respondents indicated that lack of resources is also one of the factors of not finishing legacy system modernization projects on time.

Interestingly, by far the largest percentage (63.6%) of the respondents indicated that legacy system modernization projects frequently face time constraints to finish (31.8% for "Challenging" and 31.8% for "Very Challenging").

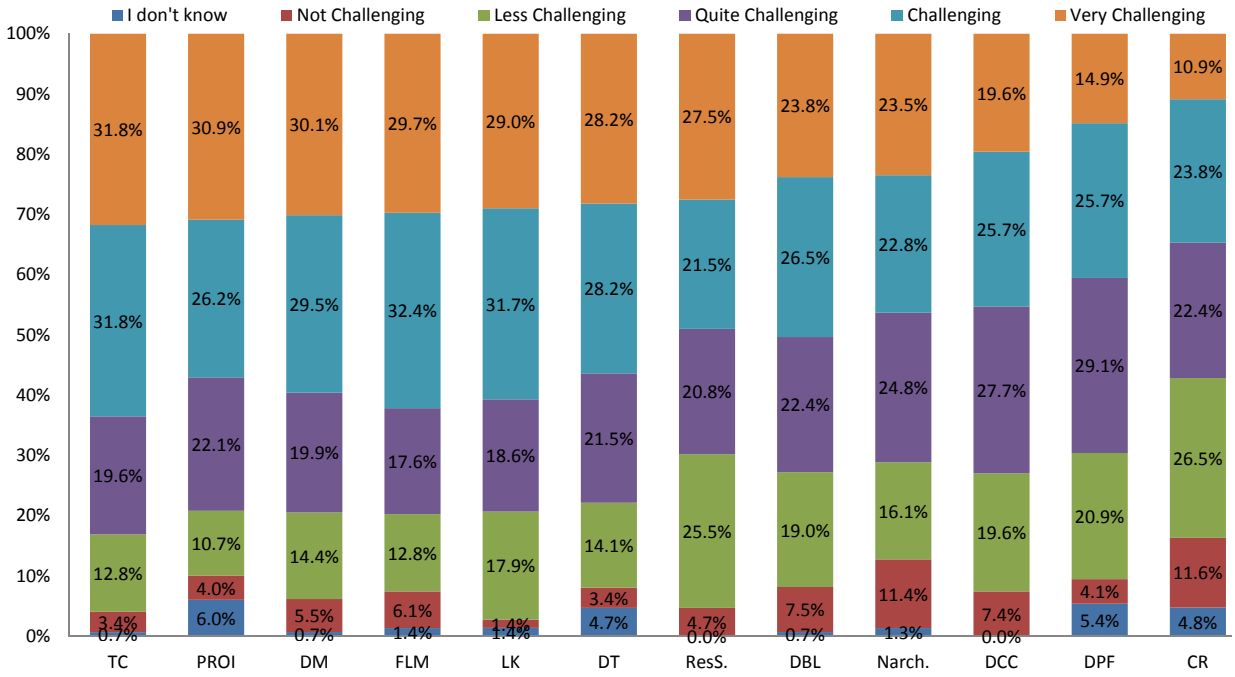

Figure 6.4: Challenges of Legacy System Modernization (*Legends: TC:–Time constraint to finish modernization; PROI:–Predicting ROI; DM:–Data Migration; FLM:–Funding modernization project; LK:–Lack of knowledge; DT:–Difficult to test; ResS.:–Resistance from staff; DBL:–Difficult to extract business logic; Narch.:–Non-evolvable system architecture; DCC:–Difficult to communicate the consequences; DPF:–Difficult to prioritize the functionality; CR:–Cultural resistance from organization)*

6.3.4.2 Data Migration

The informants also revealed that data migration in legacy system modernization project is also challenging. P11, who was conducting a modernization project in an insurance company, stressed the importance of the data migration as: *"The main risk in modernization is that the data migration, which cannot be done perfectly. Errors are made and you have some risk that your new system is disturbed after modernization".* Some of the informants indicated that the difficulty of data migration is due to the old databases that the legacy systems use. P17 explained this as: *"Data migration is really difficult because legacy system doesn't support modern databases or doesn't have relation database model".* To mitigate the risk of legacy system modernization project failure, P14 suggested planning for data migration upfront. He said: *"If you are doing migration of your legacy, you have to prepare a good data migration strategy".*

59.6% of the respondents of the survey agreed that data migration is one of the challenges of legacy system modernization projects in industry (cf. Figure 6.4).

6.3.4.3 Complex System Architecture

Dealing with the complex system architecture of legacy systems is one of the important challenges that the practitioners face while modernizing legacy systems. In the interviews, the informants used "Poor architecture", "Monolithic architecture" or "Ill–designed" to express complex architecture. P1 expressed his opinion about system architecture as: *"If you have ill-designed legacy system or old fashioned design as a monolith system, that might still be a challenge".* The informants frequently expressed that the complex architecture is a result of ad-hoc maintenance/upgrade of the past. P9 explained this as: *"Not only the development part, we have to think about the architecture of the system also. Because if people are bumping against architecture, changing the earlier architecture".* P2 further illustrated the challenge as: *"And what you also find is that one application is developed that it uses also the database of another one. And it's not through a normal interface but it's via back door to get some data over there".* P18 said: *"Because the program is really hard-coded, it is not configurable that makes modernization difficult".* Furthermore, the result of such complex architecture has an impact on testing during modernization. P12 argued that testing is challenging during modernization because: *"It can be difficult to extract all the use cases to test. It also requires a lot of work to compare functionality from legacy system to the new application. It can be time consuming and difficult work".*

In Figure 6.4, we represent the two contributing challenges of "Complex system architecture" (i.e., "Non-evolvable architecture", and "Difficult to test") individually, as presented in the online survey. Cumulatively, "Non-evolvable architecture" is perceived as a serious challenge by 46.3% of the respondents while 56.4% expressed the same for "Difficult to test".

6.3.4.4 Lack of Knowledge

The informants from the interview perceived lack of knowledge of the legacy systems as one of the important challenges. P11 explained this as: *"It is complex, because you have a lot of different expertise needed to modernize such as people who understand database environment and operating systems, middleware, enterpriser services bus of architecture. You also need people who understand business functionality to transform business functions from legacy systems to another system".* Additionally, lack of knowledge, particularly documentation can be a big risk in legacy system modernization as P20 said: *"If documen-*

tation is lacking then it is a bigger risk to migrate. Because you don't know what is going on in the old system and the risk to migrate is bigger". Nevertheless, P18 indicated the use of reverse engineering techniques to mitigate the lack of (up-to-date) documentation challenge. He explained *"For the old systems, we reverse engineer the old system and document them well so we know what the requirements are and can help to build new system".*

As per survey results, 60.7% of the respondents have indicated that lack of knowledge is either "Challenging" or "Very Challenging" for legacy system modernization.

6.3.4.5 Difficult to Extract & Prioritize Business Logic

As an effect of lack of knowledge, identifying business logic within legacy systems becomes more challenging. The informants revealed that extracting business logic is one of the challenges in legacy system modernization. P20 expressed this as *"To extract all the rules and details in there [legacy systems] is really difficult".* P12 expressed his experience as: *"The company or the project team has to extract exactly the internal functionality of this legacy application. It can be difficult to extract, to document, and to implement it [business logic] properly".* P12 further suggested using automated techniques to initially extract diagrams to better understand the business logic. He explained this as: *"So I think it [business rules] can be really helpful to provide insight into the internal working of the system, to extract it to a human readable diagrams or documentation".* Additionally, P18 said: *"Yes, you can only do that if you know exactly what the thing does. That is [business logic extraction] is the hard part".* Within the critical business processes supported by the legacy systems, prioritizing the extraction of such business logic is also equally important.

Note that, we depict the "Difficult to extract & prioritize business logic" challenge comprising of "Difficult to extract business logic", and "Difficult to prioritize the functionality", as presented in the online survey. As per the results of the survey (cf. Figure 6.4), 50.3% of the respondents agree that extracting business logic from the legacy systems is an obstacle (26.5% for "Challenging" and "23.8% for "Very Challenging") and 40.6% agree that prioritizing the functionality (25.7% for "Challenging" and "14.9% for "Very Challenging").

6.3.4.6 Resistance from Organization

Resistance to change from the organization (including users and technical staff of legacy systems) towards a new technology is a well-documented phenomenon as they fear that their expertise and professional experience with legacy systems may become redundant due to modernization. Surprisingly, not only is there resistance from technical staff, but the informants also reported that cultural resistance to adapt to new technology is another challenge in legacy system modernization. P12 explained the resistance from staff as: *"Sometimes they see legacy systems as their baby and they tend to know every aspect of it. Sometimes it is difficult to work with them while modernizing because they might not share their knowledge".* Most of the informants observed that the resistance of the existing technical staff is due to job security. P6 explained this as: *"Because what is our need if we have a new system, which is working not with COBOL. Who is gonna [going to] need me anymore, so they ditch me after it [modernization] is done. So, why should I cooperate?"* Regarding cultural resistance, informants identified that adapting to new technology is difficult for the staff. P11 strongly emphasized that cultural adaption is one of the serious challenges. He stated: *"Sometimes people do not like changes. Not only in the business*

organizations, but also in IT organization. So you need to persuade them for the need of transformation [modernization]".

For the "Resistance from organization" challenge, we depict the challenges individually (i.e., "Resistance from staff", and "Cultural resistance of organization") in Figure 6.4 as presented in the online survey. 49% of the respondents indicated that "Resistance from staff" is a challenge (21.5% for "Challenging" and 27.5% for "Very Challenging"). Similarly, "Cultural resistance from organization" is indicated by 34.7% of the respondents with 23.8% for "Challenging" and 10.9% for "Very challenging".

6.3.4.7 Addressing Soft Factors of Modernization

In this study, we adopted the concept of soft factors in modernization from Murer et al. [196] that considers non-technological factors such as people, communication, and business values of organization. The practitioners identified three soft factors as challenges. First, communicating the reasons/consequences of legacy system modernization, followed by securing funding from the top management. Often the informants indicated that the earlier challenge has an implication on the latter. P10, a legacy system modernization consultant, expressed his view as: *"I think top management doesn't understand the issue and they don't give budget for it [legacy modernization]".* Additionally, the informants indicated that communicating the consequences of the modernization is also a challenge that is often centered on the return on investment (ROI). P20 explained the difficulty of predicating ROI as: *"They [Top management] are always looking for a short term Return on Investment. Once you put the money in, they want to earn it back".* An effective way to convince the top management is to identify appropriate business cases and explain the need/consequences of legacy system modernization. P19 explained the need to use business cases as: *"You have to somehow come up with the business case that says what is my current cost, what is the cost of migration, what the new total cost of ownership, and that you have to predict the Return of Investment. A business case can have soft components like improve maintenance or improve performance because they represent business value".*

For the "Addressing soft factors of modernization" challenge, we depict the contributing challenges individually (i.e., "Predicting ROI of modernization", "Difficult to communicate consequences of modernization", and "Funding modernization projects") in Figure 6.4 as presented in the online survey. 57.1% of the respondents indicated that "Predicting ROI of modernization" is a challenge (26.2% for "Challenging" and 30.9% for "Very Challenging"). Similarly, "Difficult to communicate consequences of modernization" is indicated as a challenge by 45.3% of the respondents with 25.7% for "Challenging" and 19.6% for "Very challenging". 62.1% of the respondents expressed that securing funding for modernization is a challenge (32.4% for "Challenging" and 29.7% for "Very Challenging").

6.4 Discussion

A number of pervasive findings have emerged from the current research regarding legacy systems and their modernization, many of which are new and surprising while some resonate with the findings of other researchers. This section discusses the confirmation/contradiction of the findings with the academic research and reflects on threats to validity of this research.

6.4.1 Core System vs. Obsolete System

Evidence from current research indicates that practitioners perceive legacy systems as core systems, rather than obsolete systems– as generally perceived in academia. Practitioners do agree that there exist many problems with legacy systems such as that they are inflexible, and costly to maintain. Notwithstanding these issues, legacy systems are crucial to the execution of day-to-day business processes of organizations whatever the business domain they operate in. Legacy systems are often the "back office" systems, which are rigid and inflexible compared to the "front office" systems. For instance, in financial business, COBOL is predominantly used as "back office" systems to process millions of batch transactions per day, which is crucial to any financial institution. The practitioners perceive legacy systems as crucial systems as they are business critical, reliable, and have been running in production for decades. Within the life cycle of the legacy systems, they have been well tested and practically run without errors to execute business processes.

Surprisingly, the main consideration for practitioners to determine if a system is legacy depends on whether the functionalities of the system are still in-line with the business-IT alignment of the organization. This finding contrasts with the observations made in academia, in which factors such as inflexibility [32, 43], expensive to maintain [32, 22], and even use of obsolete programming languages [22, 41] are frequently used to decide if a system is legacy. However, the findings of this study stand in contrast with this. We observed mixed results from the interview (53%) and survey findings (54.1%) who agree that programming language is one of the factors to determine if a system is legacy. Such a mixed observation has led us to further investigate if there is any interesting correlation. Consequently, we performed a Pearson Chi-square test to check if there exists any association between the role of the respondents (7 different categories) and the choice of programming language as a deciding factor for a system being legacy. The analysis with (ρ= 2.9), which is far less than the critical value (α= 12.59), revealed that such an association does not exist. Further, it is interesting to observe that the practitioners could not formulate a concrete definition for a legacy system– a frequently observed issue in academia (e.g., Alderson & Shah [7], Cornelissen et al. [71]).

6.4.2 Legacy System is a Cash Cow

As identified by Adolph [4], the findings of this study justify the notion of cash cow used for legacy system– systems that have been bringing in revenue for years for organizations. Despite various problems, this study revealed that practitioners perceive legacy systems as critical systems that execute the day-to-day business processes. One of the findings with respect to the "business critical" characteristic conforms with the observation made by Brodie [41]. However, other characteristics such as "reliable system", "proven technology" are merely discussed in academia. In general, legacy systems are presented as a quagmire in academia, often reporting about the problems associated with them and hence, urge for their modernization. But, in industry the legacy systems are still perceived as the "backbone" system of organizations. These systems are developed, tested and have been processing millions of records for decades on a daily basis. Apart from the "business critical" characteristic, the reliability of the legacy systems is the other predominant factor that keeps legacy systems still alive in industry. Frequently, the informants of the interviews indicated that legacy system are cash cows as: "*generate a lot of revenue for banks*"– P11, "*the most profitable systems*"–P7, and "*great business value"*–P10.

6.4.3 If It Ain't Broken, Don't Fix It

The aphorism "If it ain't broken, don't fix it" suitably captures the belief among practitioners. Not only in the interviews, but the respondents of the survey also stated in the "Others" section of the survey that if the legacy systems are working "well", then legacy system modernization projects are unlikely to be initiated despite various problems. Some of the responses include "*they have been working- why fix it?"* by a developer with 17 years of experience with legacy system, and "*we didn't fix it last year, and survived. Why should this year be different?*" by a chief technical officer of a software development company.

One of the most significant problems raised by the practitioners is scarcity of knowledge, including legacy experts and (up-to-date) documentation of the legacy systems. Evidence in academia shows that lack of knowledge is one of the enablers for legacy system modernization [32, 147, 7, 22]. Apart from issues with documentation, erosion of soft knowledge [196]– existing knowledge in the form of skills and experiences within the technical staff (original developers, maintainer, users)– is also another problem that has triggered legacy system modernization in industry. Over the years, soft knowledge has become scarce resulting in knowledge erosion due to factors such as ageing, retirement, and staff changing jobs. Furthermore, Khadka et al. [147] report on how various mergers and acquisitions of the supplier of Tandem NonStop based COBOL has triggered legacy system modernization in a bank. Additionally, the other significant driver for legacy system modernization identified by practitioners is "high maintenance cost"– a well reported issue of legacy systems in academia [43, 32, 22, 283].

With respect to drivers of the legacy system modernization, the findings of our study are in-line with the problems reported in academia.

6.4.4 Business vs. Technical Aspects: A Tale of Two Perspectives

Surprisingly, the study findings reveal that the challenges of legacy system modernization in industry largely emerge from the business perspective and also complements most of the technological challenges reported in academia. Challenges such as "Funding modernization projects", "Resistance from organization", "Predicating ROI", "Timing constraints to finish" strongly relate to business perspectives of legacy system modernization. This observation also holds true for the drivers of modernization. In particular, the "Become flexible to change" driver is identified by 85% and the "Faster time-to-market" by 68% of the respondents in the survey reflect the importance of the business aspect of legacy system modernization. Furthermore, the need for a business case to communicate the necessity and consequences of modernization to the top management also reflects the importance of business aspects of modernization. Such evidence clearly indicates that legacy system modernization is not just a technical endeavor, but also a business endeavor. Lately, legacy system modernization research has also focused on considering the business and organization aspects together with the technical aspects (e.g., Murer et al. [196], Nasr et al. [202]).

6.4.5 Limitations and Threats to Validity

Adolph et al. [3] argue that explorative qualitative research is often viewed with discomfort in software engineering with regards to research validity, and verification as assessing the validity of qualitative research is a challenging task [107]. In the following subsections, the most relevant threats to validity

are discussed.

6.4.5.1 Credibility

Credibility refers to the fact that the findings of the research are free from any potential research bias. To mitigate the internal validity and to strengthen the credibility of the research, various measures have been adopted. To start with, sampling of the informants for the interview and survey was performed in which only the informants having experiences with legacy systems and legacy system modernization were included. Furthermore, the research closely followed the GT guidelines, including careful coding, memoing and categorization. Each coding and categories were cross-validated with the other researchers in the process of developing. Additionally, the results of the interviews were triangulated with an online survey filled in by 198 respondents. To increase the transparency of the research, various artifacts such as interview records, interview transcripts, interview protocol, details of the coding process and the results of the survey are made available[31].

6.4.5.2 Generalizability

Generalizability is concerned with to what extent the findings can be generalized. One of the potential threats to the external validity is the fact that all the interview informants are from the Dutch industry. This choice may somehow bias the results. However, the distribution of the informants includes different company sizes, different level of experience, different roles, and variation in the domain of the companies (refer Table 6.1) arguably increases the generalizability of the findings. Furthermore, the findings of the research have been validated by respondents from more than 30 different countries in the survey, which also increases the confidence of generalizability.

6.5 Related Work

This section describes related work aimed at assessing legacy systems and legacy system modernization along with empirical legacy system modernization research.

6.5.1 Legacy Systems and Their Characteristics

Legacy systems have been researched over the past three decades. The seminal work in this area is the first law of continuing change by Belady & Lehman [19], followed by the concept of preventative maintenance [180]. Both of these works put emphasis on addressing changes to make software systems more maintainable. Following the concepts of software evolution & maintenance, Brodie & Stonebraker describe legacy systems as "*any systems that cannot be modified to adapt to constantly changing business requirements and their failure can have a serious impact on business*" [43]. Brodie [41] mentions various characteristics of legacy systems such as mission critical, hard to maintain, inflexible and brittle. Bisbal et al. [32] listed problems of legacy systems such as legacy systems run on obsolete hardware, are expensive to maintain, suffer from lack of documentation and understanding of system, and are difficult to extend and integrate with other systems. Alderson and Shah [7] describe the issue regarding the lack of legacy experts/resources. From the aforementioned definitions and characteristics of legacy systems, it is evident that legacy systems are perceived as a serious problem in academia. The current study, in contrast, takes

[31] `www.servicifi.org`

a different approach in identifying the characteristics that still keep them operational in the industry. Nevertheless, the study also explores various issues of the legacy systems that are in-line with the issues identified in the academia. In fact, we explicitly identified overlaps between the issues of legacy systems as observed in academia and in industry such as high maintenance cost, lack of resources, inflexibile systems. In this research, we have presented these issues as drivers for legacy system modernization.

6.5.2 Legacy System Modernization and Challenges

A plethora of legacy system modernization approaches have been reported. Brodie & Stonebraker presented the DARWIN method [42] as an incremental approach for migrating legacy systems. The RENAISSANCE approach [286] delivered a systematic method for legacy system modernization. Wu et al. [296] described the Butterfly method that uses a gateway-free approach unlike Brodie & Stonebraker's gateway approach [43]. Weiderman et al. [289] presented a system modernization approach that leverage middleware and wrapping technology. The use of reverse engineering techniques in legacy system modernization has been reported by Quilici [217] and Weide et al. [288]. Bisbal et al. analyzed the existing legacy system modernization approaches in their survey [33]. Seacord et al., in their book [241], presented a risk-managed approach to legacy system modernization. Various methods/techniques have been used to modernize legacy systems such as architectural pattern languages [106, 116], feature modeling [187, 181], iterative engineering [29], wrappers [268, 289], and aspects [193]. In the last decade, the advancement of web-based technologies has fostered legacy system modernization (e.g. [264, 164, 44, 75]). In particular, the service-oriented architecture (SOA) has been a popular target architecture for legacy system modernization and various legacy to SOA modernization approaches have been reported (e.g., Khadka et al. [145], Razavian & Lago [222], Almonaies et al. [8]).

With respect to legacy system modernization challenges, Brodie [41] listed various technical challenges of legacy system modernization that are more influenced by academic research experiences. Sneed [254] discussed recurring risks such as performance loss, architectural mismatch, testing bottleneck, current staff's rejection that are associated with a reengineering projects. Van Deursen et al. [276] presented an overview of techniques to facilitate legacy system modernization and the issues of modernization, particularly aiming at identifying objects. In a research roadmap, Bennett & Rajlich [23] argued to generalize the legacy system modernization approaches beyond source code, and to include data and objects of the current legacy systems.

The legacy system modernization research and the challenges identified in the academia are largely technology-oriented. They provide different techniques/methods to facilitate legacy system modernization and point out various challenges faced in the course of applying those techniques/methods. Our research not only identified various business and organizational issues, but also confirmed the technical observations made by researchers.

6.5.3 Empirical System Legacy Modernization Research

Sneed [250] reported a case study of modernization of a commercial application system for distributing books and other publications. Colosimo et al. [67] performed two controlled experiments to evaluate migration of COBOL-based system to a J2EE, web-based system. Murer et al. [196] presented their experience with an evolutionary approach to migrate legacy software of Credit Suisse Bank. Their findings largely complement our results as they focus not only in technical details of the modernization,

but also on the business and organizational aspects. Apart from technical issues, the authors in particular identify the importance of knowledge existing within the technical staff, role of cultural values of the organization in the modernization process. Nasr et al. [202] present their experience with realizing two SOA migration case studies in which they identify various business issues such as resistance to change from the organization, time constraints to finish modernization along with technical issues such as lowering the maintenance cost, and increased flexibility. From an industrial perspective, Torchiano et al. [271] surveyed 59 Italian companies to identify the state-of-the-practice in software modernization and explained that the main factor influencing the modernization process is human factor followed by technological factors and issues related with data inconsistency. Razavian & Lago [223] conducted interviews to understand how legacy to SOA migrations are conducted in industry aiming at identifying which modernization activites/processes are prevalent in industry. In a report published by NASCIO [198], a survey of 29 states in the US is reported with the aim to identify the state-of-art of legacy systems, and the challenges faced during their modernization.

The findings of these studies resonate with the results of our study. The findings of this study undoubtedly support the results of these studies, yet we have employed different research methods (qualitative and quantitative methods) to further increase the reliability of the findings. In most of the aforementioned studies such as [222, 198, 271], a single research method is used. With respect to the observational studies [202, 67, 250], we refer to future work to validate our findings.

6.6 Concluding Remarks

Although legacy systems and their modernization have been extensively researched, this chapter addresses the relative absence of empirical studies of industrial perception of legacy systems and their modernization. To the best of our knowledge, this is the first attempt to empirically investigate the perceived benefits of legacy systems, problems associated with legacy systems that initiate modernization, and the challenges faced during modernization from an industrial perspective. With the current status of our research some of our findings provide empirical evidence for existing hypothesis from the literature while others provide new insights that extend the body of knowledge on software modernization.

What this study adds to the discourse is evidence that legacy systems are not necessarily a quagmire, but are business critical, reliable and proven systems that effectively execute the day-to-day business of organizations. Such perceived benefits of legacy systems are the factors that keep them alive in industry. Not to mention that the practitioners are largely motivated by the "If it ain't broken, don't fix it" aphorism towards legacy system modernization. In addition, the study identifies drivers of and the challenges for legacy system modernization. The drivers of the legacy system modernization in industry confirm the observation made such as lack of knowledge, high maintenance costs, and achieving flexibility. However, the challenges of legacy system modernization faced by the practitioners are not just technical, but also organizational and highly motivated by business considerations.

In summary, this study documents the following contributions:

- We document the industrial perception of legacy systems and their modernization.
- We identify the perceived benefits of the legacy systems, drivers of modernization, and challenges that the industry faces during modernization.
- We report the differences in perception of legacy systems between the industry and academia.

The findings reported in this study are the outcome of an empirical research. We, therefore, want to convey that the findings and the conclusions drawn are suggestive, rather than conclusive. We do not claim to have comprehensively addressed all the perceived benefits and the drivers of legacy systems, and the challenges faced during legacy system modernization in the industry. However, the detailed grounded theory approach that we have adopted for analyzing the interview data, followed by the validation of the findings via an online survey, increase our confidence to claim that the findings and conclusions drawn represent a significant view.

The findings have the following implications: with these findings in hand, academics can (re)focus their efforts in legacy system modernization to include, besides the technological scope, the organizational issues, and business drivers for modernization, and in order to benefit from academic research results, practitioners should examine the academic tools, techniques, and methods developed for legacy system modernization, and initiate collaborations.

As to future work, we aim at validating the findings of the current study using observational participating research in a number of real world modernization scenarios. In addition, we believe that this study has highlighted "practitioners' issues", which the research community can address in order to facilitate technology and knowledge transfer.

Chapter 7

Post Migration Analysis of Legacy System Modernization

Abstract

Software modernization has been extensively researched, primarily focusing on observing the associated phenomena, and providing technical solutions to facilitate the modernization process. Software modernization is claimed to be successful when the modernization is completed using those technical solutions. Very limited research, if any, is reported with an aim at documenting the post-modernization impacts, i.e., whether any of the pre-modernization business goals are in fact achieved after modernization. In this research, we attempt to address this relative absence of empirical study through five retrospective software modernization case studies. We use an explanatory case study approach to document the pre-modernization business goals, and to decide whether those goals have been achieved. The intended benefits for each of the five cases we considered were all (partially) met, and in most cases fully. Moreover, many cases exhibited a number of unintended benefits, and some reported detrimental effects of modernization.

7.1 Introduction

In the software engineering, it is widely accepted that real world software must be continually adapted or enhanced to remain operational [19]. The need for constant adaptation or enhancement within an operational software system is triggered by various factors such as adapting to new business requirements, changes in legislation, advancement in technology [218]. Lehman's laws of software evolution suggest that operational software systems must often reflect these changes, otherwise they become progressively less useful to the stakeholder [19]. Hence, evolving these operational software systems by constantly adapting to changes is critical and requires significant resources [188, 23]. Failure to take remedial changes gradually makes them costly to operate and maintain, thereby turning them into legacy software systems– systems that significantly resist modification and are less maintainable [22].

In terms of the software life cycle, Comella et al. [68] categorize software evolution into three activities: maintenance, modernization, and replacement. Despite the fact that some researchers and practitioners use software evolution and software maintenance interchangeably [23], this research distinguishes these two terms and instead adopts the categorization (i.e., maintenance, modernization, and replacement) proposed by Comella et al. [68].

Software modernization has been extensively researched in academia, primarily to increase maintainability, increase flexibility and reduce costs [23]. Hence, a plethora of software modernization methods exist. The majority of these aim to address technical aspects of modernization, i.e., providing technical solutions to perform or to facilitate the software modernization process [143]. Furthermore, these technical solutions of software modernization are labeled as "successful" once the modernization process is proven to be technically feasible. As per our knowledge, very limited, if any, (empirical) assessments of the impact of software modernization in terms of business goals exist. Nasr, Gross & van Deursen [202] indicate the need for a distinction between the technically feasible outcome and the impact of software modernization. Lack of empirical evidences of impact software modernization can contribute to different expectations and wrong estimations of resources. This knowledge gap can potentially lead to delays or even failures of software modernization projects. However, measuring the impacts of software modernization is not trivial.

In this chapter, we present five retrospective case studies of software modernization with the aim of documenting the impacts of modernization. We adopt an explanatory case study research method (seeking an explanation of a situation or problem for pre- and post-event studies [233]) to explore the pre- and post- modernization situation. The contribution of this research is two-fold: first, it documents the expected benefits and the impacts of software modernization by analyzing pre- and post- modernization situations, and second, it compares the post modernization impacts with the benefits of software modernization as claimed by academic research.

Section 7.2 of this chapter reviews related work. Section 7.3 presents the case study setup and research method used for this research. Section 7.4 provides the five case study reports. Section 7.5 presents the findings of the cases. Section 7.6 discusses the threats to validity of the research. Finally, Section 7.7 provides concluding remarks and relevant future research directions.

7.2 Related Work

7.2.1 Software Evolution and Software Modernization

Software evolution and software maintenance are mature research domains in software engineering. Despite, several researchers and practitioners use software evolution as a preferable substitute for software maintenance [23], this chapter distinguishes these two terms. Godfrey & German [104] provide a distinction between software evolution and software maintenance; the former is used to describe the phenomena associated with modifying existing software systems, whereas the latter describes the activities to make the existing software systems easier to manage and change. Adhering to this distinction between software evolution and software maintenance, we adopt the concept of software evolution discussed by Comella et al. [68] in terms of the software life cycle. They categorize software evolution activities into three: maintenance, modernization, and replacement. When a software system is deployed, maintenance activities are used to keep it operational. But, as the software system becomes increasingly outdated, maintenance becomes too challenging and costly, thereby requiring a modernization effort to do extensive changes rather than maintenance. Finally, when the old system can no longer be evolved, it is then replaced. Hence, software modernization is the process of evolving existing software systems by replacing, re-developing, reusing, or migrating the software components and platforms, when traditional maintenance practices can no longer achieve the desired results [142].

Research within software evolution and software maintenance has primarily focused on building an understanding of software evolution and maintenance by (empirically) observing the phenomena [104]. Such observations include studies of large scale industrial software systems (e.g., Belady & Lehman et al. [19], Gall et al. [94]), and open source systems (e.g., Godfrey & Tu [105], Koch [156]). Furthermore, empirical methods such as surveys (e.g., Kemerer & Slaughter [140], Kagdi et al. [136]) have been used to understand software evolution. A summary of empirical studies performed to understand open source software evolution is reported by Fernandez-Ramil et al. [88]. Various methods and techniques have been used to understand software evolution such as a change-based approaches (e.g., Robbes & Lanza [230]), software visualization techniques (e.g., Lanza [163]), program transformation (e.g., Baxter [18]), mining software repositories (e.g., Kagdi et al. [136]). Similarly, numerous (empirical) studies have been reported to understand the software maintenance phenomenon (e.g., Singer [243], Bianche et al. [28]). A plethora of software modernization approaches have been published, including literature reviews. For instance, Comella et al. [68] reported black-box modernization approaches; Razavian & Lago [222], Almonaies et al. [8] published reviews on migrating legacy systems to SOA; and recently Jamshidi et al. [129] reported legacy to cloud modernization approaches.

Despite this large number of (empirical) research, efforts have been focused on observing the phenomena [104] [218], and addressing technical aspects to facilitate software evolution, maintenance and modernization [143].

7.2.2 Benefits of Software Modernization

For the past three decades, the software evolution community has proposed several methods to modernize legacy software systems. These modernization methods aim at helping organizations achieve various benefits such as reduced costs, increased flexibility, and improved maintainability. To get a systematic

overview of such benefits as suggested in academia, we conducted a literature review using a backward snowballing approach [128], i.e., using the reference list of a paper to identify new papers to include. Table 7.1 depicts a non-exhaustive list of benefits, referred to as "claimed benefits" hereinafter, as a result of backward snowballing. We have observed that there is a clear absence of empirical research that measures these benefits, although limited research has been validated with industrial case studies to assess the applicability.

Table 7.1: Claimed benefits identified in the literature

Claimed Benefits	**Research Paper**
Cost reduction	[202, 8, 143, 142, 246, 21, 32, 85, 41, 44, 95, 161, 34, 23, 42, 296, 250]
Increased reusability	[202, 8, 32, 85, 161, 34, 296, 276]
Increased agility	[202, 8, 222, 32, 85, 187, 34, 276]
Increased flexibility	[202, 8, 222, 142, 21, 44, 296]
Improved performance	[32, 187, 4, 95, 161, 34, 296]
Increased maintainability	[21, 85, 41, 193, 42, 250]
To remain competitive	[129, 85, 75, 95, 161, 296]
Increased availability	[142, 129, 85, 41, 95, 42]
Faster-time-to-market	[222, 142, 187, 23, 42]
Increased interoperability	[143, 129, 85, 41, 172]

To summarize, there has been a very limited, if any, research assessing the post-modernization situation and there is a need for empirical research in collaboration with industry to assess whether any of the claimed benefits are met [202]. In this chapter we address the two key gaps in current understanding: the lack of empirical case study research documenting the claimed benefits as established in the pre-modernization phase, and to which extent those benefits were in fact met after modernization.

7.3 Case Study Design

We report on five retrospective case studies of software modernization. We chose retrospective cases for several reasons. First, the objective of this research explicitly requires successful software modernization cases that have been performed in the past. Second, the retrospective nature makes it possible to get an in-depth understanding of a contemporary phenomenon where the investigator has little control over events, thereby reducing research bias [297]. We have adopted an explanatory case study research method, primarily seeking the rationale for initiating software modernization and documenting the impacts of software modernization. Despite the fact that case studies are originally used primarily for exploratory purposes [91], Runeson & Host [233] argue that explanatory case studies are more suitable for investigating the pre- and post-event situations. Our research aims at documenting pre- and post-modernization situations, and thereby fits well with the latter.

Data collection in this case study is performed by: (i) consulting documentation to identify the need for and goals of the software modernization, and (ii) semi-structured interviews to understand the rationale and impacts of software modernization. To identify the objectives of the modernization, we started with consulting project documents including, but not limited to, project initiation documents (PIDs), business cases for modernization, modernization strategy documents, project management reports, and intermediate milestone deliverables, whenever available. As for semi-structured interviews, we conducted nine interviews. The interviews were informal and, whilst grouped around three themes, were designed

to allow the conversation to follow the respondents' interests. The interviewees typically included a project manager and/or a technical manager. In one of the cases, we also interviewed an IT director and a finance manager. We decided to involve people with different roles to obtain a varied outlook on the situation. For instance, an IT Director potentially provides a better rationale/business case for software modernization while a technical manager can elaborate on the technical issues and impacts of modernization. Prior to the interviews, each interviewee was introduced to an interview protocol, a document detailing the objectives of the interview with relevant questions grouped into themes, and a glossary of the technical terms to attain a common understanding. The interview protocol included a brief introduction to the research, questions regarding personal background of the interviewee (name, current role and responsibility, expertise, and experience), and interview questions categorized into three themes: pre-modernization situation, modernization process, and post-modernization situation.

Table 7.2: Details of the interviewees

SNo	**Role**	**Experience**	**Company**
P1	Technical Lead	34	Infra Co.
P2	Developer	29	
P3	Finance Manager	28	
P4	ICT Business Coordinator	13	Aviation Co.
P5	Business System Analyst	34	
P6	IT Director	30	
P7	Project Manager	10	Public Service
P8	Migration Lead	24	Gov. Office
P9	Project Manager	20	Finance Co.

All the interviews were conducted in English and lasted between 60-120 minutes, except one that was conducted in Dutch and later translated to English. All the interviews were conducted either in-person or via skype (for the international case studies). These interviews were recorded and then transcribed. In case more information or clarifications were needed, the interviewee and/or relevant sources identified by the interviewee were consulted in-person or via email. Table 7.2 depicts the anonymized details of the interviewees with role, years of IT experiences, and the domain of the company. Nvivo 10[32] is used as an instrumentation tool to facilitate the interview analysis process. After the individual case studies were analyzed, we used cross-case analysis (CCA) [297, 242], a data analysis method that analyzes multiples cases studies seeking for empirical evidence on a specific fact, synthesizing data, drawing inferences, and providing recommendations [297, 81]. In this research, CCA is used to identify similarities and differences among the case studies to develop concrete findings based on them.

7.4 Case Studies

We analyzed five software modernization cases within Europe that were completed at least two years ago. Four of them are based in the Netherlands and one in Portugal. The case companies come from different domains: two are from the public sector (Government organizations), two are industrial companies, and one is a financial company. We start with identifying the pre-modernization business goals, referred to as "expected benefits" hereinafter and then document the impact after modernization, referred to as "observed benefits". We do not focus on the modernization process itself. For each case study, we provide a brief summary of the case company and the pre-modernization scenario, list the expected benefits, and

[32] www.qsrinternational.com/products_nvivo.aspx

describe the impacts after modernization. The documented benefits are supported by relevant data i.e., interview quotes and relevant texts from the documentation. To anonymize the products, [**legacy system**] and [**modernized system**] are used in the quotes and some textual corrections are made within [], whenever necessary to increase understandability.

To provide an impression of the size of the legacy systems prior modernization, Table 7.3 depicts some of the available details. For the "Public Service" case, details of the system were not available.

Table 7.3: Details of the legacy systems

Company	**Language**		
	Name	**Size (LoC)**	**# Module**
Infra Co.	Progress	over 50K	Not Available
Aviation Co.	Progress	51370	1803
Public Service	COTS	Not Available	Not Available
Gov. Office	COBOL	over 1 million	2500
Finance Co.	COBOL	9.289 millions	3548

7.4.1 Case I: The Infra Co. Case

The Infra Co. is a company based in the Netherlands that develops innovative solutions for the consumer market, in particular, installation, construction and special products. The company had a COBOL-based legacy information system (IS) that was migrated to Progress[33] in 1990. The Progress-based IS was a highly performant character-based application with sales order, purchase order, warehouse management, financial management and marketing management modules. The IS was a client-server based application running on SUSE Linux on HP hardware. In 2007, Infra Co. started a project to modernize the existing Progress-based IS from console-based to GUI and to re-host the IS in VMware-based virtualized servers. The modernization project lasted for around 12 months. This case description is based on the latter modernization project.

7.4.1.1 Enhance Usability

The primary objective was to modernize the existing character-based user interface system. Furthermore, incremental development with different departments led to 3 different user interfaces for the same backend. Aiming to improve the user interface, the case company initiated the modernization process. P1 expressed this as *"In our case the modernization was mainly the modernization of user interface because we are thinking for 90% we could re-use the code which was behind the user interface."* P2 emphasized the need of modernization as *"We encountered many problems so we decided to modernize our [applications] which were reliable character user interface. And [we] modernize it into [**modernized system**]."*

With the modernization, the company transformed their character-based application to [**modernized system**] based GUI. P2 mentioned *"The goal was to get one interface that succeeded. Flexibility is good because it is easy and fast to change anything within [**modernized system**]. So that's easy. [..] I think we have reached what we wanted, I think. I think the users are happy with the programs."* P1 expressed the impact of enhanced usability. For instance, he stated *"But users got used to it very quickly [..] It's nice to work with [...] I think much more user friendly [...] key of F1 was same everywhere, the buttons*

[33]Currently known as ABL–`www.progress.com/openedge/features/abl`

were with same functions and they got used to it very quickly." P2 supported P1 as *"After working few weeks with new system, it gets easy [..] started getting used to the system very fast."*

7.4.1.2 Product Consolidation

The Infra. Co. has a subsidiary in Belgium that has its application portfolio. Historically, the applications were the same but over time incremental developments in the Netherlands and Belgium resulted in diverse applications accessing the same data. The company used this modernization to consolidate these applications into one. P1 expressed this as *"We started with this character [based] system and then [..] we started to program with the [legacy system] in windows environment and again 5 years later we started with [**modernized system**] but we never converted old software with [to] the new one. So we ended up with 3 systems next to each other and they all have different functions while they use the same database and you can say that programs used for finances were character interface programs, programs used of sales department were [**legacy system**] tools [..] and the programs for purchase department were created in [different platform]."* P2 indicated that it was important to consolidate those applications as *"of course it was different environment, [we want to] bring it back to one environment. [**legacy system**] has its own databases, [different platform] has own, we have our own database. So that is also an important issue [..] and for maintenance, updates; it is easier to have one rather than 3 environments."*

After modernization, the company has now consolidated the application into one with new interfaces, a separate business logic layer and a database layer. P1 mentioned that with the product consolidation, the development team has benefited *"Well it [Legacy application] was bit messy for users so [we] cleaned up and [now we have] one nice system in which we could develop more programs."* He also indicated that testing of application has become significantly easier as *"So testing is far more easy."*

7.4.1.3 Increase Maintainability

Maintenance of the legacy system was difficult as mentioned by P1 as *"the [**legacy system**] software, maintaining was very very hard for us, for development point of view."* The difficulty to maintain the legacy systems was due to the fact that individual sister companies were running their own silo systems. P1 expressed that as *"we had 3 companies with 3 different environments and also with also different Progress version, it was very hard to maintain."* P2 further explained *"So we had 3 interfaces and some people had 3 buttons in[on] screen and they have to open all 3 to use all the programs they have. That was not easy for maintenance."*

The new system turned out to be highly maintainable as compared to the legacy application. P2 mentioned this as *"It is easier to maintain programs, biggest advantage I mean."* The impact of modernization on increased maintainability was expressed by P1 as *"so we had to get all things in one table as this was not easy to maintain 2 tables for the same thing. That was difficult in the beginning."* With the new system, modification of the programs for end users was simple to achieve. P1 expressed this as *"About maintainability, [it] is quite easy. We did a lot of work on, it's a one second [one second of] work, if he needs [an] extra right, it's very easy to do."*

7.4.1.4 Unintended Benefits and Detrimental Effects

After modernization, transparency in the organization has increased. End users are more clear on what their daily job is, and collaboration has been on the rise. P1 mentioned *"For users, it doesn't become more flexible but more clear."* and P3 as *"departments are using the same system to collaborate and it is much better than before. Yes, for more transparency."* Furthermore, the company has achieved a maintenance cost reduction by reducing the number of programmers from 5 to 2 and lower maintenance activities. P1 expressed this as *"So there were 5 persons who could develop on that system and now we are only 2 [...] So the cost reduction is mainly in less time we spent on maintenance of the system."*

When it comes to detrimental effects, the company did observe some user resistance in the initial stages. The users were used to the character-based interface. P2 indicated this behavior as *"Most users were not really happy when going to the Windows [**modernized system**] surroundings [environment]. Because they were so used to those [character-based] screens."* P1 raised concerns over performance and mentioned that the new system was not as performant as the legacy system and said *"About performance, well it's not really bad but its not really fast but fast enough [..] Because it's [**legacy system** is] a character interface the performance is good and per definition faster than windows. And that was the biggest advantage."*

7.4.2 Case II: The Aviation Co. Case

The Aviation Co., headquartered in the Netherlands, excels in the aerospace market with over 50 years of experience in selling aircraft parts to customers in more than 100 countries. The company had a "Quotation Management" application built around 2001 that was running on the three continents where the company has branches. The application used a Progress database and was coupled with three other information systems within the company. The modernization project lasted for around 11 months.

7.4.2.1 Increase Flexibility

The primary goal of this modernization was to enhance flexibility by decomposing the monolithic legacy system, in order to increase the possibilities of reuse of system modules. P5 expressed this as *"So breaking up into reusable modules, and splitting layers like user interface, business logic, database access. [..] and in separate re-usable modules as well, so we can interchange and allowing business logic which is proprietary to the business systems to be accessed by outside parties like customers or suppliers."* P5 sees this as an opportunity to expand their business by making the existing landscape more flexible. He says *"Before, we were combining business logic, screens and database layers to the same setup code, [...] or when you develop a new application we can reuse the business logic layer in the form of a black box and also consuming from another application so we made our software also more flexible for future developments."*

After successful modernization, the company not only decomposed their legacy landscape but also started offering their modules as services (i.e., Software-as-a-Service). P4 mentioned that the new system is able to provide services as *"[...]one case where we have to use modernization also[is] to package the application to provide it as a service to one of our customers. Before we were the only user of it. And we had developed an in–house inventory tool but now we were selling it as a service, Software-as-a-Service also to one of our customer."*

7.4.2.2 Increase Maintainability

One of the key business goals was to increase maintainability of the legacy systems. The legacy systems were running independently in three geographical locations (the US, Hong Kong and the Netherlands) creating maintainability problems. P5 mentioned *"And where modifications have to be made on several locations because the software is not modularized."* He further detailed that the applications have evolved independently *"[..] sources are copied and modifications are built into the copied files and where that leads to [independent evolution] when a problem appears in regular process and sources are modified because and this modification is necessary in other files as well."*.

Modernization provided a completely new framework and standard ways of developing applications, thereby enhancing maintainability. P4 explained this as *"The programming standards over time have been documented and is now a formal document which is also shared with the vendors, so they start working from that."* He further added: *"What is interesting is that, after the modernization making simple changes has become little easier than before [...] Because people know the code, know the processes, use cases are defined after the modernization so it is easier to maintain."*

7.4.2.3 Increase Usability

With this modernization, the company aimed to improve user-friendliness. P4 highlighted this business goal as *"While in modernization phase we were able to also think of the easiness of working, accessibility of the application, usability of the application."* He expressed that the business needed to be more competitive and indicated that there is a need to have a "smart" looking new application as: *"You wanted to look smart. Business software is always ugly, very annoying to use and that to has become one of my targets when I was managing sales. I was one to have applications which look smart."* P6 stated that *"That was the first decision where modernization took place. I think what influenced modernization is [that] end user experience becomes key in applications."*

During software modernization, the Aviation Co. made architectural changes transforming monolithic legacy code to a layered architecture, including a presentation layer to represent the user interface. P4 expressed *"User experience has become more and more topic. Therefore our decision to come up with multi-layer applications"* He further firmly stated that the usability has increased after modernization by stating *"Modernized system was more user friendly than the old one."*

7.4.2.4 Unintended Benefits

Several unintended benefits were reported by the interviewees. One of them was indirect maintenance cost reduction, which was never their main business goal, yet it was achieved. P5 highlighted this as *"Cost reduction [was] not a business goal. It is a side effect because we have less overhead in maintenance but it's not a business goal to have ICT shrink because we have one source instead of."* Similarly, the company observed increased availability of the new application as compared to the older one. P4 expressed this as *"[...] there is a separate database for intermediate storage and actual data like stock data, part information there is an exchange with core database with messaging files, which makes front end 24/7 available. Independent of availability of the backend system."*

From the organizational perspective, the company has observed organizational flexibility. P5 men-

tioned that the behavior of the organization has also changed. P6 supported P5's observation as *"Modernization is not only the technical part but it is also the adoption in your organization"*. Similarly, better transparency was also observed in that users were better able to diagnose problems within the new application. P4 commented on this as *"In a later stage we noticed that the new system was working better than the old system and even later stage it was obvious that errors that were reported from the application appeared to be master data errors [rather] than application errors. So, [yes] that was transparent."*

7.4.3 Case III: The Public Service Case

The Dutch public service office initiated a project in 2010 to modernize its legacy commercial-off-the-shelf (COTS) application. Due to high maintenance cost and limited vendor support, the public service office decided to modernize the COTS application to an Oracle platform. The legacy application was a heavily customized COTS package and was responsible for managing finances of the office. After successful modernization, the new Oracle-based software system is built upon an enterprise service bus (ESB) that facilitates easy integration of software applications from different government agencies. The project lasted for three years.

7.4.3.1 Reduce Maintenance and Operational Cost

One of the expected benefits is to save cost as mentioned by P7 *"From business perspective they were looking for integration and the cost and time saving of this."* Also from a licensing perspective, the organization was searching for a cheaper option as stated by P7: *"But [new system] was cheaper [..] it will almost likely to be quite [a bit] cheaper because the licenses per user was cheaper in this case. That's why they chose [new system] in first place."*

After modernization, there was a reduction in maintenance and operational cost. This was achieved with significantly lower licensing cost from Oracle as compared to the cost of running [COTS]. The interviewee confirmed that significant savings were achieved after modernization by indicating that the licenses per user was cheaper. The other factor that contributed to cost saving was by reducing number of manpower. P7 mentioned this aspect as *"cost saving, efficiency and number of employees, that if you have a new process that is smoother easier and quicker than you don't need that many employees."*

7.4.3.2 Phase out Legacy Technology

The existing system was a heavily customized [COTS] system, incrementally customized for more than 6 years. With all these customizations, the legacy systems was fit for purpose but required significant efforts to integrate with new systems. P7 expressed this as *"This [COTS] system , existed [for] 6 years or so and is heavily customized. A lot of custom effort to get the way they wanted to do it but it involved lots of customization. The point is that after 6 years later, it was perfect for what they were doing."* Furthermore, the system was not supported by the vendor. P7 explained: *"Well, first of all this was an old system. We were on an older version which was not supported so [legacy system] was to upgrade."* These drawbacks significantly increased cost and hence the organization decided to phase out their legacy technology.

As a result of modernization, the [COTS] system was replaced by an Oracle platform with an ESB. P7 explained the differences by drawing the legacy and the current architectures. In the new architecture,

an ESB is used to integrate various applications.

7.4.3.3 Unintended Benefits and Detrimental Effects

As an effect of modernization, the organization has experienced improved organizational flexibility. Some of the geographically separated departments of the organization were merged to one. P7 expressed this as *"Eventually because of this [modernized] system, like I said these departments were geographically separate, but they started working together. But they also moved geographically to one location."* The effect of improved organizational flexibility was also reflected by changes in the process. The interviewee expressed this as *"So there was an updated model in who is allowed to do what within the system especially the process is changed. Then process got integrated and people got different roles within the flow of the system."* Regarding detrimental effects, P7 mentioned that there was some user resistance reported after operationalization of the new system.

7.4.4 Case IV: The Gov. Office Case

This is a Government office of Portugal which was running its administration module built in COBOL and DB2, both operating on IBM mainframes. The system was running 24/7 IMS applications to serve users via terminal emulators (directly) or via web-services (indirectly). Prior to modernization, a proof-of-concept was successfully done. Then the modernization project was started in 2010 and involved re-hosting and migrating the COBOL application running on an IBM mainframe to a Linux environment, converting code in one COBOL dialect to another, and migrating IMS data to an Oracle database. The modernization project lasted for six months.

7.4.4.1 Reduce Operational Cost

As per available documentation, the primary objective of the modernization was to reduce the maintenance and operational cost due to the use of mainframe technology. One of the initial documents state this as *"ongoing maintenance costs were well above 1M euros/year, and kept increasing."* P8 confirmed that the project was aimed at reducing costs as *"The main purpose was to reduce cost immediately."*

After the successful modernization, the organization made significant cost savings. The organization reported that the maintenance cost was reduced by more than 80%, primarily due to phasing out mainframe systems. P8 reported that *"The costs reduced enormously with [the] same functionality, with more or less same performance at much lower cost."*

7.4.4.2 Phase out Legacy Technology

With this modernization, the organization also wanted to phase out their legacy technology (i.e., the mainframe and COBOL). The key reason for this was the ageing mainframe and COBOL manpower within the organization. P8 mentioned *"Just had two COBOL programmers– one of them was already retired and he would go there part-time to solve problems and to develop some new functions. This was also a concern."*

Due to the strategy taken to modernize the legacy system, this goal was partially realized. The organization opted to first re-host the legacy application from mainframe to Linux but keeping the

COBOL within minimal configuration changes. With the cost savings from re-hosting, the organization planned to phase out COBOL in the future. P8 mentioned this as *"we only moved the application out of mainframe and the application was still in COBOL [..]the new system was more flexible and [..] you could develop in other languages like Java and integrate with the application. This was not possible with the mainframe."*

7.4.4.3 Unintended Benefits and Detrimental Effects

It is interesting to observe that there were several unintended (both positive and negative) impacts observed. Several unintended improvements were realized such as improved queries, automatic archiving, improvements to withstand much larger loads than before. Nevertheless, the performance of the new system was below that of the old mainframe system due to the firewall and security mechanisms used in the new Linux environment.

7.4.5 Case V: The Finance Co. Case

The Finance Co. is a Dutch financial group. In 2012, the company initiated a modernization project for its payments system built in COBOL and running on an IBM mainframe. The payments application used a DB2 database with approximately 28TB of historical data. CICS was used for transaction monitoring and approximately 10K jobs to run batches. The overall application had 11M LoC for batch programs and 8M LoC for online transaction processing. The modernization process involved re-hosting and migrating the application from the IBM mainframe to an AIX Unix platform, re-developing the existing CICS code for online transactions in Java, migrating DB2 to an Oracle 10 database, and testing. The modernization project lasted for 10 months.

7.4.5.1 Reduce Operational Cost

The Finance Co. had its payments system running on an IBM mainframe and the operational cost, particularly licensing cost of the mainframe-based applications, were steadily increasing. To reduce operational costs, the company started the modernization project. The project manager (P9) worded this as *"The Finance Co. saw the exploitation [operational] costs getting higher and higher and the application itself was limited in its extensibility [..] But the system ran into limitations on the mainframe. They could not grow, well they could, if they invested in expanding the mainframe. But they wanted to cut costs on operating the system."* The project documentation also emphasized "Reduction in operation cost" as one of the main business objectives.

Financial reports after modernization indicate that due to this modernization project, the operational costs have been reduced significantly.

7.4.5.2 Increased Performance

The other key aim of modernization was to increase performance. The processing capacity of the mainframe was reaching its limits within its existing configuration. P9 expressed this as *"The time they needed for the daily process of their mutations was too long. The window they had from 7 pm to 7am [out of office time] started to get close to not being enough. If there was one little problem, the process would need too much time. Once every month, for their big batch, they needed even 60 hours. Concluding, the*

total lead-time needed every night was too high. A project goal was to lower this time."

After modernization, the company compared the batch window on the mainframe to that of the AIX Unix environment. The performance gain on the AIX Unix environment was more than a factor of three.

7.4.5.3 Phase out Legacy Technology

The key consideration on achieving cost reduction was by phasing out the mainframe ecosystem. The mainframe ecosystem incurred high licensing cost and the company was approaching the end of the existing contract. The company took this opportunity to discontinue the mainframe. P9 expressed this concern as *"... had the idea already to move to another platform, on the mainframe were some products with high licensing costs. The Finance Co. already researched this, how can we move the applications to Java, but ran into problems."*

Prior to this modernization, the company tried modernizing the COBOL to Java, but that was not successful. Hence, in this project they opted to initially re-host the existing COBOL application from mainframe to AIX Unix and IBM DB2 database to an Oracle database. With this modernization, the company successfully re-hosted their operational environment but did continue to use COBOL.

7.4.5.4 Unintended Benefits

After modernization, the company benefited from increased flexibility by becoming vendor independent (Mainframe ecosystem) and opened up possibilities to adopt new advanced technologies.

7.5 Findings

7.5.1 Cross-Case Synthesis

Table 7.4 depicts the findings of the cross-case analysis of the five software modernization case studies. In addition to the summary of expected benefits of all five cases, we also list some of the modernization activities that were in fact performed as well as some that we know were not performed. By listing the activities, we aim to provide a high-level view of the commonalities and differences between the five cases in terms of activities. The added value of listing the modernization activities is that they sketch a more detailed picture of the extent of the modernization, and allow us to put the associated benefits better in context. Finally, in the rightmost column of the table we list any side effects that have been observed after modernization. These side effects can be both positive (unintended benefits, marked with +), and negative (detrimental effects, marked with −).

7.5.1.1 Expected vs. Observed Benefits

As can be observed in Table 7.4, most of the expected benefits are met after software modernization. The expected benefits include both technical goals such as enhanced usability, increased maintainability, product consolidation, and increased performance as well as business goals such as reducing costs, and phasing out legacy technology. We note that organizational benefits (like organizational flexibility and transparency) that are mentioned mostly as the unintended benefits, are also much less commonly mentioned as claimed benefits in the literature.

In two of the cases (the Gov. Office and Finance Co.), the expected benefit (i.e., *"Phase out legacy technology"*) was only partially met. These two exceptions are interesting, because in both cases this was in fact the main goal of modernization. In the Gov. Office case, phasing out legacy technology was a goal for two reasons: a lack of skilled manpower and the high cost involved in keeping the legacy technology operational. In the Financial Co., the high operational cost was the main reason to phase out legacy technology. In both cases, the choice was made to re-host the legacy COBOL system to different platform. The idea was to reduce operational cost by re-hosting the legacy system on a cheaper platform, in order to save money and thereby create the budget to initiate programming language modernization (porting the COBOL source code to a more modern technology). As indicated in the individual case study, significant cost savings (>60%) were reported. The process of re-hosting the legacy systems to a different platform did require various additional activities: in the case of Finance Co., assembler code was re-written in Java, Rexx scripts were converted to Unix-Rexx, development based on VisualAge was changed to the Enterprise Generation Language that generates Java wrappers. In case of the Gov. Office, the COBOL source code required minor syntactic adaptations, IMS database migration to Oracle database, and the re-development of the libraries.

7.5.1.2 Unintended Benefits

Interestingly, the case companies observed some unintended benefits– benefits that were not originally set down as goals of the modernization. The unintended benefits include recurring goals such as reduced maintenance cost, increased availability, and increased flexibility. Furthermore, three of the cases (Infra Co., Aviation Co. and Public Service) observed organizational benefits in terms of transparency and flexibility. This indicates that a software modernization can bring about significant changes in the organization.

We also observed some interesting opportunities for one of the case companies that arose as a side effect of modernization. The Aviation Co. decomposed their legacy software in a way that allowed them to expose their capabilities as SaaS to one of their customers. Additionally, they used the process of modernization to initiate an improvement to the software development process by introducing programming standards and enforcing quality control checks.

In spite of all these benefits, some detrimental effects were also observed. In two of the cases (Infra. Co. and Public Service) user resistance was reported after modernization. Despite "increase performance" is listed as a claimed benefit (cf. Table 7.1), the opposite was observed for the Gov. Office case: the performance of the new system was lower as compared to the original mainframe system due to additional firewall and security mechanisms on the new platform.

7.5.2 Claimed vs. Observed Benefits

This chapter provides an opportunity to compare the claimed benefits (cf. Table 7.1) with the observed and unintended benefits (cf. Table 7.4). Most of these observed and unintended benefits complement and are in–line with the claimed benefits. The observed and unintended benefits form a subset of the claimed benefits of software modernization. However, we do observe some differences. For example, the benefits related to organizational benefits (Organizational flexibility and transparency) attain relatively little attention in the academic literature. On the other hand, the list of claimed benefits contains some

Table 7.4: Cross-Case Analysis of five case studies

Case	Modernization Act.	Exp. Ben.	Obsrv. Ben.	Other Observation
Infra. Co.	Operational platform change	Enhance usability	Yes	+Organizational transparency
	User interface modernization	Product consolidation	Yes	+Maintenance cost reduction
	Code optimization	Increase maintainability	Yes	−User resistance
Aviation Co.	Operational platform change	Increase flexibility	Yes	+Maintenance cost reduction
	User interface modernization	Increase maintainability	Yes	+Increased availability
	Architectural modernization	Enhance usability	Yes	+Organizational flexibility
				+Organizational transparency
Public Service	Architectural modernization	Phase out legacy technology	Yes	+Organizational flexibility
	Operational platform change	Reduce maintenance & operational cost	Yes	−User resistance
	Database migration	Increase flexibility	Yes	
Gov. Office	Code conversion	Reduce operational cost	Yes	+Improved queries
	Database migration	Phase out legacy technology	Partially	+Increased load threshold
	Operational platform change			−Decreased system performance
	Code analysis			
Financial Co.	Operational platform change	Reduce operational cost	Yes	+Increased flexibility
	Code conversion	Phase out legacy technology	Partially	
	Database migration	Increase performance	Yes	
	Code analysis			

benefits reported due to modernizing to specific architectures/platforms. For instance, modernization to a service-oriented architecture (SOA) promises to deliver software modernization specific benefits such as increased flexibility, reduced costs, increased productivity, increased reusability, faster-time-to market, loose coupling [202, 83, 206]. Since none of our case studies include modernization to SOA, we are not able to assess some of these claims.

7.5.3 Lessons Learned

The work reported here is indicative and the sample (case studies) are not large enough to claim comprehensiveness. Despite being an initial research initiative to empirically explore benefits of software modernization, the following lessons can be of interest:

- Wider applicability: Industry can utilize software modernization not only to reduce maintenance cost and to phase out obsolete technology but also for other (business) opportunities. For instance, software modernization can provide an opportunity to redefine the business model of the company. We observed this for Aviation Co. where software modernization enabled the company to offer their modules as services through SaaS. Furthermore, they also used this opportunity to improve their software development process by introducing standards. In the Infra. Co. case, software

modernization was used to consolidate their products.

- Technical vs. organizational aspects: Apart from possible technical improvements, software modernization can be used to improve organizational aspects such as bringing transparency and flexibility. This was observed in two of the cases.
- User resistance was observed in two of the cases. This suggests that any software modernization should also address the soft skill aspects to mitigate such resistances by conducting training and capacity building programs, and by holding seminars to create the necessary awareness about software modernization.
- In two of the cases (Public service and Finance Co.), a phased approach of mainframe-based software modernization, i.e., initially re-hosting the mainframe-based systems to economical platforms and thereby saving maintenance cost for language modernization in future, was undertaken. This is a worthwhile software modernization alternative that industry can adopt.

7.6 Validity

Qualitative research studies are often viewed with discomfort in software engineering [3] with respect to validity as assessing the validity of the qualitative research is a challenging task [107]. Yin [297] argues that the quality of the case studies based on explanatory research should be judged on the basis of the following types of validity:

7.6.1 Construct Validity

Construct validity concerns the validity of the research method and focuses on whether the constructs (i.e., questions, terminology) are interpreted and measured correctly. This is a clear threat to our research as maintaining consistent terminologies and their definitions throughout multiple case companies is a challenge. To minimize this threat, we initially provide an interview protocol that includes a brief introduction to the research, interview questions, and an explanation of the terminology used. After transcribing the interviews, the interviewees were contacted by email, if required.

7.6.2 External Validity

concerns the domain to which the results can be generalized. With respect to external validity, we do not claim comprehensiveness of the finding; it is rather indicative. The diversity of the case companies in terms of domain, company size and geography gives confidence that the findings represent a significant view. Furthermore, the cross-case analysis method that we adopted to synthesize the findings arguably increases the generalizability. However, more empirical studies will have to be designed and executed to extend the validity of the findings. In particular, when we compare the claimed benefits from the literature (see Table 7.1) with the expected and unintended benefits of the five cases, then it is clear that we do not cover them all. In particular, among the claimed benefits we also find benefits reported due to modernizing to specific architectures/platforms. For instance, modernization to a service-oriented architecture (SOA) promises to deliver software modernization specific benefits like increased flexibility, reduced costs, increased productivity [202] along with SOA specific benefits such as increased reusability, faster-time-to market, loose coupling, statelessness [83, 206]. Since none of our cases involved modernization to SOA, we are not able to assess these claims.

7.6.3 Reliability

is concerned with demonstrating that the results of the study can be replicated. This threat is mitigated by maintaining a case study database[34] that contains all the relevant information used in the case study. This case study database consists of anonymized interview transcripts, Nvivo coding, interview protocols and backward snowballing. We believe that these artifacts contribute towards the transparency.

A few remarks should be made on the retrospective nature of the case study. A problem relating to this type of study is hindsight bias– a belief that an event is more predictable after it becomes known than it was before it became known [232]. In this research, it means that interviewees might tend to re-construct the business goals based on the results of the impacts. We minimized hindsight bias by using multiple interviewees within same company and documentation, whenever possible.

7.7 Conclusion

Research on software modernization suggests many "claimed" benefits of modernization with limited empirical evidences to support the claims. A plethora of technical solutions of software modernization are labeled as "successful" once the modernization process is proven to be technically feasible. There is limited research that assesses whether these technically "successful" solutions do meet the expected benefits, i.e., the pre-modernization business goals.

In this chapter, we address this gap of empirical evidence by discussing five (retrospective) case studies of software modernization. We have documented the "expected benefits" of each case and considered whether these "expected benefits" were in fact met after modernization. In general, the outcome of these five case studies suggest that the "expected benefits" were observed after modernization. Interestingly, we found that the case companies also observed several "side effects" of modernization: most of them were "unintended benefits", but also a few detrimental effects, primarily, due to user resistance and decreased performance. Among the reported unintended benefits we also found benefits that received relatively little attention in software modernization literature, in particular, organizational transparency and organizational flexibility.

To summarize, this chapter has the following contributions:

- reports upon five retrospective modernization case studies within different organizations,
- documents the pre-modernization business goals as "expected benefits" and identifies whether these benefits were in fact met, and provides a comparative analysis of "claimed" and "expected" benefits of modernization.

To the best of our knowledge, this empirical research is the first to explore and validate the observable benefits of software modernization. It is clear that more empirical studies have to be performed in collaboration with industry to further extend and strengthen the findings. In particular, when we compare the claimed benefits from the literature to the expected benefits of our case studies, then some benefits are still missing. Furthermore, study of successful case studies about modernization to a specific architecture/platform (e.g., legacy to SOA or cloud, code conversion) can provide insights into more specific benefits. In case of software modernization, we believe that an empirical study of failure cases is invaluable to fully understand modernization impacts, making this an important topic for future work.

[34] Available at `https://servicifi.wordpress.com/ICSME-2`

Part III

Conclusion

Chapter 8

Conclusions and Outlook

Legacy software modernization has been widely researched for decades. Despite numerous modernization methods and techniques, legacy software systems are still running the core business process in industry. With the promises of new technologies such as mobile computing, cloud computing, legacy software system modernization has become increasingly important in academia and in industry. Several research projects such as REMICS [192], ARTIST [26], MODAClouds [10] have indicated the importance of modernizing legacy software systems. Similarly, legacy software modernization has been in the top 10 priorities of companies, as identified by Gartner [97]. Acknowledging the importance of legacy software system modernization, we started the "ServiciFi" project[35] with the objectives to develop a modernization method, and to identify challenges being faced by the financial sector. Subsequently, the findings are used to develop techniques [236] to identify and extract services from monolithic code [234, 235]. This dissertation is the outcome of the former objective of developing modernization method and identifying challenges faced by the financial sector. In this research, we have identified that academic research on legacy software modernization is focused on technical aspects and not catering for business aspects. This shortcoming of legacy software modernization methods from academia prevents a wide acceptance of such methods in industry. In this concluding chapter, we provide the summary of this research by elaborating these findings.

We revisited software modernization research with two main research objectives: first, to develop a consolidated software modernization method, and second, to document how software systems and their modernization are perceived in industry. Based on the two objectives, the dissertation is divided into two parts. The first part of the thesis deals with a practical problem [291], and as a result, a structured software modernization process is developed. The first objective is focused on combining technical and business issues that persist in software modernization in the context of modernization towards a SOA system. The second part of the dissertation deals with a knowledge problem [291], and as a result, the research is focused on understanding how software modernization is perceived and performed in practice. Despite decades of research in software modernization, billions of lines of code of legacy software are still operational. With the second research objective, we documented the industrial perception of legacy software systems, and their modernization challenges.

In this chapter, we reflect upon the overall research findings and revisit the research questions presented in Chapter 1, summarize the answers this research provides, and discuss additional pointers to

[35] `http://servicifi.org/`

further research.

8.1 Revisiting Research Questions

Based on the two research objectives discussed in Chapter 1, this dissertation is divided into two parts. Each part is based on a research objective and addresses a main research question. Each research question is further divided into multiple research sub-questions. In this section we present each research objective and answer the research (sub-)questions. The first research objective is presented below:

Research Objective 1 (RO1)–"Develop a software modernization approach that includes technical and business aspects."

This research objective aims at developing a software modernization method (particularly, for the modernization to a SOA system) that consolidates technical and business aspects. Research in software modernization domain is focused on addressing technical aspects and less attention is given to others. Hence, a holistic view of consolidated method of software modernization is missing. To address this objective, RQ 1 is formulated as:

RQ 1 How can a modernization process be designed that facilitates enterprises in modernizing software systems?

This question is answered in Part I of this dissertation by considering SOA system as a target architecture. To address this question, we initially develop a legacy software modernization method that combines technical and business aspects of legacy software modernization. Then, a state of the art of modernization of legacy software systems to SOA system is provided. Finally, a structured modernization approach is developed. RQ 1 consists of three sub-questions. A summary of the answers to the sub–questions is provided below:

RQ 1.1 What are the (essential) steps to combine business aspects and technical aspects in software modernization?

This research question is answered in Chapter 2. The primary focus of this research question is to identify essential steps of combining business aspects and technical aspects related to software modernization. Up to now, research has been focused either to develop supporting technology to perform a technical modernization project or to determine modernization feasibility. However, a consolidated method that combines both aspects was missing. To answer this research question, we employed a method engineering approach to identify essential steps and developed a consolidated method by reusing method fragments from existing modernization methods. The consolidated method is known as the "ServiciFi" method. As Part I of this dissertation focuses on modernizing to a SOA system, the existing modernization methods used in developing the ServiciFi method are based on Service-orientation. In total, we used three service-oriented development methods (SODDM [208], WSIM [166] and SOMA [12]) to identify re-usable method fragments.

The ServiciFi method consists of five phases:

- **Project initiation phase**: performs the assessment of the viability of the modernization by analyzing the technical and economical feasibility. Technical feasibility is decided with an "as-is" analysis of the existing systems and the economical feasibility is done using a cost-benefit analysis.
- **Candidate service identification phase**: focuses on identifying candidate services to satisfy the requirements detailed in the earlier phase.
- **Service specification phase**: specifies the detailed capabilities of the "to-be" services. Such capabilities are derived from the existing legacy services and by mapping with the relevant third party services, if required.
- **Service construction and testing phase**: focuses on extracting source code from the legacy system using concept slicing techniques and performing the necessary tests.
- **Deployment, monitoring and management phase**: details post-development activities including, but not limited to, publishing, provisioning, versioning, and monitoring.

The project initiation phase focuses on including business aspects such that feasibility analysis (cost and technical) and the rest of the phases are focused on executing the technical modernization process. The ServiciFi method was initially evaluated with eight experts from industry and academia. Furthermore, the ServiciFi method is evaluated with two case studies to extract services from existing systems. With these two evaluation mechanisms, the ServiciFi method has been shown to be feasible and practical.

RQ 1.2 What is the state of the art of software modernization in academia?

Software modernization, in particular modernization to a SOA system, has been extensively researched in the last decade. This has resulted in a large body of knowledge of which there exists no comprehensive overview. In this research question, we identify peer-reviewed research articles from academia to gain knowledge of the state of the art of modernizing software to a SOA system. We use a systematic literature review (SLR) research method to gather a historical overview methods and techniques used in software modernization. To answer RQ 1.2, we identify and collect 121 scientific articles reported from 2000 to 2011. The articles are evaluated based on an evaluation framework.

The findings of the SLR indicate that only two scientific articles address both modernization feasibility (business aspect) and technology support for modernization. Most of the articles (97) focus on realizing modernization by providing tools and techniques to execute the modernization process. These implementation techniques are broadly divided into either integration or migration. The integration category largely includes techniques such as wrappers, adopters and middleware whereas migration focuses on internal restructuring and modification of the existing systems. Techniques to migrate legacy code include program slicing, model transformation and code transformation.

In addition to presenting a historical overview of legacy to SOA modernization approaches, this research also identifies several issues such as (i) the need of a structured modernization method combining technical and business aspects, (ii) automation of the structured method via integrating various tools and techniques, (iii) post-modernization experience reporting,

and (iv) addressing soft factors of software modernization. Chapter 3 provides the details of the SLR.

RQ 1.3 How can a structured legacy to SOA software modernization process be developed from existing methods and techniques?

Following the first two sub-questions, the RQ 1.3 research question identifies the need of a structured software modernization process. In Chapter 4, we extend the evaluation framework of Chapter 3 and the "ServiciFi" method of Chapter 2 to develop a consolidated software modernization process consisting of six phases. The six phases of the process are the following:

- **Legacy system understanding phase**: involves creating an inventory of the existing "as-is" features of the systems, primarily to understand the existing (legacy) software landscape.
- **Target system understanding phase**: focuses on developing a target "to-be" architecture blueprint with major components of the SOA environment and SOA-related standards.
- **Migration feasibility determination phase**: identifies the feasibility of the modernization from technical and economical perspectives.
- **Candidate service identification phase**: enables identifying potential legacy code so as to maximize reusability and leveraging existing legacy features.
- **Implementation phase**: involves the execution of the modernization process based on the chosen modernization strategy and availability of tool support.
- **Deployment & provisioning phase**: deals with deployment and management of the services after exposing the legacy application as a service.

Furthermore, within each of the six phases of the process, we present a rationale to justify the need of each phase, current practices within each phase, and challenges that require further attention. The proposed structured process is then evaluated by (i) migrating features of two simple yet representative applications to SOA, and (ii) by mapping activities reported in literature.

The outcome of RQ 1 is a structured software modernization method that combines technical and business aspects of software modernization to SOA. Initially, we developed a method by identifying essential steps to combine technical and business aspects. The next step was to investigate the state of the art of legacy to SOA modernization methods. This step indicated that the modernization methods reported in academia cater for technical solutions to software modernization and less attention is given to business and organization aspects. Based on this, we developed a structured software modernization method using a method engineering approach [39]– an approach to design, construct, and adapt methods, techniques and tools for the development of information systems. As Part I of this dissertation is focused on SOA systems, the structured software modernization method is developed from existing SOA development methods. With the method, we have answered RQ 1.

Software modernization has been a widely discussed topic in industry as well. Legacy software systems are still operating in critical domains such as financial institutions, government offices, and aviation. Despite a plethora of software modernization approaches, legacy software systems and their

modernization are still prevalent in industry. Research indicates that academic research in software modernization are too abstract to adopt in industry. Furthermore, large scale software modernization projects often overrun budget and time in practice. This indicates a knowledge gap within academia and industry in understanding software modernization. In this dissertation, this knowledge gap is addressed by the following research objective.

Research Objective 2 (RO2)–"Identify how software modernization is perceived and conducted in practice."

This research objective identifies how legacy software systems and software modernization are perceived in practice. This research objective leads us to empirically investigate why legacy systems are still prevalent in industry and what challenges practitioners face in software modernization. To address this objective, RQ 2 is formulated as follows:

RQ 2 What are the perceptions of practitioners about software modernization?

This question is answered in Part II of this dissertation. To address this question, we initially report on a case study of large scale software modernization of a financial application. Then, we empirically investigate how legacy software systems and its modernization is perceived in the industry. Finally, we perform five retrospective software modernization case studies to document post-modernization effects. RQ 2 consists of three sub-questions. A summary of the answers to the sub–questions is provided below:

RQ 2.1 How is large scale software modernization performed in practice?

This research question establishes a context to explore how software modernization is conducted in practice. We have observed that a limited number of case studies of large scale software modernization is reported in academia. In this research question, we conduct a case study of a large scale software modernization process and explore the challenges faced in practice. The case study is conducted in a Dutch bank which plans to modernize its legacy application into a SOA–based environment.

The case study describes a 4-phase migration process that the bank adopted. For each phase, we investigated techniques used, best practices that contribute to the success, and challenges faced during modernization. The modernization process emphasized the need for a separate governing body, known as the "Migration Program Management Committee", primarily to provide a governance on the modernization process and any other change management process. This governing body is also responsible for establishing architecture principles and aligning the business goals with the architectural requirements.

The modernization process extensively utilizes reverse engineering techniques such as call dependency graphs, and program analysis to understand the complexity of the existing systems. The findings of these reverse engineering techniques are used to decide upon selecting modernization strategies, and to harvest knowledge from the existing systems. Furthermore, the

findings of the case study also highlighted on adopting a pragmatic realization approach based on various factors such as business value, business priority, and the technical qualities of the existing systems. A key take-away from this case study is the fact that modernization of a large scale software systems is not only a technical endeavor, but also requires addressing significant business issues. For example, involving the technical staff in the modernization process, and providing necessary training to enhance their expertise have proven successful in this case.

RQ 2.2 What are the discrepancies between the perception of legacy software systems and their modernization in academia and industry?

The software engineering community recognizes the problems introduced by legacy software systems such as them being obsolete, inflexible, expensive to maintain, having out-dated documentation, and yet these systems are mission critical to the daily operations of many enterprises. A plethora of research has been reported on modernizing these legacy software systems, primarily to reduce maintenance costs and to reuse existing features. However, technology consulting firms estimate that there are still billions of lines of legacy code in operation. Industry does acknowledge the drawbacks of legacy software systems, however, they still extensively rely upon their legacy systems. This discrepancy of knowledge within academia and industry is explored by this research question in Chapter 6 in which we conduct an exploratory study to identify the perceived benefit of legacy software systems, the main drivers for legacy software modernization, and the challenges professionals face with the modernization of legacy software systems. We interviewed 26 industrial practitioners and analyzed the data to understand their perception of legacy software systems and their modernization. The result of the interviews are triangulated (data) with a structured survey to validate the findings.

We observed that academia and industry do share commonalities with respect to the characteristics of the legacy software systems. Despite acknowledging that legacy software systems are resistant to modification and are typically old systems, the practitioners value their legacy systems highly. The practitioners identify that the legacy software systems are core and business critical, and have been operating in production for years. Over the years, these systems have been well tested, optimized and hence are reliable systems for business. Evidence from this research indicates that practitioners perceive legacy systems as core systems, rather than obsolete systems– as generally perceived in academia. Furthermore, the practitioners argue that their legacy software systems are the cash cows– systems that bring in revenues.

With respect to (legacy) software modernization, the practitioners believe in "if it ain't broke, don't fix it" aphorism. Practitioners are reluctant to incur the cost and risk involved in modernizing their legacy software systems. However, they do believe that scarcity of knowledge, including legacy experts and (up-to-date) documentation of the legacy systems, is one of the key challenges that they are currently facing. Furthermore, we observe that the drivers of legacy software modernization and challenges faced while modernizing are not only due to technical reasons, but also due to business issues. Various challenges such as "Funding modernization projects", "Resistance from organization", "Predicting return-on-investment", "Timing constraints to finish" strongly relate to business perspectives of legacy system modernization. Similarly, several drivers including "Become flexible to change" and "Faster time

to market" represent business drivers. Nevertheless, the practitioners do acknowledge drivers such as "High maintenance cost", "Prone to failure" and challenges such as "Data migration" problems, "Complex system architecture", "Difficult to extract & prioritize business logic" as technical issues related with legacy software system modernization.

RQ 2.3 How often are pre-modernization business goals achieved after a "technically" successful software modernization?

Software modernization is claimed to be successful when a modernization process is completed using technical solutions. Instead, such technically "successful" modernization indicates that the transition towards new software is successful because this proves to be a technically feasible approach. However, this technically "successful" modernization is not necessarily an indication of achieving the business goals as aimed for prior to software modernization. Very limited research, if any, is reported with an aim at documenting the post-modernization impacts, i.e., whether any of the pre-modernization business goals are in fact achieved after software modernization. With this research question, we address this relative absence of empirical study through five retrospective software modernization case studies. We use an explanatory case study approach to document the pre-modernization business goals, and to decide whether those goals have been achieved after modernization.

Practitioners from the five case companies were interviewed in order to get an overview of pre- and post- modernization business goals, termed as "expected benefits" and "observed benefits" respectively. In general, the outcome of these five case studies suggest that the "expected benefits" were observed after modernization. Interestingly, many cases exhibited a number of unintended benefits, and some reported detrimental effects of modernization, due to user resistance and decreased performance.

In two of the cases, user resistance is reported indicating that any software modernization should also address the organizational aspects, referred to as soft-skill aspects [196], so as to mitigate such resistances. Moreover, it is interesting to observe that some of the case companies used software modernization not only to achieve recurring benefits such as reducing maintenance cost, phasing out obsolete technology, but also used to explore other business opportunities such as redefine business models by adopting a SaaS model, improve the software development process, and consolidate different products. We further investigate claimed benefits from academia and compared the software modernization benefits (i.e., "expected", "observed", and "unintended" benefits) of case studies. There was no significant gap between academia and industry.

The findings of RQ 2 provide a different viewpoint of legacy software system and its modernization. Practitioners value their legacy software system despite acknowledging the well-known drawbacks of legacy systems. It is interesting to note that challenges and motivations of legacy software modernization in industry acknowledge the business aspects in addition to technical aspects. The post–modernization result from the five retrospective case studies has revealed a new area of research in software modernization. This has enabled to understand the actual business impact of software modernization, i.e., meeting pre-modernization business goals.

8.2 Contributions and Implications

In this section, we present the contributions and implications of this dissertation. This dissertation has three main contributions and each contribution has different implication for the software evolution research community, and the industry. The three contributions and their implications are described below.

- **Structured software modernization method**– The result of this research is a structured software modernization method that consolidates technical, business, and organization aspects. Software modernization methods are broadly focused on (i) developing supporting technology to address the technical modernization (i.e., implementation techniques to conduct modernization), and (ii) developing techniques to determine software modernization feasibility. Chapter 2 demonstrates the feasibility of consolidating these two aspects of software modernization by identifying essential steps in the context of legacy to SOA modernization. The consolidated approach is further extended to develop a structured software modernization method consisting of six phases in Chapter 3 and 4. A method engineering approach is used to develop the structured process by reusing the method fragments of existing software modernization and development methods.

  The structured software modernization method developed in this dissertation is primarily focused on legacy to SOA modernization. The modernization method is developed from existing software modernization methods, hence is sufficiently generic to be used for any software modernization projects. More importantly, the research community can extend the modernization method and adapt it accordingly. This is possible due to the use of method engineering while developing the structured software modernization method. The consolidation of technical and business aspects of legacy software modernization also allows industry to use this method in practice.

- **Industry viewpoint of legacy software systems and their modernization**– Software modernization has steadily gained importance both in academia and in industry. There is an extensive body of knowledge already in academia that is based on research in which legacy systems are predominantly perceived as obsolete systems. However, this research identifies legacy systems as core systems and practitioners value their legacy systems highly. For the first time, this research documented the industrial perception of legacy software systems and their modernization. In particular, an initial attempt is made to identify the perceived benefits of legacy systems, drivers of software modernization, and challenges faced during modernization.

  The industry viewpoint has implications for the software evolution research community and for industry. As for software evolution research, the industrial perception of legacy software systems and their modernization provide practical challenges that industry is facing when modernizing their legacy software systems. Based on these issues and challenges, the research community can align its research agenda. In addition, the result of this research acts as a bridge between the research community and industry to facilitate technology and knowledge transfer in the software modernization domain.

- **Post software modernization impacts**– This dissertation initiates a new research area in the software modernization domain. Until now, research in software modernization has been focused on providing technical solutions to facilitate the software modernization process. Software modernization is claimed to be "successful" when the modernization is completed using those technical

solutions. This technically "successful" represent a successful transition, but not necessarily the business goals set prior to the modernization. As per our knowledge, this dissertation for the first time attempts to empirically study the post-modernization impacts within five case companies.

The findings of the post modernization impacts have implications both for research and for industry. For research, new case studies have to be conducted in order to establish more concrete results in exploring post modernization impacts. The five case studies and their results are the initial steps towards establishing a separate research direction so as to further extend the research in understanding the impact of modernization. As for industry, the preliminary findings of the five case studies indicate that industry can utilize software modernization not only to reduce maintenance cost and to phase out obsolete technology but also for other (business) opportunities such as improving software development process, changing business models (adopting SaaS offerings) . Within the context of the five case studies, we also observed a phased approach of mainframe-based software modernization, which can be a worthwhile alternative that industry can adopt. The phased approach emphasizes on initially re-hosting the mainframe-based systems to economical platforms and thereby saving maintenance cost for system modernization in the future.

8.3 Limitations and Future Works

The findings of this dissertation are the result of individual research, following the design science research framework. Most of the research questions involve knowledge problems, thereby requiring empirical research methods such as interviews, case studies and surveys, wherever applicable. In case of practical problems, the artefact is developed by rigorously following appropriate methods (e.g., method engineering) and taking into account the methods' particular validity constraints. Furthermore, artefacts developed within the scope of practical problems are either evaluated by experts or by conducting experimentation. In a research scenario, where there is need of selection of participants or case companies, the selection process in itself could not be fully controlled by the researcher, albeit measures such as snowball sampling are followed to select participants. Because of this, it cannot be excluded that participants were somehow either positively or negatively biased towards the research domain. This hinders the studys credibility, and further research is necessary to confirm the results. To mitigate this limitation, validity measures are rigorously followed.

Validation of research results is crucial throughout the entire research process. Each chapter of this dissertation details the possible threats to validity of the findings and thus contributes collectively to the credibility of the findings. Additionally, the validity of the results is ensured by rigorously following the research protocols (e.g., case study protocol, interview protocol) that are set up beforehand and employing design science approach in which the results are constantly evaluated during each iteration of the cycle. The research performed in this dissertation includes a limited number of cases or subject and can never cover a significant sample population. Hence, the findings of the research may not be generalizable to the entire population. However, this research attempts to ensure the validity of the results by adhering to the mitigation measures for each research method. The findings of this research can be strengthened further by conducting more case studies and including larger sample population, wherever possible.

The artefacts (e.g., the ServiciFi methods, and the structured software modernization methods) produced in this dissertation are developed by re-using method fragments from existing software mod-

ernization/development methods reported in literature. This potentially excludes the requirements or considerations from the industry, thereby inhibiting adoption in practice. To mitigate this risk, we have evaluated the ServiciFi methods using expert validation.

Based on the implications and limitations, we have identified several new avenues for future research. They are discussed below.

- **Extending the structured software modernization method:** The structured software modernization method proposed in this thesis is developed from existing software modernization methods/frameworks using the method engineering approach. Hence, there is possibility that the structured method does not sufficiently cover the issues identified in the industry. While conducting software modernization case studies in Part II of this dissertation, we identified some possible improvements. One of the improvements concerns the inclusion of a business case for the software modernization project. A business case identifies reasons for initiating a project. For a software modernization project, a business case would potentially represent the need for the project, with long term and short term benefits that project would bring to the organization. Furthermore, a business case also documents the risks associated with the project and alternative considerations for any potential failure. Such a business case document can be an effective medium to communicate with management.

  Additionally, the structured software modernization method can be extended to include guidelines on adopting modernization strategies. In general, a large scale software modernization in industry does not only depend upon one strategy, but rather involves (a combination of) multiple strategies such as wrapping, replacement with COTS product, re-redevelopment, and migration. Such guidelines can include technical details of the existing systems, business strategies and financial constraints. The case company presented in Chapter 4 adopted a similar approach.

- **Tool-set support for the structured software modernization method:** Software modernization is a complex endeavor for any enterprise that involves risks to day-to-day operations. Additionally, software modernization not only involves technical aspects, but requires equal participation from business and management. Adopting any modernization process involves automation of the process, whenever possible. For instance, various tools are being developed in isolation for legacy system understanding, candidate service identification and implementation phases based on reverse engineering techniques. In fact such automation would be expensive and needs significant engineering efforts in integrating the existing or any newly developed tools. Advancements in model-driven engineering have shown some potentials towards this automation. Some of the recent examples include Software EvolutioN SErvice Integration (SENSEI) [132], Software Analysis as a Service (SOFAS) [102], Q-MIG [133], and MARBLE [213].

  As a part of our research, we have also developed an integrated set of language independent tools to analyzed legacy software systems called GELATO– GEneric LAnguage TOols for model-driven analysis of legacy software systems [234]. Within the scope of our project, we have developed a model-driven based toolset that facilitates "as-is" understanding via topic modeling [236]. This has resulted in a web-based application called ITMViz [235] to facilitate program comprehension and transformation of legacy software systems. ITMViz is a part of the GELATO toolset. In future, we aim at extending GELATO to extract legacy code so as to expose them as services in a language independent fashion.

- **Conducting in–vivo research with industry:** As observed throughout the dissertation, there has been a limited number of case study-based in-vivo research within software modernization research. We have attempted to provide details of some of the large scale industrial software modernization cases, but more in–vivo research in collaboration with industry is required. As argued by Van Geet & Demeyer [278], in-vitro research when expanded to in-vivo provides confidence towards applying the research findings in reality, thereby increasing adopted by industry. In the context of software modernization, significant research is conducted in in-vitro settings. Hence, the research results are too abstract to implement in industry and do not fit the industrial purposes. To increase technology and knowledge transfer between academic research and industry in the context of software modernization, large scale in-vivo research in software modernization must be conducted.

  We have captured the industrial perception of legacy software systems and their modernization in this dissertation. The result of the industrial perception reported some discrepancies within academia and practice and also highlights the issues and challenges faced in practice. Such issues and challenges reported by the industry can be utilized by the research community to conduct large scale in–vivo research in collaboration with industry.

- **Exploring post-modernization impact research:** One of the key contributions of this dissertation is the pioneering of empirically investigating the post modernization impact. In chapter 7, we conducted five retrospective case studies to document pre-modernization business goals and validated if those goals were met after successful modernization. Given the large set of claimed benefits (i.e., benefits reported by academia with limited evidences), the case studies only validated a limited sub-set of the claimed benefits. Hence, we envision abundant opportunities to conduct more case studies and validated the claimed benefits. Additionally, the research reported in Chapter 7 is not based on any specific architecture/platform, hence dedicated case studies for specific architecture/platform (e.g., legacy software systems to SOA or cloud, code conversion/transformation) can provide further insights and helps to validate benefits specific to those architectures.

  Furthermore, the findings of Chapter 7 are based on successful software modernization case studies. However, we do strongly believe that any empirical study of failure cases is invaluable to fully understand modernization impacts. In particular, such case studies will provide data to investigate why large scale software modernization overrun budget and time in industry.

With the advancement of cloud and mobile, the industry is experiencing pressure to modernize its legacy software systems. The topic of legacy software modernization will still be relevant not only in academia, but also in industry. Hence, this research becomes more relevant in a way that it presents an extensible legacy software modernization method and documents the industrial perspective of legacy software modernization. Several areas that are identified in this research become more relevant if legacy software modernization research is performed by considering the challenges actually faced by industry.

Part IV

Finale

Bibliography

[1] A. Abran and H. Nguyenkim. Measurement of the maintenance process from a demand-based perspective. *Journal of Software Maintenance: Research and Practice*, 5(2):63–90, 1993.

[2] ADM. Architecture-Driven Modernization, 2010. Retrieved on Jan 2012, from www.adm.omg.org.

[3] S. Adolph, W. Hall, and P. Kruchten. Using grounded theory to study the experience of software development. *Empirical Software Engineering*, 16(4):487–513, 2011.

[4] W. S. Adolph. Cash cow in the tar pit: Reengineering a legacy system. *IEEE Software*, 13(3):41–47, 1996.

[5] S. Alahmari, D. De Roure, and E. Zaluska. A model-driven architecture approach to the efficient identification of services on service-oriented enterprise architecture. In *EDOCW'10*, pages 165–172. IEEE, 2010.

[6] S. Alahmari, E. Zaluska, and D. De Roure. A service identification framework for legacy system migration into SOA. In *SCC'10*, pages 614–617. IEEE, 2010.

[7] A. Alderson and H. Shah. Viewpoints on legacy systems. *Communications of the ACM*, 42(3):115–116, 1999.

[8] A. Almonaies, J. Cordy, and T. Dean. Legacy system evolution towards service-oriented architecture. In *International Workshop on SOA Migration and Evolution (SOAME'10)*, pages 53–62. IEEE, 2010.

[9] V. Andrikopoulos, S. Benbernou, and M. P. Papazoglou. Managing the evolution of service specifications. In *AISE*, pages 359–374. Springer, 2008.

[10] D. Ardagna, E. Di Nitto, P. Mohagheghi, S. Mosser, C. Ballagny, F. D'Andria, G. Casale, P. Matthews, C.-S. Nechifor, D. Petcu, A. Gericke, and C. Sheridan. Modaclouds: A model-driven approach for the design and execution of applications on multiple clouds. In *Modeling in Software Engineering (MISE), 2012 ICSE Workshop on*, pages 50–56, June 2012.

[11] E. C. Arranga and F. P. Coyle. Cobol: Perception and reality. *IEEE Computer*, 30(3):126–128, 1997.

[12] A. Arsanjani, S. Ghosh, A. Allam, T. Abdollah, S. Ganapathy, and K. Holley. SOMA: A method for developing service-oriented solutions. *IBM Systems Journal*, 47(3):377–396, 2008.

[13] L. Aversano, L. Cerulo, and C. Palumbo. Mining candidate web services from legacy code. In *WSE'08*, pages 37–40. IEEE, 2008.

[14] S. Bajracharya, T. Ngo, E. Linstead, Y. Dou, P. Rigor, P. Baldi, and C. Lopes. Sourcerer: a search engine for open source code supporting structure-based search. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications(OOPSLA'06)*, pages 681–682. ACM, 2006.

[15] S. Balasubramaniam, G. A. Lewis, E. Morris, S. Simanta, and D. Smith. Smart: Application of a method for migration of legacy systems to soa environments. In *Proceedings of the 6th International Conference on Service-Oriented Computing (ICSOC'08)*, pages 678–690, Berlin, Heidelberg, 2008. Springer-Verlag.

[16] L. Bao, C. Yin, W. He, J. Ge, and P. Chen. Extracting reusable services from legacy object-oriented systems. In *IEEE International Conference on Software Maintenance (ICSM'10)*, pages 1–5. IEEE, 2010.

[17] B. V. Batlajery, R. Khadka, A. M. Saeidi, S. Jansen, and J. Hage. Industrial perception of legacy software system and their modernization. TR UU-CS-2014-004, Utrecht University, 2014.

[18] I. D. Baxter, C. Pidgeon, and M. Mehlich. DMS®: Program transformations for practical scalable software evolution. In *International Conference on Software Engineering (ICSE'04)*, pages 625–634. IEEE Computer Society, 2004.

[19] L. A. Belady and M. M. Lehman. A model of large program development. *IBM Systems Journal*, 15(3):225–252, 1976.

[20] M. Beller, G. Gousios, A. Panichella, and A. Zaidman. When, how, and why developers (do not) test in their ides. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 179–190. ACM, 2015.

[21] H. C. Benestad, B. Anda, and E. Arisholm. Understanding software maintenance and evolution by analyzing individual changes: a literature review. *Journal of Software Maintenance and Evolution: Research and Practice*, 21(6):349–378, 2009.

[22] K. Bennett. Legacy systems: Coping with success. *IEEE Software*, 12(1):19–23, 1995.

[23] K. H. Bennett and V. T. Rajlich. Software maintenance and evolution: a roadmap. In *Future of Software Engineering*, pages 73–87. ACM, 2000.

[24] J. Bergey, L. O'Brien, and D. Smith. Options analysis for reengineering (oar): A method for mining legacy assets. Technical Report CMU/SEI-2001-TN-013, SEI, 2001.

[25] J. Bergey, D. Smith, N. Weiderman, and S. Woods. Options analysis for reengineering (oar): Issues and conceptual approach. Technical Report CMU/SEI-99-TN-014, SEI, 1999.

[26] A. Bergmayr, H. Bruneliere, J. Canovas Izquierdo, J. Gorronogoitia, G. Kousiouris, D. Kyriazis, P. Langer, A. Menychtas, L. Orue-Echevarria, C. Pezuela, and M. Wimmer. Migrating legacy software to the cloud with ARTIST. In *Proceedings of the 17th Europrean Conference on Software Maintenance and Reengineering (CSMR'13)*, pages 465–468, March 2013.

[27] P. Bhallamudi and S. Tilley. Soa migration case studies and lessons learned. In *2011 IEEE International Systems Conference (SysCon'11)*, pages 123–128. IEEE, 2011.

[28] A. Bianchi, D. Caivano, F. Lanubile, F. Rago, and G. Visaggio. An empirical study of distributed software maintenance. In *International Conference on Software Maintenance (ICSM'02)*, pages 103–109. IEEE, 2002.

[29] A. Bianchi, D. Caivano, V. Marengo, and G. Visaggio. Iterative reengineering of legacy systems. *Transactions on Software Engineering*, 29(3):225–241, 2003.

[30] T. Biggerstaff, B. Mitbander, and D. Webster. Program understanding and the concept assignment problem. *Communications of the ACM*, 37(5):72–82, 1994.

[31] D. Binkley. Source code analysis: A road map. In *International Conference on Software Engineering (ICSE'00)*, pages 104–119. IEEE, 2007.

[32] J. Bisbal, D. Lawless, B. Wu, and J. Grimson. Legacy information systems: Issues and directions. *IEEE Software*, 16(5):103–111, 1999.

[33] J. Bisbal, D. Lawless, B. Wu, J. Grimson, R. Wade, V.and Richardson, and D. OSullivan. A survey of research into legacy system migration. TR TCD-CS-1997-01, Trinity College Dublin, 1997.

[34] J. Bisbal, D. Lawless, B. Wu, J. Grimson, V. Wade, R. Richardson, and D. O'Sullivan. An overview of legacy information system migration. In *Asia Pacific Software Engineering Conference and International Computer Science Conference (APSEC/ICSC'97)*, pages 529–530. IEEE, 1997.

[35] T. F. Bissyandé, L. Réveillère, Y.-D. Bromberg, J. L. Lawall, and G. Muller. Bridging the gap between legacy services and web services. In *Middleware'10*, pages 273–292. Springer, 2010.

[36] H. Boeije. A purposeful approach to the constant comparative method in the analysis of qualitative interviews. *Quality and Quantity*, 36(4):391–409, 2002.

[37] R. W. Bons, R. Alt, H. G. Lee, and B. Weber. Banking in the internet and mobile era. *Electronic Markets*, 22(4):197–202, 2012.

[38] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer. Simple object access protocol (soap) 1.1, 2000. http://www.w3.org/TR/SOAP/.

[39] S. Brinkkemper. Method engineering: engineering of information systems development methods and tools. *Information and Software Technology*, 38(4):275–280, 1996.

[40] S. Brinkkemper, M. Saeki, and F. Harmsen. Meta-modelling based assembly techniques for situational method engineering. *Information Systems*, 24(3):209–228, 1999.

[41] M. L. Brodie. The promise of distributed computing and the challenges of legacy systems. In *Advanced Database Systems*, pages 1–28. Springer, 1992.

[42] M. L. Brodie and M. Stonebraker. DARWIN: On the incremental migration of legacy information systems. TR TR-022-10-92-165, GTE Labs Inc, 1993.

[43] M. L. Brodie and M. Stonebraker. *Migrating legacy systems: Gateways, interfaces & the incremental approach.* Morgan Kaufmann Publishers Inc., 1995.

[44] G. Canfora, A. Cimitile, A. De Lucia, and G. A. Di Lucca. Decomposing legacy programs: A first step towards migrating to client–server platforms. *Journal of Systems and Software*, 54(2):99–110, 2000.

[45] G. Canfora and M. Di Penta. New frontiers of reverse engineering. In *Future of Software Engineering*, pages 326–341. IEEE, 2007.

[46] G. Canfora and M. Di Penta. Service-oriented architectures testing: A survey. In *Software Engineering*, pages 78–105. Springer, 2009.

[47] G. Canfora and M. Di Penta. Service-oriented architectures testing: A survey. In *Software Engineering*, pages 78–105. Springer, 2009.

[48] G. Canfora, M. Di Penta, and L. Cerulo. Achievements and challenges in software reverse engineering. *Communications of ACM*, 54(4):142–151, 2011.

[49] G. Canfora, A. R. Fasolino, G. Frattolillo, and P. Tramontana. Migrating interactive legacy systems to web services. In *European Conference on Software Maintenance and Reengineering (CSMR'06)*, pages 24–36. IEEE, 2006.

[50] G. Canfora, A. R. Fasolino, G. Frattolillo, and P. Tramontana. A wrapping approach for migrating legacy system interactive functionalities to service oriented architectures. *JSS*, 81(4):463–480, 2008.

[51] S. Cetin, N. Ilker Altintas, H. Oguztuzun, A. H. Dogru, O. Tufekci, and S. Suloglu. Legacy migration to service-oriented computing with mashups. In *ICSEA'07*, pages 21–31. IEEE, 2007.

[52] K. Channabasavaiah, K. Holley, and E. Tuggle. Migrating to a service-oriented architecture. *IBM DeveloperWorks*, 16, 2003.

[53] K. Channabasavaiah, E. Yuggle, and K. Holley. Mirgating to a service-oriented architecture. Online, Dec 2003. Available from:`http://www.ibm.com/developerworks/webservices/library/ws-migratesoa/`.

[54] N. Chapin, J. E. Hale, K. M. Khan, J. F. Ramil, and W.-G. Tan. Types of software evolution and software maintenance. *Journal of software maintenance and evolution: Research and Practice*, 13(1):3–30, 2001.

[55] R. N. Charette. Why software fails [software failure]. *Spectrum, IEEE*, 42(9):42–49, 2005.

[56] F. Chen, S. Li, H. Yang, C.-H. Wang, and W. Cheng-Chung Chu. Feature analysis for service-oriented reengineering. In *12th Asia-Pacific Software Engineering Conference (APSEC'05)*, pages 8–18. IEEE, 2005.

[57] F. Chen, H. Yang, B. Qiao, and W.-C. Chu. A formal model driven approach to dependable software evolution. In *30th Annual International Computer Software and Applications Conference(COMPSAC'06)*, volume 1, pages 205–214. IEEE, 2006.

[58] F. Chen, Z. Zhang, J. Li, J. Kang, and H. Yang. Service identification via ontology mapping. In *COMPSAC'09*, pages 486–491. IEEE, 2009.

[59] L. Cherbakov, G. Galambos, R. Harishankar, S. Kalyana, and G. Rackham. Impact of service orientation at the business level. *IBM Systems Journal*, 44(4):653–668, 2005.

[60] E. J. Chikofsky, J. H. Cross, et al. Reverse engineering and design recovery: A taxonomy. *Software, IEEE*, 7(1):13–17, 1990.

[61] M. Chuba. Assessing the next 50 years for the ibm mainframe. Online, 2014. `http://www.gartner.com/technology/reprints.do?id=1-1SJXH56&ct=140401&st=sb`.

[62] S. Chung, J. B. C. An, and S. Davalos. Service-oriented software reengineering: SoSR. In *HICSS'07*, pages 172–182. IEEE, 2007.

[63] S. Chung, P. Young, and J. Nelson. Service-oriented software reengineering: Bertie3 as web services. In *Proceedings of the IEEE International Conference on Web Services (ICWS'05)*, pages 837–838. IEEE, 2005.

[64] A. Cimitile, A. De Lucia, G. Antonio Di Lucca, and A. Rita Fasolino. Identifying objects in legacy systems using design metrics. *Journal of Systems and Software*, 44(3):199–211, 1999.

[65] G. Coleman and R. OConnor. Using grounded theory to understand software process improvement: A study of irish software product companies. *Information and Software Technology*, 49(6):654–667, 2007.

[66] G. Coleman and R. OConnor. Investigating software process in practice: A grounded theory perspective. *Journal of Systems and Software*, 81(5):772–784, 2008.

[67] M. Colosimo, A. D. Lucia, G. Scanniello, and G. Tortora. Evaluating legacy system migration technologies through empirical studies. *Inf. Soft. Tech.*, 51(2):433–447, 2009.

[68] S. Comella-Dorda, K. Wallnau, R. C. Seacord, and J. Robert. A survey of black-box modernization approaches for information systems. In *In the Proceedings of the International Conference on Software Maintenance (ICSM'00)*, pages 173–183. IEEE, 2000.

[69] F. Consulting. Application modernization: Procrastinate at your peril! Online: `http://www.enterprisecioforum.com/en/whitepaper/application-modernization-procrastinate`, 2011.

[70] T. A. Corbi. Program understanding: Challenge for the 1990s. *IBM Systems Journal*, 28(2):294–306, 1989.

[71] B. Cornelissen, A. Zaidman, A. van Deursen, L. Moonen, and R. Koschke. A systematic survey of program comprehension through dynamic analysis. *Transactions on Software Engineering*, 35(5):684–702, 2009.

[72] F. Cuadrado, B. García, J. Dueas, and H. Parada. A case study on software evolution towards service-oriented architecture. In *Proceeddings of the 22nd International Conference on Advanced Information Networking and Applications-Workshops (AINAW'08)*, pages 1399–1404. IEEE, 2008.

[73] T. H. Davenport. Putting the enterprise into the enterprise system. *Harvard business review*, 76(4), 1998.

[74] A. De Lucia, G. A. Di Lucca, A. R. Fasolino, P. Guerra, and S. Petruzzelli. Migrating legacy systems towards object-oriented platforms. In *Proceedings in the International Conference on Software Maintenance (ICSM'97)*, pages 122–129. IEEE, 1997.

[75] A. De Lucia, R. Francese, G. Scanniello, and G. Tortora. Developing legacy system migration methods and tools for technology transfer. *Software: Practice and Experience*, 38(13):1333–1364, 2008.

[76] J. Dedrick and J. West. Why firms adopt open source platforms: a grounded theory of innovation and standards adoption. In *Proceedings of the workshop on standard making: A critical research frontier for information systems*, pages 236–257. Seattle, WA, 2003.

[77] S. Demeyer, S. Ducasse, and O. Nierstrasz. *Object-oriented reengineering patterns*. Elsevier, 2002.

[78] P. J. Denning. A new social contract for research. *Communications of the ACM*, 40(2):132–134, 1997.

[79] J. Dvorak. Conceptual entropy and its effect on class hierarchies. *Computer*, 27(6):59–63, 1994.

[80] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian. Selecting empirical methods for software engineering research. In *Guide to Advanced Empirical Software Engineering*, pages 285–311. Springer, 2008.

[81] K. M. Eisenhardt. Building theories from case study research. *Academy of management review*, 14(4):532–550, 1989.

[82] EPC. SEPA–vision and goal. Online, 2014. `http://www.europeanpaymentscouncil.eu/index.cfm/about-sepa/sepa-vision-and-goals/`.

[83] T. Erl. *Service-oriented architecture: concepts, technology, and design*. Prentice Hall, 2006.

[84] L. Erlikh. Leveraging legacy system dollars for e-business. *IT professional*, 2(3):17–23, 2000.

[85] A. Erradi, S. Anand, and N. Kulkarni. Evaluation of strategies for integrating legacy applications as services in a service oriented architecture. In *IEEE International Conference on Services Computing(SCC'06)*, pages 257–260. IEEE, 2006.

[86] R. Fang, L. Lam, L. Fong, D. Frank, C. Vignola, Y. Chen, and N. Du. A version-aware approach for web service directory. In *ICWS'07*, pages 406–413. IEEE, 2007.

[87] L. Feijs, R. Krikhaar, and R. van Ommering. A relational approach to support software architecture analysis. *Software: Practice and Experience*, 28(4):371–400, 1998.

[88] J. Fernandez-Ramil, A. Lozano, M. Wermelinger, and A. Capiluppi. Empirical studies of open source evolution. In *Software Evolution*, pages 263–288. Springer, 2008.

[89] F. Fleurey, E. Breton, B. Baudry, A. Nicolas, and J.-M. Jézéquel. Model-driven engineering for software migration in a large industrial context. In *Model Driven Engineering Languages and Systems*, pages 482–497. Springer, 2007.

[90] B. Flyvbjerg. Five misunderstandings about case-study research. *Qualitative inquiry*, 12(2):219–245, 2006.

[91] B. Flyvbjerg. Five misunderstandings about case-study research. *Qualitative inquiry*, 12(2):219–245, 2006.

[92] A. Fuhr, T. Horn, V. Riediger, and A. Winter. Model-driven software migration into service-oriented architectures. *Computer Science-Research and Development*, 28(1):65–84, 2011.

[93] A. Fuhr, A. Winter, U. Erdmenger, T. Horn, U. Kaiser, V. Riediger, and W. Teppe. Model-driven software migration: Process model, tool support, and application. In A. D. Ionita, M. Litoiu, and G. Lewis, editors, *Migrating Legacy Applications: Challenges in Service-Oriented Architecture and Cloud Computing Environments*, pages 154–185. IGI Global, 2012.

[94] H. Gall, M. Jazayeri, R. R. Klosch, and G. Trausmuth. Software evolution observations based on product release history. In *International Conference on Software Maintenance (ICSM'97)*, pages 160–166. IEEE, 1997.

[95] H. Gall, R. Klösch, and R. Mittermeir. Object-oriented re-architecturing. In *European Software Engineering Conference (ESEC'95)*, pages 499–519. Springer, 1995.

[96] I. Garcia-Rodriguez de Guzman, M. Polo, and M. Piattini. An ADM approach to reengineer relational databases towards web services. In *14th Working Conference on Reverse Engineering(WCRE 2007)*, pages 90–99. IEEE, 2007.

[97] Gartner. Cio agenda insights 2013. Online, 2013. `www.gartner.com/imagesrv/cio/pdf/cio_agenda_insights2013.pdf`.

[98] Gartner. Insights from the 2013 gartner cio agenda report. Online, 2013. `http://www.gartner.com/newsroom/id/2304615`.

[99] Gartner. High failure rates in insurance legacy modernization challenge cios. Online: `https://www.gartner.com/doc/2653016/high-failure-rates-insurance-legacy`, 2014.

[100] Gartner. Market trends: Banking, worldwide, 2014. Online, 2014. `http://www.gartner.com/document/2646424`, source = `http://www.gartner.com/document/2646424`.

[101] J. V. Geet and S. Demeyer. Feature location in COBOL mainframe systems: An experience report. In *25th IEEE International Conference on Software Maintenance (ICSM'09)*, pages 361–370. IEEE, 2009.

[102] G. Ghezzi and H. C. Gall. A framework for semi-automated software evolution analysis composition. *Automated Software Engineering*, 20(3):463–496, 2013.

[103] B. G. Glaser and A. L. Strauss. *The discovery of grounded theory: Strategies for qualitative research.* Aldine Transaction, Chicago, Illionios, 1967.

[104] M. W. Godfrey and D. M. German. The past, present, and future of software evolution. In *Future of Software Maintenance (FoSM'08)*, pages 129–138. IEEE, 2008.

[105] M. W. Godfrey and Q. Tu. Evolution in open source software: A case study. In *International Conference on Software Maintenance (ICSM'00)*, pages 131–142. IEEE, 2000.

[106] M. Goedicke and U. Zdun. Piecemeal legacy migrating with an architectural pattern language: A case study. *Journal of Software Maintenance and Evolution: Research and Practice*, 14(1):1–30, 2002.

[107] N. Golafshani. Understanding reliability and validity in qualitative research. *The Qualitative Report*, 8(4):597–607, 2003.

[108] N. Gold, M. Harman, D. Binkley, and R. Hierons. Unifying program slicing and concept assignment for higher-level executable source code extraction. *Software: Practice and Experience*, 35(10):977–1006, 2005.

[109] L. D. Goodwin and N. L. Leech. The meaning of validity in the new standards for educational and psychological testing: Implications for measurement courses. *Measurement and evaluation in Counseling and Development*, 2003.

[110] Grammatech. Codesurfer. Online, 2011. Available at: `http://www.grammatech.com/products/codesurfer/academic.html`.

[111] M. Greiler, A. van Deursen, and M. Storey. Test confessions: a study of testing practices for plug-in systems. In *34th International Conference on Software Engineering (ICSE'12)*, pages 244–254. IEEE, 2012.

[112] Q. Gu and P. Lago. Service identification methods: a systematic literature review. In *Towards a Service-Based Internet*, pages 37–50. Springer, 2010.

[113] G. Gui and P. D. Scott. Coupling and cohesion measures for evaluation of component reusability. In *Proceedings of the International workshop on Mining Software Repositories (MSR'06)*, pages 18–21. ACM, 2006.

[114] M. Harman, N. Gold, R. Hierons, and D. Binkley. Code extraction algorithms which unify slicing and concept assignment. In *Proceeding of the 9th Working Conference on Reverse Engineering*, pages 11–20. IEEE, 2002.

[115] D. Harris, A. Yeh, and H. Reubenstein. Extracting architectural features from source code. *Automated Software Engineering*, 3(1):109–138, 1996.

[116] W. Hasselbring, R. Reussner, H. Jaekel, J. Schlegelmilch, T. Teschke, and S. Krieghoff. The dublo architecture pattern for smooth migration of business information systems: An experience report. In *26th International Conference on Software Engineering*, pages 117–126. IEEE, 2004.

[117] R. Heckel, R. Correia, C. Matos, M. El-Ramly, G. Koutsoukos, and L. Andrade. Architectural transformations: From legacy to three-tier and services. In *Soft. Evol.*, pages 139–170. Springer, 2008.

[118] I. Heitlager, T. Kuipers, and J. Visser. A practical model for measuring maintainability. In *6th International Conference on Quality of Information and Communications Technology*, pages 30–39. IEEE, 2007.

[119] J. Henrard, D. Roland, A. Cleve, and J.-L. Hainaut. An industrial experience report on legacy data-intensive system migration. In *International Conference on Software Maintenance*, pages 473–476. IEEE, 2007.

[120] A. Hevner and S. Chatterjee. *Design research in information systems: theory and practice*, volume 22. Springer, 2010.

[121] A. Hevner, S. March, J. Park, and S. Ram. Design science in information systems research. *MIS Quarterly*, 28(1):75–105, 2004.

[122] R. Hoda, J. Noble, and S. Marshall. Organizing self-organizing teams. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 285–294. ACM, 2010.

[123] R. Hoda, J. Noble, and S. Marshall. Using grounded theory to study the human aspects of software engineering. In *Human Aspects of Software Engineering*, page 5. ACM, 2010.

[124] P. Hoyer, M. Gebhart, I. Pansa, S. Link, A. Dikanski, and S. Abeck. A model-driven development approach for service-oriented integration scenarios. In *Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns (COMPUTATIONWORLD'09*, pages 353–358. IEEE, 2009.

[125] J. Hutchinson, J. Whittle, M. Rouncefield, and S. Kristoffersen. Empirical assessment of mde in industry. In *Proceedings of the 33rd International Conference on Software Engineering (ICSE'11)*, pages 471–480. ACM, 2011.

[126] ING. Nederlander die mobiel bankiert heeft meer controle over geldzaken. online, 2012. `https://www.ing.nl/nieuws/nieuws_en_persberichten/2012/08/nederlander_die_mobiel_bankiert_heeft_meer_controle_over_geldzaken.aspx`.

[127] ISO/IEC. Software Engineering– Software Life Cycle Processes–Maintenance. Technical Report 14764-2006, IEEE, 2006.

[128] S. Jalali and C. Wohlin. Systematic literature studies: database searches vs. backward snowballing. In *International symposium on Empirical software engineering and measurement (ESEM'12)*, pages 29–38. ACM, 2012.

[129] P. Jamshidi, A. Ahmad, and C. Pahl. Cloud migration research: A systematic review. *Cloud Computing, IEEE Transactions on*, 1(2):142–157, 2013.

[130] A. Jansen and J. Bosch. Software architecture as a set of architectural design decisions. In *WICSA'05*, pages 109–120. IEEE, 2005.

[131] S. Jansen and S. Brinkkemper. *Information Systems Research Methods, Epistemology and Applications*, chapter Applied multi-case research in a mixed-method research project: Customer configuration updating improvement, pages 1–29. IGI Global, 2008.

[132] J. Jelschen. Sensei: Software evolution service integration. In *Software Evolution Week-IEEE Conference onSoftware Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE)*, pages 469–472. IEEE, 2014.

[133] J. Jelschen, G. Pandey, and A. Winter. Towards quality-driven software migration. In *Software Engineering (Workshops)*, pages 8–9, 2014.

[134] Y. Jiang and E. Stroulia. Towards reengineering web sites to web-services providers. In *European Conference on Software Maintenance and Reengineering (CSMR'04)*, pages 296–305. IEEE, 2004.

[135] S. Johnston. Modeling service-oriented solutions. Online, Jul 2005. Available from: `http://www.ibm.com/developerworks/rational/library/jul05/johnston/`.

[136] H. Kagdi, M. L. Collard, and J. I. Maletic. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *Journal of Software: Maintenance and Evolution*, 19(2):77–131, 2007.

[137] A. Kalsing, G. do Nascimento, C. Iochpe, and L. Thom. An Incremental Process Mining Approach to Extract Knowledge from Legacy Systems. In *14th IEEE International Enterprise Distributed Object Computing Conference (EDOC'10)*, pages 79–88. IEEE, 2010.

[138] Y. Kanellopoulos, C. Tjortjis, I. Heitlager, and J. Visser. Interpretation of source code clusters in terms of the iso/iec-9126 maintainability characteristics. In *European Conference on Software Maintenance and Reengineering*, pages 63–72. IEEE, 2008.

[139] R. Kazman, L. O'Brien, and C. Verhoef. Architecture reconstruction guidelines. Tech. Rpt. CMU/SEI-2001-TR-026, CMU/SEI, 2001.

[140] C. F. Kemerer and S. Slaughter. An empirical approach to studying software evolution. *Software Engineering, IEEE Transactions on*, 25(4):493–509, 1999.

[141] R. Khadka. Service identification strategies in legacy-to-soa migration. In *Proceedings of the 27th IEEE International Conference on Software Maintenance (ICSM'11)*, 2011.

[142] R. Khadka, B. V. Batlajery, A. Saeidi, S. Jansen, and J. Hage. How do professionals perceive legacy systems and software modernization? In *the 36th International Conference on Software Engineering (ICSE'14)*, pages 36–47. ACM, 2014.

[143] R. Khadka, G. Reijnders, A. Saeidi, S. Jansen, and J. Hage. A method engineering based legacy to SOA migration method. In *International Conference on Software Maintenance (ICSM'11)*, pages 163–172. IEEE, 2011.

[144] R. Khadka, A. Saeide, A. Idu, J. Hage, and S. Jansen. Legacy to soa evolution:evaluation results. Technical Report UU-CS-2012-006, Department of Information and Computing Sciences, Utrecht University, 2012.

[145] R. Khadka, A. Saeidi, A. Idu, J. Hage, and S. Jansen. Legacy to SOA evolution- a systematic literature review. In A. D. Ionita, M. Litoiu, and G. Lewis, editors, *Migrating Legacy Applications: Challenges in Service Oriented Architecture and Cloud Computing Environments*, pages 40–71. IGI Global, 2012.

[146] R. Khadka, A. Saeidi, R. Jansen, J. Hage, and R. Helms. An evaluation of service frameworks for the management of service ecosystems. In *PACIS'11*, page 10. AIS, 2011.

[147] R. Khadka, A. Saeidi, S. Jansen, J. Hage, and G. Haas. Migrating a large scale legacy application to SOA: Challenges and lessons learned. In *20th Working Conference on Reverse Engineering*, pages 425–432. IEEE, 2013.

[148] R. Khadka and B. Sapkota. An evaluation of dynamic web service composition approaches. In *Proceeding of the 4th International Workshop on Architectures, Concepts and Technologies for Service-Oriented Computing (ACT4SOC'10)*, pages 67–79, Athens, Greece, 2010.

[149] R. Khadka, B. Sapkota, L. Ferreira Pires, M. Van Sinderen, and S. Jansen. Model-driven approach to enterprise interoperability at the technical service level. *Computers in Industry*, 64(8):951–965, 2013.

[150] R. Khadka, B. Sapkota, L. F. Pires, M. Sinderen, and S. Jansen. Model-driven development of service compositions for enterprise interoperability. In M. Sinderen and P. Johnson, editors, *Enterprise Interoperability*, pages 177–190. Springer, 2011.

[151] R. Khadka, B. Sapkota, L. F. Pires, M. van Sinderen, and S. Jansen. Model-driven development of service compositions for enterprise interoperability. In M. van Sinderen and P. Johnson, editors, *Enterprise Interoperability*, volume 76, pages 177–190. Springer, 2011.

[152] V. Khusidman and W. Ulrich. Architecture-driven modernization: Transforming the enterprise draft v.5. Technical report, OMG, 2007. Available from: http://www.omg.org/cgi-bin/doc?admtf/2007-12-01.

[153] B. Kitchenham. Procedures for performing systematic reviews. Technical Report TR/SE-0401, Keele University, 2004.

[154] B. Kitchenham, O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman. Systematic literature reviews in software engineering–a systematic literature review. *Information and software technology*, 51(1):7–15, 2009.

[155] B. Kitchenham and S. L. Pfleeger. Principles of survey research: Part 5: Populations and samples. *Software Engineering Notes*, 27(5):17–20, 2002.

[156] S. Koch. Software evolution in open source projects–a large-scale investigation. *Journal of Sofware: Maintenance and Evolution*, 19(6):361–382, 2007.

[157] T. Kokko, J. Antikainen, and T. Systa. Adopting soa–experiences from nine finnish organizations. In *13th European Conference on Software Maintenance and Reengineering*, pages 129–138. IEEE, 2009.

[158] K. Kontogiannis, G. Lewis, and D. Smith. A research agenda for service-oriented architecture. In *International Workshop on Systems Development in SOA Environments (SDSOA'08)*, pages 1–6. ACM, 2008.

[159] K. Kontogiannis, G. A. Lewis, D. B. Smith, M. Litoiu, H. Muller, S. Schuster, and E. Stroulia. The landscape of service-oriented systems: A research perspective. In *International Workshop on Systems Development in SOA Environments (SDSOA'07)*, pages 1–1. IEEE, 2007.

[160] R. Koschke. Software visualization in software maintenance, reverse engineering, and re-engineering: a research survey. *J. Soft. Maint. Evol.*, 15(2):87–109, 2003.

[161] J. Koskinen, J. J. Ahonen, H. Sivula, T. Tilus, H. Lintinen, and I. Kankaanpaa. Software modernization decision criteria: An empirical study. In *9th European Conference on Software Maintenance and Reengineering (CSMR'05)*, pages 324–331. IEEE, 2005.

[162] G. Koutsoukos, L. Andrade, J. Gouveia, and M. El-Ramly. Service Extraction. Technical Report 016004, Sensoria Project, 2006. Available from:`http://www.pst.ifi.lmu.de/projekte/Sensoria/del_12/D6.2.a.pdf`.

[163] M. Lanza. The evolution matrix: Recovering software evolution using software visualization techniques. In *IWPSE*, pages 37–42. ACM, 2001.

[164] J. Lavery, C. Boldyreff, B. Ling, and C. Allison. Modelling the evolution of legacy systems to web-based systems. *Journal of Software Maintenance and Evolution: Research and Practice*, 16(1-2):5–30, 2004.

[165] C. Lawrence. Adapting legacy systems for soa. Online, Jun 2007. Available from: `http://www.ibm.com/developerworks/webservices/library/ws-soa-adaptleg/`.

[166] S. P. Lee, L. P. Chan, and E. W. Lee. Web services implementation methodology for soa application. In *Proceeding of the 4th IEEE International Conference on Industrial Informatics*, pages 335–340, 2006.

[167] M. M. Lehman. Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, 68(9):1060–1076, 1980.

[168] M. M. Lehman. Laws of software evolution revisited. In *Software process technology*, pages 108–124. Springer, 1996.

[169] M. M. Lehman, J. Ramil, and G. Kahen. Evolution as a noun and evolution as a verb. In *Proceedings of the SOCE 2000 Workshop on Software and Organisation Co-evolution*, pages 12–13, 2000.

[170] G. Lewis, E. Morris, L. O'Brien, D. Smith, and L. Wrage. Smart: The service-oriented migration and reuse technique. Technical Report CMU/SEI-2005-TN-029, CMU/SEI, Sept 2005. Available from:`http://www.sei.cmu.edu/reports/05tn029.pdf`.

[171] G. Lewis, E. Morris, and D. Smith. Analyzing the reuse potential of migrating legacy components to a service-oriented architecture. In *European Conference on Software Maintenance and Reengineering (CSMR'06)*, pages 9–18. IEEE.

[172] G. Lewis, E. Morris, and D. Smith. Service-oriented migration and reuse technique (SMART). In *Workshop on Software Technology and Engineering Practice*, pages 222–229. IEEE, 2005.

[173] G. Lewis and D. Smith. Service-oriented architecture and its implications for software maintenance and evolution. In *FoSM'08.*, pages 1–10. IEEE, 2008.

[174] G. Lewis and D. Smith. Research challenges in the maintenance and evolution of service-oriented systems. In A. D. Ionita, M. Litoiu, and G. Lewis, editors, *Migrating Legacy Applications: Challenges in Service Oriented Architecture and Cloud Computing Environments*, pages 13–39. IGI Global, 2012.

[175] G. Lewis, D. Smith, N. Chapin, and K. Kontogiannis. Maintenance and evolution of service-oriented systems and cloud-based environments (mesoca'09). Technical Report 1424448972, SEI, 2009.

[176] S. Li, S. Huang, D. Yen, and C. Chang. Migrating legacy information systems to web services architecture. *Journal of Database Management*, 18(4):1–25, 2007.

[177] S. Li and L. Tahvildari. E-bus: a toolkit for extracting business services from java software systems. In *Companion of the 30th international conference on Software engineering (ICSE-I'08)*, pages 961–962. ACM, 2008.

[178] Z. Li, X. Anming, Z. Naiyue, H. Jianbin, and C. Zhong. A soa modernization method based on tollgate model. In *International Symposium on Information Engineering and Electronic Commerce(IEEC'09)*, pages 285–289. IEEE, 2009.

[179] B. P. Lientz and E. B. Swanson. Software maintenance management: a study of the maintenance of computer application software in 487 data processing organizations. 1980.

[180] B. P. Lientz, E. B. Swanson, and G. E. Tompkins. Characteristics of application software maintenance. *Communications of the ACM*, 21(6):466–471, 1978.

[181] J. Liu, D. Batory, and C. Lengauer. Feature-oriented refactoring of legacy applications. In *28th International Conference on Software Engineering*, pages 112–121. ACM, 2006.

[182] Y. Liu, Q. Wang, M. Zhuang, and Y. Zhu. Reengineering Legacy Systems with RESTful Web Service. In *Proceedings of the 32nd Annual IEEE International Computer Software and Applications Conference (COMPSAC'08)*, pages 785–790. IEEE, 2008.

[183] Z. Mahmood. The Promise and Limitations of Service-Oriented Architecture. *International Journal of Computers*, 1(3):74–78, 2007.

[184] A. Marchetto and F. Ricca. Transforming a java application in an equivalent web-services based application: toward a tool supported stepwise approach. In *10th International Symposium on Web Site Evolution (WSE'08)*, pages 27–36. IEEE, 2008.

[185] M. L. Markus and C. Tanis. *Framing the domains of IT research: Glimpsing the future through the past*, chapter The enterprise systems experience-from adoption to success, pages 207–173. Pinnaflex Educational Resources Cincinnati, OH, 2000.

[186] A. Martin, R. Biddle, and J. Noble. Xp customer practices: A grounded theory. In *Agile Conference, 2009. AGILE'09.*, pages 33–40. IEEE, 2009.

[187] A. Mehta and G. T. Heineman. Evolving legacy system features into fine-grained components. In *24th International Conference on Software Engineering*, pages 417–427. ACM, 2002.

[188] T. Mens, Y.-G. Gueheneuc, J. Fernandez-Ramil, and M. D'Hondt. Guest editors' introduction: Software evolution. *IEEE Software*, 27(4):22–25, 2010.

[189] Microfocus. Academia needs more support to tackle the it skills gap. Online, 2013. `http://www.microfocus.com/about/press/pressreleases/2013/pr070320131001.aspx`.

[190] Microsoft. The business value of legacy modernization: Custom research note. Technical report, Microsoft, July 2007. Available from: `www.microsoft.com/mainframe`.

[191] R. Millham. Migration of a legacy procedural system to service-oriented computing using feature analysis. In *International Conference on Intelligent and Software Intensive Systems (CISIS'10)*, pages 538–543. IEEE, 2010.

[192] P. Mohagheghi, A. J. Berre, A. Sadovykh, F. Barbier, and G. Benguria. Reuse and migration of legacy systems to interoperable cloud services-the remics project. *Proceedings of Mda4ServiceCloud*, 10, 2010.

[193] M. Mortensen, S. Ghosh, and J. M. Bieman. Aspect-oriented refactoring of legacy applications: An evaluation. *Transactions on Software Engineering*, 38(1):118–140, 2012.

[194] MOSCOW. MoSCoW Prioritisation. Online, 2011. Available from: http://www.coleyconsulting.co.uk/moscow.htm.

[195] H. A. Müller, J. H. Jahnke, D. B. Smith, M.-A. Storey, S. R. Tilley, and K. Wong. Reverse engineering: a roadmap. In *Future of Software Engineering*, pages 47–60. ACM, 2000.

[196] S. Murer, B. Bonati, and F. J. Furrer. *Managed Evolution*. Springer, 2011.

[197] M. Nakamura, A. Tanaka, H. Igaki, H. Tamada, and K.-i. Matsumoto. Constructing home network systems and integrated services using legacy home appliances and web services. *IJWSR*, 5(1):82–98, 2008.

[198] NASCIO. Digital states at risk modernizing legacy systems. Technical report, NASCIO, 2008.

[199] NASCIO. State CIO Top Ten Policy and Technology Priorities for 2014. Online, 2013. http://www.nascio.org/publications/documents/NASCIO_StateCIOTop10For2014.pdf5.

[200] K. Nasr, H. Gross, and A. van Deursen. Adopting and Evaluating Service-Oriented Architecture in Industry. In *14th European Conference on Software Maintenance and Reengineering (CSMR'10)*, pages 11–20. IEEE, 2010.

[201] K. A. Nasr, H.-G. Gross, and A. van Deursen. Realizing service migration in industry: lessons learned. *Journal of Software: Evolution and Process*, 2011.

[202] K. A. Nasr, H.-G. Gross, and A. van Deursen. Realizing service migration in industry–lessons learned. *Journal of Software: Evolution and Process*, 25(6):639–661, 2013.

[203] P. Newcomb and G. Kotik. Reengineering procedural into object-oriented systems. In *20th Working Conference on Reverse Engineering (WCRE'95)*, pages 237–237. IEEE Computer Society, 1995.

[204] NYTimes. ING to Cut 1,700 Jobs and Invest $248 Million in Digital Push. Online. http://dealbook.nytimes.com/2014/11/25/ing-to-cut-1700-jobs-invest-248-million-in-digital-push/?_r=0.

[205] L. O'Brien, D. Smith, and G. Lewis. Supporting migration to services using software architecture reconstruction. In *13th IEEE International Workshop on Software Technology and Engineering Practice(STeP'05)*, pages 81–91. IEEE, 2005.

[206] M. Papazoglou. *Web services: principles and technology*. Addison-Wesley, 2008.

[207] M. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: a research roadmap. *International Journal of Cooperative Information Systems*, 17(2):223–255, 2008.

[208] M. Papazoglou and W. Van Den Heuvel. Service-oriented design and development methodology. *International Journal of Web Engineering and Technology*, 2(4):412–442, 2006.

[209] M. P. Papazoglou. The challenges of service evolution. In *Advanced Information Systems Engineering*, pages 1–15. Springer, 2008.

[210] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: State of the art and research challenges. *Computer*, 40(11):38–45, 2007.

[211] D. L. Parnas. Software aging. In *Proceedings of the 16th international conference on Software engineering (ICSE'94)*, pages 279–287. IEEE Computer Society Press, 1994.

[212] M. Perepletchikov, C. Ryan, K. Frampton, and Z. Tari. Coupling metrics for predicting maintainability in service-oriented designs. In *18th Australian Software Engineering Conference (ASWEC'07)*, pages 329–340. IEEE, 2007.

[213] R. Pérez-Castillo. Marble: Modernization approach for recovering business processes from legacy information systems. In *28th IEEE International Conference on Software Maintenance (ICSM'12)*, pages 671–676. IEEE, 2012.

[214] R. Pérez-Castillo, I. G.-R. de Guzmán, O. Ávila-García, and M. Piattini. Marble: a modernization approach for recovering business processes from legacy systems. In *International Workshop on Reverse Engineering Models from Software Artifacts (REM'09)*, pages 17–20, 2009.

[215] M. Petticrew and H. Roberts. *Systematic reviews in the social sciences: A practical guide*. John Wiley & Sons, 2008.

[216] M. Polan. Web service provisioning. Online, January 2002. Available from: `http://www.ibm.com/developerworks/library/ws-wsht`.

[217] A. Quilici. Reverse engineering of legacy systems: A path toward success. In *17th International Conference on Software Engineering*, pages 333–336. ACM, 1995.

[218] V. Rajlich. Software evolution and maintenance. In *Proceedings of the on Future of Software Engineering*, pages 133–144. ACM, 2014.

[219] M. Rambold, H. Kasinger, F. Lautenbacher, and B. Bauer. Towards autonomic service discovery: a survey and comparison. In *SCC'09*, pages 192–201. IEEE, 2009.

[220] B. Ramesh, J. Pries-Heje, and R. Baskerville. Internet software engineering: A different class of processes. *Annals of Software Engineering*, 14(1-4):169–195, 2002.

[221] M. Razavian. *Knowledge-driven Migration to Services*. PhD thesis, Vrije Universiteit Amsterdam, 2013.

[222] M. Razavian and P. Lago. A frame of reference for SOA migration. In *Towards a Service-Based Internet*, pages 150–162. Springer, 2010.

[223] M. Razavian and P. Lago. A survey of SOA migration in industry. In G. Kappel, Z. Maamar, and H. R. Motahari-Nezhad, editors, *Service-Oriented Computing*, volume 7084 of *LNCS*, pages 618–626. Springer, 2011.

[224] M. Razavian and P. Lago. A lean and mean strategy for migration to services. In *WICSA/ECSA'12*, pages 61–68. ACM, 2012.

[225] RBS. RBS IT Failure Cost Hits £175m. Online, 2012. `http://www.techweekeurope.co.uk/workspace/rbs-it-failure-175m-98130`.

[226] P. Reason. Three approaches to participative inquiry. 1994.

[227] V. Reddy, A. Dubey, S. Lakshmanan, S. Sukumaran, and R. Sisodia. Evaluating legacy assets in the context of migration to SOA. *Software Quality Journal*, 17(1):51–63, 2009.

[228] G. Reijnders, R. Khadka, S. Jansen, and J. Hage. Developing a legacy to soa migration method. Technical Report UU-CS-2011-008, Department of Information and Computing Sciences, Utrecht University, 2011.

[229] F. Ricca and A. Marchetto. A quick and dirty meet-in-the-middle approach for migrating to soa. In *Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops*, pages 73–78. ACM, 2009.

[230] R. Robbes and M. Lanza. A change-based approach to software evolution. *Electronic Notes in Theoritical Computer Science*, 166:93–109, 2007.

[231] E. Roch. Soa benefits, challenges and risk mitigation. Online, May 2006. Available from: `http://it.toolbox.com/blogs/the-soa-blog/soa-benefits-challenges-and-risk-mitigation-8075`.

[232] N. J. Roese and K. D. Vohs. Hindsight bias. *Perspectives on Psychological Science*, 7(5):411–426, 2012.

[233] P. Runeson and M. Höst. Guidelines for conducting and reporting case study research in software engineering. *Emp. Soft. Eng.*, 14(2):131–164, 2009.

[234] A. Saeidi, J. Hage, R. Khadka, and S. Jansen. Gelato: Generic language tools for model-driven analysis of legacy software systems. In *20th Working Conference on Reverse Engineering (WCRE'13)*, pages 481–482. IEEE, 2013.

[235] A. Saeidi, J. Hage, R. Khadka, and S. Jansen. Itmviz: Interactive topic modeling for source code analysis. In *23rd IEEE International Conference on Program Comprehension (ICPC'15)*, pages xx–xx. IEEE, 2015.

[236] A. M. Saeidi, J. Hage, R. Khadka, and S. Jansen. A search-based approach to multi-view clustering of software systems. In *Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on*, pages 429–438. IEEE, 2015.

[237] R. Salama and S. G. Aly. A decision making tool for the selection of service oriented-based legacy systems modernization strategies. In *International Conference on Software Engineering Research and Practice (ICSERP'08)*, 2008.

[238] J. Schelp and S. Aier. Soa and ea-sustainable contributions for increasing corporate agility. In *42nd Hawaii International Conference on System Sciences (HICSS'09)*, pages 1–8. IEEE computer society, 2009.

[239] D. Scholler. Hype cycle for application architecture. TR Tech. Rep. G00200962, Gartner, Inc, 2012.

[240] Scitools. Understand tool. Online, 2011. Available at: `http://www.scitools.com/index.php`.

[241] R. C. Seacord, D. Plakosh, and G. A. Lewis. *Modernizing legacy systems: software technologies, engineering processes, and business practices.* Addison-Wesley Professional, 2003.

[242] C. Seaman. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on Software Engineering*, 25(4):557–572, 1999.

[243] J. Singer. Practices of software maintenance. In *International Conference on Software Maintenance (ICSM'98)*, pages 139–145. IEEE, 1998.

[244] D. Smith, L. OBrien, and J. Bergey. Using the options analysis for reengineering (oar) method for mining components for a product line. In *Software Product Lines*, pages 316–327. Springer, 2002.

[245] D. B. Smith, H. A. Müller, and S. R. Tilley. The year 2000 problem: Issues and implications. Technical Report CMU/SEI-97-TR-002, CMU/SEI, 1997.

[246] H. Sneed. Planning the reengineering of legacy systems. *Software*, 12(1):24–34, 1995.

[247] H. Sneed. COB2WEB a toolset for migrating to web services. In *10th International Symposium on Web Site Evolution (WSE'08)*, pages 19–25. IEEE, 2008.

[248] H. Sneed. A pilot project for migrating COBOL code to web services. *International Journal on Software Tools for Technology*, 11(6):441–451, 2009.

[249] H. M. Sneed. Migration of procedurally oriented cobol programs in an object-oriented architecture. In *Proceedings in the International Conference on Software Maintenance (ICSM'92.*

[250] H. M. Sneed. Software renewal: A case study. *IEEE Software*, 1(3):56–63, 1984.

[251] H. M. Sneed. Understanding software through numbers: A metric based approach to program comprehension. *Journal of Software Maintenance: Research and Practice*, 7(6):405–419, 1995.

[252] H. M. Sneed. Encapsulating legacy software for use in client/server systems. In *Proceedings of the 3rd Working Conference on Reverse Engineering (WCRE'96)*, pages 104–119. IEEE, 1996.

[253] H. M. Sneed. Object-oriented cobol recycling. In *Proceedings of the 3rd Working Conference on Reverse Engineering (WCRE'96)*, pages 169–178. IEEE, 1996.

[254] H. M. Sneed. Risks involved in reengineering projects. In *6th Working Conference on Reverse Engineering*, pages 204–211. IEEE, 1999.

[255] H. M. Sneed. Encapsulation of legacy software: A technique for reusing legacy software components. *Annals of Software Engineering*, 9(1-2):293–313, 2000.

[256] H. M. Sneed. Encapsulation of legacy software: A technique for reusing legacy software components. *Annals of Software Engineering*, 9(1-2):293–313, 2000.

[257] H. M. Sneed. Integrating legacy software into a service oriented architecture. In *Proceedings of 10th European Conference on Software Maintenance and Reengineering (CSMR'06)*, pages 3–14. IEEE, IEEE Computer Society, 2006.

[258] H. M. Sneed. Migrating from COBOL to Java. In *International Conference on Software Maintenance*, pages 1–7. IEEE, 2010.

[259] H. M. Sneed and R. Majnar. A case study in software wrapping. In *Proceedings of the International Conference on Software Maintenance (ICSM'98)*, pages 86–93. IEEE, 1998.

[260] H. M. Sneed and S. H. Sneed. Creating web services from legacy host programs. In *WSE'03*, pages 59–65. IEEE, 2003.

[261] T. Souder and S. Mancoridis. A tool for securely integrating legacy systems into a distributed environment. In *Proceedings of 6th Working Conference on Reverse Engineering (WCRE'99)*, pages 47–55. IEEE, 1999.

[262] SQuaRE. Systems and software quality requirements and evaluation (SQuaRE)– system and software quality models. Standard ISO/IEC 25010, ISO, 2011.

[263] A. Strauss and J. Corbin. *Basics of qualitative research: Grounded theory procedures and techniques*, volume 1. Sage, 1990.

[264] E. Stroulia, M. El-Ramly, and P. Sorenson. From legacy to web through interaction modeling. In *18th International Conference on Software Maintenance*, pages 320–329. IEEE, 2002.

[265] T. Sucharov and P. Rice. The burden of legacy. Online: `http://www.ncc.co.uk/article/?articleid=15665`, 2008.

[266] A. Sulaiman, N. I. Jaafar, and S. Mohezar. An overview of mobile banking adoption among the urban community. *International Journal of Mobile Communications*, 5(2):157–168, 2007.

[267] E. B. Swanson. The dimensions of maintenance. In *Proceedings of the 2nd international conference on Software engineering*, pages 492–497. IEEE Computer Society Press, 1976.

[268] P. Thiran, J.-L. Hainaut, G.-J. Houben, and D. Benslimane. Wrapper-based evolution of legacy information systems. *Transactions on Software Engineering and Methodology*, 15(4):329–359, 2006.

[269] S. Thorne. Data analysis in qualitative research. *Evidence Based Nursing*, 3(3):68, 2000.

[270] S. R. Tilley, D. B. Smith, and S. Paul. Towards a framework for program understanding. In *4th International Workshop on Program Comprehension (WPC'96)*, pages 19–28. IEEE, 1996.

[271] M. Torchiano, M. Di Penta, F. Ricca, A. De Lucia, and F. Lanubile. Migration of information systems in the Italian industry: A state of the practice survey. *Information and Software Technology*, 53(1):71–86, 2011.

[272] M. C. Tremblay, A. R. Hevner, and D. J. Berndt. The use of focus groups in design science research. In *Design Research in Information Systems*, pages 121–143. Springer, 2010.

[273] A. Umar and A. Zordan. Reengineering for service oriented architectures: A strategic decision model for integration versus migration. *Journal of Systems and Software*, 82(3):448–462, 2009.

[274] I. van de Weerd and S. Brinkkemper. *Handbook of Research on Modern Systems Analysis and Design Technologies and Applications*, chapter Meta-modeling for situational analysis and design methods, pages 38–58. Idea Global Publishing, 2008.

[275] I. van de Weerd, S. de Weerd, and S. Brinkkemper. Developing a reference method for game production by method comparison. In J. Ralyt, S. Brinkkemper, and B. Henderson-Sellers, editors, *Situational Method Engineering: Fundamentals and Experiences*, volume 244 of *IFIP International Federation for Information Processing*, pages 313–327. Springer Boston, 2007.

[276] A. van Deursen, P. Klint, and C. Verhoef. Research issues in the renovation of legacy systems. In *2nd International Conference on Fundamental Approaches to Software Engineering*, pages 1–21. Springer, 1999.

[277] J. Van Geet and S. Demeyer. Lightweight visualisations of COBOL code for supporting migration to SOA. *Symposium on Software Evolution*, 8, 2008.

[278] J. Van Geet and S. Demeyer. Reverse engineering on the mainframe: Lessons learned from" in vivo" research. *IEEE Software*, 27(4):30–36, 2010.

[279] R. Van Noorden. The trouble with retractions. *Nature*, 478(7367):26–28, 2011.

[280] M. van Sinderen. Challenges and solutions in enterprise computing. *Enterprise Information Systems*, 2(4):341–346, 2008.

[281] M. van Sinderen. Challenges and solutions in enterprise computing. *Enterprise Information Systems*, 2(4):341–346, 2008.

[282] M. van Sinderen and M. Spies. Towards model-driven service-oriented enterprise computing. *Enterprise Information Systems*, 3(3):211–217, 2009.

[283] N. Veerman. Revitalizing modifiability of legacy assets. *Journal of Software Maintenance and Evolution: Research and Practice*, 16(4-5):219–254, 2004.

[284] P. Vemuri. Modernizing a legacy system to SOA-feature analysis approach. In *IEEE Region 10 Conference TENCON*, pages 1–6. IEEE, 2008.

[285] G. Visaggio. Ageing of a data-intensive legacy system: symptoms and remedies. *Journal of Software Maintenance and Evolution: Research and Practice*, 13(5):281–308, 2001.

[286] I. Warren and D. Avallone. *The renaissance of legacy systems*. Springer, 1999.

[287] J. Webster and R. T. Watson. Analyzing the past to prepare for the future: Writing a literature review. *Management Information Systems Quarterly*, 26(2):3, 2002.

[288] B. W. Weide, W. D. Heym, and J. E. Hollingsworth. Reverse engineering of legacy code exposed. In *17th International Conference on Software Engineering*, pages 327–331. ACM, 1995.

[289] N. H. Weiderman, J. K. Bergey, D. B. Smith, and S. R. Tilley. Approaches to legacy system evolution. TR CMU/SEI-97-TR-O14, DTIC Document, 1997.

[290] M. Weiser. Program slicing. In *Proceedings of the 5th International Conference on Software Engineering (ICSE'81)*, pages 439–449. IEEE, 1981.

[291] R. Wieringa. Design science as nested problem solving. In *Proceedings of the 4th international conference on design science research in information systems and technology*, page 8. ACM, 2009.

[292] R. Wieringa and J. Heerkens. The methodological soundness of requirements engineering papers: a conceptual framework and two case studies. *Requirements engineering*, 11(4):295–307, 2006.

[293] C. Wohlin. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, page 38. ACM, 2014.

[294] WSO2. Wso2/c++ web service framework. Online, 2011. Available from: `http://wso2.com/products/web-services-framework/cpp/`.

[295] B. Wu, D. Lawless, J. Bisbal, J. Grimson, V. Wade, D. O'Sullivan, and R. Richardson. Legacy systems migration-a method and its tool-kit framework. In *APSEC & ICSC'97*, pages 312–320, 1997.

[296] B. Wu, D. Lawless, J. Bisbal, R. Richardson, J. Grimson, V. Wade, and D. OSullivan. The butterfly methodology: A gateway-free approach for migrating legacy information systems. In *3rd IEEE International Conference on Engineering of Complex Computer Systems*, pages 200–205. IEEE, 1997.

[297] R. Yin. *Case study research: Design and methods*. Sage Publications, Inc, 2009.

[298] Z. Zhang, R. Liu, and H. Yang. Service identification and packaging in service oriented reengineering. In *International Conference on Software Engineering and Knowledge Engineering (SEKE'05)*, pages 219–26, 2005.

[299] Z. Zhang and H. Yang. Incubating services in legacy systems for architectural migration. In *Asia-Pacific Software Engineering Conference (APSEC'04)*, pages 196–203. IEEE, 2004.

[300] Z. Zhang, H. Yang, and W. Chu. Extracting reusable object-oriented legacy code segments with combined formal concept analysis and slicing techniques for service integration. In *Proceedings of 6th International Conference of Software Quality (QSIC'06)*, pages 385–392. IEEE, 2006.

[301] Z. Zhang, D.-D. Zhou, H.-J. Yang, and S.-C. Zhong. A service composition approach based on sequence mining for migrating e-learning legacy system to soa. *International Journal of Automatic Computing*, 7:584–595, 2010.

[302] J. Ziemann, K. Leyking, T. Kahl, and D. Werth. Soa development based on enterprise models and existing it systems. In P. Cunningham, editor, *Exploiting the Knowledge Economy: Issues, Applications and Case Studies*. IOS Press, Amesterdam, 2006.

[303] C. Zillmann, A. Winter, A. Herget, W. Teppe, M. Theurer, A. Fuhr, T. Horn, V. Riediger, U. Erdmenger, U. Kaiser, et al. The SOAMIG Process Model in Industrial Applications. In *European Conference on Software Maintenance and Reengineering (CSMR'11)*, pages 339–342. IEEE, 2011.

[304] Y. Zou and K. Kontogiannis. Migration to object oriented platforms: A state transformation approach. In *Proceedings in the International Conference on Software Maintenance (ICSM'02)*, pages 530–539. IEEE, 2002.

List of Figures

List of Tables

Publication List

Journal

R. Khadka, B. Sapkota, L. Ferreira Pires, M.J. van Sinderen and S. Jansen. *Model-driven Approach to Enterprise Interoperability at the Technical Service Level.* Computer in Industry, 64 (8) 951-965, Elsevier, 2013.

T. Baars, R. Khadka, H. Stefanov, S. Jansen, R. Batenburg and E. van Heusden E. *Chargeback for cloud services.* Future Generation Computer Systems. 41: 91-103, Elsevier, 2014.

Conference

R. Khadka, G. Reijnders, A. Saeidi, S. Jansen, and J. Hage. *A method engineering based legacy to SOA migration method.* In the 26th IEEE International Conference on Software Maintenance (ICSM 2011), pages 163–172. IEEE, 2011.

R. Khadka, A. Saeidi, A. Idu, J. Hage, and S. Jansen. *Legacy to SOA evolution: a systematic literature review.* In A. D. Ionita, M. Litoiu, and G. Lewis, editors, Migrating Legacy Applications: Challenges in Service Oriented Architecture and Cloud Computing Environments, pages 40–71. IGI Global, 2012.

R. Khadka, G. Reijnders, A. Saeidi, S. Jansen, and J. Hage. *A structured legacy to SOA migration process and its evaluation in practice.* In the IEEE 7th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA 2013), pages 2–11. IEEE, 2013.

R. Khadka, A. Saeidi, S. Jansen, J. Hage, and G. Haas. *Migrating a large scale legacy application to SOA: Challenges and lessons learned.* In the 20th Working Conference on Reverse Engineering (WCRE 2013), pages 425–432. IEEE, 2013.

R. Khadka, B. V. Batlajery, A. Saeidi, S. Jansen, and J. Hage. *How do professionals perceive legacy systems and software modernization?* In the 36th International Conference on Software Engineering (ICSE 2014), pages 36–47. ACM, 2014.

R. Khadka, P. Shrestha, B. Klein, A. Saeidi, S. Jansen, J. Hage, E. van Dis, and M. Bruntink. *Does software modernization deliver what it aimed for? A post modernization analysis of five software modernization case studies.* In the 31st International Conference on Software Maintenance and Evolution (ICSME 2015), pages 477–486. IEEE, 2015.

A. Saeidi, J. Hage, R. Khadka, and S. Jansen. *A Search-based Approach to Multi-View Clustering of Software Systems.* In the proceedings of the 22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER 2015), pages 429–438, IEEE, 2015.

R. Khadka, A. Saeidi, S. Jansen, J. Hage, and R. Helms. *An Evaluation of Service Frameworks for the Management of Service Ecosystems.* In the 15th Pacific Asian Conference on Information Systems (PACIS 2011), Brisbane, 2011.

R. Khadka, B. Sapkota, L. Ferreira Pires, M. J. van Sinderen, and S. Jansen. *Model-Driven Development of Service Compositions for Enterprise Interoperability.* In the Proceedings of the International

IFIP Working Conference on Enterprise Interoperability (IWEI 2011), Stockholm, Sweden, pages 23-24, Springer, 2011.

Workshop

R. Khadka, B. Sapkota. *An Evaluation of Dynamic Web Service Composition Approaches.* In the 4th International Workshop on Architectures, Concepts and Technologies for Service Oriented Computing (ACT4SOC 2010), pages 67-79, SciTePress, 2010.

R. Khadka, B. Sapkota, L. Ferreira Pires, M. J. van Sinderen, and S. Jansen. *WSCDL to WSBPEL– A Case Study of ATL-based Transformation.* In the 3rd International Workshop on Model Transformation with ATL (MtATL 2011), CEUR, 2011.

H. Stefanov, S. Jansen, R. Batenburg, E. van Heusden, and R. Khadka. *How to do Successful Chargeback for Cloud Services.* In the 8th International Workshop on the Economics and Business of Grids, Clouds, Systems, and Services (GECON 2011), pages 61-75, Springer, 2011.

Tools

A. Saeidi, J. Hage, R. Khadka, R. and S. Jansen. *ITMViz: Interactive Topic Modeling for Source Code Analysis.* In the proceedings of the 23rd IEEE International Conference on Program Comprehension (ICPC 2015), pages 295-298, IEEE, 2015.

A. Saeidi, J. Hage, R. Khadka, and S. Jansen. *GElATO: GEneric LAnguage TOols for Model-Driven Analysis of Legacy Software Systems.* In the proceedings of the 20th Working Conference on Reverse Engineering (WCRE 2013), pages 481-482, IEEE, 2013.

Summary

Legacy software systems are those that significantly resist modification and evolution while still being valuable to its stakeholders, to the extent that their failure has a detrimental impact on business. Despite several drawbacks, the world still depends on legacy software systems for vital societal functions, from banking to health. Even though these systems have collected seemingly insurmountable technical debt, legacy software systems keep providing valuable and useful features to support critical business processes. Legacy software systems and their modernization have been extensively researched. After decades of research, a plethora of methods and techniques has been proposed to facilitate legacy software system modernization. These modernization methods are predominantly aimed at providing a technical solution to facilitate understanding or assisting the modernization process. However, a legacy software modernization method should also consider business and organizational perspectives. In the current body of knowledge, such a consolidated modernization method is missing. Furthermore, despite numerous software modernization methods, legacy systems are still vital to run day-to-day processes in industry. Research indicates that legacy modernization methods reported in academia do not fit the purposes of industry. To address the aforementioned two issues, this dissertation consists of two research questions and hence, the dissertation is divided into two parts.

The first research question of this dissertation reads:

How can a modernization process be designed that facilitates enterprises in modernizing software systems?

To answer this question, the first part focuses on developing a systematic legacy software modernization method, particularly aimed at modernizing towards a Service-Oriented Architecture (SOA). To develop a systematic method, we initially investigate the combination of technical and business perspectives of legacy to SOA modernization. We leverage the method engineering approach to combine method fragments from different existing modernization methods in Chapter 2. The consolidated legacy to SOA modernization method is evaluated and enhanced by interviewing experts and further evaluated with two case studies. In Chapter 3, we aim at systematically investigating techniques and methods that are reported in academia regarding legacy to SOA software modernization. We use the systematic literature review method to document a historical overview of legacy to SOA modernization approaches and provide an overview of available methods and techniques used. In Chapter 4, we develop a phase-wise structured method for legacy to SOA modernization. For each phase, we present a rationale to justify the need of it, current practices, and challenges that require further attention. This research is based on the rationale that there is a need for a structured legacy to SOA modernization method that incorporates not only the technical issues but also the business and organizational issues.

The answer to the first research question is a software modernization method that combines technical and business aspects of software modernization to SOA. Initially, we developed a method by identifying essential steps to combine technical and business aspects. The next step was to investigate the state of the art of legacy to SOA modernization methods. The software modernization method is then further extended using the state of the art of software modernization in academia. An overview of the literature, analyzed in a structural manner, is used in to gather a historical overview methods and techniques used in software modernization leading to a development of a phase-wise structured method. The results of the overview indicate that less attention is given to business aspects of legacy software system modernization

in academic research.

The second research question of this dissertation reads:

What are the perceptions of practitioners about software modernization?

To answer the second research question, the second part focuses on understanding how legacy software systems and their modernization are perceived in industry. In particular, we investigate what characteristics of legacy software systems still keep them operational, what are the key drivers for modernization, what key challenges are faced in the modernization process and what business objectives/goals are met after conducting legacy software modernization. To establish a context, Chapter 5 starts with investigating a large scale legacy to SOA modernization case in a Dutch financial institution with the aim of understanding what methods and techniques are used and what challenges are faced in an industrial context. In Chapter 6, we investigate how legacy systems and their modernization are perceived in industry. We conduct interviews with 26 practitioners to understand what keeps them operational in industry, what drivers lead to modernization and what challenges are faced during modernization? We use a grounded theory approach to analyze the interviews and then use a structured survey to triangulate the findings. In Chapter 7, we investigate what it means for a legacy system modernization to be "successful" from a business perspective. As of now, legacy software modernization is claimed to be successful when the technical modernization is completed. However, there has been limited research on investigating the post-modernization results

The findings of the second research question provide a different viewpoint of legacy software system and its modernization. Practitioners value their legacy software system despite acknowledging the well-known drawbacks of legacy systems. It is interesting to note that challenges and motivations of legacy software modernization in industry acknowledge the business aspects in addition to technical aspects. The post–modernization result from the five retrospective case studies has revealed a new area of research in software modernization. Based on the post-modernization investigation of five cases, the results indicate that most of the pre-modernization business goals are met.

This dissertation revisits legacy software system modernization and highlights the importance of business aspects of software modernization along with the technical ones. These perspectives are aligned in a structured legacy software modernization method. Additionally, this dissertation also highlights the industrial viewpoint of legacy modernization by exploring how legacy systems and their modernization are perceived in practice. To summarize, this dissertation has contributed to three key research areas, (i) establishing the importance of business aspects of software modernization, (ii) documenting industrial perspective of software modernization, and (iii) identifying a new research domain within software modernization to document post software modernization impacts.

Nederlands Samenvatting

Legacy software-systemen zijn systemen die moeilijk te modificeren en evolueren zijn terwijl ze nog steeds waardevol zijn voor belanghebbenden, in die zin dat hun falen een nadelig effect op het bedrijf heeft. Ondanks meerdere nadelen is de wereld nog steeds afhankelijk van legacy software-systemen voor essentile maatschappelijke functies, van het bankwezen tot de gezondheidszorg. Ondanks dat deze systemen een schier onoverkomelijke technische bagage hebben opgebouwd, blijven legacy software-systemen waardevolle en nuttige functies bieden voor de ondersteuning van kritische bedrijfsprocessen. Legacy software-systemen en hun modernisering zijn extensief onderzocht. Na tientallen jaren onderzoek, is een plethora van methoden en technieken voorgesteld om de modernisering van legacy softwaresystemen te faciliteren. Deze moderniseringsmethoden zijn voornamelijk gericht op het leveren van technische oplossingen om het moderniseringsproces te begrijpen of assisteren. Een legacy software-moderniseringsmethode zou echter ook organisationele perspectieven moeten incorporeren. Binnen de huidige kennis mist een dergelijke moderniseringsmethode. Verder zijn legacy-systemen, ondanks vele software moderniseringsmethoden, nog steeds essentiel in dagelijkse processen in de industrie. Onderzoek wijst uit dat legacy-moderniseringsmethoden uit de wetenschap niet overeenkomen met de doelen van de industrie. Om deze bovengenoemde twee problemen te behandelen, bestaat deze dissertatie uit twee onderzoeksvragen en uit twee delen.

De eerste onderzoeksvraag luidt:

Hoe kan een moderniseringsproces worden ontworpen dat bedrijven faciliteert bij het moderniseren van softwaresystemen?

Om deze vraag te beantwoorden richt het eerste deel zich op het ontwikkelen van een systematischce legacy software-moderniseringsmethode, die speciaal gericht is op het moderniseren van een Service-Oriented Architecture (SOA). Om een systematische methode te ontwikkelen, onderzoeken we in de eerste plaats de combinatie van technische en bedrijfsperspectieven op legacy-naar-SOA-modernisering. We gebruiken de method engineering aanpak om methodefragmenten van verschillende betaande moderniseringsmethoden te combineren in hoofdstuk 2. De geconsolideerde legacy-naar-SOA-moderniseringsmethode wordt gevalueerd en verbeterd door middel van het interviewen van experts en twee case studies. In hoofdstuk 3 richten we ons op het systematisch onderzoeken van technieken en methoden uit de wetenschap m.b.t. legacy-naar-SOA-modernisering. We gebruiken de systematic literature review methode om een historisch overzicht te vormen van legacy-naar-SOA-moderniseringsmethoden en -technieken. In hoofdstuk 4 ontwikkelen we een gefaseerde gestructureerde methode voor legacy-naar-SOA-modernisering. Voor iedere fase presenteren we een verklaring om de noodzaak ervan, de huidige praktijk, en toekomstige uitdagingen te verdedigen. Dit onderzoek is gebaseerd op de aanname dat er behoefte is aan een gestructureerde legacy-naar-SOA-moderniseringsmethode die niet alleen technische problemen behelst maar ook bedrijfskundige en organisationele.

Het antwoord op de eerste onderzoeksvraag is een software-moderniseringsmethode die technische en bedrijfskundige aspecten van legacy-naar-SOA modernisering combineert. In de eerste stap ontwikkelden we een methode door essentile stappen te identificeren om technische en bedrijfsmatige aspecten te combineren. Deze software-moderniseringsmethode is verder uitgebreid door middel van state-of-the-art moderniseringskennis uit de wetenschap. Een overzicht van de literatuur wordt gebruikt om een gefaseerde gestructureerde methode te ontwikkelen. Dit overzicht laat zien dat er minder aandacht

gegeven wordt aan bedrijfsmatige aspecten van legacy software-modernisering in academisch onderzoek.

De tweede onderzoeksvraag van deze dissertatie luidt:

Wat zijn de percepties van gebruikers over softwaremodernisering?

Om de tweede onderzoeksvraag te beantwoorden, richt het tweede deel zich op begrijpen hoe legacy software-systemen en hun modernisering in de industrie beschouwd worden. We onderzoeken de perceptie in de industrie van legacy-systemen en legacy software-modernisering. In het bijzonder onderzoeken we welke karakteristieken van legacy software-systemen ervoor zorgen dat ze operationeel gehouden worden, wat de key drivers voor modernisering zijn, welke uitdagingen aangegaan worden in het moderniseringsproces, en welke bedrijfsmatige doelen worden behaald na legacy software-migratie. Om een context te geven begint hoofdstuk 5 met het onderzoeken van een grootschalige legacy-naar-SOA-moderniseringscase in een Nederlands financiel instituut, met als doel om te begrijpen welke methoden en technieken gebruikt worden en welke uitdagingen aangegaan worden in een industrile context. In hoofdstuk 6 onderzoeken we hoe legacy-systemen en hun modernisering worden beschouwd in de industrie. We interviewen 26 gebruikers om te begrijpen waarom ze operationeel gehouden worden in de industrie, welke drivers tot modernisering leiden en welke uitdagingen aangegaan worden tijdens modernisering. We gebruiken een grounded theory-aanpak om de interviews te analyseren en vervolgens een gestructureerde enqute om de resultaten te trianguleren. In hoofdstuk 7 onderzoeken we wat het betekent voor een legacy-systeemmodernisering om "succesvol" te zijn vanuit een bedrijfskundig perspectief. Momenteel wordt een legacy-softwaremodernisering als succesvol beschouwd als de technische modernisering compleet is. Er is echter weinig onderzoek gedaan naar resultaten na afloop van de modernisering.

De uitkomsten van de tweede onderzoeksvraag bieden een verschillend perspectief op legacy software-systemen en hun modernisering. Gebruikers waarderen hun legacy software-systemen ondanks dat ze zich bewust zijn van bekende nadelen. Het is interessant om te bemerken dat uitdagingen en motivaties van legacy software-modernisering in de industrie zowel de bedrijfsmatige als de technische aspecten erkennen. De postmoderniseringsresultaten uit de vijf retrospectieve case studies heeft een nieuw onderzoeksgebied in softwaremodernisering blootgelegd. De postmoderniseringsresultaten wijzen uit dat de meeste pre-moderniseringsdoelen behaald worden.

Deze dissertatie herbeschouwt de modernisering van legacy software-systemen en belicht het belang van bedrijfsmatige aspecten van softwaremodernisering naast de technische. Deze perspectieven worden naast elkaar gezet in een gestructueerde legacy software-moderniseringsmethode. Daarnaast belicht deze dissertatie ook het industrile perspectief van legacy-modernisering door te onderzoeken hoe legacy-systemen en hun modernisering in de praktijk worden beschouwd. Samenvattend draagt deze dissertatie bij aan meerdere onderzoeksgebieden: (i) het bepalen van het belang van bedrijfsmatige aspecten van softwaremodernisering, (ii) het documenteren van het industrile perspectief op softwaremodernisering, en (iii) het identificeren van een nieuw onderzoeksdomein binnen softwaremodernisering om de impact van software modernisering te documenteren.