POLYTECHNIQUE MONTREAL
affiliated to University of Montreal

Service Identification To Support The Migration Of Legacy Systems To SOA

MANEL ABDELLATIF
Département de Software Engineering And Computer Science

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*
Génie informatique

May 2021

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Cette thèse intitulée :

Service Identification To Support The Migration Of Legacy Systems To SOA

Présentée par **Manel ABDELLATIF**
en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

DEDICATION

All my beloved family,
Thank you for your endless love, sacrifice and support.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisors Dr. Yann-Gaël Guéhéneuc and Dr. Naouel Moha, for their continuous support and encouragement during my PhD. Their positive outlook and constructive feedbacks in my research inspired me and gave me confidence. I would like to thank them for their endless guidance and the sleepless nights we spent working together before deadlines. I would also thank Dr. John Mullin for accepting to supervise my thesis.

I also extend thanks to Dr. Hafedh Mili and Dr. Ghizlane El Boussaidi for their insightful comments and suggestions that made an important part of this thesis.

I also extend thanks to the members of my examination committee: Dr. David Lo, Dr. Hanifa Boucheneb and Dr. Martine Bellaïche for taking the time to review my work.

I would like to thank various interns for their excellent work during their internship to conduct our experiments and investigating results. I would like to thank all the people participating in the experiments.

I thank all the members of Ptidej team and Latece, especially Anas Shatnawi, Rafik Tighilt and Geoffrey Hecht for the simulating discussions, collaborations and all the activities that we did together.

My endless thanks go to my parents, my grandmother, my brother and my sister for their love, encouragement and sacrifices. Without them, I could not get through the hard times that I faced during my PhD. Thank you for all of the life lessons that taught me how to become who I am now.

I do not know how to say thank you to my soul mate, my dearest husband and my best friend for his never ending encouragement and unfailing love. It has taken a long time to achieve this goal, and he has always been there for me; "thank you for every thing, loving you has been my favourite adventure so far".

Finally, this thesis would not have been possible without the financial support of Fonds de recherche du Québec (FRQNT) and he Ministry of Higher Education and Scientific Research in Tunisia, for which I am incredibly grateful.

RÉSUMÉ

L'évolution des systèmes logiciels est devenue une activité centrale dans de nombreuses entreprises. Malgré leurs défis bien connus, les systèmes légataires demeurent un élément vital et important dans de nombreuses entreprises, car les connaissances intégrées dans ces systèmes est d'une valeur significative. Les systèmes légataire ne peuvent pas être simplement supprimés ou remplacés car ils exécutent généralement de manière efficace et précise une logique métier critique et complexe. Outre leurs avantages bien connus, les systèmes légataires souffrent de plusieurs inconvénients liés par exemple à leur coût de maintenance, leur évolutivité et leur flexibilité. Par conséquent, il existe un besoin croissant de migrer les logiciels légataires vers des plates-formes plus flexibles et modernes sans perdre leurs valeurs commerciales .

La migration des systèmes légataires vers l'architecture orientée services (SOA) est considérée comme l'une des alternatives promises pour la modernisation des systèmes légataires. En effet, l'infrastructure SOA en général a permis de développer des applications complexes et inter-organisationnelles en intégrant des composants fonctionnels réutilisables, relativement indépendants, généralement hétérogènes et distribués.

L'identification des services est considérée comme le processus le plus difficile dans le processus de migration. Il consiste à identifier à partir des systèmes légataires les fonctionnalités de service potentielles et les artefacts réutilisables qui peuvent être regroupés tout en ayant une logique métier précieuse. Le défi d'un tel processus est d'identifier à partir des systèmes légataires les services potentiels qui peuvent être développés de manière rentable, sont adaptés à la réutilisation, faciles à entretenir et permettent de personnaliser les applications migrées finales par une sélection et une orchestration appropriées des services.

Cette thèse soutient la migration des systèmes légataires vers SOA en (1) analysant l'état des pratiques d'identification de services dans l'académie et l'industrie tout en étudiant l'écart entre les approches d'identification de services dans les deux domaines, et (2) en proposant une approche d'identification des services sensible au type qui repose sur l'analyse du code source, car d'autres sources d'informations peuvent ne pas être disponibles ou ne pas être synchronisées avec le code réel. Notre approche ascendante basée sur le code utilise des critères de regroupement guidé par les types de services. Nous utilisons une catégorisation des types de services qui s'appuie sur des taxonomies de services et décrivons des règles de détection caractérisant les services ainsi que leur types à partir de l'analyse statique du code source.

ABSTRACT

The evolution of software systems has become a central activity in many businesses. Despite their well-known challenges, legacy systems are still a vital and important component in many enterprises because the knowledge embedded in such systems is often of significant values. Legacy systems cannot be simply removed or replaced as they effectively and accurately execute critical and complex business logic. Besides their well-known advantages, legacy systems still suffer from several drawbacks that are related to their maintenance cost, scalability, flexibility, etc. Therefore, there is a rising need for migrating legacy software to more flexible and modern platforms without losing their business values.

The migration of legacy systems to Service-Oriented Architecture (SOA) is considered one of the promised alternatives for legacy system modernization. In fact, SOA infrastructure in general have made possible to develop complex and inter-organizational applications by integrating functional components that are reusable, relatively independent, generally heterogeneous, and distributed.

Service identification is considered as the most challenging process in the overall migration process. It consists of identifying from legacy software systems potential service functionality and reusable artifacts that may have valuable business logic. The challenge of such process is to identify from legacy software systems potential services that can be developed in a cost-effective manner, are suitable for reuse, easy to maintain, and provide capability to customize end migrated applications by proper selection and orchestration of services.

This thesis supports the migration of legacy systems to SOA by (1) analyzing the state of the practices of service identification in academia and industry while studying the gap between academic and industrial SIAs, and (2) proposing a type-sensitive service identification approach that relies on source code analysis, because other sources of information may be unavailable or out of sync with the actual code. Our bottom-up, code-based approach uses service-type specific functional-clustering criteria. We use a categorization of service types that builds on published service taxonomies and describes the code-level patterns characterizing types of services.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

A2A	Architecture2Architecture
AD	Activity diagram
BPM	Business Process Model
DB	Database
DFD	Data Flow Diagram
Doc	documentation
Hu.Exp	Human Expertise
LogT	Log Traces
Ont	Ontology
SMD	State Machine Diagram
SC	Source Code
SIA	Service Identification Approach
SOA	Service-Oriented Architecture
UAI	User Application Interaction
UC	Use Case

Publications

Portions of the material in this dissertation have previously appeared in the following publications.

1. Manel Abdellatif, Rafik Tighilt, Naouel Moha, Yann-Gaël Guéhéneuc, Hafedh Mili, Ghizlane Elboussaidi, Jean Privat: Identifying Reusable Services in Legacy Object-oriented Systems: A Type-sensitive Identification Approach. IEE Transactions on Software Engineering (submitted)

2. Manel Abdellatif, Anas Shatnawi, Hafedh Mili, Naouel Moha, Ghizlane El-Boussaidi, Geoffrey Hecht, Jean Privat, Yann-Gaël Guéhéneuc: A taxonomy of service identification approaches for legacy software systems modernization. J. Syst. Softw. 173: 110868 (2021)

3. Manel Abdellatif, Rafik Tighilt, Naouel Moha, Hafedh Mili, Ghizlane El-Boussaidi, Jean Privat, Yann-Gaël Guéhéneuc: A Type-Sensitive Service Identification Approach for Legacy-to-SOA Migration. ICSOC 2020: 476-491

4. Anas Shatnawi, Hafedh Mili, Manel Abdellatif, Yann-Gaël Guéhéneuc, Naouel Moha, Geoffrey Hecht, Ghizlane El-Boussaidi, Jean Privat: Static Code Analysis of Multilanguage Software Systems. CoRR abs/1906.00815 (2019)

5. Geoffrey Hecht, Hafedh Mili, Ghizlane El-Boussaidi, Anis Boubaker, Manel Abdellatif, Yann-Gaël Guéhéneuc, Anas Shatnawi, Jean Privat, Naouel Moha: Codifying Hidden Dependencies in Legacy J2EE Applications. APSEC 2018: 305-314

6. Manel Abdellatif, Geoffrey Hecht, Hafedh Mili, Ghizlane El-Boussaidi, Naouel Moha, Anas Shatnawi, Jean Privat, Yann-Gaël Guéhéneuc: State of the Practice in Service Identification for SOA Migration in Industry. ICSOC 2018: 634-650

7. Anas Shatnawi, Hafedh Mili, Ghizlane El-Boussaidi, Anis Boubaker, Yann-Gaël Guéhéneuc, Naouel Moha, Jean Privat, Manel Abdellatif: Analyzing program dependencies in Java EE applications. MSR 2017: 64-74

The following publications are not directly related to the material in this dissertation, but they were produced in parallel to the research contained for this dissertation.

1. Manel Abdellatif, Rafik Tighilt, Abdelkarim Belkhir, Naouel Moha, Yann-Gaël Guéhéneuc, Éric Beaudry:A multi-dimensional study on the state of the practice of REST APIs usage in Android apps. Autom. Softw. Eng. 27(3): 187-228 (2020)

2. Rafik Tighilt, Manel Abdellatif, Naouel Moha, Hafedh Mili, Ghizlane El-Boussaidi, Jean Privat, Yann-Gaël Guéhéneuc: On the Study of Microservices Antipatterns: a Catalog Proposal. EuroPLoP 2020: 34:1-34:13

3. Abdelkarim Belkhir, Manel Abdellatif, Rafik Tighilt, Naouel Moha, Yann-Gaël Guéhéneuc, Éric Beaudry: An observational study on the state of REST API uses in Android mobile applications. MOBILESoft@ICSE 2019: 66-75

CHAPTER 1 Introduction

1.1 Research Context

During the past decades, there has been an evolution in computing and information technology. Technological advances and the use of information technology increased at a rapid pace. These advances present many significant opportunities for enterprises to grow and to expand, especially when offering new ways of conducting businesses. The current economic situation has, in part, been made possible by advances in information and communications technology, which have reduced the cost and increased the speed of communications across the globe, reducing the existing barriers of time and space. These advances impacted all areas of economic and social life.

The actual business environment is characterized by rapid changes in business models, mergers and acquisitions, laws and regulations, and changes in the supporting information and communication technology infrastructure [1]. It is also characterized by mobility, globalization, and increased competition [2]. It leads businesses to keep modernizing their information systems to cope with new business requirements and improve their business operations.

Lehman's law of software evolution claims that, without active countermeasures to software aging, the quality of a software system gradually degrades as the system evolves. Besides, productivity of software organizations and software quality generally fall short of expectations as software systems suffer bugs and symptoms of aging. Failure to make continuous, gradual, and remedial changes on these systems makes them costly to operate and hard to evolve. Without explicit and immediate support for evolution and modernization, software systems become impossibly complex and unreliable. They become *legacy systems*.

Yet, legacy systems are vital in many businesses because the knowledge embedded in these systems is hidden but of significant value. They cannot simply be removed or replaced because they effectively and accurately execute critical and complex business logic despite high maintenance costs, poor flexibility, and scalability issues [3]. Therefore, there is a need for modernizing legacy systems, without losing their business values, to more flexible, modern and loosely coupled architectures such as *Service Oriented Architectures* (SOA).

1.2 Thesis Statement

The migration of legacy systems to Service Oriented Architecture (SOA) is one avenue for modernizing legacy systems. SOA enables the development of complex, inter-organizational, yet flexible applications by integrating services that are reusable, relatively independent, generally heterogeneous, and distributed [4]. However, the migration of legacy systems to SOA is a difficult and complex process that must consider many factors: the choice of a migration process, the service identification method, the quality measurements of the generated services, the implementation and integration challenges [5–7], etc.

The migration of a legacy system to a SOA usually follows six typical steps [8] that include:

1. Legacy system understanding: this step aims at acquiring information about the features and functionalities of the legacy system;
2. Target system understanding: this step enables the design of major components of the target SOA, the standards to be used, the expected quality of services, etc.;
3. Migration feasibility determination: this step concerns legacy-to-SOA migration feasibility from technical, economical, and organizational perspectives to mitigate the risk of migration failure;
4. Service identification, this step consists in identifying reusable groupings—clusters of functionalities in the system that qualify as *candidate services* in the target architecture to promote software reuse and avoid development from scratch;
5. Implementation and integration, during which step developers write the services in the target SOA;
6. Service deployment and maintenance, this steps includes activities such as service publishing, service discovery, versioning, testing, etc.

Service identification (SI) is one of the most critical steps of the migration process because it establishes the foundation for the later steps and the development of the target SOA system [8, 9]. The identified services must meet a range of expectations concerning their capability, quality of service, and efficiency of use [3].

We identified the three following potential problems from the literature related to service identification:

Problem 1: Gap between academia and industry for the service identification approaches Due to the importance of service-identification approaches (SIAs) and their impact on the success of legacy migrations to SOA, the literature proposed several approaches for identifying services in legacy systems. The selection of a SIA that is suitable for some practitioners among all other SIAs is however difficult and depends on several factors, e.g., the available legacy artifacts, the process of analyzing these legacy artifacts, the available inputs, the desired outputs and the usability degree of the approach. As a result, practitioners need a comprehensive view of existing SIAs to select the one fulfilling their needs.

Problem 2: No highly-automated service identification approach that only relies on source-code analysis Service identification consists in identifying reusable groupings/-clusters of functionalities in the legacy system that qualify as *candidate services* in the target architecture. Several SIAs have been proposed in the literature [10–16]. However, they have limited identification accuracy. They often require several types of inputs (e.g., business process models, use cases, activity diagrams, etc.) that may not be available with legacy systems and they are not highly automated.

Problem 3: A lack of type-sensitive service identification approach that only relies on static analysis of the source-code of legacy systems Many existing source-code SI approaches use similar *functional-clustering criteria*, typically *cohesion* and *coupling*, which lead to candidate services that are often architecturally irrelevant for the new SOA-based system. Yet, there exists service types that could improve the identification accuracy by narrowing the search space through the types and their associated code-patterns. Service types could be used to classify service candidates according to a hierarchical-layered schema and offers the possibility to prioritize the identification of specific types of services according to the business requirements of the migration process.

Based on these problems, we formulate our thesis statement as follows:

There is a gap between academia and industry in their service identification approaches to migrate legacy systems to SOA. To fill this gap, service identification should use static analyses of the source code and be driven by service types. Service types could be used to classify service candidates hierarchically and to prioritize the identification of specific types of services according to the business requirements of the migration process.

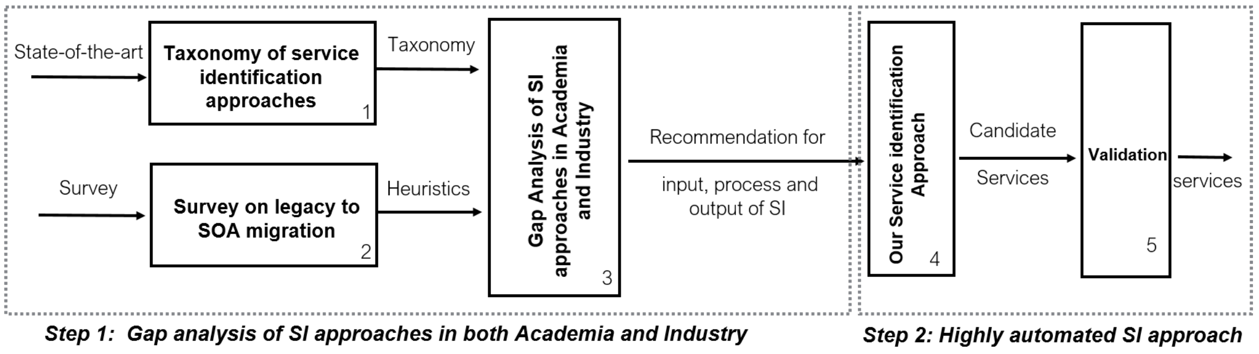

State-of-the-art
Taxonomy of service identification approaches 1
Taxonomy
Survey
Survey on legacy to SOA migration 2
Heuristics
Gap Analysis of SI approaches in Academia and Industry 3
Recommendation for input, process and output of SI
Our Service identification Approach 4
Candidate Services
Validation 5
services
Step 1: Gap analysis of SI approaches in both Academia and Industry
Step 2: Highly automated SI approach

Figure 1.1 Overview of our thesis work

1.3 Research Methodology

To answer our thesis, we propose a type-sensitive service-identification approach to help migrating legacy object-oriented systems to SOA. Figure 1.1 presents an overview of our proposed service identification approach with the five steps.

- **Step 1: Taxonomy of service identification approaches.** We study the state of the practices of SIAs in academia through a systematic literature review. Based on this study we provide a multi-dimensional taxonomy of service identification approaches that consider the used inputs, the applied processes and the generated outputs of academic SIAs.
- **Step 2: Survey on legacy to SOA migration.** We study the state of the practices of SIAs in industry by performing a survey and interviews with practitioners who migrated legacy systems to SOA. We report some migration strategies that have been adopted by the participants. We also study the reasons of migrating legacy systems to SOA as well as the importance of service identification in industrial migration projects. Finally we report the used inputs the applied processes and the generated outputs of industrial service identification approaches.
- **Step 3: Gap analysis of SIAs in academia and industry.** We compare academic and industrial SIAs and derive several derive recommendations for how to identify services in legacy systems to support their migration to SOA.
- **Step 4: A type-sensitive service-identification approach.** Based on the recommendations in step 3, we propose *ServiceMiner* our type-sensitive service identification

approach that relies on the static analysis of the source code of legacy object-oriented systems while identifying specific types of services.

- **Step 5: Validation.** We validate the approach on two case studies. We build *ground truths* for our case studies, i.e., service-oriented versions of the systems to be migrated, against which we compare the results of our identification approach. We compare the identified services to the ground truth to assess the reliability of the approach and finally compare the identification results with three state-of-the-art service identification tools.

1.4 Thesis Contributions

Contribution 1: A systematic literature review on service identification approaches. We propose a systematic literature review (SLR) of published SIAs, with focusing on *bottom-up* and *hybrid* approaches that use existing software artifacts. We chose to focus on bottom-up and hybrid approaches because previous studies [17, 18] and our own preliminary study showed that companies often have only source code as most up-to-date source of information about their legacy software systems. We analysed a total of 41 papers retained from a first set of 3,246 papers. Based on this SLR, we generated a taxonomy of SIAs, i.e., a multi-layer classification of SIAs. This classification helps practitioners in selecting a suitable service identification approach that corresponds to their migration needs.

Contribution 2: Study of the state of the practices of services identification in industry. We surveyed 45 industrial practitioners and interviewed eight of them to collect, analyze, and report their experiences with the migration of legacy systems. Our results showed that reducing maintenance costs and improving the flexibility and interoperability of legacy systems are the main motivations to migrate these systems to SOA. They also showed that SI is perceived by practitioners as an important step for the migration, in particular to identify reusable code in the legacy systems. In addition, they showed that SI is a process driven by business value rather than quality criteria, even though some practitioners consider some quality criteria (mainly reusability, granularity, and loose coupling). Finally, our results showed that SI should be driven by service types and it remains a manual process in which human experts'feedbacks is essential.

Contribution 3: A type-sensitive service identification approach to support the migration of legacy systems to SOA. We propose *ServiceMiner*, a bottom-up SIA, which relies on source-code analysis, because other sources of information may be unavailable or out of sync with the actual code. ServiceMiner relies on a categorization of service types and code-level patterns characterizing types of services. We evaluate *ServiceMiner* on two case studies: an open-source enterprise-scale legacy ERP system, and an inventory management system. Then, we compare our results to those of three state-of-the-art approaches. We show that ServiceMiner identifies architecturally-significant services with, on average, 80.5% precision, 76% recall, and 78.2% F-measure.

1.5 Other Contributions

Beside proposing an approach to identify services, we also studied how services are implemented in practice. We studied and identified microservice antipatterns that developers should avoid when implementing a microservice-based system. We also conducted a multi-dimensional study of the state of the practice of service usage by the clients, and more specifically in mobile apps.

The following contributions are not directly related to our thesis statement, but were realised in parallel to the research presented in this thesis.

Contribution I: Specification and detection of microservices anti-patterns. The software industry is currently moving from monolithic architectures into microservice-based architectures, which involve independent, reusable, and fine-grained services. However, the lack of understanding of the core concepts of microservice architectures may lead to the introduction of poorly designed solutions, called antipatterns. Microservice antipatterns may affect the quality of services and hinder the maintenance and evolution of systems. The specification and detection of microservice antipatterns could help in evaluating and assessing the design quality of such systems. Several research works studied patterns and antipatterns in microservice-based systems. However, the automatic identification of such antipatterns is still in its infancy. We proposed MARS (Microservice Antipatterns Research Software), a fully-automated approach supported by a framework for specifying and identifying microservice antipatterns. Using MARS, we specified and identified 16 microservice antipatterns in 24 microservice-based systems. Results showed that MARS can detect microservice antipatterns with an average precision greater than 68% and a recall greater than 78%.

Contribution II: A multi-dimensional study of the state of the practices of REST APIs usage in Android Apps. REST APIs are gaining a tremendous attraction in industry and a growing usage in mobile platforms. They are well suited for providing content to apps running on small devices, like smartphones and tablets. Several research works studied REST APIs development practices for mobile apps. However, little is known about how Android apps use/consume these APIs in practice. Consequently, we proposed a multi-dimensional study on the state of the practice of REST APIs usage in Android apps. We follow three directions: analysing of Android apps, mining Stack Overflow posts on REST APIs usage in Android apps, and surveying Android developers about their usage of REST APIs in their mobile apps. We (1) built a catalog of Android REST mobile clients practices, (2) proposed an automatic approach to detect these practices, (3) analyzed 1595 Android apps downloaded from the Google Play store, (4) mined 12,478 Stack Overflow posts to study REST APIs usage in Android apps, and (5) conducted an online survey with 118 Android developers to understand their usage of these practices. We reported that only two good practices are widely considered by Android developers when implementing their mobile apps. These practices are network connectivity awareness and JSON versus XML response parsing. We also reported Android developers' recommendations for the use of third-party HTTP libraries and their role in implementing the recommended practices.

1.6 Thesis Organization

This thesis is structured as follows. Chapter 2 presents a systematic literature review about existing service identification approaches. In chapter 3 we study the state of industrial practices related to legacy-to-SOA migration in general and service identification in particular. Chapter 4 presents a comparative analysis of service identification approaches in academia and industry and outlines several recommendations based on this study. In chapter 5, we propose *ServiceMiner*, our type-sensitive service identification approach and discuss its identification results. Finally, in chapter 6, we conclude with some future works while highlighting several recommendations for how to conduct service identification to support legacy-to-SOA migration.

CHAPTER 2 LITERATURE REVIEW

2.1 Introduction

The migration of legacy software systems to SOA is difficult because it depends on many factors, e.g., the choice of the migration process, the service-identification approach, the desired quality characteristics of the generated services, the implementation and integration of the services, etc., which we discuss in details later. Also, the modernization of legacy systems may have some side effects that could affect the expected or claimed benefits of the migration of legacy systems [19, 20]. Such side effects could be the decrease of the system's performance, users resistance to the new technology/system, the unexpected high cost of the modernization, the increasing time to finish the migration, etc.

An organization may adopt one of three strategies to migrate legacy software systems to SOA. It can migrate its legacy systems through a *top-down*, forward-engineering strategy by: (1) performing a high-level decomposition of its domain artifacts, (2) modelling the needed services that will take part of the targeted SOA, (3) implementing those services, and (4) implementing the process that orchestrates all these services.

An organization may also want to use a *bottom-up* strategy to re-engineer its legacy software systems to a service-oriented style by: (1) extracting all the dependencies of their legacy system, (2) mining the existing applications for reusable functionality that could qualify as services, (3) packaging these functions as *services* to enable their reuse and to delete their dependencies to the legacy infrastructures, and (4) rewriting some existing applications to *use* the newly-identified services.

An organization may also adopt a *hybrid* strategy and reuse its legacy artifacts by: (1) grouping the functions of the applications into coarse functional blocks, (2) mapping those functional blocks to available services while deleting their dependencies to the legacy infrastructure, and (3) implementing the process that orchestrates all these services.

Service identification is central to all aforementioned three migration strategies, and has been recognized by practitioners as the most challenging step of the overall migration process [8, 17]. The services identified through a Service Identification Approaches (SIAs) must meet a range of expectations regarding their capabilities, quality of service, efficiency of use, etc. [3], which we also discuss in details later. To the best of our knowledge, all bottom-up and hybrid SIAs focus solely on identifying services in legacy software systems, not in ensuring that they can be then called *"as identified"* by different clients immediately. Indeed, once services become available, multiple clients may call them simultaneously, which may and may not cause problems in the services themselves (because they store some states) or related databases (because they do not take into account multiple clients/tenants). Challenges of turning the identified legacy code with side effects (including *multitenancy*, *data consistency*, and *statefulness*) into autonomous and self-contained services must be considered after their identification, as part of the whole migration process of legacy software systems [21].

Due to the importance of SIAs and their impact on the success of legacy migrations to SOA, the literature proposed several approaches for identifying services in legacy systems. The selection of a SIA that is suitable for some practitioners among all other SIAs is however difficult and depends on several factors, e.g., the available legacy artifacts, the process of analyzing these legacy artifacts, the available inputs, the desired outputs and the usability degree of the approach. As a result, practitioners need a comprehensive view of existing SIAs to select the identification approach fulfilling their needs.

This chapter is based on these papers [22, 23]. In the following, we propose a systematic literature review (SLR) of published SIAs, with focusing on *bottom-up* and *hybrid* approaches that use existing software artifacts. We chose to focus on bottom-up and hybrid approaches because previous studies [17,18] and our own preliminary study showed that companies often have only source code as most up-to-date source of information about their legacy software systems.

Based on this SLR, we also present a taxonomy of SIAs, i.e., a multi-layer classification of SIAs. This classification helps researchers and practitioners in selecting a suitable service identification approach that corresponds to their migration needs.

2.1.1 Research Questions

Through our SLR, we study the SIAs following four dimensions: the used inputs, the applied processes, the resulting outputs, and the usability degree of the approaches. We set out to answer the following research questions:

- **RQ1: What are the inputs used by SIAs?** We aim to identify the different inputs used by SIAs that are based on software systems analyses. We aim to classify the targeted SIAs based on the artifacts used for the identification.
- **RQ2: What are the processes followed by SIAs?** We aim to describe the processes that underlie the service identification approaches reported in the literature. This entails gathering information about, (1) the techniques used to identify candidate

services, (2) the desired quality metrics, (3) the direction of the identification, (4) the automation level, and (5) the type of analysis used.

- **RQ3: What are the outputs of SIAs?** We aim to report information about the generated outputs of service identification approaches in terms of the targeted service types.

- **RQ4: What is the usability of SIAs?** We aim to study the usability degree of service identification approaches in the literature based on the systems used to validate the results, the accuracy of the identification method (when reported), the tool support, and the quality of the reported identification results.

We answer these questions and conclude that the state-of-the art SIAs are still at their infancy. This is due to four main reasons: (1) the lack of validation on real enterprise-scale systems, (2) the lack of tool support, (3) the lack of automation of SIAs, and (4) the lack of assessment of the quality of the identified services. The results also show that the proposed SIAs generally ignore the economic aspects of the identification phase such as the implementation and maintenance costs, the re-factoring costs, and time-to-market issues. We believe that more work should be done to automate state-of-the-art SIAs and consider enterprise-scale systems to validate the proposed approaches. We also believe that regardless of the sought quality attributes, SIAs should provide means to assess the quality of the identified services and consider economic aspects in their identification process.

2.1.2 Outline

The remainder of this chapter is structured as follows. Section 2.2 presents background on services. Section 2.3 describes our SLR methodology. Section 2.4 describes the inputs used by SIAs. Section 2.5 describes the processes that underlie the studied SIAs. Section 2.6 surveys the outputs of SIAs. Section 2.7 describes the usability level of these SIAs. Section 2.8 synthesizes the comparison between the studied SIAs. Section 2.9 describes related work. Finally, Section 2.10 concludes our work.

2.2 Background

2.2.1 Reusability = Usefulness + Usability

The idea of building new applications from reusable artifacts is not new [24]. Such artifacts can be analysis-level artifacts (e.g. software models, analysis patterns), design-level artifacts (e.g. architectural styles, design patterns, reference architectures), source-level artifacts (libraries, frameworks), or executables (compiled code, components, services). Reusability has many advertised advantages, including:

- *Enhanced productivity*: by reusing existing artifacts of level *i*, we save on development tasks *up to* development stage *i*. By reusing analysis models (or patterns), we save on analysis; by reusing executable components/services, we save on analysis, design, coding, and testing of the functionality implemented by the components/services.
- *Enhanced quality*: not only do *domain engineering* methodology suggests that reusable components be thoroughly tested, but re*used* artifacts, whether by design or by accident, will have been tested in *many* different contexts.
- *Enforcement of patterns* inherent in the reusable artifacts, be they analysis level patterns, architectural patterns, design patterns, coding styles, etc.

Mili *et al.* argued in [24] that *reusability* is a combination of two qualities:

- *Usefulness*, which represents the extent to which a reusable artifact can be used in different contexts. A *sorting* function/procedure or a *collection* data structure are examples of very *useful* utility (domain-independent) artifacts.
- *Usability*, which represents the extent to which the functionality is packaged in a way that facilitates its use in those contexts where it is *useful*. If I am programming in Java, a C sort routine is of little use to me.

Service identification deal with issues of *usefulness*. Its aim is to identify those clusters of functionality that are used/invoked in *many* places within the same application, or across *many* different applications. When the clusters are identified and validated, they should be repackaged. Service packaging deals with *usability*, as it attempts to wrap/package that reusable functionality behind *service interfaces*, which will facilitate their reuse (remoteness, technology independence, etc.).

2.2.2 What is a Service?

In the literature, many definitions have been proposed for defining services [25–29]. Each definition describes a service based on different details (e.g., granularity, communication mechanism, composition, etc.). Thomas Erl [29], an SOA pioneer, identified eight characteristics of services :

- *Standardized service contracts.* As software components, services define their capabilities using a standard, implementation-neutral language.

- *Loose coupling.* The services are loosely coupled, and any dependencies are explicitly stated in their service contracts.

- *Abstraction.* Whereas loose coupling refers to dependencies between services, abstraction refers to dependencies between a service provider and a service consumer. The consumer should not depend on the implementation details of the service.

- *Reusability.* Services embody reusable functionality that can service many consumers. In other work, we defined reusability as usefulness and usability (Reference). Usefulness refers to how often the provided functionality is needed while usability refers to how easy it is to use. Usability embodies many aspects, including the existence of (standardized) service contracts (see above), as well as discoverability, composability, and interoperability, discussed below.

- *Autonomy.* From the perspective of the consumer, services should be perceived as self-contained components with total control over their resources and environment. The consumer should be able to assume that the service needs no more than the parameters specified in its service contract to do its job. Naturally, behind the scenes, a service may in turn depend on other services. For example, business services can depend on a layer of shared technical services.

- *Statelessness.* We can understand statelessness of services in two complementary ways. To be able to 'service' many consumers, a service should not have to rely on implicit state information about its consumers; all of the data needed to service a particular consumer's request should be explicitly passed as parameter. The second aspect of statelessness is related to multiple interactions with the same consumer. This means that a consumer can invoke the operations of the service as many times as they want, in any order they wish, and always get the same result. In practice, of course, these two conditions are seldom attainable-and not necessarily desirable. In fact , Erl argues that "*Applying the principle of service statelessness requires that measures of realistically attainable statelessness be assessed, based on the adequacy of the surrounding technology architecture to provide state management delegation and deferral options*" [29].

- *Discoverability.* This refers to the ability of services to document and advertise their capabilities so that service consumers can find them. The documentation of the capabilities of a service needs to be expressed in a domain language that is distinct from the language used to express the service contract.

- *Composability.* This refers to dual capability of services to, a) be composed at arbitrary levels of aggregation to form more complex services, and b) address many needs. This, in turn, influences two design aspects of services, a) the modalities for interacting with the service, and b) the way the capabilities of the service are distributed among its operations.

2.3 Search Methodology

In this section, we describe the design methodology of our systematic literature review as well as the mechanisms and data that we analyse to answer our research questions. We follow the procedures proposed by Kitchenham et al. [30] for performing systematic reviews.

Figure 2.1 depicts our methodology. We first collected research papers based on search queries. We started by identifying relevant query terms based on our research questions and the context of our work: *service identification*, *SOA*, and *migration*. Then, for each keyword, we identified a set of related terms and synonyms using an online synonym finder tool[1]. Based on these terms, we defined the following search string:

> (service identification OR service mining OR service packaging) AND (migration OR modernization OR transformation OR re-engineering) AND (legacy OR existing systems OR Object-Oriented)

We executed this search query in different scientific search engines, such as Google Scholar, ACM Digital Library, and IEEE Xplore Digital Library, Engineering Village, etc.

[1]https://www.synonym-finder.com/

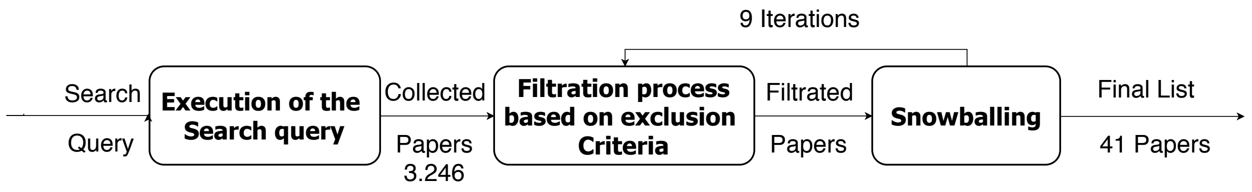


Figure 2.1 Paper selection

Our search queries returned a total of 3,246 unique references. We then filtered these references, first, based on their titles, second, based on their abstracts, and finally, based on their contents. Two of the authors manually and independently at first analyzed all the papers and then reconciled any differences through discussions. We excluded from our review the papers that met one of the following criteria:

- Papers not written in English.
- Papers not related to service identification.
- Papers about *top-down* SIAs.
- Papers that did *not* propose a technique or a methodology for service identification.

Based on these exclusion criteria, we reduced the number of references and retain 26 papers that focus on SIAs that analyze software artifacts. We believe that our search string may not cover all query terms related to service identification (e.g., *microservices*, *decomposition*, *restructuring*, etc.) and thus we risk to miss important studies. To minimize these threats, we (1) included in our search string the most important keywords related to *service identification*, and (2) applied forward and backward snowballing [31, 32] to minimize the risk of missing important papers. Forward snowballing refers to the use of the bibliographies of the papers to identify new papers that are referenced. Backward snowballing refers to the identification of new papers citing the papers being considered. We iterated the backward and forward snowballing and apply for each candidate paper our exclusion criteria. We stopped the iteration process when we have found no new candidate paper. We performed a total of nine iterations and added 15 papers. We thus obtained 41 papers that describe different SIAs, presented in Table 2.1.

2.4 RQ1: What are the inputs used by SIAs?

Using suitable inputs for service identification is crucial to the quality of the identified services and thus the migration process [33]. When it comes to legacy systems, not all software-related artifacts (e.g., use cases, business process models, activity diagrams, etc.) are always available. Consequently, as depicted in Table 2.1, many SIAs in the literature relied on different types of inputs. When considering bottom-up and hybrid approaches, they all use source code or related models, as well as other types of input. We classify the inputs into three main categories: (1) executable models of the systems, (2) non-executable models of the systems, and (3) domain artifacts. We discuss them in turn, below.

Method	Ex. Rep. of the Soft.			Non Ex. Rep. of the Soft.							Domain Artifacts		
				Runtime Artifacts		Model Artifacts							
	SC	DB	TEST	LogT	UAI	BPM	UC	AD	DFD	SMD	Ont	Hu.Exp	Doc
Service Identification Based on Quality Metrics [14]	x												
A spanning tree based approach to identifying web services [13]	x						x	x					
Generating a REST Service Layer from a Legacy System [12]	x												
A service identification framework for legacy system migration into SOA [34]	x					x		x					
Reusing existing object-oriented code as web services in a SOA [35]	x					x							
Mining candidate web services from legacy code [36]	x												x
From objects to services: toward a stepwise migration approach for Java applications [37]	x		x			x							x
Migrating interactive legacy systems to web services [10]					x		x			x			
MDCSIM: A method and a tool to identify services [38]	x					x				x		x	
Reverse engineering relational databases to identify and specify basic Web services with respect to service oriented computing [39]		x											
Identifying services in procedural programs for migrating legacy system to service oriented architecture [40]	x								x				x
A service-oriented analysis and design approach based on data flow diagram [41]	x								x				
Service discovery using a semantic algorithm in a SOA modernization process from legacy web applications [42]	x					x					x	x	
Incubating services in legacy systems for architectural migration [43]	x				x	x							x
Migrating to Web services: A research framework [44]	x												x
Service Identification and Packaging in Service Oriented Re-engineering [45]	x					x							
A wrapping approach and tool for migrating legacy components to web services [46]	x												
Extracting reusable object-oriented legacy code segments with combined formal concept analysis and slicing techniques for service integration [16]	x										x		
Using dynamic analysis and clustering for implementing services by reusing legacy code [47]	x			x		x						x	
Service Mining from Legacy Database Applications [48]		x											
An approach for mining services in database oriented applications [49]		x									x		
Using user interface design to enhance service identification [50]	x				x								
A method to identify services using master data and artifact-centric modeling approach [51]	x	x				x						x	
Multifaceted service identification: Process, requirement and data [15]		x				x						x	
The service modeling process based on use case refactoring [52]	x						x					x	
Extracting reusable services from legacy object-oriented systems [53]	x		x	x			x					x	x
Locating services in legacy software:information retrieval techniques, ontology and FCA based approach [54]	x				x						x		
Microservices Identification Through Interface Analysis [55]	x				x								
Functionality-Oriented Microservice Extraction Based on Execution Trace Clustering [56]	x		x	x	x							x	
Bottom-up and top-down cobol system migration to web services [18]	x											x	x
Extraction of microservices from monolithic software architectures [57]	x												
Service Cutter: A Systematic Approach to Service Decomposition [11]	x						x						
An approach to align business and IT perspectives during the SOA services identification [58]	x					x						x	
Discovering Microservices in Enterprise Systems Using a Business Object Containment Heuristic [59]	x	x		x								x	
A heuristic approach to locate candidate web service in legacy software [60]	x												
Identifying Microservices Using Functional Decomposition [61]	x						x						
Towards the understanding and evolution of monolithic applications as microservices [62]	x												
From Monolithic Systems to Microservices: A Decomposition Framework based on Process Mining [63]	x			x		x							
Function-Splitting Heuristics for Discovery of Microservices in Enterprise Systems [64]		x		x		x	x						
From a Monolith to a Microservices Architecture: An Approach Based on Transactional Contexts [65]	x												
Re-architecting OO Software into Microservices A Quality-centered approach [66]	x												

Table 2.1 Inputs of Service Identification Approaches (***SC*** *for Source Code,* ***DB*** *for Database,* ***LogT*** *for Log Traces,* ***UAI*** *for User Application Interaction,* ***BPM*** *for Business Process Model,* ***UC*** *for Use Case,* ***AD*** *for Activity diagram,* ***DFD*** *for Data Flow Diagram,* ***SMD*** *for State Machine Diagram,* ***Ont*** *for Ontology,* ***Hu.Exp*** *for Human Expertise,* ***Doc*** *for documentation)*

2.4.1 Executable Models

Executable models of the systems include source code and database schemas and test cases.

Source Code

"If the map and the terrain disagree, trust the terrain".
—Swiss Army Aphorism

With legacy systems, documentation (the map) is often missing or out of date. The source code (the terrain) becomes the only reliable source of information about the system. Source code is the most commonly used software artifact by the existing SIAs, due to its availability. SIAs that use source code as input identify business capabilities of the existing legacy systems and expose them as reusable services. Such SIAs rely on reverse and re-engineering processing to (1) extract dependencies between program elements such as variables, functions, modules/classes, etc.; (2) recover other kinds of information such as data flow diagrams, use cases, business process models, state machine diagrams, etc.; (3) map the source code to other artifacts such as business process models, use cases and database schemas, to complete the system map; and, usually, (4) apply clustering techniques to extract reusable services.

For legacy object-oriented systems, some SIAs rely on the relationships among classes to analyze the system structure and identify highly cohesive and loosely coupled reusable parts that could be exposed as services. For example, Adjoyan et al. [14] relied on the analysis of dependencies between the classes of legacy object-oriented software systems. They proposed a fitness function that takes into account the type of relationship between the classes and assigns a score for each relationship. They then applied an agglomerative clustering technique to group classes into candidate services. Aversano et al. [36] mined candidate services from the analysis of legacy source-code. They applied reverse-engineering techniques to extract UML diagrams of systems and analyse the signatures of related methods to identify candidate services.

Other SIAs identify services by analysing the source code of non-object-oriented software systems. For example Rodriguez et al. [18] reported the analysis of a large legacy system in an Argentinian government agency written in COBOL and running on IBM mainframes. They analysed the legacy source code to identify the transactions to be migrated to services. These transactions are then translated into Java code, which is easier to expose as Web services.

Although the identification of candidate services based on source code analysis leads to reusable and fine grained services, a combination of this kind of input with other artifacts (e.g., business processes, documentation, databases, etc.) can be used to identify services with more business values.

Databases

Architecturally, the database layer is important to manage the persistence of data. Database contents, schemas and transactions are the artifacts used by database-related SIAs [39,48,49]. These approaches identify data/entity services that provide access to, and management of, the persistent data of the systems(C.f. Section 2.6).

For example, Baghdadi et al. [39] identified entity services by extracting SQL statements from systems. They then re-factored these statements and added them to the specification of a list of candidate services using *CRUD operations patterns* (Create, Read, Update and Delete). Saha et al. [48] relied on identifying instances of database-access patterns (database related operations) to identify reusable services. Using specific quality metrics, they refined database-related operations and wrapped them into data/entity services. Interactions between the application to migrate and the database have been also used by Del et al. [49] to identify pieces of functionalities that can be exported as services. They performed the identification using clustering techniques and formal concept analysis.

Although the identification of candidate services based on the study of database queries or schema leads to reusable and fine grained services–which can only be entity services (cf. Section 2.6), a forward-engineering process is needed to build more coarse-grained services, that combine these finer-grain services, into business services.

Test Cases

A test case can be defined as a specification of the inputs, execution conditions, testing procedure, and expected output results that must be executed to achieve a testing objective, such as to verify compliance with a specific requirement.

We found only three SIAs that use test cases, among other inputs, to identify reusable services [37, 53, 56]. For example, Bao et al. [53] use test cases as an intermediate input for service identification. They first analysed the legacy system source code and manually identified candidate use cases that correspond to potential reusable services. Then, they derived test cases from these use cases and used them to drive the execution of legacy-software systems. They used dynamic analysis techniques to analyze the execution log traces and generate coarse-grained code segments for each candidate use case that corresponds to an identified service. Also, Jin et al. [56] only used test cases to execute different paths of the system and generate the corresponding log traces. They analysed these log files to get all classes and method invocations of the system. They then applied a clustering algorithm to group high cohesive and loosely coupled group of classes that will be mapped into services.

As shown by Table 2.1, test cases are rarely used by SIAs. They are only used as an intermediate artifact to guide the service identification process, probably because test cases are seldom available, and when they are, they cover only a small portion of the system.

2.4.2 Non-executable Models

We distinguish between two categories of non-executable models: runtime artifacts extracted during the execution of the systems, and non-executable models that describe the architecture of the systems. We discuss them in turn below.

Runtime Artifacts

Runtime artifacts are extracted during the executions of the systems. They contain log traces and user-application interactions (e.g., user interfaces).

Log Traces Execution traces of legacy software systems depict the dynamic behavior of the systems. Six SIAs rely on log traces to extract sequence calls related to specific execution scenarios [47, 53, 56, 59, 63, 64]. These approaches identify pieces of legacy code executed during a set of business processes [47] or use cases [53], which are usually identified manually by business analysts. Then, they suggest those pieces of code as potential implementations of services. For example, Fuhr et al. [47] applied mapping techniques of legacy code to business processes. They used log trace analyses and clustering techniques. They cluster the classes identified in the log traces according to their usage during the business processes.

We note that SIAs do not rely *solely* on log traces to identify services; they usually combine them with other types of inputs such as business process models, use cases, or human expertise.

User Interactions User-interface inputs capture the relationship between users and the system's functionalities. User interfaces usually embody data requirements and workflows [67]. If the workflow model of a system is not available, knowledge extracted from its user interfaces is useful to recover its underlying business logic [43, 50].

We found five SIAs that analyse users' interactions with user interfaces to help identify services [10, 43, 50, 54–56]. For example, Mani et al. [50] proposed an XML-based representation called the Unified User Interface Design Specification (UUIDS) to describe user interfaces, including data bindings and navigation events. They use this representation to automate the analysis of user interfaces and retrieve useful information for candidate services requirements.

The analyses of user interactions help to retrieve navigational information through the operations performed by users. They also help to identify reusable tasks with high business values, which could become services. However, SIAs based on user interactions are hardly automated. Further, they require a model of the tasks, as input, which may not be readily available.

Model Artifacts

Model artifacts abstract the structure and execution behavior of systems. They include business process models, use cases, activity diagrams, and state machine diagrams, which are discussed in turn below.

Business Process Model (BPMs) They describe sets of activities and tasks that accomplish an organizational goal [68]. BPMs have been used extensively by SIAs because of their ability to describe the business logic of legacy software systems at a high-level of abstraction. Business processes can be modeled with the Business Process Model and Notation (BPMN) and executed through their corresponding Business Process Execution Language (BPEL). The decomposition of business processes is a common strategy to identify services [15]. Business process-driven SIAs usually decompose business processes into tasks. These tasks are then clustered and exposed as services.

For example, Alahmari et al. [34] identified services based on analyzing business process models. These business process models are derived from questionnaires, interviews and available documentations that provide atomic business processes and entities on the one hand, and activity diagrams that provide primitive functionalities on the other hand. The activity diagrams are manually identified from UML class diagrams extracted from the legacy code using IBM Rational Rose. Different service granularity levels are distinguished, as they pertain to atomic business processes and entities. Related atomic processes and entities are grouped together within the same service candidates to maximize cohesion of candidate services and minimize coupling between them. Fuhr et al. [47] relied on business process models to correlate classes of legacy object oriented systems. Each activity in the business process model is executed. The classes that are called during the execution of a task are considered to be related. The identification of services is based on a clustering technique where the similarity measurement is based on how many classes are used together in the activity executions.

In the context of service identification, BPMs help to understand and capture the broad functional domains of legacy systems and how they interact with each other. Furthermore, business process-driven approaches identify high-level candidate services (based on process and tasks activities). However, the major problem with relying on BPMs to identify services is that such models are not always available especially for legacy software systems.

Use Cases They help to identify, at a high-level of abstraction, the interactions between users and systems to achieve goals. Use cases depicts functional requirements as well as sequences of actions that can be used for service identification [69]. We found seven SIAs that use such artifact [10, 11, 13, 52, 53, 61, 64].

For example, Bao et al. [53] analyze of the relationships between use-case elements to identify reusable services. They consider independent use cases of object-oriented systems are *candidate* services. If a use case A extends a use case B, they consider B as a candidate service, whereas A is not. Further, if use case A specializes (inherits from) use case B, then A is considered as a candidate service, whereas B is not.

The main reasons for SIAs to rely on use cases is that they offer systematic and intuitive means of capturing functional requirements with a focus on value to the users. However, to the best of our knowledge, SIAs based on use cases are difficult to automate to the extent that they often rely on human expertise.

Activity Diagram They show interactions in systems as well as the different steps involved in executing tasks [70]. Only two SIAs use activity diagrams to identify services [13, 34]. For example, Al Ahmari et al. [34] extracted, from activity diagrams, useful information and transform them to BPMN using mapping rules. They then analysed the business process models to extract reusable services. They used activity diagrams of legacy systems as input but concretely relied on analysing the BPMNs to identify reusable services in the system.

None of the identified SIAs relied only on activity diagrams. Other types of inputs are usually used such as source code, BPMs, and use cases to complement the identification process of candidate services.

Data Flow Diagram A *Data Flow Diagram* (DFD) is a graphical representation of *functional dependencies*, based on the analysis of data flows, between business functions or processes [71]. The main entities of a DFD are (1) the data stores storing data for later use, (2) external entities representing the source or the destination of the data, (3) processes manipulating the data, and (3) the data flows. Only two SIAs use DFDs to identify reusable services [40, 41].

For example, Zhao et al. [41] rely on DFDs to identify services. They start by elaborating DFDs based on the system source code analysis. They recommend to design new DFDs for coarse-grained processes and to delete from the diagrams the fine-grained ones. They map each process of the elaborated DFDs to a service. They finally recommend to design a composite service that will capture the operations provided by identified services and allow these operations to be invoked in a defined workflow structure.

DFDs can describe the business logics of a software system. However, they are not always available nor straightforward to generate from legacy systems. SIAs based on DFDs of ill-structured systems do not guarantee as well the identification of relevant services [40, 41]. Further, DFDs cannot represent dynamic dependencies because they are only based on the source code of software systems.

State Machine Diagram A *State Machine Diagram* (SMD) shows a dynamic view of a system and describes the different states that entities can have during their lifetimes [72]. We found that only two SIAs use state machine diagrams as inputs [10, 38]. Canfora et al. [10] used these diagrams to model the interactions between users and systems, whereas Huergo et al. [38] used them to model the life-cycle of master data, which they define as any information considered to play a key role in the operation of a business.

Although state machine diagrams are ideal for describing the behavior of a limited number of objects, they are not suitable for SIAs that are dealing with large systems due to the state-explosion problem. Further, they are seldom available, and are not easy to obtain from source code or documentation.

2.4.3 Domain Artifacts

Domain artifacts provide knowledge about the application domain of the systems. They include software documentation, human expertise, and ontologies.

Documentation

Software documentation describes and documents systems at different levels of abstraction [73]. Software documentation includes textual descriptions as well as diagrams and models, such as the ones discussed above. Software documentation can guide SIAs by reducing the search space for candidate services by describing key functionalities of the systems. Some SIAs rely on software documentation to better understand the system at hand, which helps to identify reusable services [18, 36, 40, 43, 44, 53]. For example, Aversano et al. [36] proposed a SIA that analyses the Javadoc documentation of systems to calculate lexical similarity between the classes or methods of the systems; they then used that similarity to identify clusters of functionality that can map to services. Rodriguez et al. [18] described an industrial case study in which the documentation of a COBOL system was used to understand the system and to identify business rules in the code.

As with many other inputs (e.g., business process models, log traces, use cases, etc.), software documentation is not always available, and often outdated or out of sync with the source code of legacy systems.

Human Expertise

Human expertise appears in different ways in SIAs. It has been used to fine tune the parameters of various service identification algorithms (see e.g. [13]). It has also been used to define the business logic and translate it into business processes [15, 34, 74]. It is also needed to analyse use cases and identify candidate services [53]. Finally, human expertise is needed to define data flow diagrams of the system to then identify candidate services [40, 41, 44].

Human expertise in SIAs limits the automation of service identification approaches and it appears in most of SIAs at different steps of the identification process.

Ontologies

An ontology is a structured set of terms representing the semantics of a domain, whether through metadata or elements of a knowledge domain [75]. Several SIAs use ontologies to identify services [15, 49, 54, 76].

For example, Djeloul et al. [54] proposed a WordNet-based technique to identify services. They built queries by analysing users interfaces. They then used WordNet to expand the queries and identify pieces of code participating in services. They also used information-retrieval techniques, such as vector-space model and latent-semantic analysis, to map queries to the relevant code.

Chen et al. [76] started by analyzing the source code of systems and used three types of ontologies: a domain concept ontology, a functionality ontology, and a software-component ontology. They used formal and relational concept analysis to map source code of legacy systems to the ontologies they specified to identify candidate services.

The major challenge of ontology-based SIAs lies in defining the proper ontologies for the system. Also, the high cost of developing ontologies in terms of time, effort and resources remain a well-known bottleneck in the ontology development process [77]. Finally, ontology-based SIAs are complex and require a lot of human expertise.

2.5 RQ2: What are the processes followed by SIAs?

A service-identification process applies one or more identification *techniques* (e.g, wrapping, clustering, formal concept analysis, etc.) that target a set of *quality metrics* (e.g, coupling, cohesion, granularity, etc.) based on a predefined identification *direction* (i.e, bottom-up, top-down or hybrid). Human expertise defines the *automation degree* of the process, based on specific *analysis types* (e.g, static, dynamic, lexical, etc.).

2.5.1 Techniques of SIAs

We classified techniques of SIAs into six types:

- Wrapping: A black-box identification technique that encapsulates the legacy system with a service layer without changing its implementation. The wrapper provides access to the legacy system through a service encapsulation layer that exposes only the functionalities desired by the software architect [10, 78].
- Genetic Algorithm: A metaheuristic for solving optimization problems that is based on “natural selection". It relies on the calculation of a fitness function to reach an optimal (or near-optimal) solution. By definition, an optimal solution is a feasible solution where the fitness function reaches its maximum (or minimum) value [79].
- Formal concept analysis (FCA): A method for data analysis where we derive implicit relationships between objects in a formal way. It is also considered as a principled way of grouping objects that have common properties [80]. To use FCA, we should first specify the context denoted by a triple *C=(E, P, R)* where E is a set of finite elements, P is a set of finite properties and R is a binary relation based on E and P. Also a *formal concept* is defined as a grouping of all the elements that share a common set of properties. A partial order could be defined on the formal concepts through the use of *concept lattices* [81] that also offer a structured visualization of the concepts hierarchy.
- Clustering: It consists of classifying and partitioning data into clusters (also called groups, categories or partitions) that share common properties. These clusters are built based on the internal homogeneity of their elements and the external separation between them. In fact, elements in the same cluster should be similar to each other while elements in different clusters should not [82].

- Custom heuristics: Some authors proposed their own heuristic algorithms, instead of using predefined algorithms, to decompose legacy software into SOA.
- General guidelines: In our context, it refers to approaches that only propose best practices, lessons learned or recommendations for service identification.

In the following, we describe and discuss the use of these techniques to identify services from legacy systems.

Wrapping

Wrapping-based SIAs use this technique for encapsulating a legacy system (or subset thereof) with a service layer and exporting its functionalities without changing its implementation [10]. Seven SIAs use/propose wrapping techniques [10,12,18,35,44,46,48]. For example, Canfora et al. [10] proposed a wrapping methodology to expose the interactive functionalities of systems as services. The wrapper acts as an interpreter of a Finite State Automaton (FSA) that describes the interaction model between the system interfaces and their users. Also, Sneed et al. [35] proposed an automatic wrapping technique based on the analysis of the public method interfaces of object-oriented code. They transform the public method interfaces into a relational table. Then based on this table, they generate WSDL interfaces that describe the functionalities of web services. Finally, they generate from the definitions of WSDL service interfaces the corresponding BPEL scripts to manage the service, as well as the corresponding test script to test the service. Wrapping techniques do not require to understand fully the architectures/implementations of the legacy software systems. It avoids the decomposition of the systems into reusable services.However, the underlying systems still must be maintained and so still need legacy expertise.

Genetic Algorithms

We found only three SIAs that rely on Genetic Algorithms to identify services from legacy software systems [13,15,60]. For example, Jain et al. [13] used Genetic Algorithms to identify services in legacy source code. They proposed an identification technique that is based on spanning trees. They used these representations to provide developers with a set of possible solutions for the identification problem. They also used a multi-objective genetic algorithm to refine the initial set of service decompositions. The multi-objective Genetic Algorithm relied on a fitness function that takes into consideration a set of managerial goals (i.e., cost effectiveness, ease of assembly, customization, reusability, and maintainability) to get a near-optimal solution for the service identification problem. Abdelkader et al. [60] proposed also a Genetic Algorithm-based SIA. However, they only take into consideration the functional cohesion of a set of legacy system modules.

Although Genetic Algorithm-based SIAs may yield near-optimal solutions of reusable services, these SIAs do not guarantee to obtain systematically the optimal services that (1) maximize (or minimize) the fitness function, and (2) are architectally relevant for the identification problem. Also, the relevance of the identified services highly depend on the choice of the objectives/managerial goals of the identification.

Formal Concept Analysis

SIAs based on formal concept analysis basically rely on ontologies and/or concept lattices [81] to identify services [16, 49, 76]. These SIAs usually rely on concept lattices to order the identified formal concepts and/or to visualise these concepts as well as the specified ontologies–when used. For example, Zhang et al. [16] used formal concept analysis and program slicing to identify services in object-oriented systems. They begin by mapping the program entities (classes, methods) into elements and properties, using documentation and human expertise. They then applied the Ganter algorithm [83] to build the concept lattices. Finally, they visualized, interpreted and analyzed these concepts to get meaningful, useful, and reusable services. Also, Del et al. [49] identified database-related features to be exported as services. They started by collecting database queries, using the dynamic execution of the database oriented systems. They then performed an analysis of the queries fields (i.e., the `SELECT` and the `FROM` clauses) and constraints (i.e., the `WHERE` clauses). They built a formal context using the concept lattice technique [84]. They used FCA to group related queries into concepts and map them to candidate services.

The big challenge of using FCA for service identification consists in well identifying the concepts related to the entities of legacy systems. A proper setting of the formal context and their entities is required to ensure proper identification of reusable services. Also, the lack of automation in setting the formal context of the system may hinder the use of FCA algorithms to identify services in enterprise-scale systems.

Clustering

SIAs use clustering to group classes or functionalities in legacy systems and consider each group as a candidate service. In general, they combine clustering techniques and custom heuristics. SIAs based on clustering belong to either one of two categories: classes clustering [11, 13–15, 43, 47, 55–57, 59, 61, 62, 64–66] or functionalities clustering techniques [45, 48].

The main clustering techniques used in the literature are k-means [47, 85] and hierarchical-agglomerative clustering [43, 86] .

For example, Zhang et al. [43] proposed an agglomerative hierarchical clustering technique to extract reusable services from object-oriented legacy code. They started by analyzing legacy source code to calculate the similarity between the source code entities. The similarity metric consider the relationship between classes (i.e, inheritance, association, etc.) as well as the semantic similarity between them according to their names. They finally express the results in a dendrogram, which presents a hierarchic view of several possible decompositions of the system into services. Also, Fuhr et al. [47] used k-means clustering techniques to identify services according to their type. The similarity measurement is based on how many classes are used together in a targeted activity execution.

K-means clustering techniques are indeed straightforward to apply. However, their results in the context of service identification show below-average performance. On the other hand, SIAs based on hierarchical clustering techniques do not require to specify in advance the number of the needed clusters/services. However, a subjective choice of the cutting point level in the generated dendrogram is needed to get the final set of services. This could be problematic for enterprise-scale systems where the number of possibilities for cutting points could be important. The choice between K-means and hierarchical clustering depends on the application context where K-means could be a good option when practitioners already know the number of services to be identified. On the other hand, hierarchical clustering is good for the case of unknowing the number of services to be identified. In this case, the hierarchical clustering will partition the system into a number of services based on the inter and intra cluster scaling.

Custom Heuristics

Some SIAs use dedicated heuristics [11, 13, 14, 40, 41, 43, 47, 57] to identify services from legacy systems. Heuristics techniques are usually used with clustering techniques and genetic algorithms. They also rely on quality metrics to identify candidate services.

For example, Adjoyan et al. [14] proposed a fitness function based on three characteristics of services: composability, self-containment, and functionality. They grouped classes from object-oriented legacy software systems using a hierarchical-agglomerative clustering algorithm, which groups classes using the value of the fitness function. Also, Jain et al. [13] proposed a set of heuristics based on dynamic and static relationships among classes in object-oriented systems. Then, they used these heuristics with a multi-objective optimization algorithm to get sets of classes representing services.

Although the use of heuristics is common in SIAs, their main challenge consists in establishing reliable heuristics to guide the process of identifying reusable services.

General Guidelines

We found two works in the literature that propose only general guidelines for service identification [34, 44].

For example, Alahmari et al. [34] proposed to extract UML activity diagrams from legacy systems and perform a model-to-model transformation to obtain BPMN from the diagrams. They argued that having a well defined SOA migration meta-model is important to make the migration process effective. They recommended the use of ad-hoc metrics because they assist in deriving optimal services with suitable granularity. Also Sneed et al. [44] proposed several guidelines for discovering potential services, evaluating these services and extracting their code from legacy systems. They recommended the use of a highly customizable rule based decision making mechanisms to identify which portions of legacy code could be potential services. They also recommended the use of DFDs to analyse data flow of the identified portions of code and decide about its business value.

SIAs based on guidelines propose general ideas to extract services from legacy software systems. They are indeed difficult to validate and automate.

Technique	**SI Method**	**Total**
Wrapping	[10, 12, 18, 35, 44, 46, 48]	7
Genetic Algorithm	[13, 15, 60]	3
Formal Concept Analyses	[16, 49, 54, 76]	4
Clustering	[11, 13–15, 43, 45, 47, 48, 55–57, 59, 61, 62, 64–66],	17
Custom heuristics	[11–14, 36–38, 40, 41, 43, 47, 50, 52, 53, 56–58, 58, 59, 62–66, 76]	25
General Guidelines	[34, 44]	2

Table 2.2 Targeted techniques of SIAs

2.5.2 Quality of Identified Services

Achieving the desired level of quality is critical for service based architectures. As a result, some SIAs use/target some quality metrics/requirements to obtain high-quality candidate services.

Quality Requirements

We describe the quality requirements targeted by the studies SIAs as follows:

- **Reuse:** The ability of a service to participate in multiple service assemblies (compositions) [87]. Better reusability should provide better return of investment (ROI) and shorter development times [88].
- **Maintainability:** Services should ease the effort to modify their implementation, to identify root causes of failures, to verify changes, etc. [89].
- **Interoperability:** The ability of a service to communicate and be invoked by other systems/services implemented in different programming languages [4].
- **Self-containment:** A service should be completely self-contained to be deployed as a single unit, without depending on other services [14].
- **Composability:** Services should be composable with one another to be reused and integrated as services that control other services or that provide functionalities to other services [90].

Quality requirement	**SI Method**	**Total**
Reuse	[15, 18, 37, 43, 44, 52]	6
Maintainability	[45]	1
Interoperability	[44]	1
Self-containment	[14, 43]	2
Composability	[14]	1

Table 2.3 Targeted quality requirements by SIAs

As we can see in Table 2.3, a few SIAs consider quality requirements in their identification techniques. However, service reuse is the most considered requirement by these approaches. On the other hand, we notice that few studies consider the study of composability, self-containment, maintainability, and interoperability of the identified services. This could be because these quality requirements are (1) difficult to characterize and measure and (2) hardly provide useful insights to identify services.

Quality Metrics

We describe the quality metrics targeted by the studied SIAs as follows:

- **Coupling:** The dependencies among services should be minimized and the functionalities should be encapsulated to limit the impact of changes in one service to other services [89].
- **Cohesion:** Cohesion is a measure of the strength of relationship between the programming entities (e.g., modules, classes, functions, etc.) implementing a service and the functionality provided by the service [60].
- **Granularity:** An adequate granularity is a primary concern of SIAs. It can be adjusted to the scope of the functionality offered by the service [38].
- **Total number of services:** SIAs must balance between too many "small" services and not enough services [11].

Quality Metric	**SI Method**	**Total**
Coupling	[13, 14, 34, 38, 40, 41, 43, 45, 51, 55, 61–66]	16
Cohesion	[13–15, 18, 34, 38, 40, 41, 45, 51, 55, 61, 62, 64, 66]	15
Granularity	[18, 37, 38, 40, 41, 43, 45, 51, 52, 55, 61, 62, 65]	13
Number of services	[11, 14, 37, 66]	4

Table 2.4 Targeted quality metrics by SIAs

Table 2.4 shows that state-of-the-art SIAs highly rely on the use of some specific quality metrics such as loose coupling, high cohesion, and granularity. However, these SIAs fail at providing a comprehensive quality model to assess and evaluate the quality of the identified services.

2.5.3 Directions of SIAs

SIAs can follow three directions: top-down, bottom-up, and hybrid.

- A top-down process starts with high-level artifacts, e.g., domain analysis or requirement characterization of systems to define their functionalities. They do not consider low-level artifacts to identify services. Hence, we do not consider further these SIAs in our study.
- A bottom-up process starts with low-level artifacts to maximize code reuse and minimize changes. It extracts more abstract artifacts, e.g., architectures, which can be used to identify candidate services. It can also identify new services that fill implementation gaps or meet new requirements [91].

- A hybrid process combines a top-down and a bottom-up process. It starts from requirements and implementation artifacts to identify the candidate services.

Direction	SI Method	Total
Bottom-up	[11–14, 16, 18, 39–41, 44, 46, 48, 49, 54–57, 59, 60, 62, 66, 76]	22
Hybrid	[10, 15, 34–38, 42, 43, 45, 47, 50, 52, 53, 58, 61, 63–65]	19

Table 2.5 Identification process directions of service identification methods

As we focus in this SLR on SIAs that follow the bottom-up and hybrid direction, we report in Table 2.5 the distribution of SIAs over these two directions. Table 2.5 shows that there are almost equal numbers of bottom-up and hybrid SIAs in the literature. Finally we notice that bottom-up SIAs are more successful at delivering services in the short-term but they usually identify fine-grained services with limited reuse. Moreover, Hybrid SIAs tend to complement and reduce the limitations of bottom-up approaches by also considering requirements.

Analyses Types

Identification approaches may perform static, dynamic, lexical analyses or a combination thereof to identify services.

- Static analysis is performed without executing a software system. Dependencies between classes are potential relationships, like method calls and access attributes. These dependencies are analyzed to identify strongly connected classes, for example, to identify services. [12, 14, 35, 36, 39, 43–46] are examples of identification methods based only on static analysis. The main advantage of static analysis is that it depends only on the source code. It does not address polymorphism and dynamic binding.

- Dynamic analysis is performed by examining the software system at run time. Dependencies between software elements (e.g., class instantiations and accesses [53], function calls [45, 64], relationships between database tables [59], etc.) are collected during the program execution [92]. The execution is performed based on a set of cases that covers the system functionalities, called execution scenarios.

- Lexical analysis techniques suppose that the similarity between the classes should be taken into account during service identification process. This analysis plays the main role in approaches that used features location and textual similarity techniques.

Table 2.6 shows that 76% of SIAs rely on static analysis, 39% on dynamic analysis, and 21% on lexical analysis. Finally we found that 38% rely on a combination of analyses to reduce the limitations of each individual analysis.

Analysis Type	**SI Method**	**Total**
Static analysis	[11–16, 18, 35, 36, 38, 39, 42–46, 48–50, 52, 54, 55, 57–62, 65, 66, 76]	31
Dynamic analysis	[10, 13, 34, 37, 38, 40, 41, 45, 47, 50, 53, 56, 57, 59, 63, 64]	16
Lexical analysis	[16, 36, 42, 43, 49, 54, 55, 57]	8

Table 2.6 Analyses types of SIAs

2.5.4 Automation of SIAs

Automation is the degree to which a SIA needs human experts. We distinguish three levels of automation: manual, semi-automatic, and fully automatic.

- Manual SIAs depend entirely on human experts. They only provide general guidelines to experts to identify services without automating any step of the service identification process [12, 37].
- Semi-automatic SIAs need human experts to perform some of their tasks. For example, Jain et al. [13] proposed a SIA that require a human expert to provide objective functions and specify weights for each of them.
- Automatic SIA do not need any human intervention during the identification process. We did not find any approach in the literature that fully automates the identification of services in existing systems.

Table 2.7 shows that there is a lack of automation of SIAs, especially 88% of the SIAs are semi-automatic or manual.

2.6 RQ3: What are the outputs of SIAs?

In the following, we discuss the output of SIAs in terms of the target service architecture (service-based/microservice-based) and discuss the types of services considered by these approaches.

Analysis Type	**SI Method**	**Total**
Automatic	[11, 39, 48, 58, 60]	5
Semi-automatic	[10, 13–16, 18, 34–36, 38, 40–47, 49, 50, 53–57, 59, 62–66, 76]	32
Manual	[12, 37, 52, 61]	4

Table 2.7 Automation of SIAs

2.6.1 Service Architecture

Service identification approaches aim at identifying services that will be integrated in a SOA.

In the past few years, several SIAs have been interested in identifying microservices—a variant of the service-oriented architecture style—to migrate legacy systems to microservice-based systems [11, 55, 57, 61–66]. For example, Escobar et al. [62] proposed a microservice identification approach to migrate a monolithic Java Enterprise Edition (JEE) application to microservices. They performed a static analysis to cluster session and entity beans into microservices. They started by associating a cluster to each session bean. They grouped these clusters according to a *clustering threshold* that focus on structural coupling and cohesion. The distance between clusters is calculated based on the number of shared entity beans.

Mazlami et al. [57] proposed a microservices identification approach that relies on the analysis of data collected from a version control repository of a monolithic application. They also applied clustering and custom heuristics to extract loosely-coupled and high-cohesive set of classes that will be mapped to microservices. Both semantic and logical coupling metrics were considered by their clustering algorithm. In particular, they combined three metrics to identify microservices: *semantic coupling* (to identify groups of classes that belong to the same domain), *single responsibility principle* (to analyze classes that change together in commits), and *contributor coupling* (to identify classes accessed by the same development team). All these metrics were combined and used by a clustering algorithm to identify groups of classes that belong to the same domain and could represent a microservice.

We notice that microservices identification approaches rely on clustering and custom heuristics to decompose the system into small services. Although the granularity is an important characteristic for qualifying microservices, none of the studied approaches provided a comprehensive model to evaluate whether microservices are identified with the right level of granularity. Also, the granularity difference between services and microservices is still neither well defined nor clearly discussed by the studied microservices identification approaches.

2.6.2 Service Types

In this section, we will describe taxonomies of service types and detail existing type-sensitive SIAs.

2.6.3 Taxonomy of Service Types

We identified only four SIAs that identify specific types of services in existing systems [34,37, 38,47] and nine papers proposing service taxonomies [29,34,37,38,47,93–96], which classify services with hierarchical-layered schemas to support the communication among stakeholders during the implementation of SOAs. These existing taxonomies offer several service types with different classification criteria (e.g., granularity [29,34,93], reuse [47,93,95], etc.) and different names for the same service types. We studied these previous works and identified the following six service types that are generic and cover most of the existing service types.

1. **Business-process services:** (Also called business service [34, 47, 93, 95]), they correspond to business processes or use cases. These are services used by users. These services compose or use the enterprise-task, application-task, and entity services described in the following. Examples of business-process services include flight booking services, hotel booking services and sales order services.

2. **Enterprise-task services:** (Also called capabilities [93]), they are of finer granularity than business-process services. They implement generic business functionalities reused across different applications. Examples of Enterprise-task services include online payment service and tax calculation.

3. **Application-task services:** (Also called task, activity or composite service [34, 37, 39, 95]), they provide functionalities specific to one application. They exist to support reuse within one application or to enable business-process services [93]. Examples of Application-task services include quoting request and invoicing services that take part in the sales order business process of a typical ERP system.

4. **Entity services:** (Also called information or data services [29, 34, 95]), they provide access to and management of the persistent data of legacy software systems. They support actions on data (CRUD) and may have side-effects (i.e., they modify shared data). Examples of entity services include clients, bank accounts, and products.

5. **Utility services:** They do not support directly the business-process services but provide some cross-cutting functionalities required by domain-specific services [38, 47, 95]. Examples of typical utility services include notification, logging, and authentication.

6. **Infrastructure services:** They allow users deploying and running SOA systems. They include services for communication routing, protocol conversion, message processing and transformation [34]. They are sometimes provided by an Enterprise Service Bus (ESB). They are reused in more services than utility services. Examples include publish-subscribe services, message queues, and ESB.

Considering the different categories of services identified above, we propose a multidimensional taxonomy that aims at supporting the identification of services from the legacy source code. The dimensions of this taxonomy are:

- Domain: domain-specific (business) versus domain-neutral (technical).
- Granularity: fine-grained versus coarse-grained. The granularity of a service depends on the complexity of the capability/function the service provides.
- Scope of reuse: Enterprise versus Application. This dimension is dependent on the two previous ones: domain and granularity. Fine-grained services are likely to be more shareable as they may be composed to provide different coarse-grained services that may not belong to the same application. Likewise domain-neutral services, in particular infrastructure services, may be reused across different applications.
- Side-effects: computation-only services versus services with side-effects. Services with side-effects are those that manage the application state; i.e. they modify the persistent data of the application. These are mainly Entity services. To maintain data consistency, these services require the implementation of some transactional and/or compensation mechanisms.

Table 2.8 summarises the properties of the different categories of services according to the dimensions of our taxonomy. In the context of a legacy to SOA migration process, service categories and their properties should be taken into consideration while building a service identification approach.

2.6.4 Type-sensitive SIAs

Most of SIAs identify general services of SOA without specifying different service types, e.g., [14, 26, 45]. Only a few approaches [34, 37, 38, 47] considered the identification of specific types of services in existing systems.

For example, Alahmari *et al.* [34] identified services based on analyzing business process models. These business process models are derived from questionnaires, interviews and available

Service category	Domain	Granularity	Scope of reuse	Side-effect
Business	Specific	Coarse	Application	No
Enterprise	Specific	Variable	Enterprise	No
Application	Specific	Variable	Application	No
Entity	Specific	Fine	Variable	Yes
Utility	Neutral	Variable	Variable	No
Infrastructure	Neutral	Variable	Enterprise	No

Table 2.8 A multidimensional service taxonomy

documentations that provide atomic business processes and entities on the one hand, and activity diagrams that provide primitive functionalities on the other hand. The activity diagrams are manually identified from UML class diagrams extracted from the legacy code using IBM Rational Rose. Different service granularity are distinguished in relation to atomic business processes and entities. Dependent atomic processes as well as the related entities are grouped together at the same service to maximize the cohesion and minimize the coupling. There is no details about how to identify the different service types.

Fuhr *et al.* [47] identified three types of services. These are business, entity and utility services. The services are identified from legacy codes based on a dynamic analysis technique. The authors relied on a business process model to identify correlation among classes. Each activity in the business process model is executed. Classes that have got called during the execution are considered as related. The identification of services is based on a clustering technique where the similarity measurement is based on how many classes are used together in the activity executions. The identified clusters are manually interpreted and mapped into the different service types. Classes used only for the implementation of one activity are grouped into a business service corresponding to this activity. Entity services are composed of clusters of classes that contribute to implement multiple activities but not all of them. A Cluster of classes that are used by all of the activities represent the implementation of utility services. A strong assumption regarding this approach is that business process model should be available to identify execution scenarios.

Marchetto *et al.* [37] proposed a stepwise type-sensitive SIA that extracts reusable services from legacy systems using dynamic analysis of Java source code. They proposed guidelines to identify Utility, Entity, and Task services. They executed several test scenarios and extracted reusable functional groupings qualified as candidate services. They identified Utility services by manually mining non-business functionalities and cross-cutting functionalities, which can be grouped and exposed as candidate Utility services. They extracted candidate Entity services by analyzing persistent objects and the classes using them. Finally, they considered each main functionality of the target application as a possible candidate Task service. They validated the SIA on small Java systems, limiting its generalisability to real enterprise systems. Although the proposed SIA is type-sensitive, identification is manual and based on test scenarios that may not cover all the functionalities of the system.

Huergo *et al.* [38] proposed a method to identify services based on their types. They rely on UML class diagrams of object-oriented systems from which they derive state machines to identify the states of the objects at runtime in the system. They start by manually identifying *Master Data* that they define as classes playing a key role in the operation of a business. Each Master Data is considered a candidate Entity service. Next, they derive state machines related to the identified Master Data. They analyze the transitions of the state machines and identify Task and Process services. This SI method is not fully automated and relies on the manual identification of Master Data in a system.

We notice that there is a lack of SIAs that are type-centric: only four SIAs focus on the identification of specific types of services from legacy systems. These approaches focus on identifying business [34, 38, 47], entity [37, 38] and utility services [34, 37, 38, 47]. Also, none of the studied SIAs tried to identify enterprise-task or infrastructure services through the analysis of legacy systems. These type-centric SIAs do not distinguish in their service identification process between enterprise and application-task services as the scope of reuse of the identified services is not well studied or specified.

2.7 RQ4: What is the usability of SIAs?

Figure 2.2 shows that we consider four elements to estimate the usability of SIAs: validation, accuracy, tool support, and result quality. We then introduce a measure of the usability of the SIAs based on these four elements and their values for each SIA.

2.7.1 Validation

Validation refers to the legacy software systems (if any) on which the SIA was applied. It can be industrial (e.g., real industrial systems), experimental (small, experimental systems), or none at all. We evaluate the usability of a SIA as follows. If the validation is performed on (1) industrial systems, it is “high”; (2) experimental systems, it is “medium”, else (3) it is “low”. We found that only 34% of SIAs were validated on real industrial systems, with most SIAs validated on experimental systems or not validated at all. This lack of industrial validation is a major threat to the applicability of SIAs.

2.7.2 Accuracy/Precision

We assign “high”, “medium”, and “low” values to the accuracy/precision of SIAs. We assign “high” if it is greater than 80%, medium if it is between 50% and 79% in the SIA, and low if it is less than 50%.

Although the accuracy/precision of SIAs is important, we found that only few SIAs have reported accuracy/precision (as depicted in Table 2.9).

2.7.3 Tool Support

Tool support refers to the tool(s) implementing a SIA and their maturity, if any.

We consider the tool support of a SIA as “high” if it is open-source or industry ready, “medium” if it is only a prototype, and “low” if there is little or no tool support.

2.7.4 Result Quality

Result quality is an estimation of the quality of the identified candidate services and whether or not the authors detailed well their proposed SIA. It can be “high”, “medium”, or “low”.

2.7.5 Usability

Method	**ToolSupport**	**Validation**	**Accuracy / Precision**	**Result Quality**	**Usability**
Service Identification Based on Quality Metrics [14]	Prototype	Experimental	Medium	Medium	Medium
A spanning tree based approach to identifying web services [13]	MOGA-WSI	Industry	NA	High	High
Generating a REST Service Layer from a Legacy System [12]	MIGRARIA	Experimental	NA	High	High
A service identification framework for legacy system migration into SOA [34]	Prototype	Experimental	NA	Low	Low
Reusing existing object-oriented code as web services in a SOA [35]	Industry ready	Industry	NA	High	High
Mining candidate web services from legacy code [36]	NA	Experimental	NA	Low	Low
From objects to services: toward a stepwise migration approach for Java applications [37]	NA	Experimental	NA	Low	Low
Migrating interactive legacy systems to web services [10]	NA	Case Study	NA	Medium	Low
MDCSIM: A method and a tool to identify services [38]	MDCSIM	Industry	NA	High	High
Reverse engineering relational databases to identify and specify basic Web services with respect to service oriented computing [39]	CASE	Experimental	NA	Medium	High
Identifying services in procedural programs for migrating legacy system to service oriented architecture [40]	NA	Experimental	NA	Low	Low
A service-oriented analysis and design approach based on data flow diagram [41]	SOAD	Experimental	NA	Low	Medium
Service discovery using a semantic algorithm in a SOA modernization process from legacy web applications [42]	MigraSOA	Experimental	NA	Low	Medium
Incubating services in legacy systems for architectural migration [43]	Prototype	Industry	NA	Low	Medium
Migrating to Web services: A research framework [44]	NA	No Validation	NA	Low	Low
Service Identification and Packaging in Service Oriented Reengineering [45]	Prototype	Case Study	NA	Medium	Medium
A wrapping approach and tool for migrating legacy components to web services [46]	Prototype	Case Study	NA	Low	Low
Extracting reusable object-oriented legacy code segments with combined formal concept analysis and slicing techniques for service integration [16]	Prototype	Experimental	NA	Low	Low
Using dynamic analysis and clustering for implementing services by reusing legacy code [47]	Prototype	Case Study	Meduim	Low	Low
Service Mining from Legacy Database Applications [48]	Prototype	Industry	NA	High	High
An approach for mining services in database oriented applications [49]	Prototype	Industry	High	High	High
Using user interface design to enhance service identification [50]	Prototype	Industry	NA	Medium	High
A method to identify services using master data and artifact-centric modeling approach [51]	NA	Experimental	NA	Low	Low
Multifaceted service identification: Process, requirement and data [15]	Prototype	Experimental	High	Low	Medium
The service modeling process based on use case refactoring [52]	Prototype	Case Study	NA	Low	Low
Extracting reusable services from legacy object-oriented systems [53]	Prototype	Industry	NA	Medium	High
Locating services in legacy software:information retrieval techniques, ontology and FCA based approach [54]	Prototype	Case Study	NA	Low	Low
Microservices Identification Through Interface Analysis [55]	NA	Case Study	NA	Low	Low
Extraction of microservices from monolithic software architectures [57]	Prototype	Industry	NA	High	High
Service Cutter: A Systematic Approach to Service Decomposition [11]	ServiceCutter	Experimental	NA	High	High
Bottom-up and top-down cobol system migration to web services [18]	Industry ready	Industry	NA	High	High
Functionality-Oriented Microservice Extraction Based on Execution Trace Clustering [56]	FOME	Experimental	NA	Low	Medium
An approach to align business and IT perspectives during the SOA services identification [58]	Prototype	Experimental	NA	Low	Low
Discovering Microservices in Enterprise Systems Using a Business Object Containment Heuristic [59]	Prototype	Industry	NA	Medium	High
A heuristic approach to locate candidate web service in legacy software [60]	Prototype	Experimental	NA	Low	Low
Identifying Microservices Using Functional Decomposition [61]	Prototype	Experimental	NA	Low	Low
Towards the understanding and evolution of monolithic applications as microservices [62]	Prototype	Industry	NA	High	High
From Monolithic Systems to Microservices: A Decomposition Framework based on Process Mining [63]	Prototype	Industry	NA	High	High
Function-Splitting Heuristics for Discovery of Microservices in Enterprise Systems [64]	Prototype	Industry	NA	Medium	High
From a Monolith to a Microservices Architecture: An Approach Based on Transactional Contexts [65]	Prototype	Experimental	Medium	Medium	Medium
Re-architecting OO Software into Microservices A Quality-centered approach [66]	Prototype	Experimental	NA	Medium	Medium

Table 2.9 Usability of SIAs

We consider these four preceding elements to estimate the usability of SIAs. We assign to each SIA a usability degree (UD) as follows:

$$UD = \sum_{i=1}^{4} Score_i$$

$Score_i \in \{high = 1, medium = 0, low = -1\}, \forall i \in \{1, .., 4\}$ and refers to validation, accuracy, tool support, and usability, respectively.

$If UD \geq 1, then\ UD = high.$

$If UD = 0, then\ UD = medium.$

$If UD \leq -1, then\ UD = low.$

We tried our best to consider the most important usability criteria and give a rational estimation of the usability degree of the studied SIAs. For example, as shown in Table 2.9, to calculate the usability of the SIA of Rodriguez et al. [12], we studied the scores relative to tool support, validation, identification accuracy, and quality results of the approach. This study has a high tool support through the tool named MIGRARIA (tool-support score is 1). It is validated on an experimental system (validation score is 0). There was no mention of the accuracy/precision of the approach and thus we did not consider associated scores for calculating the usability of the approach. Finally, based on our judgment of the whole approach, we estimated that this SIA has high quality results (quality result is 1). We added all these scores and obtain a usability score of two, which we qualified as a high usability degree.

Table 2.9 shows that 39% of SIAs have a high usability degree while 22% have medium usability, and 39% have low usability. These results show that the studied SIAs are still in their infancy, mainly due to (1) the lack of validation on industrial systems, (2) the lack of estimation of their accuracy/precision, (3) their lack of tool support, and (4) their lack of automation.

2.8 Discussions

In this section, we will discuss our observations about the studied SIAs in terms of the main nodes of our taxonomy: inputs, processes, outputs, and usability.

2.8.1 Validation

Figure 2.2 shows the taxonomy resulting from our answers to the research questions. This taxonomy directly derive from the previous sections.

We believe that the validation of a taxonomy is difficult for several reasons. In fact, it is a tool for researchers and practitioners and, as such, it should be used to assess its strengths and limitations. Also, a taxonomy often cannot be compared against other ones, either because they do not exist or because they have different objectives.

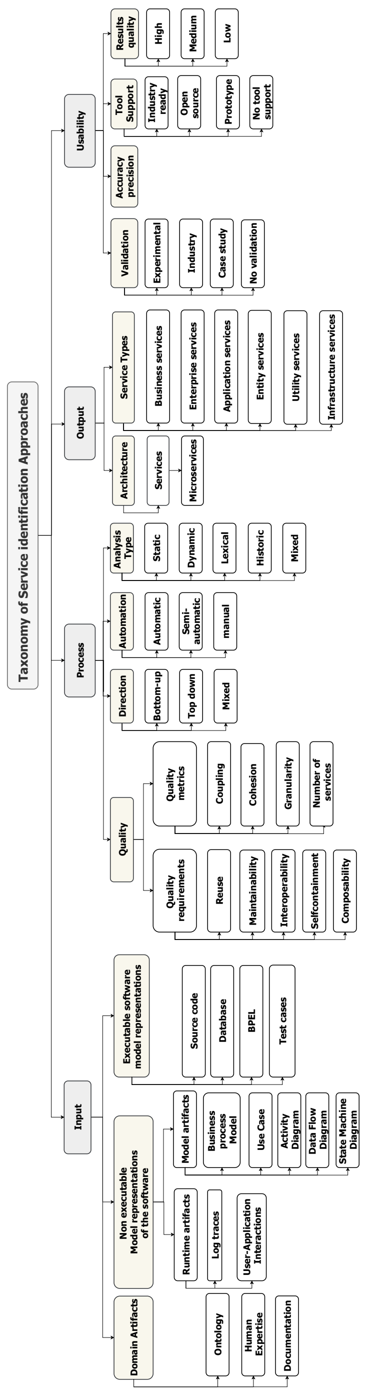


Figure 2.2 Taxonomy of service identification approaches

2.8.2 Inputs

SIAs rely on diverse types of inputs to identify services. We found that the most used inputs are source code and business-process models (BPMs). Combining multiple inputs is also common. The most used combination of inputs are also source code and BPMs [35,37,45,47]. Only 10 SIAs rely on a single input type [12,14,39,46,48,57,60,62,65,66], either source code again or databases.

2.8.3 Processes

Most SIAs rely on clustering and custom heuristics to identify services. The main challenge for these approaches is in using adequate heuristics to identify services.

The success of a SOA depends on the quality of the services. Services with low quality attributes may (1) affect reuse negatively and (2) compromise business agility and reduce return on investment [6]. Quality attributes are therefore important to identify services. However, not all service quality requirements are considered by state-of-the-art SIAs. Moreover, regardless of the adopted quality requirements, SIAs should provide means to assess/control the quality of the candidate services. Also, there are many economic factors that SIAs should take into account. Such aspects could be the implementation and maintenance cost, the refactoring cost of the system, and time-to-market. The economic aspects of the identified methods are widely ignored in the studied SIAs. We believe that more efforts should be done in SIAs to consider as well such economic aspects which play an important role to select the appropriate SIA for an organisation.

2.8.4 Outputs

We noticed that microservices architectures have been gaining a lot of consideration in the past few years as we found many studies focusing on the identification of microservices in legacy systems. The applied identification techniques are quite similar to those used for identifying services. On the other hand, few SIAs focus on the identification of specific types of services. In particular we observed that these SIAs focus on identifying business, entity, and utility services but not enterprise/application-task and infrastructure services. Also, we noticed that these type-sensitive SIAs do not distinguish between enterprise and application task services as the scope of reuse of the identified services is not well specified/studied. We believe that the identification of services according to their types is a challenging problem because (1) we have to build a taxonomy that cover all service types, (2) define detection rules/signature for each service type, and (3) target the metrics or detection rules that are appropriate for each type. We believe that not all service types have distinct signatures as two different service types may leave similar or indistinguishable signatures in the code. The taxonomy of service types may not be representative of all existing service types. To mitigate this threat, we validated our taxonomy through an industrial survey with 45 practitioners who were involved in migration projects of legacy systems to SOA [17]. None of them mentioned the identification of new/other types of services.

2.8.5 Usability

We reported that 51% of the state-of-the-art SIAs have medium or low usability degree due to (1) their lack of validation on real industrial systems, (2) their lack of tool support, and (3) their lack of automation. In particular, most SIAs consider only small examples in their validation, also confirmed by some participants in our survey [17]. The participants reported that a problem exists in the knowledge transfer between academia and industry because of the lack of consideration of enterprise-scaled systems to validate the proposed SIAs in academia.

Finally, we believe that measuring the usability of a given SIA is quite difficult. Our proposed metric may partially measure the usability of a given SIA as we do not cover all possible usability-related aspects. However, we tried our best to consider the most important usability criteria such as the tool support, the quality of SIA results, the validation of the process and the accuracy/precision of the SIA. As future work, we aim to empirically validate our proposed metric of usability with people from academia and industry to study its feasibility of quantifying/estimating the usability degree of a SIA.

2.9 Other SLRs

Several systematic literature reviews and surveys on SIAs have been proposed in the literature. In the period from 2009 to 2021, ten surveys [6, 7, 9, 96–102] on service identification were identified. Although these surveys had different goals, neither of them fully addressed all our research questions. Table 2.10 contains a summary and comparison between the most relevant surveys focusing on service identification in the literature.

For example, Boerner et al. [97], only studied business-driven SIAs techniques and focused on their strategic and economic aspects. They stressed the consideration of economic aspects when identifying services based only on top down approaches. Birkmeier et al. [103] proposed a classification of SIAs between 1984 and 2008. This SLR is indeed old, does not fully addressed our research questions and does not cover recent SIAs. Cai et al. [104] proposed another survey where they identified the most frequent activities in the state-of-the-art SIAs between 2004 and 2011. Then, Vale et al. [100] made a comparison of SIAs and a list of recommendation of the most suitable SI technique according to stakeholders' needs in the Service-Oriented Product Line Engineering context. Bani et al. [7, 9] proposed two different surveys about service identification. In the first one they studied the evaluation frameworks for 24 state-of-the-art SIAs. Then, in the second survey they only identified the challenges of 14 service identification approaches and discuss their limitations. Both studies do not fully address our research questions as we do in our SLR.

Finally Fritcsch et al. [102] provided a classification of refactoring approches of monolithic applications to microservices. They studied ten microservices identification approaches and provided a decision guide for decomposition approaches based on the microservices identification requirements.

Although there are several SLRs on service identification in the literature, none of these surveys fully addressed our research questions. Their focus differ deeply as we cover more in details state-of-the-art service identification approaches in terms of (1) the artifacts used by SIAs, (3) the processes of these approaches,(4) the outputs of these processes, and (5) the usability degree of these approaches. We also propose a taxonomy of SIAs and validate its correctness and coverage with industrial experts in legacy-to-SOA migration through surveys and one-on-one interviews.

SIA	Goal	Year	Coverage	Papers	RQ1	RQ2	RQ3	RQ4
Boerner *et al.* [97]	Business-driven SI techniques comparison with the study of their strategic and economic aspects	2009	2005-2008	5	NA	PA	PA	A
Birkmeier *et al.* [98]	Classification of service identification techniques	2009	1984*-2008	15	PA	A	PA	NA
Gu and Lago [96]	Providing the basic elements of SI to help practitioners selecting the most suitable one basic on their needs	2010	2004-2009	30	A	A	A	NA
Cai *et al.* [104]	Identify frequent used activities done in several SI research works	2011	2004-2011	41	PA	A	PA	NA
Vale *et al.* [100]	Comparison of SI methods and recommendation of the most suitable SI technique according to stakeholders' needs in the Service-Oriented Product Line Engineering context	2012	2005-2012	32	PA	PA	PA	PA
Taei *et al.* [101]	Suitable inputs identification for SI methods in small and medium enterprise	2012	2002-2010	48	PA	PA	PA	NA
Huergo *et al.* [6]	Classification of SI methods	2014	2002-2013	105	PA	A	PA	NA
Bani *et al.* [7]	Exploring existing evaluation frameworks for state-of-the-art SIAs	2018	2007-2016	23	PA	PA	NA	PA
Bani *et al.* [9]	Identifying service identification challenges in service oriented architecture	2018	2005-2016	14	PA	NA	NA	NA
Fritzsch *et al.* [102]	Classification of refactoring approaches of monolithic applications to microservices	2018	2015-2017	10	PA	PA	PA	NA
Our SLR	Focusing on bottom-up and hybrid SIAs based on the used input, the applied process, the generated output and the usability of the approach Reviewing SI from the point of view of researchers and practitioners interest	2021	2004-2021	41	A	A	A	A

Table 2.10 Systematic literature reviews of service identification in the literature (***A** for Addressed, **PA** for Partially Addressed, **NA** for Not Addressed*

2.10 Conclusion

We presented in this chapter a systematic literature review (SLR) on service identification approaches (SIAs) that use the artifacts to build legacy software systems as input. We studied the SIAs in terms of their inputs, their processes, their outputs, and their usability. We built our taxonomy on our experience with legacy software modernization, discussions with industrial partners, and the analysis of existing SIAs.

The results of our SLR show that the state-of-the art SIAs are still at their infancy mainly due to (1) the lack of validation on real enterprise-scale systems; (2) the lack of tool support, and (3) the lack of automation of SIAs. They also show that the proposed SIAs generally ignore the economic aspects of the identification phase as well as the identification by service type. Indeed despite of their importance in the migration process, only few SIAs consider the economic aspects of the service identification process such as the implementation and maintenance cost, the re-factoring cost of the system, and time-to-market. Also, most of the existing SIAs look for services based on their functional cohesion and low coupling with other parts of the applications, regardless of service types.

Furthermore, we showed that the current trend of SIAs is the identification of microservices in existing systems. However, the applied identification techniques were very similar to those used for identifying services. The granularity border between services and microservices is still not well defined nor clearly discussed by these approaches.

Finally, we found that most SIAs usually do not try to improve the quality attributes of the identified candidate services. We believe that regardless of the sought quality attributes, SIAs should provide means to assess the quality of the identified services. Also, more work should be done to automate the SIAs and consider enterprise-scaled systems to validate the proposed approaches.

These conclusions and the generated taxonomy served as a support for our following study of the state of the practices of service identification in industry, that we detail in the next chapter.

CHAPTER 3 State of the Practice of Service Identification In Industry

3.1 Introduction

Several SI approaches have been proposed in the literature of academic research [10–12]. However, these approaches are based on few evidence and out of touch with industry practices due to the little knowledge about the state-of-the-practice of SI as part of *real* migration projects. Therefore, in this chapter, we study the gap between industry and academia by understanding industrial practices and identifying best practices of legacy applications migration to SOA in general and SI in particular. Thus, we answer the following research questions:

- **RQ1.** What kind of systems are being migrated to SOA?
- **RQ2.** Why are such systems being migrated?
- **RQ3.** What approaches are being used for application migration, in general, and SI in particular?

We survey 45 SOA migration practitioners using an online survey, and interviewed eight of them to answer these questions. We identify key findings including: (1) reducing maintenance costs is a key driver in SOA migration, (2) domain knowledge and source code of legacy applications are most often used respectively in a hybrid top-down and bottom-up approach for SI, (3) SI focuses on *domain* services, (4) there is little automation–the process of migration remains *essentially* manual, and (5) RESTful services and microservices are the most frequent target architectures.

This chapter is based on the following paper [17] and is organized as follows. Section 3.2 describes the study design. The results of the online survey are presented in Section 3.3. Section 3.4 reports the results of the interview sessions, which are discussed in Section 3.5. We conclude in Section 3.6 with recommendations and best practices for SI.

3.2 Study Design

The survey presented in this chapter was conducted between October 2017 and March 2018 and aimed to investigate the state of the practice in SOA migration, in general, and service identification in particular. Our study consisted of four main phases:

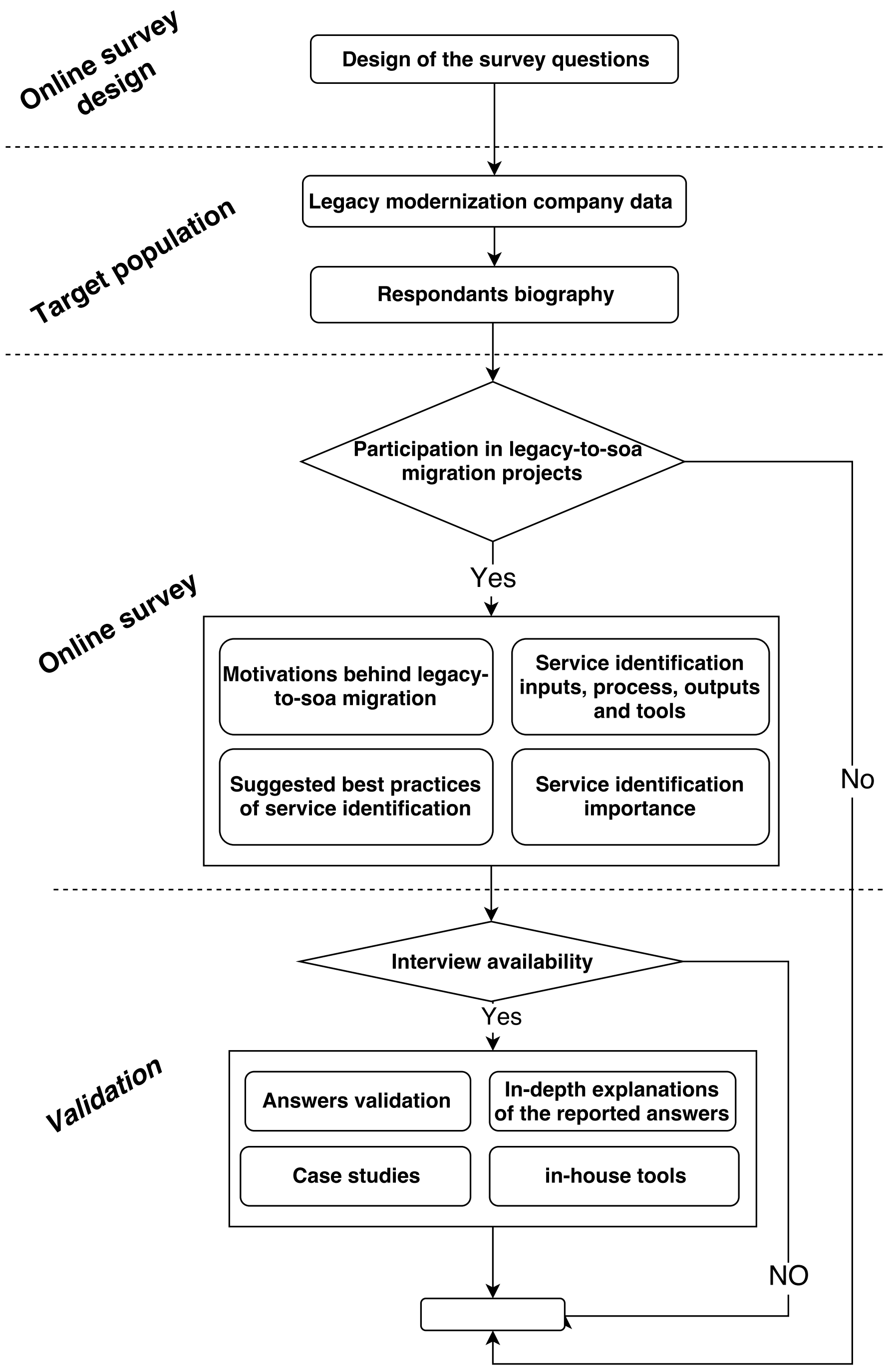

Online survey design
Design of the survey questions
Target population
Legacy modernization company data
Respondants biography
Participation in legacy-to-soa migration projects
Yes
No
Online survey
Motivations behind legacy-to-soa migration
Service identification inputs, process, outputs and tools
Suggested best practices of service identification
Service identification importance
Interview availability
Yes
NO
Validation
Answers validation
In-depth explanations of the reported answers
Case studies
in-house tools

Figure 3.1 Study design

A- Preparation of the online survey. We created a web-based survey [1] using Google forms. The survey was prepared based on our literature survey of the state-of-the art methods for SI and informal discussions with some subject matter experts. This helped identify the

[1]https://goo.gl/forms/EE31KeA7R7pUeTYI2

dimensions/aspects of the questionnaire, the individual questions, and the possible answers for each question. Before publishing the survey, we performed a pilot with six potential subjects, three from academia and three from industry, to validate the relevance of the questions, their wording, the coverage of the answers, etc. The six 'testers' went through the questions and suggested minor changes. The final survey contained six sections: 1) participants' professional and demographic data, 2) type of migrated system, and reasons for the migration, 3) general information about SI methods (perception of importance, strategy, inputs, level of automation), 4) detailed technical information about SI (technique/algorithm used, quality metrics considered), 5) Information on the types of services sought and targeted technologies, and 6) Information about the tools used, and the suggested best practices.

B- Selection of participants. We targeted developers with an industrial experience in SOA migration. Identifying and soliciting such developers was challenging. We relied on (1) information about companies that offer modernization services, (2) online presentations and webinars made by professionals that had the professional's contact information, and 3) search queries on LinkedIn profiles, such as "*legacy migration OR legacy modernization OR SOA architect OR SOA migration OR Cloud migration OR service migration OR service mining*". Once we identified potential participants, we sent them invitations via e-mail, LinkedIn, Facebook, and Twitter. We chose *not* to solicit more than three professionals from any given company to: 1) have an as wide representation as possible, and 2) to not overburden a single organization with our request.

C- Online survey. We invited 289 professionals to participate, and kindly asked them to forward our invitations to other people in their network who have experience in SOA migration and SI. The survey was completed by 47 people, two of which did not participate in SOA migration projects and whose responses were discarded, leaving us with 45 complete responses.

D- Validation. We assessed the reliability of the answers in the online survey by looking for spurious/facetious answers, contradictions between answers, etc. To be able to validate improbable answers, one question of the survey asked participants if they agreed to be contacted for a follow-up 30-minute interview, and 24 out of 45 agreed; however only eight could be interviewed in the end and the results of those interviews are shown in Section 3.4. A two-pass method [105] was used to analyze our transcripts of the individual interviews[2]. The first pass of the analysis consists of *thematic coding* to identify broad issues related to legacy-to-SOA migration in general and SI in particular. The second pass of analysis was performed using *axial coding* to identify relationships among the identified issues. Major

[2]https://goo.gl/ZYv2Ut for sample transcripts

factors were identified using *meta-codes*. The *meta-codes* were then used to identify similar patterns across the data from the multiple interviewees. Overall, the answers were plausible, and the eight detailed interviews confirmed the questionnaire answers, although provided us with more in-depth information.

3.3 Online Survey

In this section, we describe the results of our survey. We allowed multiple answers to most questions of the survey, therefore the sum of the computed percentage may exceed 100% in some cases. The given percentages are computed based on the total number of participants who answered a given question.

A- Participants. We reached a total of 45 participants who were involved in legacy-to-SOA migration projects in different capacities: 50% were software architects, 23.7% were directors of technology, and 21% were software engineers. The remaining 5.3% of participants mentioned other positions such as migration specialists, project managers and CEOs. They work in different industries: 64% were in technology and telecommunication, 20% from banking and insurance, 12.8% from health, and 3.2% from education. In terms of experience, 78% had more than 10 years of experience, and this was somewhat reflected in their age distribution: 23% were less than 35 years old, 39% were between 36 and 45, 20.5% were between 46 and 55, and 17.5% were over than 55 years old.

B- Types of legacy systems. The results show that the legacy systems included mainframe applications, transactional applications, ERP systems, monolithic client-server applications, software-analysis tools, and visualization tools; 13% of these were less than 5 year old, 18% were between 5-10 year old, and 69% were more than 10 years old. In terms of size, 62% of the systems were deemed large, 36% were medium size, and 2% were deemed small. Cobol (52.6%) and Java (57%) were the two most prominent languages for legacy systems. Figure 3.2 shows the many other languages used in the migrated applications.

Finding 1: *Practitioners migrate different types of old legacy systems implemented mainly in Cobol and Java.*

C- Motivations for Legacy-to-SOA Migration. We asked about the motivations behind the migration of legacy systems to SOA. We provided a list of reasons for the migration as shown in Figure 3.3. The most prevalent motivation was to reduce maintenance costs (82%). Practitioners reported during the interviews that the cost involved in maintaining legacy systems can be high due to (1) the poor/outdated documentation of these systems; (2) the obsolete/old programming languages used to implement these systems; (3) the de-

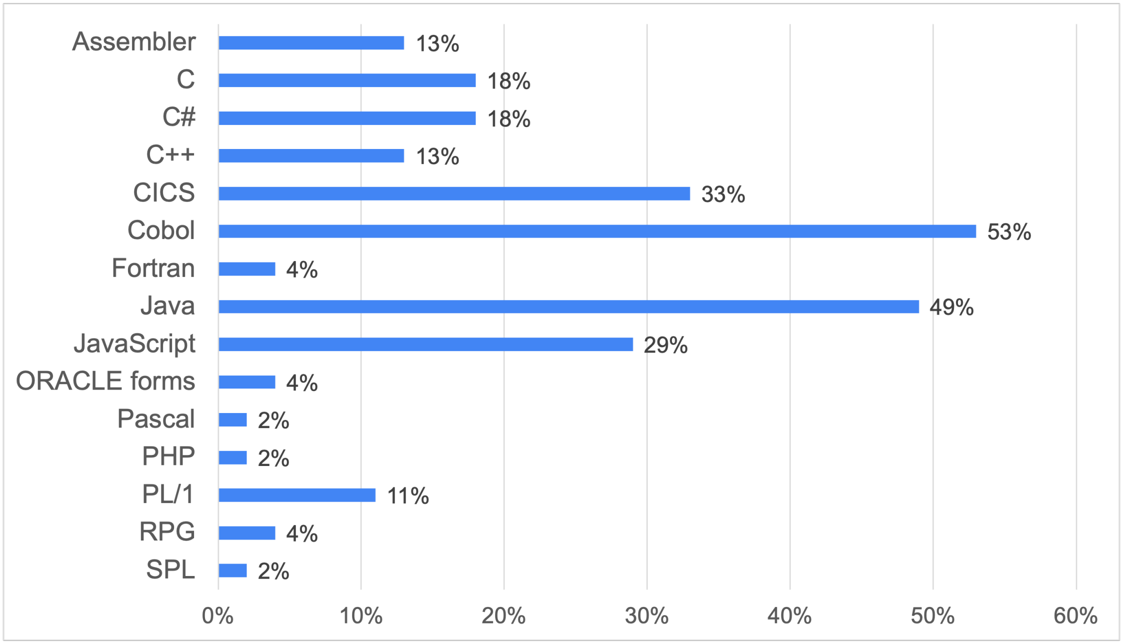


Figure 3.2 Languages of the systems migrated to SOA

cay and difficulty to understand the architectures, designs, and implementations of these legacy systems; and, (4) the lack of developers with the skills necessary to maintain these systems. The second most significant motivation to migrate legacy systems was to improve their flexibility (64%). We have been told during the interviews that practitioners have difficulties with legacy systems because they do not allow companies to have the flexibility required to carry out day-to-day tasks for evolving systems to meet new business requirements. The improvement of the interoperability of the legacy systems with the migration to SOA was the third most significant motivation (64%). During the interviews, practitioners told us that SOA eases the interoperability of heterogeneous systems by exploiting the pervasive infrastructure of the network. Thus, it offers the possibility to continue using and reusing the business capabilities provided by legacy systems in new, modern systems [29]. Improving system availability and testability as well as improving performance were other motivations of industrial legacy-to-SOA migration projects (38% each). Participants also mentioned other business and technical reasons for migrating legacy systems to SOA, such as improving business agility, having new user interfaces, and embracing new technologies.

Finding 2: *Reducing maintenance costs, improving the flexibility and interoperability of legacy systems are the main motivations to migrate legacy systems.*

D- Importance of Identifying Reusable Services from Legacy Systems. We asked about the importance of identifying *reusable* services in the source code of legacy systems

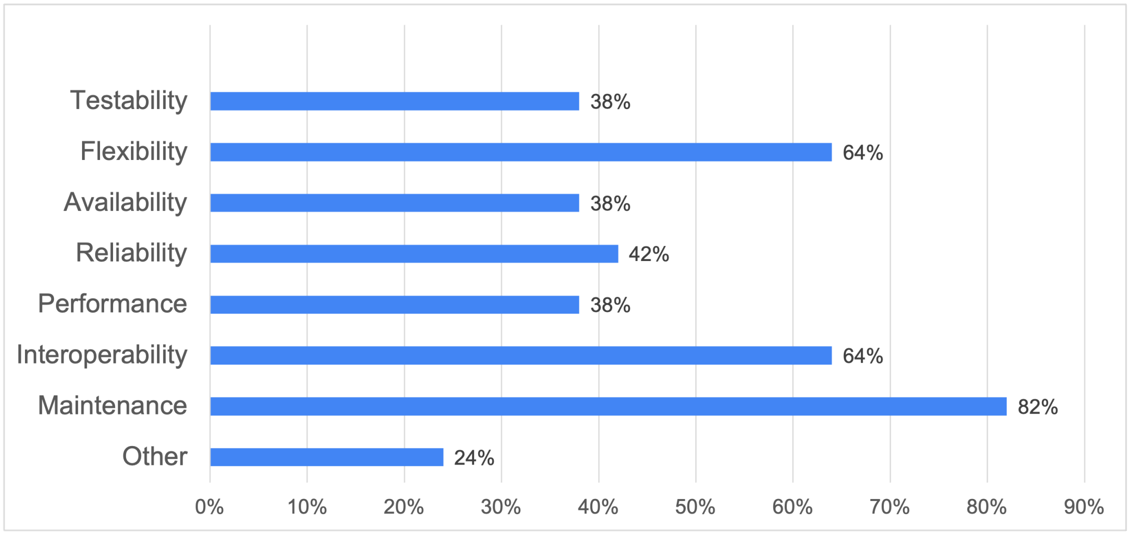


Figure 3.3 Reasons of legacy-to-SOA migration

during the migration process: 87% of the participants qualified it as important while only 13% thought that it is not. We explain this agreement by the benefits of software reuse, which (1) *increases software productivity* by shortening software-development time, (2) *reduces software development costs* by avoiding the re-implementation of existing services, (3) *reduces maintenance costs* because the reused services were functional and have been well-tested, and (4) reduces the risk of introducing new failures into the process of enhancing or creating new business services. We explain the 13% of disagreement as some participants undertook top-down migrations rather than bottom-up or mixed migrations and the former does not require identifying services in source code.

Finding 3: *Identifying services in legacy applications is an important step in legacy-to-SOA migration.*

E- Inputs of SI. Through a literature review, we identified several types of inputs used for SI. We listed these inputs in our survey and asked participants on which inputs they relied to identify services. Figure 3.4 shows that the most used inputs were source code, business process models, databases, and human knowledge. 76% of the participants relied on the recovery of the business logic of legacy systems through the analyses of the source code to identify services with high business value. 71% relied on the mapping of business processes with the legacy source code to extract reusable services through human expertise. These artifacts may help software engineers to have a better understanding of the legacy systems. Finally, participants rarely relied on ontologies, activity diagrams, state machine diagrams, and execution traces to identify services. This observation may be due to their unavailability or complexity to establish especially since our practitioners deal with large systems.

Finding 4: *Many software artifacts can be used for SI. Practitioners mostly used source code, business process models, databases, and human expertise. There is a very low interest in relying on ontologies, activity diagrams, state machine diagrams, and execution traces to identify services.*

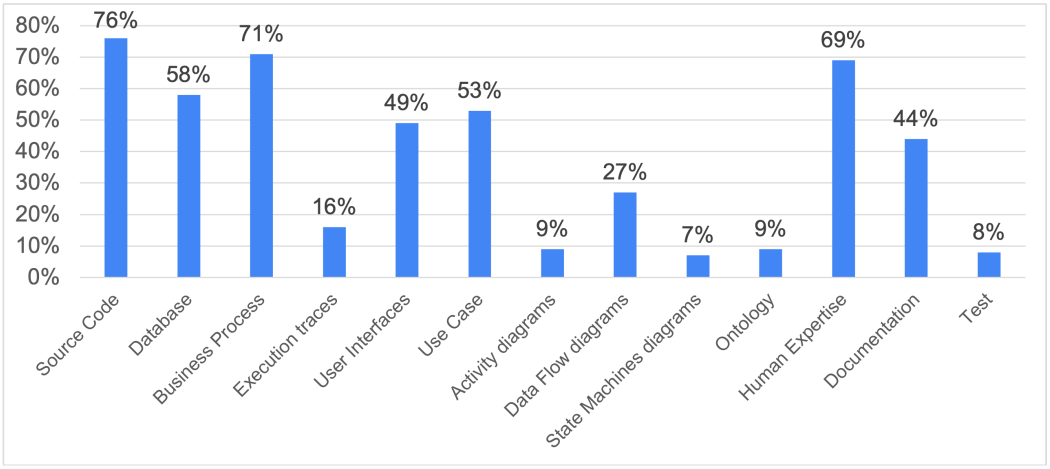


Figure 3.4 Inputs for SI in industry

F- Directions of SI. We asked participants about their choices of direction for identifying services. We proposed three directions: (1) Top-down: starting from domain-specific conceptual models, like business concepts and process models, to identify services, which are then specified and implemented through a forward engineering process; (2) Bottom-up: starting by analyzing the existing legacy system artifacts and identifying services from reusable legacy code; and (3) Mixed: starting both from domain-specific conceptual models and the analyses of the legacy system to identify services. We found that 53% of the participants use a mixed direction to identify services. Participants used almost equally top-down and bottom-up directions with 23% and 24% each. We explain these observations as follows: (1) practitioners relied on source code and business process models as reported in *Finding 4*, (2) practitioners also relied on extracting the business logic of legacy systems because documentation was not always available, (3) practitioners prioritized reuse and avoided development from scratch to reduce time and costs, and (4) practitioners faced limitations due to the lack of legacy experts/knowledge, unavailability of up-to-date documentation, program comprehension, and challenges of reverse-engineering legacy systems.

Finding 5: *Practitioners highly rely on a mixed direction to identify services during legacy-to-SOA migration process.*

G- Techniques for SI. We asked about the techniques that they used to identify services. As depicted in Figure 3.5, we found that 60% of the participants relied on clustering functionalities of the legacy systems and exposing these clusters as services. 47% of them relied on some black-box techniques, like wrapping, because they either consider the migration as an integration problem or did not want to modify the core functionalities of the legacy systems because it provided useful services. We observed a low interest in using machine-learning techniques, formal-concept analysis, or meta-heuristic algorithms to identify reusable services. Using these techniques may be challenging for practitioners because they are dealing with large systems to migrate and so the knowledge required to establish these techniques could be time consuming and may not lead to optimal results. Also these techniques are researched by academics and not mainly by professionals Finally, 9% of the participants mentioned that they did not use any techniques and performed SI manually.

Finding 6: *Functionality clustering and wrapping are the most used techniques of SI in industry.*

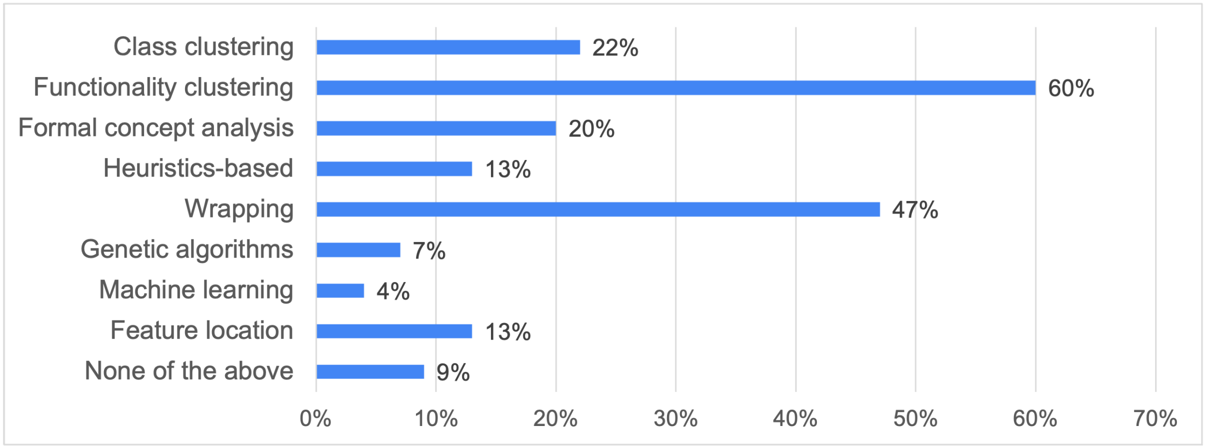


Figure 3.5 Techniques for SI in industry

H- Analyses types for SI. We asked about the types of analyses that they performed for SI (static, dynamic, textual, and–or historical analyses). We observed that 87% of the participants relied on static analyses of the source code of the legacy systems to identify services. 43% of them reported that they relied on runtime analyses. Participants also relied on textual analyses for the identification processes. Textual analyses include elements such as features identification techniques, natural language processing, legacy documentation analysis, etc. Only 18% of the participants reported that they relied on historical analyses (analyses of different versions of the legacy system) to extract candidate services, which may be due to (1) the unavailability of several versions of the legacy system and (2) the difficulty to study the evolution of a legacy system to gather valuable information to identify reusable services.

Finding 7: *Practitioners mostly relied on static analyses of the source code of their legacy systems for SI.*

I- Services Quality Criteria. We asked the participants about the quality metrics/criteria that they sought during SI. We identified the quality criteria, listed in Figure 3.6. Service reusability was the most sought quality criteria by the participants (62%), followed by service granularity (47%), and loose coupling (44%). Reusability was defined by participants as both a measure of the amount of source code reused in the services and the amount of services reused in the systems. Costs and the adaptation effort were also considered by the participants during the identification process (40% and 42% respectively). However, they did not consider self-descriptiveness, high cohesion, composability, and the total numbers of services when identifying services.

Finding 8: *Only few service quality criteria are desired by practitioners in the SI process: reusability, granularity, and loose coupling.*

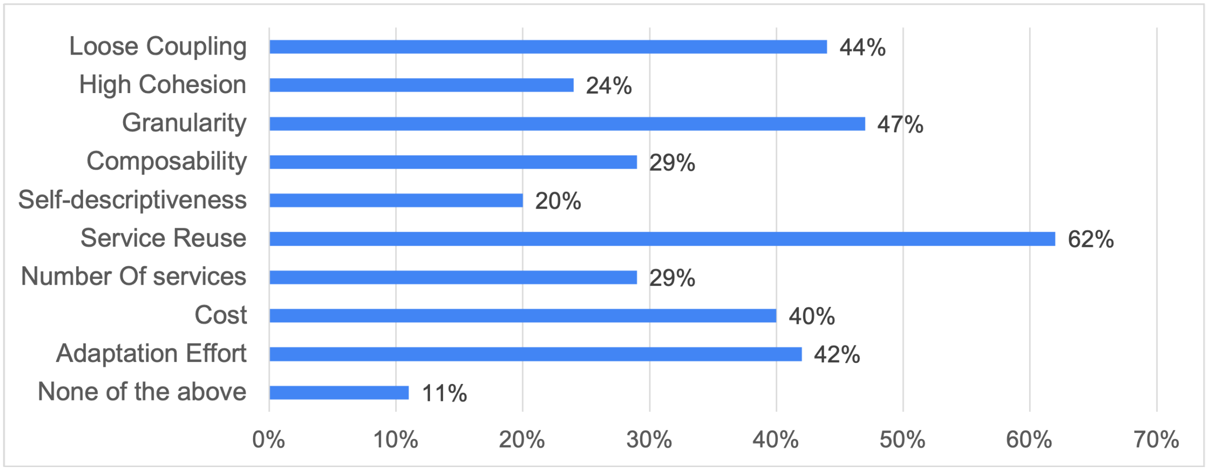


Figure 3.6 Desired services quality criteria for SI in industry

J- Types of the Identified Services. We provided practitioners with a taxonomy classifying service types into domain-specific (business) services versus domain-neutral (technical) services. The provided domain-specific services are: (1) business services, enterprise services, application services and entity services. The technical services are utility services and infrastructure services. We report the results in Figure 3.7. As domain-specific services represent the business core functionalities of SOA, they were the most targeted services (i.e., business and application services) during the SI processes compared to technical services. Utility and infrastructure services were the less targeted services because they are SOA-specific services and utility services are relatively easy to implement.

Finding 9: *SI is a business-driven process that prioritized the identification of domain-specific services rather than technical services.*

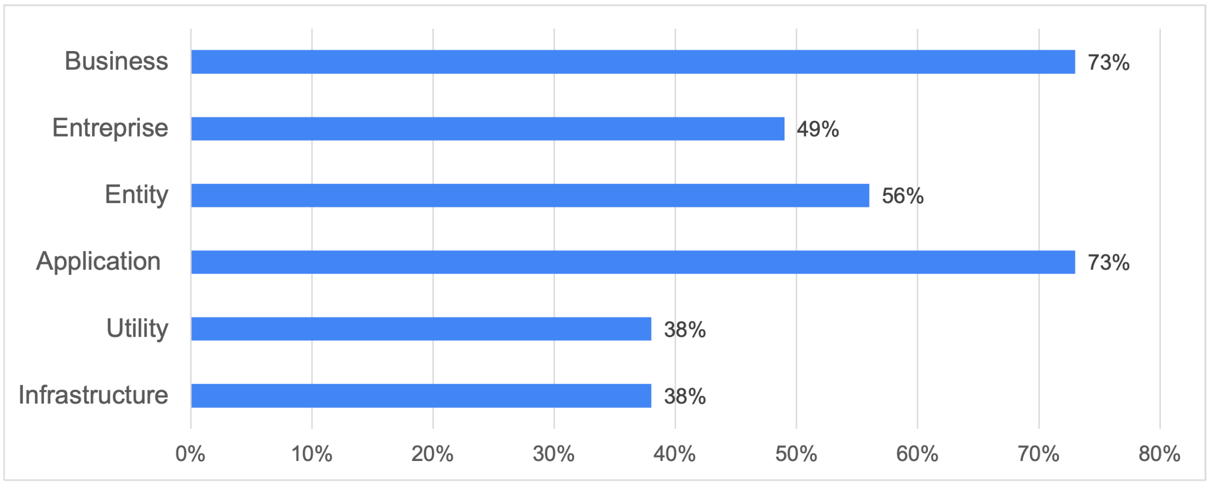


Figure 3.7 Types of the migrated services

K- Service Technologies. We asked the participants about the services technologies that they targeted during migration. We found that 75% of them use REST services, 60% use SOAP and only 4.5% use Service Component Architecture (SCA). Surprisingly, half of the participants reported that they focused on identifying microservices in legacy systems. While there is no precise definition of this architectural style, microservices are gaining interest among organizations, especially with the growth of the Cloud and DevOps paradigms [106].

Finding 10: *Restful services are the most targeted service technology in legacy-to-SOA migration.*

L- Automation of SI. We asked the participants about the degree of automation of their SI techniques as well as the tools they used to this end[3]. We found that many different tools are being used as well as manual analyses and in-house tools put together for the migration of particular legacy systems. However, not one set of tools supports adequately the migration of systems to SOA. We also found that the majority of the techniques used by the participants to identify services are either semi-automatic (51%) or manual (42.3%). Only three participants (6.7%) mentioned the use of tools to identify automatically reusable services. We highly believe based the reported tools that these fully-automatic approaches deal with re-engineering tasks as well as wrapping techniques that automatically expose

[3]we report the list of tools in https://goo.gl/ZYv2Ut

legacy systems functionalities as services.

Finding 11: *There is a lack of automation of SI techniques in industry but input from human experts is essential to annotate/qualify intermediate or final results of SI.*

M- Threats to the Validity. *Construct validity* threats refers to the extent to which operationalizations of a construct (in our case the survey and interview questions and terminology) do actually measure what the theory claims. To minimize this threat, we used both open and closed questions in the survey and tried to minimize the ambiguities through our pilot study as we mentioned in section 3.2.

Internal validity acquiescence bias is a kind of response bias where respondents have a tendency to agree with all the questions in the survey or to indicate a positive connotation. It is sometimes referred to the tendency of a respondent to agree with a statement when in doubt. We mitigate this threat by doing interview sessions to validate the survey answers. We also eliminated responses where participants selected all the possible choices for all the questions. We also mitigate this threats by checking the responses to questions that are related to each others (e.g. the used input for SI and the identification direction, etc.). Finally, we decided not to have incentives for participating in our survey to minimize social desirability bias.

External validity the survey participants might not be representative of the general population of software developers migrating legacy systems to SOA. Thus, the generalizability of our survey might be limited. The mitigation of this threat to validity is very challenging because (1) we are targeting practitioners with very specific technical skills; and (2) professionals are in general not eager to communicate the details of their in-house tools and techniques of modernization approaches. To mitigate this risk, we advertised our survey through various channels (e.g., LinkedIn, Twitter, Facebook and email) and targeted professionals from different legacy modernization companies. Also, to the best of our knowledge, our sample size is one of the largest such size among many papers in empirical software engineering in general and modernization in particular. Participants could freely decide whether to participate in the study or not (self-selection). They were informed about the topic of the survey, the estimated time to complete the survey, the research purpose of the study and the guarantee of the anonymity of their identity and that of their answers.

3.4 Interview Sessions

Eight participants among the 45 agreed to carry phone interviews. Table 3.1 describes the profile of these participants. The initial purpose of the interviews was to ask the participants to elaborate on some of their answers or resolve contradictions among their answers. However, the interviews often ended with discussions on issues not addressed in the survey. The interviews also allowed participants to rectify some of their answers and for our part to obtain presentations and white papers about their migrations. We now summarize salient facts gathered from the interviews in terms of the adopted migration strategies and directions of SI.

Participant	**Profession**	**Experience**	**Country**
P1	Technical Solution Architect	25 years	Germany
P2	Legacy modernization and enterprise IT architect	18 years	India
P3	Mainframe Modernization Specialist	33 years	USA
P4	Legacy and data Center senior consultant	30 years	Italy
P5	Software modernization expert	15 years	Canada
P6	IT Architect	20 years	Canada
P7	Director of technology	12 years	Canada
P8	Software Engineer	7 years	France

Table 3.1 Information about the participants in the interview sessions

3.4.1 Migration Strategies

We asked the interviewed participants about their adopted migration strategies to migrate legacy software systems to SOA. We identified three strategies: *rehosting*, *legacy system re-architecture*, and *rehosting followed by re-architecture*.

A- Rehosting (adopted by P1, P3, P6) consists of moving a legacy system with minimal changes from one platform, typically legacy mainframes, to more modern alternatives such as Linux, Unix, or Windows in two ways: (1) by running emulators or virtual machines of the source platform on the target platforms (e.g., a VMS or AS400 emulator/virtual machine on Linux) or (2) by rewriting the parts of the systems that interface with the target platforms. The business logic and data of the legacy systems remain unchanged on the new platform. Rehosting is done when the hardware or software platforms become too costly to support –or are no longer supported –by the manufacturer/vendor. The systems can be *wrapped* within services once they are integrated on the new platforms.

B- Legacy systems re-architecture (adopted by P1, P2, P5) is a migration strategy in three steps applied each on three different layers: the *application-code layer*, which contains the legacy code in Cobol, PL1, etc.; the *information layer*, which gathers data access through files, databases etc.; and, the *business-process layer* which describes the business logic of the system. The three migration steps are: (1) *Legacy system discovery and migration planning*, it focuses on cataloging and understanding all the assets in the legacy systems, *“we are importing the code in our toolset. We are looking for dependencies and capturing business processes. We just take a look if everything is complete”* said P1; (2) *Design*, this phase consists in designing the new system, a “*future case analysis repository*” that contains enhancements to the legacy business processes and all the modernized data and future SOA model are stored in the *information layer*; and (3) *Target system development and test*, this phase *“is a very classic software development phase just to develop and test the new SOA based system”* said P1.

C- Legacy systems re-hosting and re-architecture (adopted by P1, P4, P7, P8) aims to build new SOAs that yield the business values of the legacy systems while minimizing costs related to legacy hardware and ensuring a progressive and incremental replacement of the legacy code. This migration strategy is mainly used to “*minimize disruption while ensuring business continuity*” said P8. It avoids the “*big-bang*” migration strategy by (1) re-hosting the legacy systems to modern platforms to minimize hardware costs, (2) creating wrappers to hide the internal legacy functionalities, and (3) replacing progressively the legacy code.

3.4.2 Directions of SI

We detail in this section the adopted directions of SI by our interviewees.

A- Bottom-up strategies (P1, P3, P4, and P5) consist of identifying artifacts of the legacy code that implement reusable business functions to be repackaged as services: *“Through the bottom-up SI strategy we want to reuse the existing legacy code certainly, but not the architecture. Most of legacy systems that we deal with have about 25 millions lines of code. If we want to write them again, it can take years”* said P5. The artifacts used by bottom-up approaches include the source code, data flow analyses, legacy system interfaces, databases, documentations, and human expertise. Reverse-engineering tools were used to understand the legacy systems and extract their business logic, especially when there is a lack of documentation and experts. Several interviewees reported using both in-house and open-source tools to reverse-engineer systems. For example P5 used an in-house tool based on the Knowledge Discovery Model (KDM) to obtain call and data-flow graphs of COBOL systems. P5 relies on functionality clustering and pattern matching to identify reusable services: *“We are searching for patterns and we are looking for business rules or business logic that match with these patterns and heuristics. We are doing data flow analysis with slicing to identify reusable business functions that can be grouped and deployed as services”.* Many interviewees (P1, P3, P4, and P5) also relied on techniques for detecting code clones to identify reusable services. *“What we also do in many cases is looking for duplicate code pattern because in many cases business rules are duplicated, you need to decide what to take out of this”* said P4.

B- Top-down strategies (P7) starts from the analysis of domain-specific conceptual models and requirements to specify the services of the targeted SOA. P7 recommended to use this strategy when (1) legacy source code is not available, (2) legacy source code is not reusable, (3) cost of re-engineering and integrating legacy systems is high, and (4) organizations are mature enough in terms of business processes. P7 reported that they adopted a semi-automatic top-down strategy for SI to migrate a legacy banking system to SOA. They used BPMN process models of the banking legacy system as input. They begun by identifying the entity-services and the application services. They then moved to higher-level services, such as task-centric services, and finally developed an orchestration layer that represented business services. This strategy is based on the analysis of "*information*" used in each activity of the business processes. P7 explained that *"information could be a document, reports, windows, screens, an entity etc. that is required in the execution of an activity"*. To identify entity services from business processes, *key information* manipulated in the business process models was identified. An *information* is considered as *key* by P7, if it meets at least one of the following conditions: (1) its number of occurrences exceeds a given threshold and (2) it is related to a highly solicited activities.

C- Mixed strategies (P1, P2, P3, P4, P5, P6, and P8) rely on reverse-engineering techniques to document the legacy systems, extract the business logics, and identify reusable pieces of code that can be exposed as services. They also rely on forward-engineering techniques to define the business processes of the target SOAs and to design and implement the services. P2 said: *"Sometimes if the source code is available and documentation is not, we use some parser based tools to reverse-engineer these applications. These tools will create some documentation from the code and then that documentation is used to do the forward engineering and complete the targeted SOA road map"*. He also argued that *"through this documentation we create use cases for forward engineering to complete the identification, the design and the implementation of the services"*. P1 said *"we document everything in our system and then at the very end we identify the business rules mark them in the code and you can extract them afterwards [...] we have a list of business processes and core code description and we also document this, and based on this we are creating our service-oriented material"*.

D- Final choice of the identified services is a manual process driven by subject-matter experts. P5 said: *"We make proposition about the services that we identify and ask the customer if it makes sense. Sometimes at technical level we have better knowledge than the customer but not from business process level"*.

3.4.3 Threats to Validity

Internal validity. Social desirability is a bias that leads any respondent to deny undesirable traits and report traits that are socially desirable. To minimize this threat, we did not put any incentives for the participants to participate in the interviews. We also guaranteed the interviewees their anonymity and emphasized that all the reported information will be only for research purposes.

External validity. Information from our interviews is not generalizable as the number of the interviewees is a bit on the low end for software engineering studies. However, it is acceptable given that it is unquestionably difficult to find interviewees in legacy-to-SOA migration domain. We only sought to obtain a better understanding of the results of the online survey. Also, Table 3.1 shows that our interviewees are experts in legacy-to-SOA migration and, thus, that our sample is still reliable because we are dealing with subject-matter experts.

Conclusion validity. The information from our interviews also show some threats to the validity of our conclusion because some interviewees contradict each other or, for one interviewee, change their answers to the survey. However, this threat is acceptable because we use these interviews with the purpose to mitigate and discuss the answers to the survey.

3.5 Discussions

After analysing the survey and interview data, we highlight the following facts.

Importance of Service Identification From Legacy Systems. We observed that SI is an important step in the overall legacy-to-SOA migration process for most practitioners, especially when it comes to the context of SI from legacy systems. As emphasized by P4 “It is important because we are able to identify the reusable of the code. SI is considered as the main helmet to measure the impact of the migration[...] you need to understand the migration cost which is in many cases too expensive, you have to cut the cost by identifying reusable pieces of the legacy code in a cost-effective way”. Thus, the agreement about the importance of identifying reusable services in industry can be explained by the benefits of software reuse. However it should be noted that SI is not always fine-grained as mentioned by P6 “We basically wrap the legacy system and expose all its functionalities as services”.

Business-value Driven Service Identification. We notice that not all service quality criteria are equally targeted by practitioners. Unlike academia, efforts in industrial SI strategies are made to deal with business constraints such as the recovering of the business logic of legacy systems and extracting reusable functionalities with high business value. There are big investments by practitioners to preserve the business logic of legacy systems rather than to care about service quality constraints. As it is stated by P2, SI is mainly driven by the customers business needs: “Our customers do not really focus on these features. I am not saying that these quality criteria are not necessary but because of the business constraints, considering service quality metrics become a lower priority comparing to timing to finish the project and return in investment issues”. Also, technical constraints may hinder the consideration of quality metrics : targeting quality metrics while identifying reusable pieces of code that can be exposed as services may not be suitable for all legacy technologies like mainframe legacy systems for example: “For banking mainframes systems it is not easy to use that kind of approach since we are dealing with routines” stated P4.

Automation and Experts Feedbacks. The full automation of SI process is not the primary focus of practitioners. It is even the case of big modernization companies as it is stated by P1 “In our SI methodology we are not doing everything automatic, automation is about 70% of all the migration project”. However, there is automation in wrapping and reverse engineering techniques to document and extract the business logic of legacy systems when the documentation is absent. Feedback loop with business analysts and customers is considered essential by practitioners to decide about the pertinence of a candidate identified service. Practitioners also do not take the risk to try to fully automate the SI process as it is a challenging problem with unpredictable results, time consuming and needs a lot of research investments.

Gap between Academia and Industry. None of the interviewed practitioners mentioned the use of research papers or academic resources for their migration projects. From the point of view of practitioners, "academics do not see the larger picture of the real industrial problems and challenges they are facing” as stated by P2. The lack of cost-effective academic SI technique and the lack of validation on real enterprise-scale systems is a problem that hinders knowledge transfer between academia and industry in the context of legacy-to-SOA migration.

3.6 Conclusion

We presented in this chapter a state of the practice of SI in industry to support the migration of legacy software systems to SOA. We surveyed 45 industrial practitioners and interviewed eight of them to collect, analyze, and report their experiences with the migration of legacy systems. Our results showed that reducing maintenance costs and improving the flexibility and interoperability of legacy systems are the main motivations to migrate these systems to SOA. They also showed that SI is perceived by practitioners as an important step for the migration, in particular to identify reusable code in the legacy systems. In addition, they showed that SI is a process driven by business value rather than quality criteria, even though some practitioners consider some quality criteria (mainly reusability, granularity, and loose coupling). Finally, our results showed that SI remains a manual process in which human experts' feedback is essential to annotate/qualify intermediate or candidate services.

Based on these observations, in the next chapter we study the gap between academic and industrial SIAs and derive several recommendations that will be considered in our identification approach.

CHAPTER 4 Gap Analysis between Academia and Industry

4.1 Introduction

In this chapter, we investigate gaps between academia and industry for service identification based on our SLR (Chapter 2) and survey (Chapter 3). We compare the inputs, processes, and outputs of SIAs in academia and industry. We derive several recommendations for identifying services in legacy systems. We rely on these recommendations to propose our SIA that we detail in chapter 5.

This chapter is based on the following paper [23]. In Section 4.2, we compare the used inputs for services identification in academia and industry. In Section 4.3, we study the processes of academic and industrial SIAs in terms of the used techniques and the type of analysis. In Section 4.4, we compare the outputs of academic and industrial SIAs. In Section 4.4, we list the recommendations that we derive. Finally Section 4.6 concludes the chapter.

4.2 Inputs of SIAs

We identified several types of inputs used for SI in academia and industry, e.g., source code, database, business process, user interface etc. Figure 4.1 shows that both academic and industrial SIAs mostly rely on source code, business process models, and human expertise, possibly due to the availability of such artifacts with legacy systems. We also noticed that both academic and industrial SIAs rely on the recovery of the business logic through analyses of the source code to identify services with high business value. They also map business processes with the legacy source code to extract reusable services through human expertise.

Both academic and industrial SIAs rarely rely on ontologies, activity diagrams, state machines, and execution traces to identify services, possibly due to their unavailability or the complexity to obtain them, especially with large systems. None of the participants mentioned test cases. Only a few academic approaches relied on such inputs. We also observed discrepancies: practitioners rely a lot on use cases, user interfaces, database schemas, and data-flow diagrams while such artifacts are seldom considered by academia.

Practitioners generally migrate systems with very old technologies (COBOL, CICS, etc.) [107]. They have/develop tools to generate documentations for these systems. They rely on these tools to generate, for example, data-flow diagrams and use cases to support (1) the understanding of the legacy system and (2) the identification of candidate services.

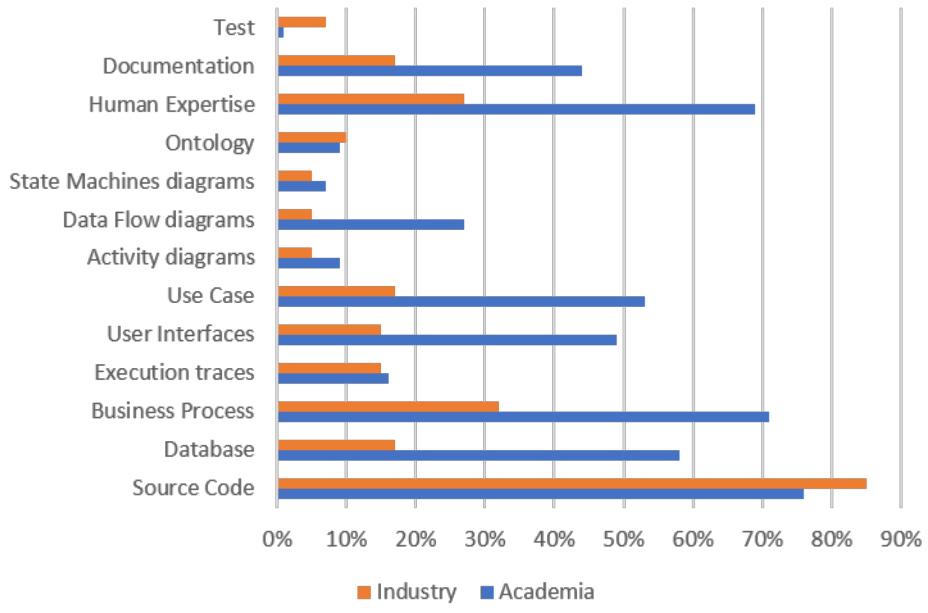


Figure 4.1 Inputs of SIAs

Finding 1: *Many software artifacts can be used for SI. The most widely used inputs by both industrial and academic SIAs are source code, business process models, and human expertise. Ontologies, activity diagrams, state machines, and execution traces are seldom used to identify services by both academia and industry.*

4.3 Processes of SIAs

We now compare the processes used in academic and industrial SIAs in terms of techniques and types of analyses (e.g, static, dynamic, lexical, etc).

Techniques of SIAs

We identified several techniques used to identify services, shown in Figure 4.2.

We found that clustering software artifacts and exposing the clusters as services is the most used technique in both academia and industry. We found discrepancies in the other techniques used by academia and industry: 47% of the participants relied on wrapping techniques because they considered migration an integration problem. They did not want to modify the legacy system *per se* because it provided reliable business functionalities. This technique is rarely considered by researchers who instead focused on custom heuristics, usually combined

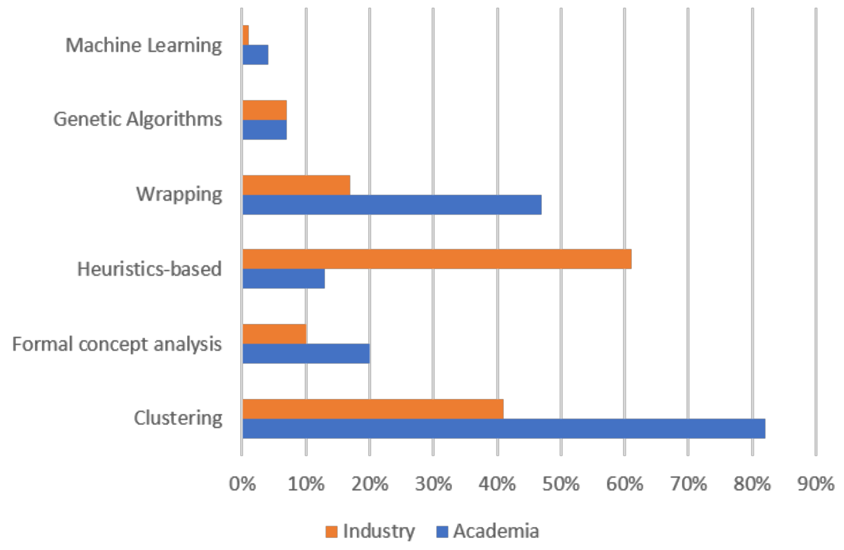


Figure 4.2 Techniques of SIAs

with clustering techniques.

We observed a low interest by both academia and industry in using machine-learning techniques, formal-concept analysis, and genetic algorithms to identify services. Using these techniques may be challenging for practitioners because they may be unfamiliar with these techniques, which may lead to sub-optimal results. More studies should be done by researchers to investigate the efficiency of such techniques in identifying services in legacy systems.

Finding 2: *Clustering is the technique most used by academia and industry to identify services. In second place, practitioners favour wrapping and researchers custom heuristics.*

Types of Analyses

SIAs may perform static, dynamic, lexical, historical analyses, or some combination thereof to identify services.

Figure 4.3 shows that both researchers and practitioners use the same types of analyses. They relied on static analyses of the source code of legacy systems. They also used dynamic analyses. Some practitioners (43%) and academic SIAs (20%) relied on textual analyses. Only 18% of the participants and 2% of academic SIAs relied on historical analyses, which may be due to (1) the unavailability of the history the legacy system and (2) the difficulty to study its evolution to obtain usable data.

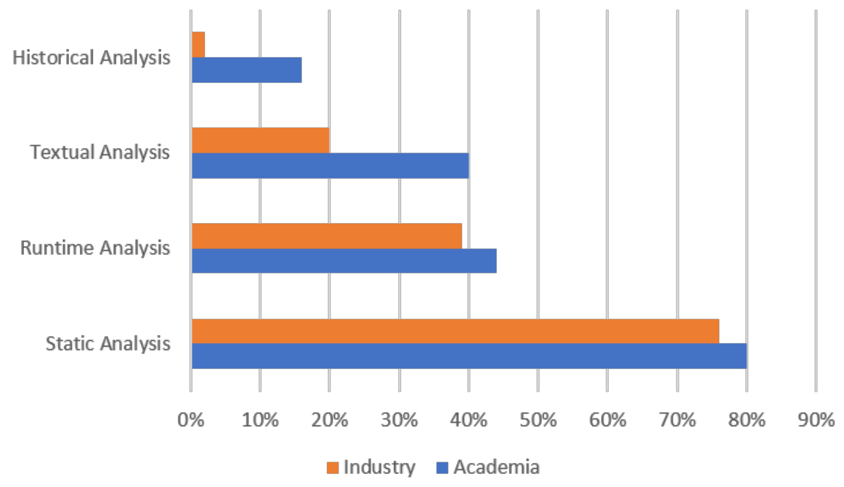


Figure 4.3 Types of analyses of SIAs

Finding 3: *Both researchers and practitioners relied mostly on static and dynamic analyses of the source code to identify services.*

4.4 Outputs of SIAs

We asked participants the types of services that they identify and search for type-sensitive SIAs in the literature. We report the results in Figure 4.4. We found that identifying services by types is mostly used in industry. There is a lack of type-sensitive academic SIAs. During interviews, several practitioners highlighted the importance of identifying the types of services when migrating to SOA, i.e., the nature and business capabilities of the identified services [107].

Because domain-specific services represent the core business functionalities of a legacy system, they were the most targeted services by SIAs, compared to domain-neutral services. Utility and Infrastructure services are SOA-specific services, with utility services relatively easy to implement.

Finding 4: *Service identification is business driven. Practitioners prioritise the identification of domain-specific rather than domain-neutral services. There is a lack of type-sensitive SIAs in the academic literature.*

We now present the results of our comparison between academic and industrial SIAs in terms of inputs, processes, and outputs.

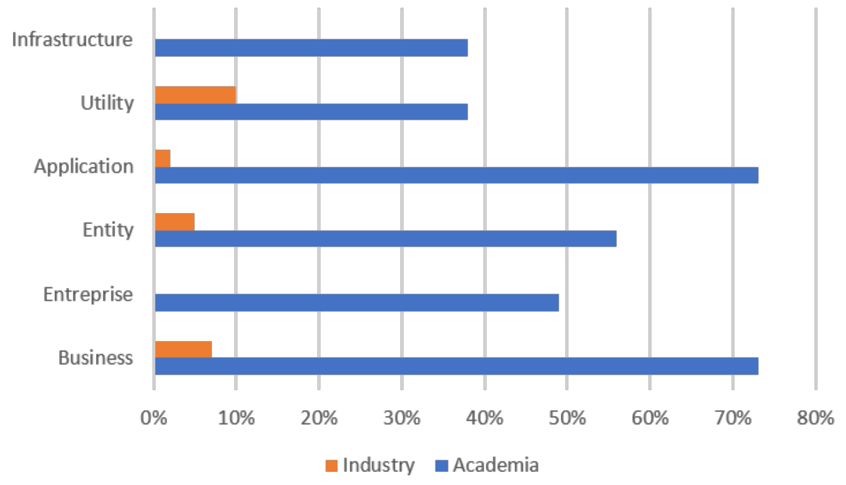


Figure 4.4 Types of services in SIAs

4.5 Recommendations

We drew several lessons from these results, which we summarize as follows.

Service identification is a business-value driven process. When identifying services we must focus on the functional clusters that implement *useful* and *reusable* business functions. We must not focus on *technical/architectural* properties, as many academic techniques do (e.g., [13–15,51]). While the research literature identifies many service types, SI must first and foremost focus on identifying domain services, i.e., entity, business, and process services that have business values.

A deep understanding of the domain and a great familiarity with the legacy systems are necessary. Because SI is driven by business values, we must have a deep understanding of the *domain*, including its *main entities* and *processes*. We must also be familiar with the legacy systems in which to identify services. While the research literature assumes that the SI techniques are independent of the legacy experts, they should allow incorporating seamlessly knowledge from experts who are familiar with the systems.

The input must be source code and production data. A Swiss army aphorism states that "If the map and the terrain disagree, trust the terrain". With legacy systems, documentation (the map) may be absent or awfully out of date. The source code (the terrain) is the only reliable source of information about what the *current* system does. Therefore, SIAs should consider analysing the source code of legacy systems (when available) to identify reusable services accurately. Source-code analysis could also be complemented by other inputs, such as business processes and human expertise.

The output must be high-value, coarse-grained services. Regardless of the targeted SOA technology, be it SOAP, RESTful, or microservices, the output of any SI technique must be high-value, coarse grained *domain services*. SI should consider the types of services. This consideration could improve the identification accuracy by narrowing the search space by types of services and their associated code-patterns. Types could be used to classify candidate services hierarchically and to prioritize the identification of specific types of services according to their business values or the requirements of the migration process.

The process must follow a (proven) methodology. Migration projects are complex endeavors, regardless of the source and target technologies. There is value in adopting or adapting an existing *SOA migration methodology* because such methodologies prescribe *processes*, *deliverables*, and *quality metrics* to guide the migration. While the research literature proposes techniques, the participants recommended using existing methodologies including Oracle's *OUM Methodology*, IBM's *Service-Oriented Modelling and Architecture* (SOMA) methodology [108], and devising SI techniques as *parts* of these methodologies. The process of a SIA should also rely on a clustering algorithm to group cohesive entities in the system that will be repackaged into services. The clustering technique should consider some heuristics related to the legacy system to tune the identification of functional groupings. The identification process should also consider quality requirements and metrics to identify high quality architecturally significant services.

4.6 Conclusion

We presented in this chapter a comparative analysis of the state of the practice of SI in academia and industry to support the migration of legacy systems to SOA. Our analysis showed that there is a gap between academia and industry regarding the proposed service-identification approaches in terms of inputs, processes, and outputs. Based on this analysis, we drew several recommendations for service identification.

We now build on these recommendations to propose our service-identification approach. Our approach relies on the static analysis of the source code of legacy systems and consider the identification of specific types of services during the identification process.

CHAPTER 5 Type Sensitive Service Identification Approach

5.1 Introduction

The maintenance and migration of legacy software systems are central IT activities in many organizations in which these systems are mission-critical. These systems embed hidden knowledge that is of significant values. They cannot be simply removed or replaced because they execute effectively and accurately critical and complex business logic. However, legacy software systems are difficult to maintain and scale because their software and hardware become obsolete [3]. They must be modernized to ease their maintenance and evolution.

A common strategy for modernizing such systems is their migration to *service-oriented architecture* (SOA), which defines a style where systems are made of services that are reusable, distributed, relatively independent, and often heterogeneous [29]. Service Identification (SI) is considered one of the most challenging steps of the migration process [8]. It consists in identifying reusable groupings—clusters of functionalities in the legacy system that qualify as *candidate services* in the target architecture.

Several SI approaches have been proposed in the literature [10–16]. However most of them have limited identification accuracy and usually require several types of inputs (e.g., business process models, use cases, activity diagrams, etc.) that may not be always available especially in the context of legacy systems. We argue that service identification should depend on service types to improve the identification accuracy by narrowing the search space through the types and their associated code-patterns.

Service types should be used to classify service candidates according to a hierarchical-layered schema and offers the possibility to prioritize the identification of specific types of services according to the business requirements of the migration process. Also, in chapter 3, we reported that several practitioners highlighted the importance of identifying service types when migrating legacy systems to SOA. They claimed that type-aware SI provides important information on the nature and business capabilities of the identified services. Besides, existing source-code SI approaches use similar *functional-clustering criteria*—typically *cohesion* and *coupling*, which lead to candidate services that are often architecturally irrelevant for the new SOA-based system.

Consequently, we propose *ServiceMiner*, a type-aware SI approach to support the migration of legacy systems to SOA. We consider a bottom-up approach relying on source code analysis, as other sources of information (e.g., business process models, use cases, activity diagrams, etc.) may be unavailable or out of sync with the actual code. We use a categorization of service types based on previous service taxonomies and describe the code-level patterns characterizing each type of service. We evaluate *ServiceMiner* on an open-source, enterprise-scale legacy ERP system and compare its results to those of two state-of-the-art SI approaches [11,13]. We show that our approach automates the identification of specific types of candidate services, which are architecturally significant for the new SOA-based system.

This chapter is based on the following papers [22,23] and is structured as follows. Section 5.2 details the service identification approach. Section 5.3 presents the experimental validation of our approach and details the obtained results. We discuss in Section 5.4 our threats to validity. Finally, we conclude in Section 5.5 with future work.

5.2 Approach

Figure 5.1 summarizes our SI approach, *ServiceMiner*, which consists of two phases: (1) a *pre-processing phase* in which we build a model of a system using source code analysis, perform an initial clustering of highly connected classes, and compute code metrics and (2) a *processing phase* in which we apply metric-based rules, called SI rules in the following, on the clusters to filtrate, reorganise, and classify them to identify candidate services and their types.

5.2.1 Pre-processing Phase

Model Generation Our SI rules in Table 5.1 use code metrics, such as fanin and fanout, computed on the *model* of the legacy system. Legacy systems come in different languages and

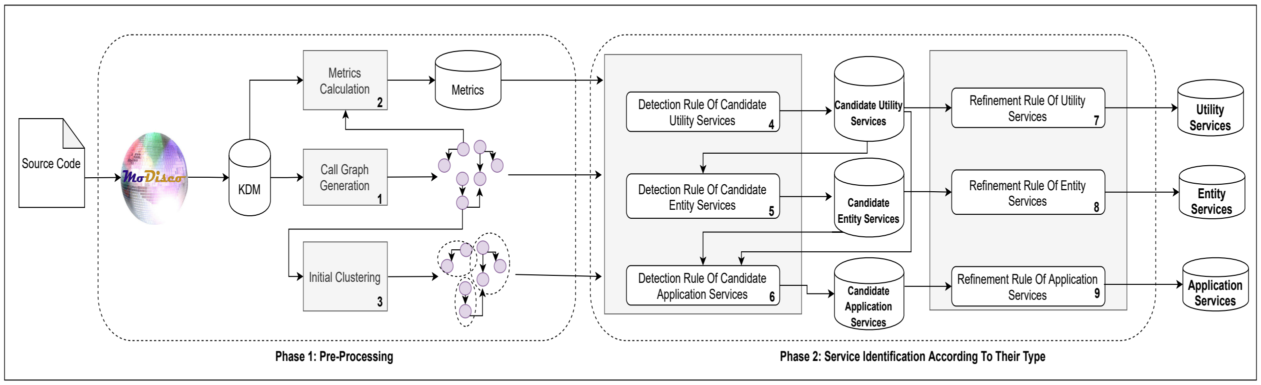

Source Code
MoDisco
KDM
Metrics Calculation 2
Metrics
Call Graph Generation 1
Initial Clustering 3
Phase 1: Pre-Processing
Detection Rule Of Candidate Utility Services 4
Detection Rule Of Candidate Entity Services 5
Detection Rule Of Candidate Application Services 6
Candidate Utility Services
Candidate Entity Services
Candidate Application Services
Refinement Rule Of Utility Services 7
Refinement Rule Of Entity Services 8
Refinement Rule Of Application Services 9
Utility Services
Entity Services
Application Services
Phase 2: Service Identification According To Their Type

Figure 5.1 Overview of our SIA

may combine several technologies. In this first step, we parse the source code of the legacy system and build its model based on the OMG *Knowledge Discovery Metamodel* (KDM) [109], which was defined to represent (legacy) systems at different levels of abstraction, regardless of languages and technologies.

We use MoDISCO [110], an Eclipse-based open-source implementation of the KDM that provides an extensible framework (1) to obtain KDM models from source code in different languages and (2) to navigate the KDM models, which we use to compute metrics on the models.

Metrics Calculation Our rules use class-level and method-level metrics. We use the models obtained in the previous step to compute class-level metrics and, to simplify the implementation of our SIA, we use *Understand*[1] to compute method-level metrics.

We also analyze the static relationships between the constituents (classes, methods, etc.) of the model. A relationship may be a *generalization*, an *aggregation*, or an *association* between classes, for example. We assign a weight to each of them according to their relative importance. The total relationship weight between a pair of related constituents is:

$$Weight(C_i, C_j) = \sum_{t=1}^{T} W_t \times NR_t$$

where C_i and C_j are the constituents, T is the number of relationships, W_t the weight of a relationship of type t, and NR_t the number of such relationship between C_i and C_j.

Initial Clustering The SI rules in Table 5.1 apply to candidate clusters that group classes contributing together to some functionalities. We rely on Kruskal's maximum spanning-tree algorithm [111] to generate our initial set of clusters because (1) it is an efficient polynomial-time algorithm for generating clusters, (2) it was used by several state-of-the-art SIAs [13,57], and (3) a free implementation of the algorithm is provided in the open-source Java library *Jgrapht*, which can be integrated into our approach.

5.2.2 Processing Phase

Not every cluster is a candidate service. To identify clusters that could become services, we first assign them a type. We first discuss service types and their code patterns qualitatively and then express them as rules, shown in Table 5.1.

[1]http://www.scitools.com

In the following, we consider the identification of only Utility, Entity, and Application services and explain how to identify these types through the analysis of the source code of legacy systems.

We exclude Infrastructure services because legacy systems are likely *not* to contain such services by their very definition. We do not distinguish between Application and Enterprise services because they only differ in terms of scope of reuse: within a single system vs. across systems. Finally, we do not consider Business services because (1) they orchestrate other services, such as Application services and (2) other sources of information, e.g., business processes, are required to detect them.

Each type of service is mutually exclusive and their detection is hierarchical: first we detect candidate Utility services. Within the remaining clusters, we detect Entity and then Application services, as follows:

- **Utility services** provide highly generic functionalities that are separate from domain-specific business processes and reusable across a range of business functionalities [29]. We detect Utility services by identifying clusters with high fanin (highly solicited/called clusters) and low fanout (clusters that do not call many others). They are *domain-neutral* so the identified clusters should not persist data or contain database queries.
- **Entity services** represent and manage *domain-specific* business entities, such as products, invoices, etc. They are *data-centric* and reusable by other *domain-specific* services, such as Application services. We classify a cluster as an Entity service with (1) high fanin, (2) low fanout, (3) persistency, (4) access to the infrastructure (e.g., database), and (5) fine grained.
- **Application services** are domain- and system-specific. They have low fanin compared to Entity and Utility services. They also use Entity services. They generally perform complex computations and handle (user) errors. We classify a cluster as Application service if it has (1) a call to at least one Entity service, (2) a high McCabe complexity, and–or (3) error handling capabilities.

The main challenge of a clustering algorithm is to group accordingly related classes in the same cluster. when we used the clustering algorithm we noticed that some initial clusters are not architecturally significant because (1) they sometimes group classes that are not in the same domain, (2) not all the classes of the system are present in the clustering, and (3) there are initial clusters that are too fine grained (e.g. clusters with only one class). To enhance the quality of the identified candidate services, we add each missing class in a cluster before

Service Type	**Detection rules**
Utility Services	Very High Fanin **AND** Very Low Fanout **AND** **Not** persistent
Entity Services	**Not** Utility service **AND** High Fanin **AND** Low Fanout **AND** Persistent **AND** Access to infrastructure **AND** Fine grained
Application Services	**Not** Utility **AND** **Not** Entity **AND** Low Fanin **AND** (Call to Entity $\geq$1 **OR** High McCabe Complexity **OR** Error Handling)

Table 5.1 Detection rules of services according to their types

applying our SI rules to identify candidate services. After that, we apply the refinement rules on clusters/candidate services with the same type.

Listing 5.1 shows the pseudo code of the used refinement rules. Because service granularity, cohesion and coupling are essential quality criteria that should be considered in a SOA-based system, we consider these elements to merge candidate services. For example, to refine candidate entity services, we select clusters that are labelled *"entity"* (Line 8). We randomly select entity candidate service (Line 12) and check if it should be merged with another candidate service with the same type. The merge is done with the most coupled service if the total size of the final entity service is less than the maximum allowed size (Lines 14 to 21). We repeat the process several times until we reach a predefined maximum number of iterations (Line 10).

Listing 5.1 shows the use of several constants to direct the refinement rules. These constants must be set by the developers using our approach to identify services. They cannot be set in general because they depend on both the analysed system and the developers' requirements. The developers would set some starting values and then would apply our approach. Then, through trials and errors, they would identify the values most appropriate for the analysed system and their requirements, e.g., the sizes of services and number of iterations.

Listing 5.1 Pseudo Code of the refinement rules

```
IC            : Set of all initial clusters
TotalTypes    : Number of service types
MaxSize_i     : Maximum size of a cluster with Type_i
MaxIterations: Number of iterations

For (i=1 To TotalTypes )

 IC_i = Set of candidate services with Type_i in IC

 For (j = 1 To MaxIterations)

   Select randomly Service_j From IC_i

   Select Service_k from IC_i Where
         size(Service_k) + size(Service_j) <= MaxSize_i
        AND Coupling (Service_j, Service_k) = MaxCoupling_j

   If NotNull(Service_k) Then
     Merge((Service_j, Service_k)
     Remove(Service_j) from IC_i
   End if
 End For
End For
```

5.3 Experimental Validation

We validate our approach via (1) a quantitative validation on two legacy systems, (2) a qualitative validation of the services related to a particular feature of one system, and (3) a comparison of the results with those of three state-of-the-art approaches on the two systems [11,13].

5.3.1 Case Studies

We choose two Java legacy systems for our validation.

Compiere is one of the few large, Java, open-source *legacy* system available on the Internet. It is a *legacy system* because it is a large ERP system with a long history, first introduced by *Aptean* in 2003[2]. It provides businesses, government agencies, and non-profit organizations with flexible and low-cost ERP features[3], such as business partners management, monitoring and analysis of business performance, control of manufacturing operations, warehouse management (automating logistics), purchasing (automating procurement to payment), materials management (inventory receipts, shipments, etc.), and sales order management (quotes, book orders, etc.). It supports different databases, including Oracle and PostgreSQL. We use Compiere v3.3 because (1) it is the first stable release of the system, (2) it was released more than 15 years ago, (3) it includes 2,716 classes for more than 530 KLOC, and (4) it is not service-oriented.

FXML-POS is an open-source[4] inventory management system in Java. It includes 56 classes for 8 KLOC. It is based on the model-view-controller architecture and provides several features related to ERP, such as purchase management, sales management, supplier management, etc. We use FXML-POS because it offers several features and it is not service-oriented.

5.3.2 Ground Truths

We must build *ground truths* for our case studies, i.e., service-oriented versions of Compiere and FXML-POS against which to compare the results of our approach.

We asked two independent Ph.D. and Master's students to identify services in the case studies. They relied on several artifacts to build manually ground truths by (1) analysing the systems, (2) understanding their core functionalities, and (3) extracting reusable classes that could become services.

They used *Understand* to recover their designs and to visualise class dependencies. They also generated views of the systems (Cf. Figures 5.2, 5.3 and 5.4), which we make available online[5]. They also reviewed extensively any documentation as well as their source code to have the best possible understanding of these systems and accurately identify services.

They found 477 services in Compiere and 22 services in FXML-POS, to which they manually assigned one of the three types Utility, Entity, or Application. These ground truths are

[2]http://www.aptean.com
[3]http://www.compiere.com/products/capabilities/
[4]https://github.com/sadatrafsanjani/JavaFX-Point-of-Sales
[5]http://si-serviceminer.com

available online[6] for study and replication.

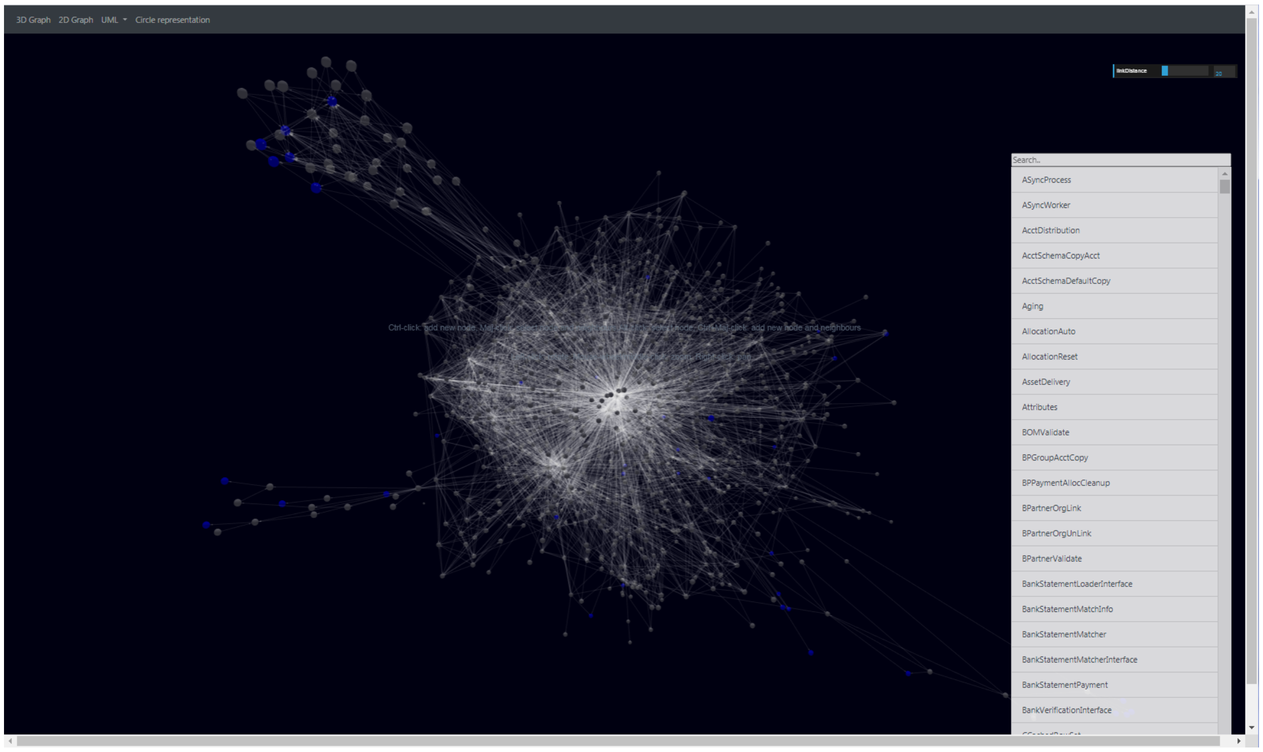
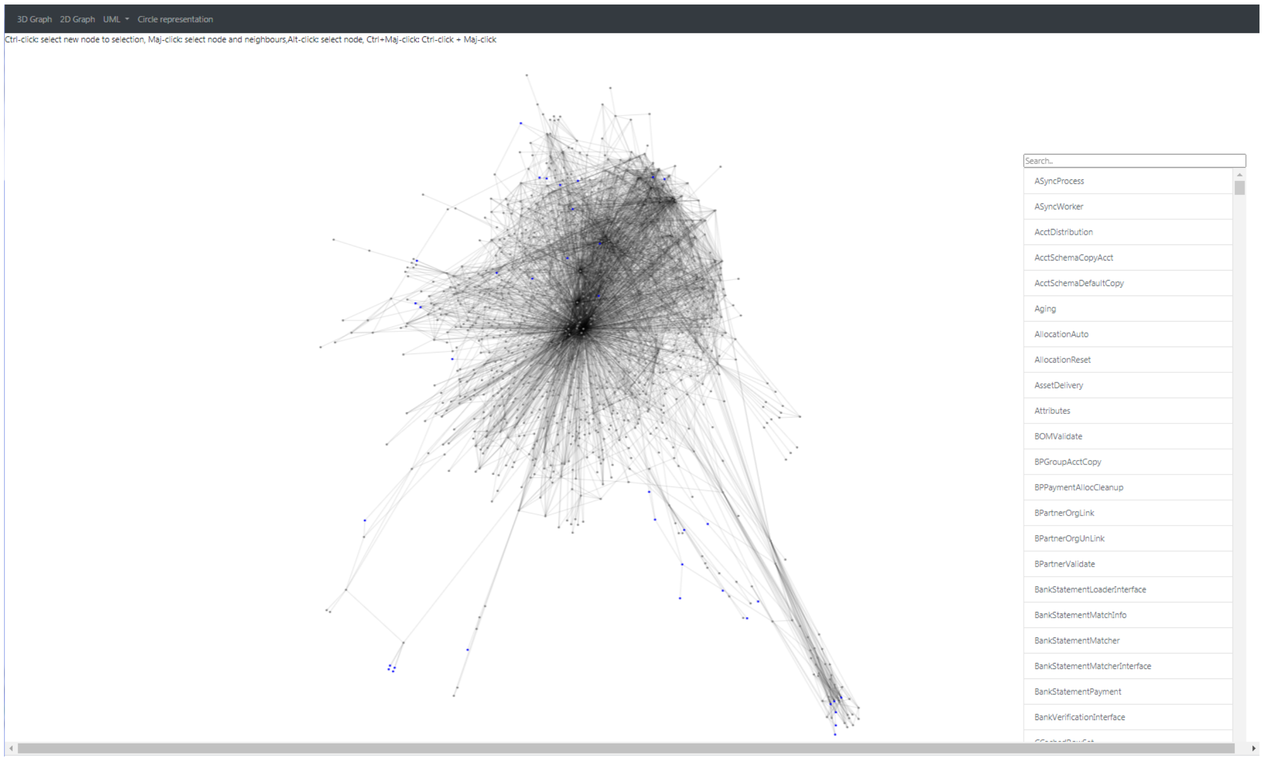


Figure 5.2 3D call graph of Compiere

[6]http://si-serviceminer.com/TSE-Replication



Ctrl-click: select new node to selection, Maj-click: select node and neighbours,Alt-click: select node, Ctrl+Maj-click: Ctrl-click + Maj-click

Figure 5.3 2D call graph of Compiere

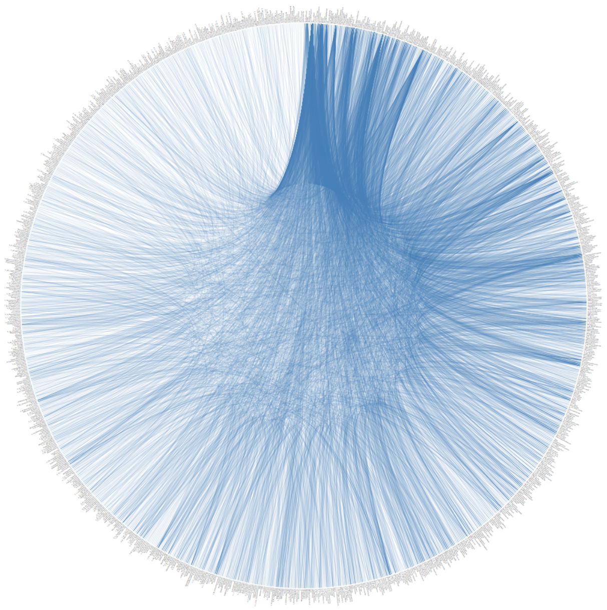


Figure 5.4 Circular representation of dependencies in Compiere

5.3.3 Evaluation metrics

We assess our approach with respect to a ground-truth architecture and in comparison to other tools using measures of clustering and information retrieval: MojoFM, A2A, precision, and recall.

MojoFM MojoFM measures the similarity between two set of (candidate) services, S_A and S_B [112] by counting the numbers of changes to apply to S_A to obtain S_B. Changes include moving classes from one cluster to another (Move) and merging clusters (Join), defined as:

$$MojoFM(S_A, S_B) = \left(1 - \frac{mno(S_A, S_B)}{max(mno(\forall S_A, S_B))}\right) \times 100$$

with $mno(S_A, S_B)$ the minimum number of Move and Join operations needed to transform A into B. $mno(\forall S_A, S_B)$ is the minimum number of Move and Join operations needed to transform an arbitrary decomposition into B. S_A is the set of services identified by one approach while S_B is either those provided by another approach or a ground truth. $MojoFM(S_A, S_B)$ reflects the amount of effort needed to transform one architecture into another: the greater the value, the greater the similarity, the less the effort.

Architecture2Architecture Architecture2Architecture (A2A) is also a measure of similarity, considering five operations to transform one set into another [113], which overcomes some of MojoFM limitations [114], defined as:

$$A2A(S_A, S_B) = \left(1 - \frac{mto(S_A, S_B)}{aco(S_A) + aco(S_B)}\right) \times 100$$

with:

$$\begin{aligned}
&mto(S_A, S_B) = \\
&remC(S_A, S_B) + addC(S_A, S_B) \\
&+ remE(S_A, S_B) + addE(S_A, S_B) + movE(S_A, S_B)
\end{aligned}$$

and:

$$\begin{aligned} &aco(S_A) = \\ &addC(S_{A_0}, S_A) + addE(S_{A_0}, S_A) + movE(S_{A_0}, S_A) \end{aligned}$$

where $mto(S_A, S_B)$ is the minimum number of operations needed to transform S_A into S_B and $aco(S_A)$ is the number of operations needed to construct S_A from an "empty" set S_{A_0}. These two functions, *mto* and *aco*, use additions $addE$, removals $remE$, and moves $movE$ of classes from one cluster to another and additions $addC$ and removals $remC$ of clusters.

Precision and Recall Precision and recall are defined as usual as:

$$Precision = \frac{TP}{TP + FP}$$

and:

$$Recall = \frac{TP}{TP + FN}$$

with TP the number of correctly identified services, FP the number of wrong services, and FN the number of missing services. We also use the F-measure, which is the harmonic average of the precision and recall:

$$F - measure = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

5.3.4 Quantitative Validation

We applied *ServiceMiner* on Compiere to show its accuracy in identifying services in legacy systems by considering the different possible combinations of the metrics in each rule. This allows us refine accordingly the detection rules and keep the ones that led to the best detection results. We measured precision, recall, and F-measure for each rule and report the results in Figures 5.5, 5.6, and 5.7.

For example, for Utility services, we considered the combinations "very high fanin", "very low fanout", and "Not persistent". As shown in Figure 5.6, the best F-measure is obtained when considering the three metrics to identify Utility services: considering clusters with only very high fanin or very low fanout or clusters that are only not persistent leads to poor precision values.

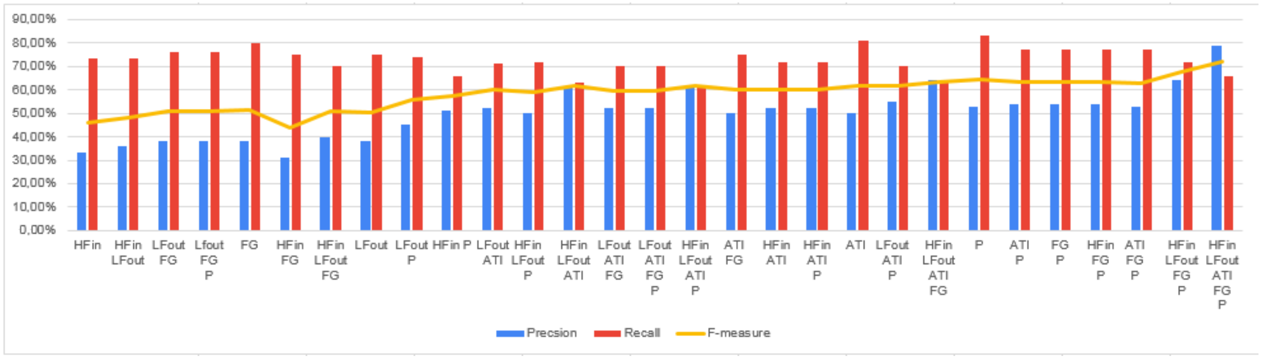


Figure 5.5 Evaluation of the detection rules of Entity Services on Compiere

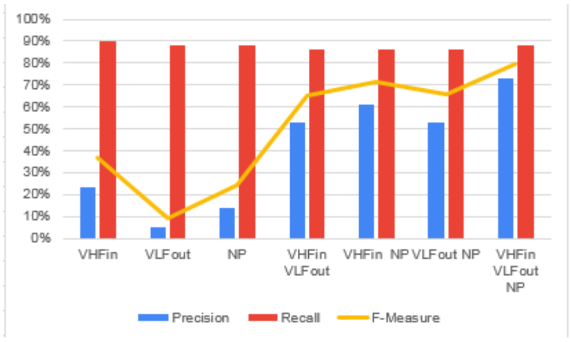


Figure 5.6 Evaluation of the detection rules of Utility Services on Compiere

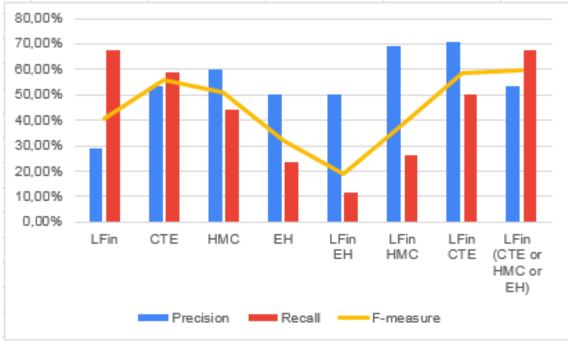


Figure 5.7 Evaluation of the detection rules of Application Services on Compiere

We repeated the same analysis for the detection rules of Entity and Application services. As shown in Figures 5.5, 5.6, and 5.7, we obtained the best F-measure values when using the detection rules detailed in Section 5.2.2.

After validating the rules on Compiere, we applied them on FXML-POS to identify the services according to their types. When analyzing the detection results of Utility services in FXML-POS, we found that the detection rule should be tuned to consider as well non-persistent clusters that have very low fanin and very low fanout. All the other rules were applied the same as in Compiere.

We identified in Compiere 445 services: 43 Application services, 299 Entity services, and 103 Utility services. For FXML-POS we identified 21 services: 9 Application services, 8 Entity services, and 4 Utility services. We report in Table 5.3 the overall average accuracy of *ServiceMiner*: a precision of 80.5%, a recall of 76%, and a F-measure of 78.2%.

For Utility services we obtained an average precision of services with a precision of 73.9%

	# Services		Precision			Recall			F-measure		
	Compiere	**FXML-POS**	**Compiere**	**FXML-POS**	**Average**	**Compiere**	**FXML-POS**	**Average**	**Compiere**	**FXML-POS**	**Average**
Utility	103	4	(75/103) 72,80%	(3/4) 75,00%	73,90%	(75/85) 88,24%	(3/3) 100,00%	94,12%	79,79%	85,70%	82,75%
Entity	299	8	(237/299) 79,30%	(8/8) 100,00%	89,65%	(237/358) 66,20%	(8/9) 88,90%	77,55%	72,15%	94,10%	83,13%
Application	43	9	(23/43) 53,50%	(7/9) 77,80%	65,65%	(23/34) 67,65%	(7/8) 87,50%	77,58%	59,74%	82,40%	71,07%
Total	445	21	(335/445) 75,30%	(18/21) 85,70%	80,50%	(335/477) 70,23%	(18/22) 81,80%	76,02%	72,67%	83,70%	78,19%

Table 5.3 Overview of Service Identification Accuracy with *ServiceMiner*

and a recall of 94.12%. The identified Utility services related to logging, Web uploading, printing, etc. For Entity services, we obtained a precision of 89.65%, a recall of 77.55%, and a F-measure of 83.13% with services for products, orders, invoices, etc. We missed some Entity services that have a low fanin/high fanout because of our choice of metrics and thresholds, which could be refined by developers when applied on their own systems. We observed a precision of 65.65%, a recall of 77.58%, and a F-measure of 71.07% for Application services, which relate to payment processing, tax calculation, and inventory management. The identification of Application services depends on the previous identifications of Entity and Utility services and, thus, false-positive Application services were mainly due to some Entity and Utility services being incorrectly labelled as Application services, such as caching-related services and Web-project deployment services.

Although it missed the identification of some services, *ServiceMiner* could reduce the developers' effort needed to identify services by avoiding the manual analysis and identification of reusable services. Our recall could be improved by setting different thresholds and iterating through the identification process.

5.3.5 Qualitative Validation

We take the example of the sales-orders management in Compiere and detail how *ServiceMiner* helps practitioners identify services related to this functionality.

Sales orders management entails quotations, sales orders, and invoices, linked to the shipment of goods to customers. The initial clustering step of our approach builds a set of candidate clusters that we filtrated with our detection rules to identify candidate services. First, we identified Utility services, related to logging and printing. These services provide cross-cutting functionalities called by other services with very low fanout and no persistence.

Second, we identified Entity services, i.e., clusters representing business entities, which related to products, invoices, business partners, warehouses, and bank statements. These entities are persistent, have access to the database, and are used by other domain-specific services (e.g., Application and Business services).

Third, we identified Application services among the remaining clusters. An example of Application service related to sales-orders processing is the payment service responsible for generating payments based on the information from an invoice, a business partner, and a bank statement Entity services. It is also responsible for handling errors, e.g., if the payment is unsuccessful.

We applied *ServiceMiner* on FXML-POS and obtained architecturally significant services as well. We applied our first detection rule on the initial clustering. We correctly identified three utility services related to login, hibernate utilities, and pdf document generation and printing. We applied then our second detection rule to identify entity services in the system. The identified entities are related to products, sales orders, purchases, employees, invoices, and product categories. Finally, we applied our third detection rule to identify Application services. These services are related to the different controllers in the system such as products, invoice, and supplier controllers, etc.

We obtained architecturally significant candidate services thanks to the application of our type-aware SIA. We believe that it can thus assist practitioners in the identification of candidate services because it highly automates the SI process of Utility, Entity, and Application services with acceptable precision and recall.

5.3.6 Comparison with State of the Art

We chose three existing approaches to compare their results against those of *ServiceMiner*: *MOGA-WSI* [13], *Service Cutter* [11] and *MicroserviceExtraction* [57]. These three were the only approaches available online.

MOGA-WSI uses spanning trees and provides candidate services with different levels of inter-service coupling. It relies on genetic and multi-objective optimisation algorithms to refine an initial set of services. It also considers a set of managerial goals, such as cost effectiveness, ease of assembly, customization, reusability, and maintainability.

Service Cutter is a graph-based approach considering 16 coupling metrics and two kinds of clustering algorithms, *Girvan-Newman* (GN) [115] and *Epidemic Label Propagation* (ELP) [116], with different (non-)deterministic behaviour.

MicroserviceExtraction is also a graph-based approach that relies on coupling metrics. It also uses an historical analysis of the system, via GitHub, to identify logically and semanticallyrelated classes. Coupling metrics are used to identify clusters in the system, which are candidate services.

We assessed our approach with respect to our ground truths and in comparison to the three approaches, using MojoFM [112], *Architecture2Architecture* (A2A) [114], precision, and recall. We relied on these metrics to study the identification results of each approach regardless of the types of services.

Table 5.4 shows the identification results. Our approach outperforms *MOGA-WSI*, *Service Cutter*, and *MicroserviceExtraction* for all the reported metrics. We tried several configurations for each approaches but all showed lower results in comparison to our SIA. We observed that these approaches generated very unbalanced services. For example, *MOGA-WSI* identified one service with 253 classes and 143 services with only one to six classes. Similarly, *Service Cutter* identified two coarse-grain services and 393 fine-grained ones. Although service identification using *Service Cutter* with *Girvan-Newman* is deterministic, we were limited to a maximum number of 30 services, which lead to poor identification results.

Some of the coupling metrics of *MicroserviceExtraction* rely on the historical analysis of the system to be migrated. However commits of Compiere and FXML-POS are not reported on GitHub. This may be also the cause of the poor identification results of the tool.

We argue that our SI approach outperforms the three other approaches because: (1) *ServiceMiner* follows a stepwise process, which identifies Utility services then Entity services and finally Application services, (2) it uses simple, straightforward metrics instead of complex goals, like maintainability, which are subjective and difficult to define and measure, and (3) the studied state-of-the-art approaches are proofs of concept, with limited applicability on enterprise-scale systems (such as Compiere).

	Services		MojoFM			A2A			Precision			Recall			F-measure		
	Compiere	**POS**	**Compiere**	**POS**	**Average**	**Compiere**	**POS**	**Average**	**Compiere**	**POS**	**Average**	**Compiere**	**POS**	**Average**	**Compiere**	**POS**	**Average**
MOGA-WSI	396	30	11.0%	18.3%	14.65%	42.0%	36.6%	39.3%	14.0%	6.6%	10.3%	13.0%	9.1%	11.1%	13.5%	7.7%	10.6%
Service Cutter (EPL)	395	38	15.7%	20.5%	18.1%	51.0%	65.3%	58.2%	12.2%	18.9%	15.6%	10.3%	31.8%	21.1%	11.2%	23.7%	17.5%
Service Cutter (GN)	30	30	21.6%	20.5%	21.05%	41.0%	66.7%	53.9%	15.6%	10.4%	13%	9.7%	13.6%	11.7%	14.1%	11.8%	13%
MicroserviceExtaction	n/a	11	n/a	11.4%	11.4%	n/a	31.2%	31.2%	n/a	63%	63%	n/a	31.8%	31.8%	n/a	42.3%	42.3%
ServiceMiner	445	21	67.0%	70%	68.5%	74.1%	91.7%	82.9%	75.3%	85.7%	80.5%	70.2%	81.8%	76%	72.7%	83.7%	78.2%

Table 5.4 Comparison results of service identification approaches

5.4 Threats to Validity

Internal Validity. Our SI approach as well as its validation depend on several algorithms and thresholds. To mitigate any threat, (1) we implemented *MOGA-WSI* based on its original papers to the best of our understanding and shared it for investigation and replication[7]; (2) we used the best identification results of the approaches; (3) we explored the use of different metrics and threshold values; and, (4) we chose the spanning-tree algorithm for its ease of use and availability. Future work should compare with other algorithms to vet further the reliability of our SIA.

We know that service detection rules must be adapted from one system to another. Our detection rules are easily customised, being flexible and extensible. We recommend to consider the same processing steps in the same order to identify services according to their types. Also, legacy systems most likely embed poor design and coding practices, e.g., code smells, that reduce the separation of concerns within/between classes, which reduces the precision/recall of static-based SIAs. The used refinement rules only add services that were divided/ignored by the clustering algorithm. Other rules could be added to enhance the quality of ill-clustered candidate services.

Construct validity. We should have relied on a real post-migration SOA-based system. However, we could not find any open-source, enterprise-scale system that was migrated to SOA. Thus, we relied on ground truths to validate quantitatively the services identified by our approach. No single "correct" SOA version exists for a given legacy system. We relied on several artifacts (e.g., official documentations, source code, etc.) and studied in-depth the systems to identify reusable sets of classes that could be packaged into services. Also, we shared our ground truths for others to confirm/infirm our claims. We put the original source code as well as the identified services online[8], which also reduces threats to reliability validity.

External validity. Our case studies may not be representative of all legacy systems, which limits the generalisability of our results. Compiere is large and complex enough to validate our approach. We considered another validation system while we continue our search for other large, open-source, legacy systems. The identified services may not be representative of all service types. Our approach is extensible to new service types. It can also be extended to identify microservices [117] by mapping each Utility and Entity service with a microservice and decomposing each identified Application service into smaller microservices that each have a single responsibility.

[7]https://github.com/MPoly2018/MOGA-WSI

[8]http://si-serviceminer.com/TSE-Replication

5.5 Conclusion

In this chapter, we proposed *ServiceMiner*, a type-sensitive SIA for the migration of legacy software systems to a service-oriented architecture (SOA). We evaluated *ServiceMiner* on a real-world legacy ERP system, Compiere, and an inventory management system, FXML-POS. We showed that, in general, *ServiceMiner* identified relevant, architecturally-significant services with 80.5% of precision, 76% of recall, and 78.2% of F-measure. We also compared ServiceMiner with three state-of-the-art SIAs. We showed that it outperformed the state-of-the-art SIAs by providing more architecturally-significant services.

CHAPTER 6 CONCLUSION

IT departments strain under the needs to automate business processes within, and across, organizational boundaries, and to develop and deploy new applications. They also contend with heterogeneous systems that become legacy and challenging to maintain. These systems embed hidden knowledge that is of significant values. They cannot be simply removed or replaced because they execute effectively and accurately critical and complex business logic. However, legacy systems are difficult to maintain and scale because their software and hardware become obsolete. They must be modernized to ease their maintenance and evolution.

A common strategy for modernizing legacy systems is to migrate them to service-oriented architecture (SOA). A key step in the migration process is the identification of reusable functionalities in the system that qualify as candidate services in the target architecture. There are several approaches related to service identification that have been proposed in the literature. However, to the best of our knowledge, there is no systematic study of the gap between academia and industry in terms of service identification approaches. The proposed approaches have in general limited identification accuracy. They usually require different types of inputs that may not be always available especially in the context of legacy systems. Finally, most of the proposed approaches do not consider the identification of specific types of services in legacy systems.

On solving the above problems, we formulated our thesis statement as below:

There is a gap between academia and industry in their service identification approaches to migrate legacy systems to SOA. To fill this gap, service identification should use static analyses of the source code and be driven by service types. Service types could be used to classify service candidates hierarchically and to prioritize the identification of specific types of services according to the business requirements of the migration process.

6.1 Summary

This thesis supports the migration of legacy systems to SOA by (1) studying the state of the practices of service identification in academia and industry , and (2) proposing *ServiceMiner* a bottom-up service identification approach that relies on source code analysis, because other sources of information may be unavailable or out of sync with the actual code. Our bottom-up, code-based approach uses service-type specific functional-clustering criteria. We use a categorization of service types that builds on published service taxonomies and describes the code-level patterns characterizing types of services.

A systematic literature review on service identification approaches. We did a systematic literature review (SLR) that covers 41 SIAs based on software-systems analyses. Based on this SLR, we created a taxonomy of SIAs and build a multi-layer classification of existing identification approaches. We started from a high-level classification based on the used inputs, the applied processes, the given outputs, and the usability of the SIAs. We then divided each category into a fine-grained taxonomy that helps practitioners in selecting a suitable approach for identifying services in legacy software systems. We build our SLR based on our experience with legacy software modernization, on discussions and experiences working with industrial partners, and analyses of existing SIAs. We identified the main challenges that SIAs need to address, to improve their quality. This classification helps practitioners in selecting a suitable service identification approach that corresponds to their migration needs.

Study of the state of the practices of services identification in industry. We conducted an online survey with 45 industrial practitioners and interviewed eight of them to report their experiences with the migration of legacy systems. Our study showed that practitioners relied on different migration strategies. We also showed that reducing maintenance costs and improving the flexibility and interoperability of legacy systems were the main motivations to migrate these systems to SOA. Practitioners consider SI an important step for the migration of legacy systems to SOA, as it promotes software reuse. Moreover, our results showed that SI is a process driven by business value rather than quality criteria, even though some practitioners consider some quality criteria (mainly reusability, granularity, and loose coupling). We showed that most of industrial SIAs remain a manual process in which human experts'feedbacks is essential. Finally, we reported that many practitioners considered the identification of specific types of services from legacy systems and concluded that SI should be driven by service types to identify more relevant and architecturally significant services.

Gap Analysis of SIAs in academia and industry. We studied academic and industrial SIAs and derived several recommendations for how to identify services in legacy systems to support their migration to SOA. These recommendations served as a support for our service identification approach that relies on static analysis of legacy systems while identifying specific types of services.

A type-sensitive service identification approach to support the migration of legacy systems to SOA. We proposed *ServiceMiner*, a type-sensitive SIA that relies on static analysis of the source-code of legacy systems, because other sources of information may be unavailable or out of sync with the actual code. *ServiceMiner* relies on a taxonomy of service types and identification rules to extract reusable services from legacy systems according to their types. We evaluate *ServiceMiner* on two case studies: an open-source enterprise-scale legacy ERP system, and an inventory management system. Then, we compared our results to those of three state-of-the-art approaches. We showed that ServiceMiner identifies architecturally-significant services with, on average, 80.5% precision, 76% recall, and 78.2% F-measure.

6.2 Discussions and Recommendations

We believe that our approach is beneficial for both researchers and practitioners interested in migrating legacy systems to SOA because (1) we highly automated the SI process, which is one of the most labour-intensive step of the migration; (2) our SIA yields architecturally-significant candidate services that can be packaged and integrated in the targeted SOA system while identifying their types; (3) it offers the possibility to prioritise the identification of specific types of services; and, (4) it is extensible to new technologies/languages, thanks to its use of the KDM metamodel as intermediate representation for C, C++, Python, etc.

Finally, we recommend to consider service types to identify services in legacy systems to improve accuracy. We also recommend to order the services to be migrated. We suggest to start first with Utility services because they are highly reusable and invoked by other services; second, to continue with Entity services because they manage and represent the business entities of the system and are used by the other services; third, to identify Application services as they compose functionalities provided by Entity services; finally, to identify Business services that manage and compose/use the previous types of services.

6.3 Future Research

In future work, we will consider the identification of other types of services, such as Enterprise and Business services. We will also perform a qualitative validation of the significance and relevance of the identified services with developers. We will use an optimization algorithm to improve the clustering process and compare other algorithms to study the reliability of our approach. We will also apply AI techniques to classify the classes and then apply clustering techniques to identify services according to their types. Finally, we will complete our

SOA migration road-map by exploring automatic service packaging techniques to efficiently package and integrate the identified services into the targeted SOA platform.

REFERENCES

[1] C. E. Passaris, "The business of globalization and the globalization of business," vol. 9, no. 1, 2006.

[2] V. Rajlich, "Software evolution and maintenance," in *Proceedings of the on Future of Software Engineering.* ACM, 2014, pp. 133–144.

[3] G. Lewis *et al.*, "Smart: The service-oriented migration and reuse technique," DTIC Document, Tech. Rep., 2005.

[4] T. Erl, *Service-oriented architecture.* Pearson Education Incorporated, 2005, vol. 8.

[5] R. Khadka *et al.*, "Migrating a large scale legacy application to soa: Challenges and lessons learned," in *2013 20th Working Conference on Reverse Engineering (WCRE).* IEEE, 2013, pp. 425–432.

[6] R. S. Huergo *et al.*, "A systematic survey of service identification methods," *Service Oriented Computing and Applications*, vol. 8, no. 3, pp. 199–219, 2014.

[7] B. Bani-Ismail and Y. Baghdadi, "A survey of existing evaluation frameworks for service identification methods: towards a comprehensive evaluation framework," in *International Conference on Knowledge Management in Organizations.* Springer, 2018, pp. 191–202.

[8] R. Khadka *et al.*, "A structured legacy to soa migration process and its evaluation in practice," in *Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA), 2013 IEEE 7th International Symposium on the.* IEEE, 2013, pp. 2–11.

[9] B. Bani-Ismail and Y. Baghdadi, "A literature review on service identification challenges in service oriented architecture," in *International Conference on Knowledge Management in Organizations.* Springer, 2018, pp. 203–214.

[10] G. Canfora *et al.*, "Migrating interactive legacy systems to web services," in *Conference on Software Maintenance and Reengineering (CSMR'06).* IEEE, 2006, pp. 10–pp.

[11] M. Gysel *et al.*, "Service cutter: A systematic approach to service decomposition," in *European Conference on Service-Oriented and Cloud Computing.* Springer, 2016, pp. 185–200.

[12] R. Rodríguez-Echeverría *et al.*, "Generating a rest service layer from a legacy system," in *Information System Development.* Springer, 2014, pp. 433–444.

[13] H. Jain, H. Zhao, and N. R. Chinta, "A spanning tree based approach to identifying web services," *International Journal of Web Services Research*, vol. 1, no. 1, p. 1, 2004.

[14] S. Adjoyan, A. Seriai, and A. Shatnawi, "Service identification based on quality metrics - object-oriented legacy system migration towards SOA," in *The 26th International Conference on Software Engineering and Knowledge Engineering, Hyatt Regency, Vancouver, BC, Canada, July 1-3, 2013.*, 2014, pp. 1–6.

[15] M. J. Amiri, S. Parsa, and A. M. Lajevardi, "Multifaceted service identification: Process, requirement and data," *Computer Science and Information Systems*, vol. 13, no. 2, pp. 335–358, 2016.

[16] Z. Zhang, H. Yang, and W. C. Chu, "Extracting reusable object-oriented legacy code segments with combined formal concept analysis and slicing techniques for service integration," in *2006 Sixth International Conference on Quality Software (QSIC'06).* IEEE, 2006, pp. 385–392.

[17] M. Abdellatif *et al.*, "State of the practice in service identification for soa migration in industry," in *International Conference on Service-Oriented Computing.* Springer, 2018, pp. 634–650.

[18] J. M. Rodriguez *et al.*, "Bottom-up and top-down cobol system migration to web services," *IEEE Internet Computing*, vol. 17, no. 2, pp. 44–51, 2013.

[19] R. Khadka *et al.*, "Does software modernization deliver what it aimed for? a post modernization analysis of five software modernization case studies," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME).* IEEE, 2015, pp. 477–486.

[20] C. Wagner, *Model-Driven Software Migration: A Methodology: Reengineering, Recovery and Modernization of Legacy Systems.* Springer Science & Business Media, 2014.

[21] A. Furda *et al.*, "Migrating enterprise legacy source code to microservices: on multitenancy, statefulness, and data consistency," *IEEE Software*, vol. 35, no. 3, pp. 63–72, 2017.

[22] M. Abdellatif *et al.*, "A taxonomy of service identification approaches for legacy software systems modernization," *J. Syst. Softw.*, vol. 173, p. 110868, 2021. [Online]. Available: https://doi.org/10.1016/j.jss.2020.110868

[23] ——, “Identifying reusable services in legacy object-oriented systems: A type-sensitive identification approach,” *IEEE Transactions on Software Engineering*, p. 14, under review 2021.

[24] H. Mili *et al.*, *Reuse-based Software Engineering: Techniques, Organization, and Controls.* New York, NY, USA: Wiley-Interscience, 2001.

[25] D. K. Barry, *Web Services, Service-oriented Architectures, and Cloud Computing: The Savvy Manager’s Guide.* Morgan Kaufmann, 2003.

[26] M. Nakamura *et al.*, “Extracting service candidates from procedural programs based on process dependency analysis,” in *Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific.* IEEE, 2009, pp. 484–491.

[27] A. Erradi, S. Anand, and N. Kulkarni, “Soaf: An architectural framework for service definition and realization,” in *Services Computing, 2006. SCC’06. IEEE International Conference on.* IEEE, 2006, pp. 151–158.

[28] A. Brown, S. Johnston, and K. Kelly, “Using service-oriented architecture and component-based development to build web service applications,” *Rational Software Corporation*, 2002.

[29] T. Erl, *SOA Principles of Service Design.* Upper Saddle River, NJ, USA: Prentice Hall PTR, 2007.

[30] B. Kitchenham, “Procedures for performing systematic reviews,” *Keele, UK, Keele University*, vol. 33, no. 2004, pp. 1–26, 2004.

[31] C. Wohlin, “Guidelines for snowballing in systematic literature studies and a replication in software engineering,” in *Proceedings of the 18th international conference on evaluation and assessment in software engineering.* ACM, 2014, p. 38.

[32] K. R. Felizardo *et al.*, “Using forward snowballing to update systematic reviews in software engineering,” in *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement.* ACM, 2016, p. 53.

[33] A. T. Zadeh *et al.*, “A systematic input selection for service identification in smes,” *Journal of Applied Sciences*, vol. 12, no. 12, p. 1232, 2012.

[34] S. Alahmari, E. Zaluska, and D. De Roure, “A service identification framework for legacy system migration into soa,” in *Services Computing (SCC), 2010 IEEE International Conference on.* IEEE, 2010, pp. 614–617.

[35] H. M. Sneed, C. Verhoef, and S. H. Sneed, "Reusing existing object-oriented code as web services in a soa," in *Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA), 2013 IEEE 7th International Symposium on the.* IEEE, 2013, pp. 31–39.

[36] L. Aversano, L. Cerulo, and C. Palumbo, "Mining candidate web services from legacy code," in *10th International Symposium on Web Site Evolution.* IEEE, 2008, pp. 37–40.

[37] A. Marchetto and F. Ricca, "From objects to services: toward a stepwise migration approach for java applications," *International journal on software tools for technology transfer*, vol. 11, no. 6, p. 427, 2009.

[38] R. S. Huergo, P. F. Pires, and F. C. Delicato, "Mdcsim: A method and a tool to identify services," *IT Convergence Practice*, vol. 2, no. 4, pp. 1–27, 2014.

[39] Y. Baghdadi, "Reverse engineering relational databases to identify and specify basic web services with respect to service oriented computing," *Information systems frontiers*, vol. 8, no. 5, pp. 395–410, 2006.

[40] M. Nakamur *et al.*, "Identifying services in procedural programs for migrating legacy system to service oriented architecture," *Implementation and Integration of Information Systems in the Service Sector*, p. 237, 2012.

[41] Y. Zhao *et al.*, "A service-oriented analysis and design approach based on data flow diagram," in *International Conference on Computational Intelligence and Software Engineering CiSE 2009.* IEEE, 2009, pp. 1–5.

[42] E. Sosa-Sánchez *et al.*, "Service discovery using a semantic algorithm in a soa modernization process from legacy web applications," in *Services (SERVICES), 2014 IEEE World Congress on.* IEEE, 2014, pp. 470–477.

[43] Z. Zhang and H. Yang, "Incubating services in legacy systems for architectural migration," in *11th Asia-Pacific Software Engineering Conference, 2004.* IEEE, 2004, pp. 196–203.

[44] H. Sneed, "Migrating to web services: A research framework," in *Proceedings of the International*, 2007.

[45] Z. Zhang, R. Liu, and H. Yang, "Service identification and packaging in service oriented reengineering." in *SEKE*, vol. 5, 2005, pp. 620–625.

[46] G. Chenghao, W. Min, and Z. Xiaoming, "A wrapping approach and tool for migrating legacy components to web services," in *First International Conference on Networking and Distributed Computing (ICNDC),2010.* IEEE, 2010, pp. 94–98.

[47] A. Fuhr, T. Horn, and V. Riediger, "Using dynamic analysis and clustering for implementing services by reusing legacy code," in *Reverse Engineering (WCRE), 2011 18th Working Conference on.* IEEE, 2011, pp. 275–279.

[48] D. Saha, "Service mining from legacy database applications," in *Web Services (ICWS), 2015 IEEE International Conference on.* IEEE, 2015, pp. 448–455.

[49] C. Del Grosso, M. Di Penta, and I. G.-R. de Guzman, "An approach for mining services in database oriented applications," in *11th European Conference on Software Maintenance and Reengineering, 2007. CSMR'07.* IEEE, 2007, pp. 287–296.

[50] S. Mani *et al.*, "Using user interface design to enhance service identification," in *Web Services, 2008. ICWS'08. IEEE International Conference on.* IEEE, 2008, pp. 78–87.

[51] R. S. Huergo, P. F. Pires, and F. C. Delicato, "A method to identify services using master data and artifact-centric modeling approach," in *Proceedings of the 29th Annual ACM Symposium on Applied Computing.* ACM, 2014, pp. 1225–1230.

[52] Y. Kim and K.-G. Doh, "The service modeling process based on use case refactoring," in *International Conference on Business Information Systems.* Springer, 2007, pp. 108–120.

[53] L. Bao *et al.*, "Extracting reusable services from legacy object-oriented systems," in *software maintenance (ICSM), 2010 IEEE International Conference on.* IEEE, 2010, pp. 1–5.

[54] M. Djeloul, "Locating services in legacy software:information retrieval techniques, ontology and fca based approach," *WSEAS Transactions on Computers*, vol. 11, no. 1, pp. 19–26, 2012/01/, legacy software;information retrieval techniques;FCA based approach;Web services technology;WORDNET ontology;formal concepts analysis;source code;.

[55] L. Baresi, M. Garriga, and A. De Renzis, "Microservices identification through interface analysis," in *European Conference on Service-Oriented and Cloud Computing.* Springer, 2017, pp. 19–33.

[56] W. Jin *et al.*, "Functionality-oriented microservice extraction based on execution trace clustering," in *2018 IEEE International Conference on Web Services (ICWS)*. IEEE, 2018, pp. 211–218.

[57] G. Mazlami, J. Cito, and P. Leitner, "Extraction of microservices from monolithic software architectures," in *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, 2017, pp. 524–531.

[58] E. Souza, A. Moreira, and C. De Faveri, "An approach to align business and it perspectives during the soa services identification," in *2017 17th International Conference on Computational Science and Its Applications (ICCSA)*. IEEE, 2017, pp. 1–7.

[59] A. A. C. De Alwis *et al.*, "Discovering microservices in enterprise systems using a business object containment heuristic," in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, 2018, pp. 60–79.

[60] M. Abdelkader, M. Malki, and S. M. Benslimane, "A heuristic approach to locate candidate web service in legacy software," *International Journal of Computer Applications in Technology*, vol. 47, no. 2-3, pp. 152–161, 2013.

[61] S. Tyszberowicz *et al.*, "Identifying microservices using functional decomposition," in *International Symposium on Dependable Software Engineering: Theories, Tools, and Applications*. Springer, 2018, pp. 50–65.

[62] D. Escobar *et al.*, "Towards the understanding and evolution of monolithic applications as microservices," in *2016 XLII Latin American Computing Conference (CLEI)*. IEEE, 2016, pp. 1–11.

[63] D. Taibi and K. Systä, "From monolithic systems to microservices: A decomposition framework based on process mining," in *8th International Conference on Cloud Computing and Services Science, CLOSER*, 2019.

[64] A. A. C. De Alwis *et al.*, "Function-splitting heuristics for discovery of microservices in enterprise systems," in *International Conference on Service-Oriented Computing*. Springer, 2018, pp. 37–53.

[65] L. Nunes, N. Santos, and A. R. Silva, "From a monolith to a microservices architecture: An approach based on transactional contexts," in *European Conference on Software Architecture*. Springer, 2019, pp. 37–52.

[66] A. Selmadji *et al.*, "Re-architecting oo software into microservices," in *European Conference on Service-Oriented and Cloud Computing.* Springer, 2018, pp. 65–73.

[67] D. Hix and H. R. Hartson, *Developing user interfaces: ensuring usability through product & process.* John Wiley & Sons, Inc., 1993.

[68] M. Weske, "Business process management architectures," in *Business Process Management.* Springer, 2012, pp. 333–371.

[69] A. Vemulapalli and N. Subramanian, "Transforming functional requirements from uml into bpel to efficiently develop soa-based systems," in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems".* Springer, 2009, pp. 337–349.

[70] J. Schmuller, *Sams teach yourself UML in 24 hours.* Sams publishing, 2004.

[71] S. W. Ambler, *The object primer: Agile model-driven development with UML 2.0.* Cambridge University Press, 2004.

[72] M. Aggarwal and S. Sabharwal, "Test case generation from uml state machine diagram: A survey," in *Computer and Communication Technology (ICCCT), 2012 Third International Conference on.* IEEE, 2012, pp. 133–140.

[73] T. C. Lethbridge, J. Singer, and A. Forward, "How software engineers use documentation: The state of the practice," *IEEE Software*, vol. 20, no. 6, pp. 35–39, 2003.

[74] E. Sosa *et al.*, "A model-driven process to modernize legacy web applications based on service oriented architectures," in *2013 15th IEEE International Symposium on Web Systems Evolution (WSE).* IEEE, 2013, pp. 61–70.

[75] S. Bechhofer, "Owl: Web ontology language," in *Encyclopedia of Database Systems.* Springer, 2009, pp. 2008–2009.

[76] F. Chen *et al.*, "Service identification via ontology mapping," in *2009 33rd Annual IEEE International Computer Software and Applications Conference*, vol. 1. IEEE, 2009, pp. 486–491.

[77] S. Zhao, E. Chang, and T. Dillon, "Knowledge extraction from web-based application source code: An approach to database reverse engineering for ontology development," in *2008 IEEE International Conference on Information Reuse and Integration.* IEEE, 2008, pp. 153–159.

[78] H. M. Sneed, “Integrating legacy software into a service oriented architecture,” in *Software Maintenance and Reengineering, 2006. CSMR 2006. Proceedings of the 10th European Conference on.* IEEE, 2006, pp. 11–pp.

[79] M. Balabanović and Y. Shoham, “Fab: content-based, collaborative recommendation,” *Communications of the ACM*, vol. 40, no. 3, pp. 66–72, 1997.

[80] G. Birkhoff, *Lattice theory.* American Mathematical Soc., 1940, vol. 25.

[81] G. Gratzer, *Lattice theory: First concepts and distributive lattices.* Courier Corporation, 2009.

[82] Rui Xu and D. Wunsch, “Survey of clustering algorithms,” *IEEE Transactions on Neural Networks*, vol. 16, no. 3, pp. 645–678, May 2005.

[83] B. Ganter, “Two basic algorithms in concept analysis,” *Formal Concept Analysis*, pp. 312–340, 2010.

[84] R. Wille, “Restructuring lattice theory: an approach based on hierarchies of concepts,” in *Ordered sets.* Springer, 1982, pp. 445–470.

[85] A. K. Jain, “Data clustering: 50 years beyond k-means,” *Pattern recognition letters*, vol. 31, no. 8, pp. 651–666, 2010.

[86] F. Murtagh and P. Legendre, “Ward’s hierarchical agglomerative clustering method: which algorithms implement ward’s criterion?” *Journal of classification*, vol. 31, no. 3, pp. 274–295, 2014.

[87] G. Feuerlicht *et al.*, “Understanding service reusability,” in *International Conference Systems Integration.* Department of Information Technologies and Czech Society for Systems Integration, 2007.

[88] V. Alkkiomäki and K. Smolander, “Anatomy of one service-oriented architecture implementation and reasons behind low service reuse,” *Service Oriented Computing and Applications*, vol. 10, no. 2, pp. 207–220, 2016.

[89] M. Perepletchikov *et al.*, “Coupling metrics for predicting maintainability in service-oriented designs,” in *2007 Australian Software Engineering Conference (ASWEC’07).* IEEE, 2007, pp. 329–340.

[90] R. Sindhgatta, B. Sengupta, and K. Ponnalagu, “Measuring the quality of service oriented design,” in *Service-Oriented Computing.* Springer, 2009, pp. 485–499.

[91] M. Bell, *SOA modeling patterns for service oriented discovery and analysis.* John Wiley & Sons, 2009.

[92] A. Shatnawi *et al.*, "Identifying software components from object-oriented apis based on dynamic analysis," in *Proceedings of the 26th Conference on Program Comprehension.* ACM, 2018, pp. 189–199.

[93] S. Cohen, "Ontology and taxonomy of services in a service-oriented architecture," *The Architecture Journal*, vol. 11, no. 11, pp. 30–35, 2007.

[94] D. Krafzig, K. Banke, and D. Slama, *Enterprise SOA: service-oriented architecture best practices.* Prentice Hall Professional, 2005.

[95] B. A. Ani and Y. Baghdadi, "A taxonomy-centred process for service engineering," *International Journal of Computer Applications in Technology*, vol. 52, no. 1, pp. 1–17, 2015.

[96] Q. Gu and P. Lago, "Service identification methods: a systematic literature review," in *Towards a Service-Based Internet.* Springer, 2010, pp. 37–50.

[97] R. Boerner and M. Goeken, "Service identification in soa governance literature review and implications for a new method," in *Digital Ecosystems and Technologies, 2009. DEST'09. 3rd IEEE International Conference on.* IEEE, 2009, pp. 588–593.

[98] D. Birkmeier, S. Klöckner, and S. Overhage, "A survey of service identification approaches-classification framework, state of the art, and comparison," *Enterprise Modelling and Information Systems Architectures*, vol. 4, no. 2, pp. 20–36, 2015.

[99] X. Cai *et al.*, "Component-based software engineering: technologies, development frameworks, and quality assurance schemes," in *Software Engineering Conference, 2000. APSEC 2000. Proceedings. Seventh Asia-Pacific.* IEEE, 2000, pp. 372–379.

[100] T. Vale *et al.*, "A study on service identification methods for software product lines," in *Proceedings of the 16th International Software Product Line Conference-Volume 2.* ACM, 2012, pp. 156–163.

[101] A. Taei Zadeh *et al.*, "A systematic input selection for service identification in smes," *Journal of Applied Sciences*, vol. 12, no. 12, pp. 1232–1244, 2012.

[102] J. Fritzsch *et al.*, "From monolith to microservices: a classification of refactoring approaches," in *International Workshop on Software Engineering Aspects of Continuous*

Development and New Paradigms of Software Production and Deployment. Springer, 2018, pp. 128–141.

[103] D. Birkmeier and S. Overhage, "On component identification approaches–classification, state of the art, and comparison," in *Component-Based Software Engineering.* Springer, 2009, pp. 1–18.

[104] S. Cai, Y. Liu, and X. Wang, "A survey of service identification strategies," in *Services Computing Conference (APSCC), 2011 IEEE Asia-Pacific.* IEEE, 2011, pp. 464–470.

[105] K. Charmaz and L. Belgrave, "Qualitative interviewing and grounded theory analysis," *The SAGE handbook of interview research*, pp. 347–365, 2012.

[106] P. Di Francesco, I. Malavolta, and P. Lago, "Research on architecting microservices: Trends, focus, and potential for industrial adoption," in *ICSA*, 2017, pp. 21–30.

[107] M. Abdellatif *et al.*, "State of the practice in service identification for soa migration in industry," in *ICSOC.* Springer, 2018, pp. 634–650.

[108] A. Arsanjani *et al.*, "Soma: A method for developing service-oriented solutions," *IBM systems Journal*, vol. 47, no. 3, pp. 377–396, 2008.

[109] G. E. Boussaidi *et al.*, "Reconstructing architectural views from legacy systems," in *WCRE*, 2012.

[110] H. Bruneliere *et al.*, "Modisco: A model driven reverse engineering framework," *IST*, vol. 56, no. 8, pp. 1012–1032, 2014.

[111] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical society*, vol. 7, no. 1, pp. 48–50, 1956.

[112] Z. Wen and V. Tzerpos, "An effectiveness measure for software clustering algorithms," in *Proceedings. 12th IEEE International Workshop on Program Comprehension, 2004.* IEEE, 2004, pp. 194–203.

[113] O. Maqbool and H. Babri, "Hierarchical clustering for software architecture recovery," *IEEE Transactions on Software Engineering*, vol. 33, no. 11, pp. 759–780, 2007.

[114] J. Garcia, I. Ivkovic, and N. Medvidovic, "A comparative analysis of software architecture recovery techniques," in *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering.* IEEE Press, 2013, pp. 486–496.

[115] M. E. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical review E*, vol. 69, no. 2, p. 026113, 2004.

[116] U. N. Raghavan, R. Albert, and S. Kumara, "Near linear time algorithm to detect community structures in large-scale networks," *Physical review E*, vol. 76, no. 3, p. 036106, 2007.

[117] S. Newman, *Building microservices: designing fine-grained systems.* " O'Reilly Media, Inc.", 2015.