A Taxonomy of Service Identification Approaches for Legacy Software Systems Modernization

Manel Abdellatif, Anas Shatnawi, Hafedh Mili, Naouel Moha, Ghizlane El Boussaidi, Geoffrey Hecht, Jean Privat, Yann-Gaël Guéhéneuc

Polytechnique Montréal, Montreal, Quebec, Canada
manel.abdellatif@polymtl.ca

Université du Québec à Montréal, Montreal, Quebec, Canada
{shatnawi.anas,mili.hafedh,moha.naouel,hecht.geoffrey,privat.jean}@uqam.ca

École de Technologie Supérieure, Montréal, Québec, Canada
ghizlane.elboussaidi@etsmtl.ca

Concordia University, Montréal, Québec, Canada
yann-gael.gueheneuc@concordia.ca

Abstract

The success of modernizing legacy software systems to Service-Oriented Architecture (SOA) depends on Service Identification Approaches (SIAs), which identify reusable functionalities that could become services. The literature describes several SIAs. However, the selection of an identification approach that is suitable for a practitioner is difficult because it depends on several factors, including the goal of modernization, the available legacy artifacts, the organization's development process, the desired output, and the usability of the approach. Accordingly, to select a suitable service identification approach, a practitioner must have a comprehensive view of existing techniques.

We report a systematic literature review (SLR) that covers 41 SIAs based on software-systems analyses. Based on this SLR, we create a taxonomy of SIAs and build a multi-layer classification of existing identification approaches. We start from a high-level classification based on the used inputs, the applied processes, the given outputs, and the usability of the SIAs. We then divide 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 validate the correctness and the coverage of our review with industrial experts who modernize(d) legacy software systems to SOA. The results show that our classification conforms to the industrial experts' experiences. We also show that most of the studied SIAs are still at their infancy. Finally, we identify the main challenges that SIAs need to address, to improve their quality.

Keywords: Service Identification, Microservices, Taxonomy, Legacy System, Migration

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 still of significant values. They cannot be removed or replaced because they execute effectively and accurately critical and complex business logic. Yet, legacy software systems suffer from several drawbacks including high maintenance costs, scalability and portability problems, and so forth [1]. Thus, these systems should be migrated to more flexible and modern architectures to retain their business values while decreasing their maintenance costs.

The migration of legacy software systems to a Service-Oriented Architecture (SOA) is one avenue for the modernization of these systems. SOA allows developing complex and inter-organizational systems by integrating and composing services that are reusable, distributed, relatively independent, and often heterogeneous [2]. Also over the past few years, increasing efforts have been made to migrate legacy systems to microservices, which are, in a SOA architecture, any services having a single responsibility, running in their own processes, and communicating with lightweight mechanisms [3]. In the following, we use the term "service" to cover any form/granularity of services, including microservices.

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 [4, 5]. 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) modeling 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 orchestrating 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 [6, 7]. 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. [1], 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).The challenges of turning such legacy code into autonomous and self-contained services include dealing with multi tenancy, data consistency and statefulness The legacy code might have side effects that violate one or more service design principles. These challenges must be considered after identifying the services, as part of the whole migration process of legacy software systems [8].

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.

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 [7, 9] 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 practitioners in selecting a suitable service identification approach that corresponds to their migration needs. We perform our SLR using our experience with legacy software modernization, discussions with industrial partners, and the analysis of 41 papers retained from a first set of 3,246 papers. We validate the correctness and coverage of our SLR through a survey and one-to-one interviews with 45 industrial experts in legacy software-systems modernization. The results show that our taxonomy conforms to the industrial experts' experiences, with a precision of 99%, and a recall of 94%.

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

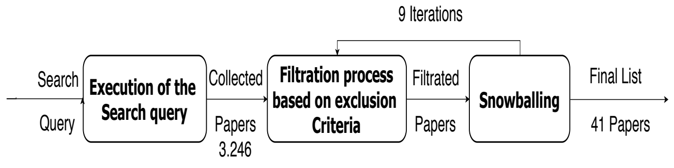

Figure 1: Paper selection

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.

1.2. Outline

The remainder of this paper is structured as follows. Section 2 describes our SLR methodology. Section 3 describes the inputs used by SIAs. Section 4 describes the processes that underlie the studied SIAs. Section 5 surveys the outputs of SIAs. Section 6 describes the usability level of these SIAs. Section 7 details the validation of our taxonomy. Section 8 synthesizes the comparison between the studied SIAs. Section 9 describes related work. Finally, Section 10 concludes our work.

2. 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. [10] for performing systematic reviews.

Figure 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] and defined the following query:

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

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 analyzed all the papers and then reconciled any differences through discussions. We excluded from our review papers meeting 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.
- Papers published before 2004 and after 2019.

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 [11, 12] 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 1.

3. 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 [13]. 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 1, many SIAs in the literature relied on different types of inputs. When

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

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.

3.1. Executable Models

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

3.1.1. 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. [19] 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. [9] 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 using source code analysis leads to reusable and fine grained services, a combination of this kind of input with other artifacts (e.g., business processes, databases, etc.) can be used to identify services with more business values.

3.1.2. 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 [23, 33, 34]. These approaches identify data/entity services that provide access to, and management of, the persistent data of the systems(C.f. Section 5).

For example, Baghdadi et al. [23] 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. [33] 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. [34] 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 5), a forward-engineering process is needed to build more coarse-grained services, that combine these finer-grain services, into business services.

3.1.3. 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 [39, 42, 20]. For example, Bao et al. [39] 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. [42] 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

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 [15]	x						x	x					
Generating a REST Service Layer from a Legacy System [16]	x												
A service identification framework for legacy system migration into SOA [17]	x					x		x					
Reusing existing object-oriented code as web services in a SOA [18]	x					x							
Mining candidate web services from legacy code [19]	x												x
From objects to services: toward a stepwise migration approach for Java applications [20]	x		x			x							x
Migrating interactive legacy systems to web services [21]					x		x			x			
MDCSIM: A method and a tool to identify services [22]	x					x				x		x	
Reverse engineering relational databases to identify and specify basic Web services with respect to service oriented computing [23]		x											
Identifying services in procedural programs for migrating legacy system to service oriented architecture [24]	x								x				x
A service-oriented analysis and design approach based on data flow diagram [25]	x								x				
Service discovery using a semantic algorithm in a SOA modernization process from legacy web applications [26]	x					x					x	x	
Incubating services in legacy systems for architectural migration [27]	x				x	x							x
Migrating to Web services: A research framework [28]	x												x
Service Identification and Packaging in Service Oriented Re-engineering [29]	x					x							
A wrapping approach and tool for migrating legacy components to web services [30]	x												
Extracting reusable object-oriented legacy code segments with combined formal concept analysis and slicing techniques for service integration [31]	x										x		
Using dynamic analysis and clustering for implementing services by reusing legacy code [32]	x			x		x						x	
Service Mining from Legacy Database Applications [33]		x											
An approach for mining services in database oriented applications [34]		x									x		
Using user interface design to enhance service identification [35]	x				x								
A method to identify services using master data and artifact-centric modeling approach [36]	x	x				x						x	
Multifaceted service identification: Process, requirement and data [37]		x				x						x	
The service modeling process based on use case refactoring [38]	x						x					x	
Extracting reusable services from legacy object-oriented systems [39]	x		x	x			x					x	x
Locating services in legacy software:information retrieval techniques, ontology and FCA based approach [40]	x				x						x		
Microservices Identification Through Interface Analysis [41]	x				x								
Functionality-Oriented Microservice Extraction Based on Execution Trace Clustering [42]	x		x	x	x							x	
Bottom-up and top-down cobol system migration to web services [9]	x											x	x
Extraction of microservices from monolithic software architectures [43]	x												
Service Cutter: A Systematic Approach to Service Decomposition [44]	x						x						
An approach to align business and IT perspectives during the SOA services identification [45]	x					x						x	
Discovering Microservices in Enterprise Systems Using a Business Object Containment Heuristic [46]	x	x		x								x	
A heuristic approach to locate candidate web service in legacy software [47]	x												
Identifying Microservices Using Functional Decomposition [48]	x						x						
Towards the understanding and evolution of monolithic applications as microservices [49]	x												
From Monolithic Systems to Microservices: A Decomposition Framework based on Process Mining [50]	x			x		x							
Function-Splitting Heuristics for Discovery of Microservices in Enterprise Systems [51]		x		x		x	x						
From a Monolith to a Microservices Architecture: An Approach Based on Transactional Contexts [52]	x												
Re-architecting OO Software into Microservices A Quality-centered approach [53]	x												

Table 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)*

cohesive and loosely coupled group of classes that will be mapped into services.

As shown by Table 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.

3.2. 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 below.

3.2.1. 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 [32, 39, 42, 46, 50, 51]. These approaches identify pieces of legacy code executed during a set of business processes [32] or use cases [39], 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. [32] 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 [54]. If the workflow model of a system is not available, knowledge extracted from its user interfaces is useful to recover its underlying business logic [35, 27].

We found five SIAs that analyse users' interactions with user interfaces to identify services [41, 35, 21, 27, 40, 42]. For example, Mani et al. [35] proposed an XML-based representation, 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.

3.2.2. 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 [55]. 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 [37]. Business process-driven SIAs usually decompose business processes into tasks. These tasks are then clustered and exposed as services.

For example, Alahmari et al. [17] 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. [32] 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 [56]. We found seven SIAs that use such artifact [15, 21, 38, 39, 44, 48, 51].

For example, Bao et al. [39] 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 [57]. Only two SIAs use activity diagrams to identify services [15, 17]. For example, Al Ahmari et al. [17] 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 [58]. The main entities of a DFD are the (1) data stores storing data for later use, (2) external entities representing sources/destinations of the data, (3) processes manipulating the data, and (3) data flows. Only two SIAs use DFDs to identify reusable services [24, 25].

For example, Zhao et al. [25] 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 [24, 25]. 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 [59]. We found that only two SIAs use state machine diagrams as inputs [21, 22]. Canfora et al. [21] used these diagrams to model the interactions between users and systems. Huergo et al. [22] used them to model the life-cycle of master data, defined 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.

3.3. Domain Artifacts

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

3.3.1. Documentation

Software documentation describes and documents systems at different levels of abstraction [60]. 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 [9, 19, 24, 27, 28, 39]. For example, Aversano et al. [19] 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. [9] 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.

3.3.2. 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. [15]). It has also been used to define the business logic and translate it into business processes [17, 37, 61]. It is also needed to analyse use cases and identify candidate services [39]. Finally, human expertise is needed to define data flow diagrams of the system to then identify candidate services [24, 25, 28].

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.

3.3.3. Ontologies

An ontology is a structured set of terms representing the semantics of a domain, whether through metadata or elements of a knowledge domain [62]. Several SIAs use ontologies to identify services [34, 37, 40, 63].

For example, Djeloul et al. [40] 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. [63] 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 [64]. Finally, ontology-based SIAs are complex and require a lot of human expertise.

4. 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.).

4.1. 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 [21, 65].

- 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 [66].

- 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 [67]. 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 with *concept lattices* [68], which 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 [69].

- Custom heuristics: Some authors proposed their own heuristic algorithms, instead of using predefined algorithms, to decompose legacy software into SOA.

- General guidelines: they refer 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.

4.1.1. 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 [21]. Seven SIAs use/propose wrapping techniques [9, 16, 18, 21, 28, 30, 33]. For example, Canfora et al. [21] 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. [18] 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.

4.1.2. Genetic Algorithms

We found only three SIAs that rely on Genetic Algorithms to identify services from legacy software systems [15, 47, 37]. For example, Jain et al. [15] 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. [47] 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.

4.1.3. Formal Concept Analysis

SIAs based on formal concept analysis basically rely on ontologies and/or concept lattices [68] to identify services [31, 34, 63]. 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. [31] 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 [70] to build the concept lattices. Finally, they visualized, interpreted and analyzed these concepts to get meaningful, useful, and reusable services. Also, Del et al. [34] 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 [71]. 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.

4.1.4. 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 [14, 15, 27, 32, 37, 41, 42, 43, 44, 46, 48, 49, 51, 52, 53] or functionalities clustering techniques [33, 29]. The main clustering techniques used in the literature are k-means [72, 32] and hierarchical-agglomerative clustering [73, 27].

For example, Zhang et al. [27] 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. [32] 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.

4.1.5. Custom Heuristics

Some SIAs use dedicated heuristics [14, 15, 24, 25, 27, 32, 43, 44] 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. [15] 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.

4.1.6. General Guidelines

We found two works in the literature that propose only general guidelines for service identification [17, 28].

For example, Alahmari et al. [17] 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. [28] 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	[9, 16, 18, 21, 28, 30, 33]	7
GeneticAlgorithm	[15, 47, 37]	3
Formal Concept Analyses	[31, 34, 40, 63]	4
Clustering	[14, 15, 29, 27, 32, 33, 37, 41, 42, 43, 44, 46, 48, 49, 51, 52, 53],	17
Custom heuristics	[14, 15, 16, 19, 20, 22, 24, 25, 45, 27, 32, 35, 38, 39, 63, 42, 43, 44, 45, 46, 50, 49, 51, 52, 53]	25
General Guidelines	[28, 17]	2

Table 2: Targeted techniques of SIAs

4.2. 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.

4.2.1. 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) [74]. Better reusability should provide better return of investment (ROI) and shorter development times [75].
- **Maintainability:** Services should ease the effort to modify their implementation, to identify root causes of failures, to verify changes, etc. [76].
- **Interoperability:** The ability of a service to communicate and be invoked by other systems/services implemented in different programming languages [77].
- **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 [78].

Quality requirement	**SI Method**	**Total**
Reuse	[9, 20, 27, 28, 37, 38]	6
Maintainability	[29]	1
Interoperability	[28]	1
Self-containment	[14, 27]	2
Composability	[14]	1

Table 3: Targeted quality requirements by SIAs

As we can see in Table 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.

4.2.2. 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 [76].
- **Cohesion:** Cohesion is a measure of the strength of the relationships among programming entities (e.g., classes, functions, etc.) implementing a service and the functionality provided by the service [47].
- **Granularity:** An adequate granularity is a primary concern of SIAs. It can be adjusted to the scope of the functionality offered by the service [22].
- **Total number of services:** SIAs must not have too many "small" services or not enough services [44].

Quality Metric	**SI Method**	**Total**
Coupling	[14, 15, 17, 22, 24, 25, 27, 29, 36, 41, 48, 49, 50, 51, 52, 53]	16
Cohesion	[9, 14, 15, 17, 22, 24, 25, 29, 36, 37, 41, 48, 49, 51, 53]	15
Granularity	[9, 20, 22, 24, 25, 27, 29, 36, 38, 41, 48, 49, 52]	13
Number of services	[14, 20, 44, 53]	4

Table 4: Targeted quality metrics by SIAs

Table 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.

4.3. 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 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 [79].
- A hybrid process combines a top-down and a bottom-up process. It uses both requirements and implementation artifacts to identify the candidate services.

As we focus in this SLR on SIAs that follow the bottom-up and hybrid direction, we report in Table 5 the distribution of SIAs over these two directions. Table 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

Direction	SI Method	Total
Bottom-up	[9, 14, 15, 16, 23, 24, 25, 28, 30, 31, 33, 34, 40, 41, 42, 43, 44, 46, 47, 49, 53, 63]	22
Hybrid	[17, 18, 19, 20, 21, 22, 26, 27, 29, 32, 35, 37, 38, 39, 45, 48, 50, 51, 52]	19

Table 5: Identification process directions of service identification methods in the literature

with limited reuse. Moreover, Hybrid SIAs tend to complement and reduce the limitations of bottom-up approaches by also considering requirements.

4.3.1. Analyses Types

SIAs may perform static, dynamic, lexical analyses, or some 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. [14, 16, 18, 19, 23, 27, 28, 29, 30] 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 [39], function calls [29, 51], relationships between database tables [46], etc.) are collected during the program execution [80]. 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 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.

4.4. 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.

Analysis Type	SI Method	Total
Static analysis	[14, 15, 16, 18, 19, 22, 23, 28, 30, 26, 27, 29, 35, 37, 38, 45, 31, 33, 34, 40, 63, 41, 9, 43, 44, 46, 47, 48, 49, 52, 53]	31
Dynamic analysis	[15, 17, 20, 21, 22, 24, 25, 39, 29, 32, 35, 42, 43, 46, 50, 51]	16
Lexical analysis	[19, 26, 27, 31, 34, 40, 41, 43]	8

Table 6: Analyses types of SIAs

- 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 [16, 20].
- Semi-automatic SIAs need human experts to perform some of their tasks. For example, Jain et al. [15] 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 7 shows that there is a lack of automation of SIAs: 88% of the SIAs are semi-automatic or manual.

Analysis Type	SI Method	Total
Automatic	[23, 33, 44, 45, 47]	5
Semi-automatic	[9, 14, 15, 17, 18, 19, 21, 22, 24, 25, 26, 27, 28, 29, 30, 31, 32, 34, 35, 37, 39, 40, 41, 42, 43, 46, 49, 50, 51, 52, 53, 63]	32
Manual	[16, 20, 38, 48]	4

Table 7: Automation of SIAs

5. 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.

5.1. 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 [41, 43, 44, 48, 49, 50, 51, 52, 53]. For example, Escobar et al. [49] 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 focuses on structural coupling and cohesion. The distance between clusters is calculated based on the number of shared entity beans.

Mazlami et al. [43] 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.

5.2. Service Types

We identified only four SIAs that identify specific types of services in existing systems [17, 20, 22, 32] and nine papers proposing service taxonomies [17, 20, 22, 32, 81, 82, 83, 84, 85], that 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 [17, 81, 82], reuse [32, 82, 84], 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. We validated our taxonomy through an industrial survey with practitioners [7] that we detail in Section 7.

1. **Business-process services:** (Also called business service [17, 32, 82, 84]), 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 [82]), 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" and "tax calculation".
3. **Application-task services:** (Also called task, activity or composite service [17, 20, 23, 84]), they provide functionalities specific to one application. They exist to support reuse within one application or to enable business-process services [82]. Examples of Application-task services include quoting request and invoicing that take part in the sales order business process of a typical ERP system.
4. **Entity services:** (Also called information or data services [17, 81, 84]), 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 management services for 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 [84, 32, 22]. 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 [17]. They are sometimes provided by an Enterprise Service Bus (ESB). They are reused in more services than utility services. Examples of Infrastructure services include publish-subscribe, message queues, and ESB.

Most of SIAs identify general services of SOA without specifying different service types, e.g., [14, 29, 86]. Only a few approaches [17, 20, 22, 32] considered the identification of specific types of services in existing systems.

For example, Alahmari et al. [17] 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 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. [32] 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.

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 [17, 22, 32], entity [20, 22] and utility services [17, 20, 22, 32]. 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.

6. RQ4: What is the usability of SIAs?

Figure 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.

6.1. 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.

6.2. 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 8).

6.3. 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.

6.4. 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".

6.5. Usability

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 8, to calculate the usability of the SIA of Rodriguez et al. [16], 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 8 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.

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 [15]	MOGA-WSI	Industry	NA	High	High
Generating a REST Service Layer from a Legacy System [16]	MIGRARIA	Experimental	NA	High	High
A service identification framework for legacy system migration into SOA [17]	Prototype	Experimental	NA	Low	Low
Reusing existing object-oriented code as web services in a SOA [18]	Industry ready	Industry	NA	High	High
Mining candidate web services from legacy code [19]	NA	Experimental	NA	Low	Low
From objects to services: toward a stepwise migration approach for Java applications [20]	NA	Experimental	NA	Low	Low
Migrating interactive legacy systems to web services [21]	NA	Case Study	NA	Medium	Low
MDCSIM: A method and a tool to identify services [22]	MDCSIM	Industry	NA	High	High
Reverse engineering relational databases to identify and specify basic Web services with respect to service oriented computing [23]	CASE	Experimental	NA	Medium	High
Identifying services in procedural programs for migrating legacy system to service oriented architecture [24]	NA	Experimental	NA	Low	Low
A service-oriented analysis and design approach based on data flow diagram [25]	SOAD	Experimental	NA	Low	Medium
Service discovery using a semantic algorithm in a SOA modernization process from legacy web applications [26]	MigraSOA	Experimental	NA	Low	Medium
Incubating services in legacy systems for architectural migration [27]	Prototype	Industry	NA	Low	Medium
Migrating to Web services: A research framework [28]	NA	No Validation	NA	Low	Low
Service Identification and Packaging in Service Oriented Reengineering [29]	Prototype	Case Study	NA	Medium	Medium
A wrapping approach and tool for migrating legacy components to web services [30]	Prototype	Case Study	NA	Low	Low
Extracting reusable object-oriented legacy code segments with combined formal concept analysis and slicing techniques for service integration [31]	Prototype	Experimental	NA	Low	Low
Using dynamic analysis and clustering for implementing services by reusing legacy code [32]	Prototype	Case Study	Meduim	Low	Low
Service Mining from Legacy Database Applications [33]	Prototype	Industry	NA	High	High
An approach for mining services in database oriented applications [34]	Prototype	Industry	High	High	High
Using user interface design to enhance service identification [35]	Prototype	Industry	NA	Medium	High
A method to identify services using master data and artifact-centric modeling approach [36]	NA	Experimental	NA	Low	Low
Multifaceted service identification: Process, requirement and data [37]	Prototype	Experimental	High	Low	Medium
The service modeling process based on use case refactoring [38]	Prototype	Case Study	NA	Low	Low
Extracting reusable services from legacy object-oriented systems [39]	Prototype	Industry	NA	Medium	High
Locating services in legacy software:information retrieval techniques, ontology and FCA based approach [40]	Prototype	Case Study	NA	Low	Low
Microservices Identification Through Interface Analysis [41]	NA	Case Study	NA	Low	Low
Extraction of microservices from monolithic software architectures [43]	Prototype	Industry	NA	High	High
Service Cutter: A Systematic Approach to Service Decomposition [44]	ServiceCutter	Experimental	NA	High	High
Bottom-up and top-down cobol system migration to web services [9]	Industry ready	Industry	NA	High	High
Functionality-Oriented Microservice Extraction Based on Execution Trace Clustering [42]	FOME	Experimental	NA	Low	Medium
An approach to align business and IT perspectives during the SOA services identification [45]	Prototype	Experimental	NA	Low	Low
Discovering Microservices in Enterprise Systems Using a Business Object Containment Heuristic [46]	Prototype	Industry	NA	Medium	High
A heuristic approach to locate candidate web service in legacy software [47]	Prototype	Experimental	NA	Low	Low
Identifying Microservices Using Functional Decomposition [48]	Prototype	Experimental	NA	Low	Low
Towards the understanding and evolution of monolithic applications as microservices [49]	Prototype	Industry	NA	High	High
From Monolithic Systems to Microservices: A Decomposition Framework based on Process Mining [50]	Prototype	Industry	NA	High	High
Function-Splitting Heuristics for Discovery of Microservices in Enterprise Systems [51]	Prototype	Industry	NA	Medium	High
From a Monolith to a Microservices Architecture: An Approach Based on Transactional Contexts [52]	Prototype	Experimental	Medium	Medium	Medium
Re-architecting OO Software into Microservices A Quality-centered approach [53]	Prototype	Experimental	NA	Medium	Medium

Table 8: Usability of SIAs

7. Taxonomy and Validation

Figure 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. Consequently, to validate our taxonomy, we performed a survey with industrial experts.

7.1. Methodology

We conducted a survey with 45 industrial experts to validate our taxonomy and also obtain their informed opinions about legacy-to-SOA migration in general and service identification in particular [7]. We conducted this survey between October 2017 and March 2018 in five main steps:

7.1.1. Preparation of the Survey

We created a Web-based survey[2] using Google Forms. We built our survey on our taxonomy: the individual questions correspond to each composite node of the taxonomy and their possible answers correspond to the leave nodes.

Before releasing the survey, we performed a pilot study with six participants, three from academia and three from industry, and validated the relevance of the questions, their wording, the coverage of their answers, etc. The six participants went through the questions and suggested few minor changes.

The final survey contained six sections: (1) participants' professional and demographic data, (2) type of migrated systems and reasons for migration, (3) general information about SIAs (perception of importance, strategy, inputs, level of automation), (4) technical information about SIAs (techniques, quality metrics), (5) types of sought services, and (6) used tools and best practices.

For example, we asked the participants the following questions: "What information do/did you use to identify services?", "What kind of identification techniques do/did you apply?", "What are the types of the migrated services?", etc. We provide a list of possible answers for each question and ask the participants to mention any other answer if he/she did not select any possibility from the provided list.

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

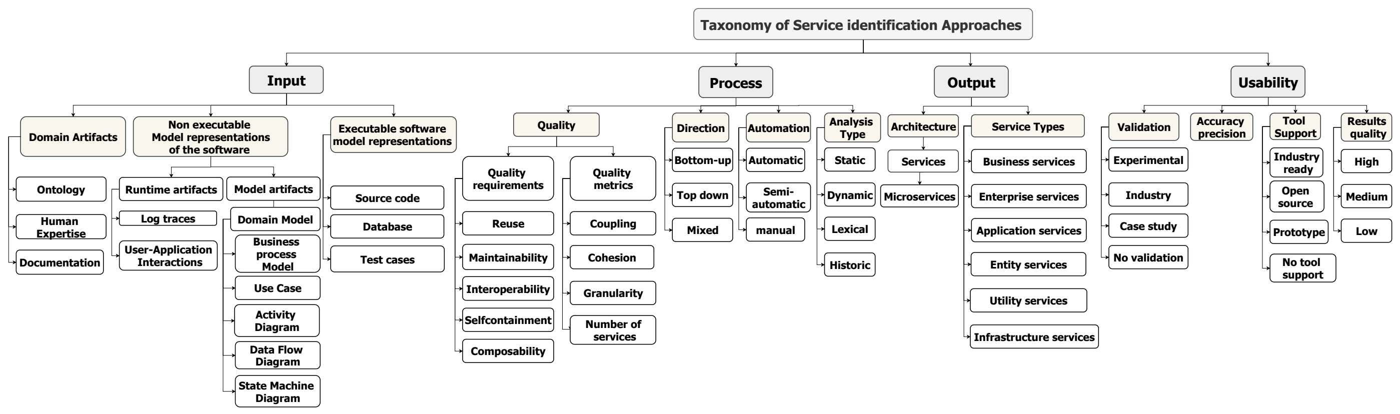

Figure 2: Taxonomy of service identification approaches

7.1.2. Selection of the 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 legacy-to-SOA migration experts, and (3) search queries on LinkedIn profiles: "legacy migration OR legacy modernization OR SOA architect OR SOA migration OR Cloud migration OR service migration OR service mining". We attempted neither to be exhaustive in our search for participants nor to cover different strata of developers working on SOA migration. As such, our sample is a random sample of convenience.

Once we identified potential participants, we sent them invitations via e-mail, LinkedIn, Facebook, and Twitter. We chose *not* to solicit more than three participants from any given company (1) to have a diversity of companies and (2) to avoid overburdening a single company.

7.1.3. Administering the Survey

We invited 289 participants and recommended about 15 minutes to complete the survey. We asked potential participants to forward our invitations to colleagues with experience in SOA migration in their social/professional networks. The survey was completed by 47 participants, two of whom did not participate in SOA migration and whose answers we removed, for *45 complete answers*.

7.1.4. Validation of the Survey

We assessed the reliability of the answers to the survey by searching for spurious/facetious answers, contradictory answers, etc. We also performed follow-up interviews with 24 of the 45 participants who agreed to such interviews by providing their e-mail addresses.

We interviewed 8 of these 24 participants. We used a two-pass method [87] to analyze our transcripts of the individual interviews[3]. We first performed a *thematic coding* to identify broad issues related to legacy-to-SOA migration in general and SIAs in particular. We then performed an *axial coding* to identify relationships among the identified issues. We identified major issues related to our taxonomy using *meta-codes*, which we then used to code manually the data of all the interviewees [7].

7.1.5. Validation of the Taxonomy

We could then measure the precision and accuracy of our taxonomy (input, process, and output of SIAs) as:

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

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

where TP (*True Positive*) corresponds to a leaf/answer identified by both a participant and our taxonomy, FP (*False Positive*) corresponds to a leaf/answer identified in our taxonomy but not mentioned by any participant, and FN (*False Negative*) corresponds to an answer from at least one participant but not identified in our taxonomy.

We do not assess the usability of the SIAs for two reasons. First, we did not want to overburden the participants with the assessment of a very subjective metric and because we believe that such a measure deserves a complete, independent study (in future work).

7.2. Participants

We reached a total of 45 participants 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% were migration specialists, project managers, and CEOs.

The participants worked in different industries: 64% were in technology and telecommunication, 20% in banking and insurance, 12.8% in health, and 3.2% in education.

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

In terms of experience, 78% had more than 10 years of experience, also reflected in their age distributions: 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 55.

7.3. Validation Results

Feature	Precision	Recall
Inputs	93%	100%
Techniques	100%	82%
Quality metrics	100%	73%
Direction	100%	100%
Automation	100%	100%
Analysis type	100%	100%
Service types	100%	100%

Table 9: Validation results of the service-identification taxonomy

Table 9 shows the validation results of our taxonomy with precision and recall: our taxonomy is conformed to the experts' experiences with a precision between 93% and 100% and a recall between 73% and 100%.

In particular, for inputs, precision is 93% and recall 100%. None of the participants mentioned the use of test scenarios to identify services. However, we kept this input in our taxonomy as it is used by two SIAs. In terms of the techniques, recall is 82%, which is acceptable, because some participants mentioned the use of in-house tools or manual identification. For quality metrics, recall is 73%, which is acceptable, because some participants mentioned the use of other economic quality metrics not considered in any of the SIAs: identification cost, adaptation effort, and time to market.

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.

8.1. 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 [18, 20, 29, 32]. Only 10 SIAs rely on a single input type [14, 16, 23, 30, 33, 43, 47, 49, 52, 53], either source code again or databases.

8.2. 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 [88]. 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 re-factoring 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.

8.3. 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 [7]. None of them mentioned the identification of new/other types of services.

8.4. 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 [7]. 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 a 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.

9. Related Work

Several systematic literature reviews and surveys on SIAs have been proposed in the literature. In the period from 2009 to 2019, ten surveys [89, 90, 85, 91, 92, 93, 88, 94, 95, 96] on service identification were identified. Although these surveys had different goals, neither of them fully addressed all our research questions. Table 10 contains a summary and comparison between the most relevant surveys focusing on service identification in the literature.

For example, Boerner et al. [89], 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. [98] 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. [97] 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. [92] 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. [95, 96] proposed two different surveys about service identification. In the first one they studied the evaluation frameworks for 24 state-of-the-art SIAs. In the second survey they only identified the challenges of 14 service identification approaches and their limitations. Both studies do not fully address our research questions as we do in our SLR.

Finally Fritcsch et al. [94] provided a classification of refactoring approaches of monolithic applications to microservices. They studied 10 microservices identification approaches and provided a guide for decomposition approaches using 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.

10. Conclusion and Future Work

We presented in this paper 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. We validated the correctness and the coverage of our taxonomy with industrial experts in legacy-to-SOA migration through surveys and one-on-one interviews. The validation results showed that our taxonomy is conformed to the industrial experts' experiences with 99% of precision and 94% of recall.

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. The results 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, we believe that more work should be done to automate the SIAs and consider enterprise-scaled systems to validate the proposed approaches.

As future work, we will generalize our survey and study top-down service identification approaches. We will study in detail SIAs that use some architecture-centric methods such as Architecture Tradeoff Analysis Method (ATAM) [99], Attribute-Driven Design (ADD) [100], and Cost Benefit Analysis Method (CBAM) [101]. These methods could assist a service identification approach by providing and evaluating architectural descriptions of the system to migrate. Also, we will study empirically the gap of the state of the practices of SIAs between academia and industry. We want to identify issues that the research community can address to ease knowledge transfer between academia and industry in the context of legacy-to-SOA migration. Finally, we believe that the identification of services according to their types is a challenging but interesting prob-

SIA	Goal	Year of publication	Covered years	Included papers	RQ1	RQ2	RQ3	RQ4
Boerner *et al.*[89]	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.*[90]	Classification of service identification techniques	2009	1984*-2008	15	PA	A	PA	NA
Gu and Lago[85]	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.*[97]	Identify frequent used activities done in several SI research works	2011	2004-2011	41	PA	A	PA	NA
Vale *et al.*[92]	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.*[93]	Suitable inputs identification for SI methods in small and medium enterprise	2012	2002-2010	48	PA	PA	PA	NA
Huergo *et al.*[88]	Classification of SI methods	2014	2002-2013	105	PA	A	PA	NA
Bani *et al.*[96]	Exploring existing evaluation frameworks for state-of-the-art SIAs	2018	2007-2016	23	PA	PA	NA	PA
Bani *et al.*[95]	Identifying service identification challenges in service oriented architecture	2018	2005-2016	14	PA	NA	NA	NA
Fritzsch *et al.*[94]	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	2019	2004-2019	41	A	A	A	A

Table 10: Systematic literature reviews of Service Identification in the literature (***A** for Addressed, **PA** for Partially Addressed, **NA** for Not Addressed*

lem. As future work, we aim to propose a type-centric service identification approach that promote better reuse at the application, enterprise, and business levels.

References

[1] Grace Lewis, Ed Morris, Liam O'Brien, Dennis Smith, and Lutz Wrage. Smart: The service-oriented migration and reuse technique. Technical report, DTIC Document, 2005.

[2] Thomas Erl. *SOA design patterns.* Pearson Education, 2008.

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

[4] Ravi Khadka, Prajan Shrestha, Bart Klein, Amir Saeidi, Jurriaan Hage, Slinger Jansen, Edwin van Dis, and Magiel Bruntink. 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)*, pages 477–486. IEEE, 2015.

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

[6] Ravi Khadka, Amir Saeidi, Slinger Jansen, and Jurriaan Hage. 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*, pages 2–11. IEEE, 2013.

[7] Manel Abdellatif, Geoffrey Hecht, Hafedh Mili, Ghizlane Elboussaidi, Naouel Moha, Anas Shatnawi, Jean Privat, and Yann-Gaël Guéhéneuc. State of the practice in service identification for soa migration in industry. In *International Conference on Service-Oriented Computing*, pages 634–650. Springer, 2018.

[8] Andrei Furda, Colin Fidge, Olaf Zimmermann, Wayne Kelly, and Alistair Barros. Migrating enterprise legacy source code to microservices: on multitenancy, statefulness, and data consistency. *IEEE Software*, 35(3):63–72, 2017.

[9] Juan Manuel Rodriguez, Marco Crasso, Cristian Mateos, Alejandro Zunino, and Marcelo Campo. Bottom-up and top-down cobol system migration to web services. *IEEE Internet Computing*, 17(2):44–51, 2013.

[10] Barbara Kitchenham. Procedures for performing systematic reviews. *Keele, UK, Keele University*, 33(2004):1–26, 2004.

[11] Claes Wohlin. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, page 38. ACM, 2014.

[12] Katia Romero Felizardo, Emilia Mendes, Marcos Kalinowski, Érica Ferreira Souza, and Nandamudi L Vijaykumar. 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*, page 53. ACM, 2016.

[13] Ali Taei Zadeh, Muriati Mukhtar, Shahnorbanun Sahran, and MR Khabbazi. A systematic input selection for service identification in smes. *Journal of Applied Sciences*, 12(12):1232, 2012.

[14] Seza Adjoyan, Abdelhak-Djamel Seriai, and Anas 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.*, pages 1–6, 2014.

[15] Hemant Jain, Huimin Zhao, and Nageswara R Chinta. A spanning tree based approach to identifying web services. *International Journal of Web Services Research*, 1(1):1, 2004.

[16] Roberto Rodríguez-Echeverría, Fernando Maclas, Vlctor M Pavón, José M Conejero, and Fernando Sánchez-Figueroa. Generating a rest service layer from a legacy system. In *Information System Development*, pages 433–444. Springer, 2014.

[17] Saad Alahmari, Ed Zaluska, and David De Roure. A service identification framework for legacy system migration into soa. In *Services Computing (SCC), 2010 IEEE International Conference on*, pages 614–617. IEEE, 2010.

[18] Harry M Sneed, Chris Verhoef, and Stephan 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*, pages 31–39. IEEE, 2013.

[19] Lerina Aversano, Luigi Cerulo, and Ciro Palumbo. Mining candidate web services from legacy code. In *10th International Symposium on Web Site Evolution*, pages 37–40. IEEE, 2008.

[20] Alessandro Marchetto and Filippo Ricca. From objects to services: toward a stepwise migration approach for java applications. *International journal on software tools for technology transfer*, 11(6):427, 2009.

[21] Gerardo Canfora, Anna Rita Fasolino, Gianni Frattolillo, and Porfirio Tramontana. Migrating interactive legacy systems to web services. In *Conference on Software Maintenance and Reengineering (CSMR'06)*, pages 10–pp. IEEE, 2006.

[22] Rosane S Huergo, Paulo F Pires, and Flávia C Delicato. Mdc-

sim: A method and a tool to identify services. *IT Convergence Practice*, 2(4):1–27, 2014.

[23] Youcef Baghdadi. Reverse engineering relational databases to identify and specify basic web services with respect to service oriented computing. *Information systems frontiers*, 8(5):395–410, 2006.

[24] Masahide Nakamur, Hiroshi Igaki, Takahiro Kimura, and Kenichi Matsumoto. Identifying services in procedural programs for migrating legacy system to service oriented architecture. *Implementation and Integration of Information Systems in the Service Sector*, page 237, 2012.

[25] Yun Zhao, Huayou Si, Yulin Ni, and Hengnian Qi. A service-oriented analysis and design approach based on data flow diagram. In *International Conference on Computational Intelligence and Software Engineering CiSE 2009.*, pages 1–5. IEEE, 2009.

[26] Encarna Sosa-Sánchez, Pedro J Clemente, Miguel Sánchez-Cabrera, José M Conejero, Roberto Rodríguez-Echeverría, and Fernando Sánchez-Figueroa. Service discovery using a semantic algorithm in a soa modernization process from legacy web applications. In *Services (SERVICES), 2014 IEEE World Congress on*, pages 470–477. IEEE, 2014.

[27] Zhuopeng Zhang and Hongji Yang. Incubating services in legacy systems for architectural migration. In *11th Asia-Pacific Software Engineering Conference, 2004.*, pages 196–203. IEEE, 2004.

[28] Harry Sneed. Migrating to web services: A research framework. In *Proceedings of the International*, 2007.

[29] Zhuopeng Zhang, Ruimin Liu, and Hongji Yang. Service identification and packaging in service oriented reengineering. In *SEKE*, volume 5, pages 620–625, 2005.

[30] Guo Chenghao, Wang Min, and Zhou Xiaoming. A wrapping approach and tool for migrating legacy components to web services. In *First International Conference on Networking and Distributed Computing (ICNDC),2010*, pages 94–98. IEEE, 2010.

[31] Zhuopeng Zhang, Hongji Yang, and William 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)*, pages 385–392. IEEE, 2006.

[32] Andreas Fuhr, Tassilo Horn, and Volker Riediger. Using dynamic analysis and clustering for implementing services by reusing legacy code. In *Reverse Engineering (WCRE), 2011 18th Working Conference on*, pages 275–279. IEEE, 2011.

[33] Diptikalyan Saha. Service mining from legacy database applications. In *Web Services (ICWS), 2015 IEEE International Conference on*, pages 448–455. IEEE, 2015.

[34] Concettina Del Grosso, Massimiliano Di Penta, and Ignacio Garcia-Rodriguez de Guzman. An approach for mining services in database oriented applications. In *11th European Conference on Software Maintenance and Reengineering, 2007. CSMR'07.*, pages 287–296. IEEE, 2007.

[35] Senthil Mani, Vibha S Sinha, Noi Sukaviriya, and Thejaswini Ramachandra. Using user interface design to enhance service identification. In *Web Services, 2008. ICWS'08. IEEE International Conference on*, pages 78–87. IEEE, 2008.

[36] Rosane S Huergo, Paulo F Pires, and Flavia 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*, pages 1225–1230. ACM, 2014.

[37] Mohammad Javad Amiri, Saeed Parsa, and Amir Mohammadzade Lajevardi. Multifaceted service identification: Process, requirement and data. *Computer Science and Information Systems*, 13(2):335–358, 2016.

[38] Yukyong Kim and Kyung-Goo Doh. The service modeling process based on use case refactoring. In *International Conference on Business Information Systems*, pages 108–120. Springer, 2007.

[39] Liang Bao, Chao Yin, Weigang He, Jun Ge, and Ping Chen. Extracting reusable services from legacy object-oriented systems. In *software maintenance (ICSM), 2010 IEEE International Conference on*, pages 1–5. IEEE, 2010.

[40] M.A.M. Djeloul. Locating services in legacy software:information retrieval techniques, ontology and fca based approach. *WSEAS Transactions on Computers*, 11(1):19 – 26, 2012/01/. legacy software;information retrieval techniques;FCA based approach;Web services technology;WORDNET ontology;formal concepts analysis;source code;.

[41] Luciano Baresi, Martin Garriga, and Alan De Renzis. Microservices identification through interface analysis. In *European Conference on Service-Oriented and Cloud Computing*, pages 19–33. Springer, 2017.

[42] Wuxia Jin, Ting Liu, Qinghua Zheng, Di Cui, and Yuanfang Cai. Functionality-oriented microservice extraction based on execution trace clustering. In *2018 IEEE International Conference on Web Services (ICWS)*, pages 211–218. IEEE, 2018.

[43] Genc Mazlami, Jürgen Cito, and Philipp Leitner. Extraction of microservices from monolithic software architectures. In *2017 IEEE International Conference on Web Services (ICWS)*, pages 524–531. IEEE, 2017.

[44] Michael Gysel, Lukas Kölbener, Wolfgang Giersche, and Olaf Zimmermann. Service cutter: A systematic approach to service decomposition. In *European Conference on Service-Oriented and Cloud Computing*, pages 185–200. Springer, 2016.

[45] Eric Souza, Ana Moreira, and Cristiano 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)*, pages 1–7. IEEE, 2017.

[46] Adambarage Anuruddha Chathuranga De Alwis, Alistair Barros, Colin Fidge, and Artem Polyvyanyy. Discovering microservices in enterprise systems using a business object containment heuristic. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 60–79. Springer, 2018.

[47] Mostefai Abdelkader, Mimoun Malki, and Sidi Mohamed Benslimane. A heuristic approach to locate candidate web service in legacy software. *International Journal of Computer Applications in Technology*, 47(2-3):152–161, 2013.

[48] Shmuel Tyszberowicz, Robert Heinrich, Bo Liu, and Zhiming Liu. Identifying microservices using functional decomposition. In *International Symposium on Dependable Software Engineering: Theories, Tools, and Applications*, pages 50–65. Springer, 2018.

[49] Daniel Escobar, Diana Cárdenas, Rolando Amarillo, Eddie Castro, Kelly Garcés, Carlos Parra, and Rubby Casallas. Towards the understanding and evolution of monolithic applications as microservices. In *2016 XLII Latin American Computing Conference (CLEI)*, pages 1–11. IEEE, 2016.

[50] Davide Taibi and Kari 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.

[51] Adambarage Anuruddha Chathuranga De Alwis, Alistair Barros, Artem Polyvyanyy, and Colin Fidge. Function-splitting heuristics for discovery of microservices in enterprise systems. In *International Conference on Service-Oriented Computing*, pages 37–53. Springer, 2018.

[52] Luís Nunes, Nuno Santos, and António Rito Silva. From a monolith to a microservices architecture: An approach based on transactional contexts. In *European Conference on Software Architecture*, pages 37–52. Springer, 2019.

[53] Anfel Selmadji, Abdelhak-Djamel Seriai, Hinde Lilia Bouziane, Christophe Dony, and Rahina Oumarou Mahamane. Re-architecting oo software into microservices. In *European Conference on Service-Oriented and Cloud Computing*, pages 65–73. Springer, 2018.

[54] Deborah Hix and H Rex Hartson. *Developing user interfaces: ensuring usability through product & process.* John Wiley &

Sons, Inc., 1993.

[55] Mathias Weske. Business process management architectures. In *Business Process Management*, pages 333–371. Springer, 2012.

[56] Anisha Vemulapalli and Nary 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"*, pages 337–349. Springer, 2009.

[57] Joseph Schmuller. *Sams teach yourself UML in 24 hours.* Sams publishing, 2004.

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

[59] Manuj Aggarwal and Sangeeta Sabharwal. Test case generation from uml state machine diagram: A survey. In *Computer and Communication Technology (ICCCT), 2012 Third International Conference on*, pages 133–140. IEEE, 2012.

[60] Timothy C Lethbridge, Janice Singer, and Andrew Forward. How software engineers use documentation: The state of the practice. *IEEE Software*, 20(6):35–39, 2003.

[61] Encarna Sosa, Pedro J Clemente, José M Conejero, and Roberto Rodríguez-Echeverría. 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)*, pages 61–70. IEEE, 2013.

[62] Sean Bechhofer. Owl: Web ontology language. In *Encyclopedia of Database Systems*, pages 2008–2009. Springer, 2009.

[63] Feng Chen, Zhuopeng Zhang, Jianzhi Li, Jian Kang, and Hongji Yang. Service identification via ontology mapping. In *2009 33rd Annual IEEE International Computer Software and Applications Conference*, volume 1, pages 486–491. IEEE, 2009.

[64] Shuxin Zhao, Elizabeth Chang, and Tharam 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*, pages 153–159. IEEE, 2008.

[65] Harry 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*, pages 11–pp. IEEE, 2006.

[66] Marko Balabanović and Yoav Shoham. Fab: content-based, collaborative recommendation. *Communications of the ACM*, 40(3):66–72, 1997.

[67] Garrett Birkhoff. *Lattice theory*, volume 25. American Mathematical Soc., 1940.

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

[69] Rui Xu and D. Wunsch. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, May 2005.

[70] Bernhard Ganter. Two basic algorithms in concept analysis. *Formal Concept Analysis*, pages 312–340, 2010.

[71] Rudolf Wille. Restructuring lattice theory: an approach based on hierarchies of concepts. In *Ordered sets*, pages 445–470. Springer, 1982.

[72] Anil K Jain. Data clustering: 50 years beyond k-means. *Pattern recognition letters*, 31(8):651–666, 2010.

[73] Fionn Murtagh and Pierre Legendre. Ward's hierarchical agglomerative clustering method: which algorithms implement ward's criterion? *Journal of classification*, 31(3):274–295, 2014.

[74] George Feuerlicht et al. Understanding service reusability. In *International Conference Systems Integration*. Department of Information Technologies and Czech Society for Systems Integration, 2007.

[75] Ville Alkkiomäki and Kari Smolander. Anatomy of one service-oriented architecture implementation and reasons behind low service reuse. *Service Oriented Computing and Applications*, 10(2):207–220, 2016.

[76] Mikhail Perepletchikov, Caspar Ryan, Keith Frampton, and Zahir Tari. Coupling metrics for predicting maintainability in service-oriented designs. In *2007 Australian Software Engineering Conference (ASWEC'07)*, pages 329–340. IEEE, 2007.

[77] Thomas Erl. *Service-oriented architecture*, volume 8. Pearson Education Incorporated, 2005.

[78] Renuka Sindhgatta, Bikram Sengupta, and Karthikeyan Ponnalagu. Measuring the quality of service oriented design. In *Service-Oriented Computing*, pages 485–499. Springer, 2009.

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

[80] Anas Shatnawi, Hudhaifa Shatnawi, Mohamed Aymen Saied, Zakarea Al Shara, Houari Sahraoui, and Abdelhak Seriai. Identifying software components from object-oriented apis based on dynamic analysis. In *Proceedings of the 26th Conference on Program Comprehension*, pages 189–199. ACM, 2018.

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

[82] Shy Cohen. Ontology and taxonomy of services in a service-oriented architecture. *The Architecture Journal*, 11(11):30–35, 2007.

[83] Dirk Krafzig, Karl Banke, and Dirk Slama. *Enterprise SOA: service-oriented architecture best practices.* Prentice Hall Professional, 2005.

[84] Bashar Al Ani and Youcef Baghdadi. A taxonomy-centred process for service engineering. *International Journal of Computer Applications in Technology*, 52(1):1–17, 2015.

[85] Qing Gu and Patricia Lago. Service identification methods: a systematic literature review. In *Towards a Service-Based Internet*, pages 37–50. Springer, 2010.

[86] Masahide Nakamura, Hiroshi Igaki, Takahiro Kimura, and Ken-ichi Matsumoto. Extracting service candidates from procedural programs based on process dependency analysis. In *Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific*, pages 484–491. IEEE, 2009.

[87] Kathy Charmaz and Liska Belgrave. Qualitative interviewing and grounded theory analysis. *The SAGE handbook of interview research*, pages 347–365, 2012.

[88] Rosane S Huergo, Paulo F Pires, Flavia C Delicato, Bruno Costa, Everton Cavalcante, and Thais Batista. A systematic survey of service identification methods. *Service Oriented Computing and Applications*, 8(3):199–219, 2014.

[89] René Boerner and Matthias 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*, pages 588–593. IEEE, 2009.

[90] Dominik Birkmeier, Sebastian Klöckner, and Sven Overhage. A survey of service identification approaches-classification framework, state of the art, and comparison. *Enterprise Modelling and Information Systems Architectures*, 4(2):20–36, 2015.

[91] Xia Cai, Michael R Lyu, Kam-Fai Wong, and Roy Ko. Component-based software engineering: technologies, development frameworks, and quality assurance schemes. In *Software Engineering Conference, 2000. APSEC 2000. Proceedings. Seventh Asia-Pacific*, pages 372–379. IEEE, 2000.

[92] Tassio Vale, Gustavo Bittencourt Figueiredo, Eduardo Santana de Almeida, and Silvio Romero de Lemos Meira. A study on service identification methods for software product lines. In *Proceedings of the 16th International Software Product Line Conference-Volume 2*, pages 156–163. ACM, 2012.

[93] Ali Taei Zadeh, Muriati Mukhtar, Shahnorbanun Sahran, and Mahmood Reza Khabbazi. A systematic input selection for service identification in smes. *Journal of Applied Sciences*, 12(12):1232–1244, 2012.

[94] Jonas Fritzsch, Justus Bogner, Alfred Zimmermann, and Stefan Wagner. 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*, pages 128–141. Springer, 2018.

[95] Basel Bani-Ismail and Youcef Baghdadi. A literature review

on service identification challenges in service oriented architecture. In *International Conference on Knowledge Management in Organizations*, pages 203–214. Springer, 2018.

[96] Basel Bani-Ismail and Youcef Baghdadi. A survey of existing evaluation frameworks for service identification methods: towards a comprehensive evaluation framework. In *International Conference on Knowledge Management in Organizations*, pages 191–202. Springer, 2018.

[97] Simin Cai, Yan Liu, and Xiaoping Wang. A survey of service identification strategies. In *Services Computing Conference (APSCC), 2011 IEEE Asia-Pacific*, pages 464–470. IEEE, 2011.

[98] Dominik Birkmeier and Sven Overhage. On component identification approaches–classification, state of the art, and comparison. In *Component-Based Software Engineering*, pages 1–18. Springer, 2009.

[99] Rick Kazman, Mark Klein, Mario Barbacci, Tom Longstaff, Howard Lipson, and Jeromy Carriere. The architecture tradeoff analysis method. In *Proceedings. Fourth IEEE International Conference on Engineering of Complex Computer Systems (Cat. No. 98EX193)*, pages 68–78. IEEE, 1998.

[100] RL Nord et al. Integrating the quality attribute workshop (qaw) and the attribute-driven design (add) method. inf. téc. Technical report, CMU/SEI-2004-TN-017, Software Engineering Institute–Carnegie Mellon . . . , 2004.

[101] Robert L Nord, Mario R Barbacci, Paul Clements, Rick Kazman, and Mark Klein. Integrating the architecture tradeoff analysis method (atam) with the cost benefit analysis method (cbam). Technical report, Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, 2003.