Chapter 3

Legacy to SOA Evolution: A Systematic Literature Review

Ravi Khadka
Utrecht University, The Netherlands

Amir Saeidi
Utrecht University, The Netherlands

Andrei Idu
Utrecht University, The Netherlands

Jurriaan Hage
Utrecht University, The Netherlands

Slinger Jansen
Utrecht University, The Netherlands

ABSTRACT

Enterprises depend on business-critical systems that have been developed over the last three decades or more, also known as legacy systems. They have several well-known disadvantages (e.g., inflexible, domain unspecific, and hard to maintain), and this is recognized by both vendors and customers of these software systems. Both vendors and customers of these systems are well aware that better and more flexible customer specific solutions can be created following the service-oriented paradigm. Hence, momentum is growing within enterprises to evolve legacy systems towards Service-Oriented Architecture (SOA). The evolution to SOA is favored because of various advantages including well established sets of open standards, platform and language independent interfaces, clear separation of service interface and implementation, and loose-coupling among services.

In the last decade, there have been significant developments in legacy to SOA evolution, and that has resulted in a large research body of which there exists no comprehensive overview. This chapter provides a historic overview, focusing on the methods and techniques used in legacy to SOA evolution. The authors conducted a systematic literature review to collect legacy to SOA evolution approaches reported from 2000 to August 2011. To this end, 121 primary studies were found and evaluated using an evaluation framework, which was developed from three evolution and modernization methods widely used in the software re-engineering domain. The evaluation constitutes the inventory of current research approaches and methods and techniques used in legacy to SOA evolution. The result of the SLR also identifies current research issues in legacy to SOA evolution and provides future research directions to address those research issues.

DOI: 10.4018/978-1-4666-2488-7.ch003
INTRODUCTION

Recently, many enterprises have focused on increasing their business flexibility and achieving cross-enterprise collaboration to remain competitive in the market, and to meet their business objectives. Enterprises are especially challenged by constant changes in the business environment and changes in the supporting Information Technology (IT) infrastructures that hinder the overall success of enterprises (van Sinderen, 2008). Furthermore, most enterprises still rely on so-called legacy system—software developed over the previous decades using 3GL programming languages like COBOL, RPG, PL/I, C, C++. Despite the well-known disadvantages, such as being inflexible and hard to maintain, legacy systems are still vitally important to the enterprises as they support complex core business processes; they cannot simply be removed as they implement and store critical business logic. Unsurprisingly, the knowledge contained in these systems is of high value to an enterprise. On the other hand, proper documentation, skilled manpower, and resources to evolve these legacy systems are scarce. Therefore, momentum is growing to evolve and reuse those legacy systems within new technological environments—Service-Oriented Architecture (SOA) being the most promising one (Bisbal, Lawless, Wu, & Grimson, 1999; Lewis, Morris, O’Brien, Smith, & Wrage, 2005).

SOA has emerged as an architectural style that enables the reuse of existing legacy assets within a new paradigm that facilitates loose coupling, abstraction of underlying logic, flexibility, reusability and discoverability (Papazoglou, 2008). The evolution from legacy to SOA can be beneficial from both economical and technical perspectives. From an economical perspective, legacy to SOA evolution fosters change management including intra-organizational changes, and changes in enterprises (Khadka, Sapkota, Pires, Sinderen, & Jansen, 2011; Papazoglou, Traverso, Dustdar, & Leymann, 2007). From a technical perspective, seamless enterprise collaboration through service composition (Khadka & Sapkota, 2010) and reduction in maintenance cost are claimed as long-term benefits (Papazoglou et al., 2007; Schelp & Aier, 2009). Motivated by these benefits, there has been significant research in legacy to SOA evolution. However, there is no systematic overview of legacy to SOA evolution, particularly focusing on the techniques, methods and approaches used to evolve legacy systems to a SOA environment. In the systematic literature review conducted by Razavian (Razavian & Lago, 2010), an overview of SOA migration families is reported. It focuses on classifying the SOA migration approaches into eight distinct families. The classification is inspired by the reengineering horseshoe method (Bergey, Smith, Weiderman, & Woods, 1999) rather than giving a historical overview of SOA migration methods. Also, a brief overview of legacy to SOA evolution is reported by Almonaies (Almonaies, Cordy, & Dean, 2010) that divides the legacy to SOA evolution approaches into four categories: replacement, redevelopment, wrapping and migration. The legacy to SOA evolution approaches reported in this research were not based on any systematic literature review process, so a complete, historical overview of the legacy to SOA evolution approaches is still lacking.

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

1. A historical overview of legacy to SOA evolution.
2. A legacy to SOA evolution process framework.
3. An inventory of methods and techniques used in various phases of legacy to SOA evolution.
4. A series of research issues and recommendations for future research directions.

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

The chapter is structured as follows: Section 2 provides the details of our research method; Section 3 presents the evaluation framework; Section 4 discusses the overview of the primary studies; Section 5 elaborates the findings of our SLR; Section 6 discusses the findings and best practices in legacy to SOA evolution; Section 7 describes the threats to validity; and in Section 8, we present the conclusions of our research and possible future research directions.

**RESEARCH METHOD**

We have adopted the procedures of conducting a systematic review process based on the guidelines proposed by Kitchenham (2004). A systematic review consists of a review protocol that details the rationale of the survey, research objectives, search strategy, selection criteria, data extraction, synthesis, and analysis of the extracted data and interpretation of the findings. Such a review process is typically appropriate in our research since it summarizes the existing contributions, identifies the gaps in the current research and avenues for further research, and provides a background to position new research activities in a research framework.

### Review Protocol

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

### Research Question

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

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

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

Search Strategy

We have constructed a search string using SOA, legacy, and migration as main keywords, and have included synonyms and related terms. The search string is then constructed using Boolean “AND” to connect the three keywords and Boolean “OR” to allow synonyms and word class variants of each keyword. The resulting search string is depicted in Listing 1.

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

Study Selection

It is likely that some of the results (study data) of a search might contain the keywords but are irrelevant to our research. For instance, a study data with the title “An evaluation of legacy systems and grid service systems of health-care domain: An initial step towards transformation to cloud-based system” is included in the result of the initial selection. In order to exclude such irrelevant studies, study selection is performed such that the study data is assessed to determine the actual relevance. A set of inclusion and exclusion criteria based on the scope of research and the quality of the studies were determined by us. The inclusion and exclusion criteria are given in Table 1.

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

Data Extraction

We extracted the study selection in a spreadsheet including the following details: title, authors, publication year, publication form (journal/conference/workshop/book chapter), name, and abstract. We conducted the first selection round based on the “title and abstract” of the study. The study was categorized as follows:

1. Relevant (study inside the scope of the research),
2. Irrelevant (study outside the scope of the research), and

Listing 1. Search string

(SOA OR “Service-Oriented” OR “Service-Based” OR “Service-Centric” OR “Service-Engineering” OR “SOSE” OR “Web service” OR “service-oriented computing”) AND (Monolith OR “legacy code” OR “Legacy system” OR “existing system” OR “legacy component” OR “legacy software” OR “monolithic system” OR “existing software” OR “pre-existing software” OR “legacy information system” OR “legacy program” OR “pre-existing assets”) AND (migration OR evolution OR modernisation OR reengineering OR re-engineering OR reuse OR “service identification” OR “candidate service identification” OR “service extraction” OR bridging OR reconstruction OR modernization OR decomposing OR “incubating services” OR integrating OR redesigning OR “service mining” OR migrating OR transformation)
Table 1. Inclusion and exclusion criteria for study selection

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

3. Moderate (unable to decide the relevancy of the paper).

For each irrelevant and moderate study, explicit reasons were provided in the spreadsheet. The moderate category was decided by repeating the review by a reviewer other than the initial reviewer and by discussing the paper with the team. The final outcome is the collection of relevant studies, which we refer to as the primary studies.

Data Synthesis

The primary studies were evaluated against the evaluation framework presented in Section 3 and various findings are reported in Section 4 and Section 5.

We conducted a review process adhering to the review protocol. Initially, we had 8493 hits when we ran the search query over the electronic libraries/indexing sources. Those 8493 articles were analyzed by five researchers to determine the relevancy based on title and abstract, which left 269 articles. These articles were then evaluated based on inclusion and exclusion criteria, which resulted in 121 primary studies. The details of the review process can be found in Idu, Khadka, Saeidi, Jansen, and Hage (2012). Figure 1 depicts the review process.

EVALUATION FRAMEWORK FOR LEGACY TO SOA EVOLUTION

To develop an evaluation framework for legacy to SOA evolution, we needed to identify the phases that are typically related to evolution/modernization of legacy systems. Based on a high number of citations (popularity), availability of documentation, and completeness of the legacy evolution/modernization process, the following methods from software re-engineering domain were used to identify the phases for our evaluation framework: the butterfly method (Wu, et al., 1997), the Renaissance method (Warren & Ransom, 2002), and the Architecture-Driven Modernization (ADM)
Legacy to SOA Evolution

Figure 1. The review process with number of studies

(Khusidman & Ulrich, 2007). The main reason for using these evolution/modernization methods is that the software re-engineering domain has been extensively researched and widely practiced in industries, as compared to SOA evolution methods. In particular, we want to reuse the concepts from those methods in the development of a new method for legacy to SOA evolution. Method engineering (Brinkkemper, 1996) allows us to reuse existing concepts from existing methods to construct new methods. Hence, we use method engineering and reuse the concepts from the three above-mentioned legacy evolution/modernization methods. We argue that reusing the methods and practices from existing standards/methods saves time and reduces the adoption problem (i.e., it is easier to adapt to the existing methods/practices than learning new methods). Due to limitations of space, we do not provide the details of the construction of the evaluation framework in this chapter. The details are reported in Idu et al. (2012). One can argue that there are sufficient relevant legacy to SOA evolution methods that could have been used to develop the evaluation framework. Most of the legacy to SOA evolution methods reported in literature, either focus on developing supporting technology (i.e., implementation techniques to expose legacy systems in SOA) or planning the legacy to SOA evolution (i.e., determining the feasibility of evolution) (Razavian & Lago, 2010). However, a legacy to SOA evolution requires the consolidation of both, developing supporting technology and planning the legacy to SOA evolution (De Lucia, Francesc, Scanniello, & Tortora, 2008; Khadka, Reijnders, Saeidi, Jansen, & Hage, 2011). In our approach, we aim at developing such a framework that combines both aspects (i.e., planning legacy to SOA evolution and implementation). Furthermore, we aim at assessing those existing legacy to SOA evolution methods using our developed evaluation method rather than using them to develop a new method.

From the three methods, we have identified phases that are common to all of them. For instance, legacy system understanding, target system understanding, evolution feasibility determination, and implementation of evolution are common phases in the above-mentioned methods. To make our evaluation framework more relevant to the SOA domain and to reflect the intent of legacy to SOA evolution, we further analyzed and identified some phases from the following service-oriented development methods: Service-Oriented Design and Development Methodology (SODDM) (Papazoglou & Van Den Heuvel, 2006), Web Service Implementation Methodology (WSIM) (Lee, Chan, & Lee, 2006), and Service-Oriented Modeling and Architecture (SOMA) (Arsanjani, et al., 2004). The details of the identification of the phases are detailed by Reijnders et al. (Reijnders, Khadka, Jansen, & Hage, 2011) using the method engineering approach. From these service-oriented development methods, we have added candidate service identification and deployment and provisioning phases to our evaluation framework. Finally, our evaluation framework includes six phases divided over two generic stages. The evaluation framework and the phases are depicted in Figure
2. The evolution planning addresses the question “what to do?” and “is evolution feasible in the given context?” The evolution implementation & management addresses the question “how to do it?” and “what techniques can be used to perform the evolution?” In the following subsections, we explain the phases of our evaluation framework.

**Legacy System Understanding**

Understanding the legacy system and its as-is situation are crucial to the success of any evolution (Seacord, Plakosh, & Lewis, 2003). This includes a detailed analysis of the legacy system and various techniques can be used. For instance, reverse engineering, program understanding, architectural recovery can be used, often with tool support to generate system artifacts. Legacy system understanding often includes analyzing the development history, interviewing the developers (if any) and current users to come to an understanding of the architecture of the legacy system. In our evaluation framework, we have defined evaluation criteria to investigate if any legacy to SOA evolution method includes legacy system understanding and to what extent this phase is discussed.

**Target System Understanding**

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

*Figure 2. The evaluation framework*
Evolution Feasibility Determination

The legacy system understanding and the target system understanding phases provide better understanding of the as-is and to-be situations, respectively. Based on this understanding, the feasibility of the evolution has to be determined and is done in the evolution feasibility determination phase. The feasibility assessments are carried out at a technical, economical, and organizational level. The technical assessment includes measuring the code complexity of the legacy system in terms of cohesion, coupling, reusability and abstraction (Reddy, Dubey, Lakshmanan, Sukumar, & Sisodia, 2009). Economical assessment includes determining economic feasibility of the evolution, for instance by using the cost-benefit analysis, as suggested by Sneed (1995a). Upon analyzing the technical and economical feasibility, the organization approves the evolution project by also considering whether its business goals are met by the intended SOA system. In our evaluation framework, we have defined evaluation criteria to determine whether a legacy to SOA evolution method includes evolution feasibility and if so, how is it performed.

Candidate Service Identification

Legacy systems are subjected to evolutionary development and bug fixing in the source code often by people who did not develop it. This typically leads to much redundancy in the code. Furthermore, poor documentation and lack of appropriate resources (e.g. developers, architects) make the understanding of source code a hard task. In such a scenario, identifying the potential services and service-rich areas in a legacy code is definitely a challenging task (Khadka, 2011). The candidate service identification phase aims at locating the service-rich areas. Various techniques can be used for this purpose. For instance, architectural reconstruction, feature location, design pattern recovery, cluster analysis techniques, concept analysis, source code visualization can be used to identify the service-rich areas in a large body of legacy code. In our evaluation framework, we have defined evaluation criteria to investigate if any legacy to SOA evolution method includes techniques to identify potential candidate services.

Implementation

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

Deployment and Provisioning

The deployment and provisioning phase is concerned with deployment and management of the services after extraction of the legacy code. Upon extraction, services are deployed in the service infrastructure. Service provisioning typically includes the after-deployment activities such as publishing, versioning of services, metering and billing of the usage of the services (Khadka, Saeidi, Jansen, Hage, & Helms, 2011). In our evaluation framework, we have defined evaluation criteria to determine whether a legacy to SOA evolution method includes deployment and provisioning. Based on the identified phases, we have derived the list of evaluation criteria given in Table 2: the first column presents the stages within an evolution, the second column lists the identified phases of our evaluation framework, the third column presents the evaluation question as evaluation criterion for each phase, and the final
column gives possible answers for each evaluation question. The answers can be of three types: Yes/No- to indicate whether the given criterion is met, narrative- to answer an open question, and scale- to quantify the degree of support for any criterion. The judgment of scale is presented in Appendix B.

**OVERVIEW OF THE PRIMARY STUDIES**

In total, we found 121 publications as our primary after evaluating against the inclusion and exclusion criteria. Figure 3 shows the distribution of primary studies published per year along with the trend-line. The positive slope of the trend-line not only indicates an increasing amount of research being carried out in legacy to SOA evolution domain, but also reflects the increase of legacy to SOA evolution approaches along with the maturity of SOA paradigm- SOA being used as architectural style after SOAP (Box, et al., 2000) was first submitted to W3C in 2000. We cannot be certain that we have covered all studies with a publication date in 2011, since studies may not have been indexed yet at the time. This is one of the possible reasons for the sharp decrease in publication in 2011.

Figure 4 presents the distribution of the primary studies across venues from which at least two articles were selected. It is very interesting to notice that the largest amount of research is reported at venues related to system maintenance, evolution and re-engineering such as CSMR, ICSM, WSE rather than core service-oriented computing venues such as SCC, ECOWS, ICSOC. This implies legacy to SOA evolution is often seen as a solution to maintenance/evolution problems of (legacy) software systems. Also, the

**Table 2. The evaluation criteria based on the evaluation framework**

<table>
<thead>
<tr>
<th>Stage</th>
<th>Phase</th>
<th>Evaluation question</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evolution planning</td>
<td>Legacy system understanding</td>
<td>Does the solution include legacy system understanding?</td>
<td>Yes/No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Which technique(s) is used for legacy system understanding?</td>
<td>Narrative</td>
</tr>
<tr>
<td></td>
<td></td>
<td>To what extent are those techniques used?</td>
<td>Scale</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Is there any tool support for legacy system understanding?</td>
<td>Yes/No</td>
</tr>
<tr>
<td></td>
<td>Target system understanding</td>
<td>Does the solution include target system understanding?</td>
<td>Yes/No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>What criteria/factors are included for target system understanding?</td>
<td>Narrative</td>
</tr>
<tr>
<td></td>
<td></td>
<td>To what extent are those criteria/factors used?</td>
<td>Scale</td>
</tr>
<tr>
<td></td>
<td>Evolution feasibility determination</td>
<td>Does the solution include evolution feasibility assessment?</td>
<td>Yes/No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>What technique(s) is used for evolution feasibility assessment?</td>
<td>Narrative</td>
</tr>
<tr>
<td>Evolution implementation &amp; management</td>
<td>Candidate service identification</td>
<td>Does the solution include candidate service identification?</td>
<td>Yes/No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>What technique(s) is used for identifying candidate services?</td>
<td>Narrative</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Is there tool support for candidate service identification?</td>
<td>Yes/No</td>
</tr>
<tr>
<td></td>
<td>Implementation</td>
<td>Does the solution provide any implementation technique for evolution?</td>
<td>Yes/No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>What technique(s) is used for implementation?</td>
<td>Narrative</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Is there tool support for the implementation?</td>
<td>Yes/No</td>
</tr>
<tr>
<td></td>
<td>Deployment and provisioning</td>
<td>Does the solution provide deployment and provisioning of the services?</td>
<td>Yes/No</td>
</tr>
<tr>
<td>Case study</td>
<td></td>
<td>What empirical evidence (industrial/experiment) is provided?</td>
<td>Narrative</td>
</tr>
<tr>
<td></td>
<td></td>
<td>In which language is the legacy system developed?</td>
<td>Narrative</td>
</tr>
</tbody>
</table>
frequency of publication in journals is also relatively low as compared to conferences or workshops, which is not surprising in such a young field. Note that we have not included the venues with less than two occurrences.

Table 3 presents the distribution of the primary studies according to the kind of source. The result shows that conferences are the most widely used method of dissemination for legacy to SOA evolution approaches. The journal papers for legacy to SOA evolution approaches score quite low as compared with the conference papers.

RESULTS

The result of our SLR is based on the evaluation criteria described in Table 2. Using our evaluation criteria, we evaluated 121 publications. Due to limitations of space, we have not included the full result of our complete evaluation in this chapter. Appendix A depicts an evaluation of a small number of articles. For the complete evaluation result, please consult Khadka et al. (Khadka, Saeidi, Idu, Hage, & Jansen, 2012). The result is primarily focused on whether the publication supports the phases of our evaluation framework, what methods and technologies are used (if supported), and whether any tool support for methods and techniques is discussed. Furthermore, the details of empirical evidence (case study) reported in each publication are also presented. In our evaluation, we created the inventory of the methods and techniques as mentioned in the publication. We did not conduct any subjective assumption for categorization. For instance, in many publications “architectural recovery” and “architectural reconstruction” of the legacy system understanding phase are considered to be identical; however, we did not combine them into one. Since we do not conduct any subjective assumption, we believe that this will reduce biasness of our findings. We present our findings with two aspects:

1. **Degree of coverage**: Indicates what stages/phases are supported by the primary studies and,
2. **Methods and techniques used**: Inventory of what methods and techniques are generally in practice in each phase.

Degree of Coverage

Out of 121 publications, 12 publications have full coverage of the evolution planning stage, i.e., 12 publications support the legacy system understanding, target system understanding and evolution feasibility determination phases. Indi-
individually, under the evolution planning stage, 66 publications support legacy system understanding, 43 publications support target system understanding and 20 publications support an evolution feasibility determination phase.

Similarly, 15 publications out of 121 have full coverage of the evolution implementation and management stage, i.e., 15 publications support the candidate service identification, implementation, and deployment and provisioning phases. Individually, 59 publications support the candidate service identification phase, 97 support the implementation phase and 22 support the deployment and provisioning phase. Interestingly, only 2 publications (Khadka, et al., 2011; Zillmann, et al., 2011) support the overall phases of our legacy to SOA evolution framework. Table 4 presents the distribution of primary studies per phase.

**Methods and Techniques**

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

Figure 5 depicts the methods and techniques that are used for the legacy system understanding phase. Reverse engineering technique is by far the most widely used technique. Documentation and Interviewing are the second and third most used techniques followed by mostly source code analysis or architectural reconstruction techniques. Based on the Scale criteria (-, +, ++, +++), 22 papers extensively discussed legacy system understanding with ++++, 18 papers with ++, and 20 papers with +. In most of the cases, multiple methods and techniques were used for legacy system understanding. An interesting observation is that most of the methods and techniques used for legacy system understanding are technical in nature such as reverse engineering, architectural recovery, program understanding. Manual techniques like documentation and interviewing...
are less common than in-depth descriptions of technical methods. One of the reasons for using methods and techniques of such technical nature is that legacy resources like documentation and developers are scarce—a widely identified problem in legacy evolution (Bennett, 1995; Bisbal, et al., 1999). Furthermore, only 26/121 papers discuss tool support for legacy system understanding. In most of the papers, multiple techniques are combined for legacy system understanding.

Figure 6 depicts the methods and techniques that are used for target system understanding. Here selection of a specific architecture is most widely used. It is interesting to note that almost all of the instances in the chart are techniques that actually represent the technological aspect of target system, while only Interviewing refers to the process (i.e., organizational) perspective. Only 13/121 papers extensively discusses the target system understanding with +++, 12/121 papers with ++, and 12/121 papers with +.

The methods and techniques that are used for the evolution feasibility determination phase are shown in Figure 7. Here Cost-Benefit Analysis (CBA) is widely used, followed by Code complexity. While CBA technique is primarily an economically oriented analysis, the other most used techniques, Code complexity and Reusability assessment, refer to a technical analysis. The details of CBA are presented by Sneed (1995a) and Umar et al. (Umar & Zordan, 2009) and the details of Code complexity is explained by Sneed (2009). The concept of Option Analysis for Re-engineering (OAR) is proposed by Bergey (2001); it has been used in SMART (Balasubramaniam, Lewis, Morris, Simanta, & Smith, 2008; Lewis, et al., 2005).

Figure 8 depicts the methods and techniques that are used for the candidate service identification phase. Manual identification is most widely used. It is also noteworthy that none of the other techniques are widely used, leading to 51 distinct techniques encountered in the primary studies other than Manual. It is interesting to note that candidate service identification has also been separately researched to foster legacy to SOA evolution.

Alahmari et al. (Alahmari, Zaluska, & De Roure, 2010) propose model-driven architecture based service identification using a SOA meta-
model to identify services in legacy code. Aversano et al. (Aversano, Cerulo, & Palumbo, 2008) combined information retrieval techniques with a similarity based metric to identify potential services in legacy systems. In Chen, Zhang, Li, Kang, and Yang (2009), the authors propose an ontology based approach in which an ontology stores knowledge of both the application domain and the legacy code. Later, formal concept analysis and relational concept analysis are used to identify the candidate services. The authors in Nakamura, Igaki, Kimura, and Matsumoto (2009) generate data flow diagrams from the legacy code using reverse engineering techniques to aid can-

![Distribution of methods and techniques used for legacy system understanding](image1)

**Figure 5. Distribution of methods and techniques used for legacy system understanding**

![Distribution of methods and techniques used for target system understanding](image2)

**Figure 6. Distribution of methods and techniques used for target system understanding**
didate service identification. The authors of Zhang, Liu, and Yang (2005) use clustering technique to analyze the architectural information and identify the related modules as potential candidate services.

The methods and techniques that are used in the implementation phase are presented in Figure 9. Wrapping is by far the most widely used. Considering the big difference between Wrapping and the other techniques used, we believe that most of the legacy to SOA evolution techniques do not focus on altering existing legacy code bases. In addition, wrapping is a fast, less risky, economical, and easy solution although the legacy system remains monolithic. The result of our evaluation shows that techniques like model transformation, program slicing, and code transformations are much less frequently used. From Table 4, we find that 97 out of 121 papers support the Implementation phase. In our evaluation, we also found out that 74 papers out of these 97 papers (i.e., papers supporting the implementation phase) also have tool support for implementation. Furthermore, 22 publications support the deployment and provisioning phase.

Figure 10 depicts the distribution of empirical studies conducted to validate the proposed legacy to SOA evolution in the primary studies. The majority of primary studies presented case studies, which were performed at an Industrial level. Interesting to note is the fact that there was a small number of studies that presented both Experimental and Industrial case studies, thus covering a wider applicability of validation. Among the industrial case studies, C++ and COBOL based legacy systems are most common: four cases for each. In the experimental case studies, Java-based systems were widely used (sixteen in all), followed by COBOL (four systems).

DISCUSSION

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

Findings and Best Practices

The evolution planning stage of the evaluation framework (see Figure 2) addresses the feasibility of evolution from business and technical perspectives. The evolution planning focuses on justifying whether the legacy system is economically and technically suitable for evolution. To a large extent, the success and failure of an evolution project depends on proper planning (Sneed, 1995a). In the context of legacy to SOA evolution, evolution planning becomes more complicated as various methods and techniques are used for candidate service identification and implementation. Figures 8 and 9 illustrate the distribution of methods and techniques used for these purposes.

Figure 8. Distribution of method and techniques used for candidate service identification

Figure 9. Distribution of method and techniques used for implementation
technical factors of the legacy systems should be well understood. Such technical factors include complexity metrics (Sneed, 1995b) and coupling and cohesion metrics for reusability (Gui & Scott, 2006; Perepletchikov, Ryan, Frampton, & Tari, 2007). In the case of legacy systems, obtaining such information is a challenging task, particularly due to the unavailability of resources and documentation. Other important factors include cost estimation for evolution and economic feasibility to determine the profitability of evolution. The economic feasibility should take into account the current expense of maintaining the legacy system and the costs predicted for maintaining the target system after evolution. Hence, evolution planning should also consider the architecture and standards of the target system.

Within evolution planning, legacy system understanding has been extensively investigated, primarily with reverse engineering techniques. The two major categories under reverse engineering in legacy system understanding are program understanding and architectural reconstruction. Program understanding (Corbi, 1989) is defined as the process of acquiring knowledge about a computer program and is extensively used for software maintenance, software evolution and software reengineering. Corbi (1989) identifies three actions that can be used to understand a program: read about the program (e.g., documentation), read the program itself (e.g., read source code) and run the program (e.g., watch execution, get trace data, etc.). All these three actions have been used in legacy to SOA evolution under various related topics, such as documentation, source code analysis, static analysis and dynamic analysis. Most of the articles state that program understanding or source code analysis techniques are used to understand the legacy codes. However, only few articles explain such program understanding techniques in detail. Source code analysis techniques have been presented well by Zhang (Zhang, Yang, & Chu, 2006); static analysis in Zillmann et al. (2011) using Flow Graph Manipulator (FGM) and dynamic analysis using JGrabLab/GReQL in Zillmann et al. (2011) and TGraph in Fuhr, Horn, Riediger, and Winter (2011). Architectural reconstruction is a process in which the architecture representations of a software system are obtained from the existing source code (Kazman, O’Brien, & Verhoef, 2001) and is widely used in the software reengineering domain. Similarly, the use of architectural reconstruction has been also been reported in legacy to SOA evolution approaches. Cuadrado et al. (2008) used the QUE-es Architecture Recovery (QAR) workflow to reconstruct the architecture of legacy systems using Jude, Omondo UML studio and Eclipse Test and Performance Tools Platform (TPTP) tool. Lewis et al. (2006) and O’Brien et al. (O’Brien, Smith, & Lewis, 2005) used the ARMIN tool to reconstruct the architecture of the DoD Command and Control (C2) legacy application such that various undocumented dependencies in the source code were identified. Li and Tahvildari (2008) used the Extracting Business Services (E-BUS) toolkit to reconstruct the architecture of various Java-based systems. Similarly, Zhang et al. (2005) uses architecture recovery to obtain design and architectural information that are used as input for service identification. Our evaluation indicates that architectural reconstruction has been used more often with tool support than program understanding. In addition, in most of the cases both program understanding and architectural reconstruction have been employed. Feature loca-
tion techniques (Chen, Li, Yang, Wang, & Chu, 2005; van Geet & Demeyer, 2010; Vemuri, 2008) have also been reported in understanding legacy systems.

The target system understanding phase intends to choose the architecture and related SOA technologies of the future system, which eventually plays an important role in the quality of the future SOA system. Lewis (2005) argues that the characteristics of the target system will temper decisions about whether legacy components can be reused. Basically, target system understanding can be viewed from two perspectives: functional characteristics and technical characteristics of the target system. The functional characteristics include the potential functionalities to-be evolved from the legacy code. This process is referred to as service design. It also defines to what level of granularity the services are to be defined and, accordingly, the orchestration of the services has to be managed to support business processes. Various functional and non-functional properties should also be considered, such as maintainability, interoperability, responsiveness, performance, security, and availability. The technical characteristics of the target environment include service technology (SOAP or REST-based), messaging technologies, communication protocols, service description languages, and service discovery mechanisms. Despite the importance, target system understanding is not described in detail in most of the articles. Rather, the articles just state that target architecture or target system is an important aspect. However, the functional characteristics of target system understanding has been well explored in SOAMIG (Fuhr, et al., 2011; Zillmann, et al., 2011) and SMART (Lewis, Morris, & Smith, 2006; Lewis, Morris, Smith, & O’Brien, 2005). The SOAMIG method describes the importance of service design, which is the result of forward engineering (design of the target architecture and the orchestration of services) and reverse engineering (potential functionalities as services from legacy system understanding). The SMART method focuses on designing the target system based on the potential functionalities as services and to assess them with the stakeholders by taking into account of various functional and non-functional characteristics of the target system. From the technical characteristics perspective, Cuadrado et al. (Cuadrado, García, Duenas, & Parada, 2008) provide a clear explanation of using the OSGi specification and service platform. The authors consider maintainability and interoperability as important criteria of the target system and accordingly use OSGi specifications to support those non-functional characteristics.

One of the important phases from the organizational perspective is evolution feasibility determination that determines the go or no-go of the evolution project. Evolution feasibility determination focuses on an economical and technical assessment of the legacy system and the target system along with the business goals that the organization wants to achieve through evolution. The evolution feasibility determination phase uses the finding of the legacy system understanding (e.g., code complexity, cohesion and coupling metrics, etc) and the findings of the target system understanding (e.g., non-functional characteristics, selection of service technology, orchestration design, etc) to determine the technical and economical feasibility. The best practices in the evolution feasibility determination phase include the cost-benefit analysis proposed by Sneed (1995a) for re-engineering projects and serves as a good starting point. This CBA model has been widely followed in legacy to SOA evolution (Khadka, et al., 2011; Sneed, 2008, 2009). Umar and Zordan (2009) extended the CBA model to include the integration costs which facilitates the strategic decision making in legacy to SOA evolution. The SMART method uses Options Analysis for Re-engineering (OAR) (Smith, O’Brien, & Bergey, 2002) to determine the so called migration feasibility decision point. Based on the SMART method and a decision
Legacy to SOA Evolution

framework by Erradi (Erradi, Anand, & Kulkarni, 2006), Salama and Aly (2008) present a decision making tool for the selection of the legacy to SOA modernization strategies, which also considers evolution feasibility.

Based on the outcome of the evolution planning, the next step is to decide how to implement the evolution and what techniques are favorable for implementation. It has been widely recognized that legacy evolution is not purely a technical problem, but involves business engineering as well (Ziemann, Leyking, Kahl, & Werth, 2006). The main challenges are how to identify business functionality as a potential service, how to evolve such business functionality as a service and finally, how to maintain and monitor the service once it is deployed. Based on these three requirements, we have identified three phases under evolution implementation and deployment stage.

Identifying service-rich areas in a huge chunk of legacy code has been a challenging task in legacy to SOA evolution. Our survey has revealed that techniques applied to locate service-rich areas can be broadly classified into two: modeling the business requirements (top-down) approach and legacy code to business functionalities (bottom-up) approach. In modeling the business requirement approach, the core business process is designed from the functionalities identified from the legacy system understanding and then the process is subdivided until that can be mapped to functionalities in legacy system. In most of such approaches, BPMN is used to model the business process (Alahmari, Roure, & Zaluska, 2010; Fuhr, et al., 2011; Li, Anming, Naileyue, Jianbin, & Zhong, 2009; Ricca & Marchetto, 2009; Zillmann, et al., 2011). The legacy code to business functionalities approach utilizes legacy code as starting point to discover existing business knowledge within legacy systems. Various techniques have been used, such as information retrieval (Aversano, et al., 2008), concept analysis (Zhang, et al., 2006), cluster analysis (Zhang & Yang, 2004), business rule recovery (Marchetto & Ricca, 2008) and pattern matching and discovery (Guzman, Polo, & Piattini, 2007; Jiang & Stroulia, 2004; Zhang, Zhou, Yang, & Zhong, 2010).

Based on the findings of the Implementation phase, the legacy to SOA evolution can be either categorized as legacy system integration or legacy system migration. Legacy system integration is an approach in which the legacy codes is not substantially modified and is used from within a new environment. The legacy systems typically remain in their original environment. Generally, techniques like wrapping, adaptors and middleware based approaches fall into the integration category, which is the predominant implementation technique as far we have seen in legacy to SOA migration (cf Figure 9). Integration is claimed to be a fast, less risky, economical and easy solution but the legacy system remains as it is (Almonaihs, et al., 2010; Umar & Zordan, 2009). Wrapping based legacy to SOA evolution are reported by Sneed (2008, 2009) in which the author has developed various tools to support the evolution; Ricca and Marchetto (2009) used wrapping to evolve ATM functionality to SOA; Zhang et al. (2008) used wrapping to evolve GUI-based legacy systems to SOA. On the other hand, the legacy system migration approach is one in which the legacy code is transformed, internally modified, or reused in a new environment. Umar and Zordan (2009) define migration as “an internal restructuring and modification of legacy systems into target systems.” The migration technique is claimed to be costly, time consuming but in the long run the organization can gradually replace the existing legacy system. Various techniques have been used to migration legacy systems to SOA (program slicing [Bao, Yin, He, Ge, & Chen, 2010; Chen, et al., 2009; Khadka, et al., 2011; Marchetto & Ricca, 2008; Zhang, et al., 2006]; model transformation techniques [Chen, Yang, Qiao, & Chu, 2006; Fuhr, et al., 2011; Hoyer, et al., 2009]). The distinction between integration
and migration is discussed by Umar et al. (Umar & Zordan, 2009) in detail with respective benefits and drawbacks.

In the deployment and provisioning phase, the evolved services have to be deployed and activities are required to manage and control the behavior of services during usage. In the context of legacy to SOA evolution, activities such as testing, versioning and monitoring are important. Service testing has been a research challenge in the SOA domain due to the dynamic binding (Canfora & Di Penta, 2006) and the fact that the source code of services might not reside within a single organization (Lewis, Smith, Chapin, & Kontogiannis, 2009). Service testing in the context of legacy to SOA evolution is even more complicated because the exposed service after evolution should perform correctly when compared to the legacy system. Some legacy to SOA evolution approaches also address the testing of exposed services (Fuhr, et al., 2011; Khadka, et al., 2011; Marchetto & Ricca, 2008; Sneed, 2008; Zillmann, et al., 2011). Due to changing business requirements, services need to evolve and this leads to multiple versions of an exposed service (Fang, et al., 2007; Khadka, et al., 2011). Service versioning is inevitable in legacy to SOA evolution as well, particularly, in legacy system integration approaches. In legacy system integration, the legacy code is exposed through interfaces, without making any changes to the original code. Later, changes made to legacy code after evolution have to be reflected in the service interfaces as well and this creates multiple versions of the original service. In addition, service monitoring for non-functional attributes becomes important while the exposed services are in use. Service versioning and service monitoring has not received much attention in legacy to SOA evolution.

An increasing number of articles from 2000 to 2011 on legacy to SOA evolution suggests that the hype is gaining momentum in academia and is still in maturing stage. It is also interesting to see that almost half of the results of the research are evaluated in an industrial context (see Figure 10). Some good examples of such research include: the SMART (Lewis, et al., 2005) which has been evaluated in migrating the Department of Defense Mission Status System and Command and Control system, the SOAMIG process model (Fuhr, et al., 2011; Zillmann, et al., 2011) in Amadeus Germany’s RAIL-system, the wrapping method (Sneed, 2008, 2009) for a COBOL-based insurance system, the migration of Java-based legacy application to SOA (Bhallamudi & Tilley, 2011), the feature analysis method for migrating a COBOL-based telecommunication systems (Millham, 2010), and a case study of adopting SOA in the transportation sector (Nasr, Gross, & Deursen, 2010).

Research Issues and Agenda

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

Legacy to SOA Evolution as a Process

Legacy to SOA evolution is a complex process, which is influenced by technical, economical and organizational factors. So, any legacy to SOA evolution requires a structured process model that can address these technical, economical and organizational factors. The need of such a structured process model has been also argued by various researchers (Kontogiannis, et al., 2007; Lewis, et al., 2009). Such a structured evolution process should include a legacy system assessment to recover knowledge, the standards and architecture of the target system, technical & economical feasibility, a risk analysis, candidate service identification, and the implementation and maintenance of the system after evolution. Our evaluation framework (see Figure 2) addresses these requirements as it covers
all the aspects necessary to support any legacy to SOA evolution project. One interesting finding of our SLR is that only two articles (Khadka, et al., 2011; Zillmann, et al., 2011) cover all aspects of legacy to SOA evolution as identified by our evaluation framework.

Automation of the Legacy to SOA Evolution Process

Upon establishing a legacy to SOA evolution process model, the next challenge is the automation of such legacy to SOA evolution process through the development of tools and techniques. As identified by various researchers, e.g., Kontogiannis, et al. (2007), Lewis et al. (2009), and Nasr et al. (2010), one of the major issues of legacy to SOA evolution is tool support for the various phases. In fact such automation would be expensive and needs a huge effort due to variation in legacy systems. As can be seen from our SLR finding, various tools and techniques have already been successfully developed and used in legacy to SOA evolution. Establishing the suitability of those tools and techniques following a legacy system assessment (technical qualities of legacy code) in the various phases is an interesting and challenging future research topic. Another issue that is worth investigating is “Can legacy to SOA evolution be carried out in language independent manner?” A potential research direction to address this issue could be model-driven legacy to SOA evolution. We are currently involved in an ongoing research project (Servicifi, 2010) that aims at generating a model of the legacy code and identifying patterns to locate service-rich areas. Such patterns are then employed in tandem with a code-query based program slicer after which the sliced out functionality can be exposed as a service. There have been other initiatives in the model-driven legacy to SOA evolution as well (ADM, 2010; Fleurey, Breton, Baudry, Nicolas, & Jézéquel, 2007; Fuhr, et al., 2011; REMICS, 2012).

After-Evolution Experience Reporting

Legacy to SOA evolution is not just about the successful technical transformation of an existing state to a new state. Most reports about the legacy to SOA evolution claim successful evolution because it was technically and economically feasible and the desired target state of SOA has been achieved (Nasr, et al., 2010). However, this “successful” evolution does not really indicate that the enterprise has achieved its business goals. Answers to various questions still remain unclear after such a “successful” evolution. Did the legacy to SOA evolution deliver the promised benefits such as increased flexibility, enhanced maintainability, and reduced costs? In many legacy to SOA evolution projects, there were explicit requirements of the enterprise (e.g., Cuadrado et al., 2008) aimed at increased usability and interoperability. Does the evolution to SOA successfully meet such requirements? As identified by Sneed (Sneed, 2006) one of the issues after evolution is performance. Such issues are still to be investigated in sufficient detail through experimental analysis.

Determining the Decomposability of Legacy Systems

One of the fundamental issues, pointed out by Brodie and Stonebraker (1998), is that the evolution of legacy system depends on its decomposability. The less decomposable a system is, the more difficult evolution will be. However, there are still no explicit factors that determine the decomposability of a legacy system. Sneed (2006, 2009) provides requirements in terms of code properties for determining the suitability of legacy code for wrapping. Similar requirements should also be investigated to determine the decomposability of a legacy system based on the legacy code and complexity. Determining the decomposability of the legacy code facilitates the evolution feasibility process and thus enables choosing the right
evolution strategy (i.e., wrapping, replacement, redevelopment, migration) (Almonaies, et al., 2010).

Evolution from Organizational Perspective

The SLR reveals that legacy to SOA evolution is primarily seen as a technical challenge, focused on finding an efficient solution for evolution. However, legacy to SOA evolution also introduces various organizational challenges such as ownership of services, responsibility of maintaining and monitoring of services and resistance from the current IT staffs to change. One of the peculiar challenges includes the adoption problem (Khadka, et al., 2011; Mahmood, 2007) in which the existing users of legacy systems may fear that their expertise may become redundant due to the introduction of SOA. Such organizational issues should also be properly investigated and considered in legacy to SOA evolution.

THREATS TO VALIDITY

In our SLR, various subjective measurements have been involved. For instance, the selection of primary studies and data extraction process of review protocol and the scale measurement in the evaluation framework itself have subjective measurements. Such subjective measurements can bias the overall result of the findings. Hence, we justify the validity of the results by discussing the possible threats to our result and the countermeasures that we have taken to minimize them. Our literature survey is subjected to the following three types of threats: threats to construct validity, threats to internal validity, and threats to external validity.

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

Internal validity refers to the extent to which the design and execution of the study are likely to prevent systematic errors. In our research, internal validity refers to the elimination of bias. In our review, the involvement of five researchers in the study selection and evaluation process minimizes the threats to internal validity. Furthermore, in each round of study selection the distribution of the studies were done in such a way that each researcher obtains a different set of studies. We have introduced three categories of studies “relevant,” “irrelevant,” and “moderate.” For each moderate study, the next categorization is done by a researcher other than the one who categorized the study as “moderate.” Another potential area of bias is the categorization of the studies into “relevant,” “irrelevant” and “moderate.” Such a threat is mitigated by clearly specifying the inclusion and exclusion criteria (see Table 1). Furthermore, the data selection (initial selection, secondary selection, and primary study) process was distributed among five researchers rather than one researcher. This step also reduces the possibility of bias.

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

CONCLUSION AND FUTURE RESEARCH

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

The resulting overview of the evaluation has created an inventory of historical contributions to the evolution of legacy to SOA, and a list of methods and techniques that are widely practiced. Due to limitations of space, only a snapshot of the result of evaluation can be presented in Appendix A. Particularly, the methods and techniques according to the phases of our evaluation framework have provided insights into existing practices in the legacy to SOA evolution process. In summary, the work described in this chapter offers the following contributions:

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

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

We have identified several possible improvements of our research as well. One of the enhancements of the current evaluation process includes double checking the evaluation result. In the presented evaluation, the primary articles were divided among five researchers and then evaluated. As an enhancement, we aim at double checking each evaluation result by at least one other researcher. This will surely reduce bias (i.e., no subjective categorization are made) and lead to a more accurate finding. In our evaluation, we reported what was reported in the article. For instance, “architectural recovery” and “architectural reconstruction” techniques can be considered to be the same and both of them again can be considered to fall under the heading of “reverse engineering.” In our evaluation, we have not made use of such subjective assumptions. In the future, we aim at refining the results of our evaluation with attribute generalization (Cornelissen, Zaidman, van Deursen, Moonen, & Koschke, 2009): a technique to generalize the values of the finding into common and related category. Furthermore, we also aim at evaluating the proposed evaluation framework with case studies and enhance it accordingly. Currently, our research is only focused on the legacy to SOA evolution reported in academia. In future, we aim to also provide similar insights into the legacy to SOA evolution approaches practiced in industry.
REFERENCES


Legacy to SOA Evolution


KEY TERMS AND DEFINITIONS

**Candidate Service Identification:** Candidate service identification refers to a process of locating service-rich areas in a legacy code.

**Legacy Evolution:** Legacy evolution is a gradual process of performing structural changes to legacy systems such that they comply with new technological and organizational requirements.

**Legacy System:** Legacy system is an old application program written using outdated techniques and programming languages such as COBOL, RGP, PL/I, but it continues to do useful work.

**Legacy System Understanding:** Legacy system understanding refers to as-is analysis of the existing legacy systems and thus enabling in better understanding of technical and functional characteristics of the legacy systems.
**Service:** A service in SOA is defined as a self-contained, platform-agonistic computational element that supports rapid, low-cost and composition of loosely-coupled software applications.

**Service-Oriented Architecture (SOA):** SOA is an architectural style that enables the reuse of existing legacy assets within a new paradigm that facilitates loose coupling, abstraction of underlying logic, flexibility, reusability, and discoverability.

**Systematic Literature Review (SLR):** SLR is a systematic approach that aims at providing an exhaustive summary of literature relevant to a research question.
### APPENDIX A

**Table 5. An overview of primary studies w.r.t the evaluation criteria**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Evolution Planning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Legacy System Understanding</td>
</tr>
<tr>
<td></td>
<td>Y/N Technique</td>
</tr>
<tr>
<td>(Salama &amp; Aly, 2008)</td>
<td>Y Source Code Analysis</td>
</tr>
<tr>
<td>(Sneed, 2009)</td>
<td>N N/A</td>
</tr>
<tr>
<td>(Zhang, Yang, Zhou, &amp; Zhong, 2010)</td>
<td>Y Reverse Engineering, Domain Analysis</td>
</tr>
<tr>
<td>(Canfora, Fasoliino, Fratolillo, &amp; Tramontana, 2008)</td>
<td>Y Reverse Engineering, Static Analysis, Dynamic Analysis</td>
</tr>
<tr>
<td>(Sneed, 2008)</td>
<td>N N/A</td>
</tr>
<tr>
<td>(Zhang, et al., 2006)</td>
<td>Y Source Code Analysis</td>
</tr>
<tr>
<td>(Vemuri, 2008)</td>
<td>Y Test Cases, Feature Analysis</td>
</tr>
<tr>
<td>(Sneed, 2006)</td>
<td>N N/A</td>
</tr>
<tr>
<td>(Fuhr, et al., 2011)</td>
<td>Y Code Parsing, Model Transformation</td>
</tr>
<tr>
<td>(Del Castillo, García-Rodríguez, &amp; Caballero, 2009)</td>
<td>Y Interviewing, Database Model Recovery</td>
</tr>
<tr>
<td>(Umar &amp; Zordan, 2009)</td>
<td>Y Strategic Analysis</td>
</tr>
<tr>
<td>(O’Brien, et al., 2005)</td>
<td>Y Architectural Reconstruction, Program Understanding</td>
</tr>
</tbody>
</table>

*continued on following page*
<table>
<thead>
<tr>
<th>Table 5. Continued</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Candidate Service Identification</th>
<th>Implementation</th>
<th>Deploy &amp; Provisioning</th>
<th>Case Study</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y/N Technique Tool supp.</td>
<td>Y/N Technique Tool supp.</td>
<td>Y/N</td>
<td>Exp./Ind.</td>
</tr>
<tr>
<td>N</td>
<td>N/A</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Y</td>
<td>Concept Analysis</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Y</td>
<td>Matching Algorithm</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Y</td>
<td>Business Value, Technical Quality Assessment, Use Cases</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Y</td>
<td>Manual</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Y</td>
<td>Manual</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>N</td>
<td>N/A</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Y</td>
<td>Formal Concept Analysis</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Y</td>
<td>Feature Analysis</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Y</td>
<td>Business Rule and Value Analysis</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Y</td>
<td>Graph Query</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Y</td>
<td>Model Driven Pattern Matching</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>N</td>
<td>N/A</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Y</td>
<td>Manual</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>N</td>
<td>N/A</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>
### APPENDIX B

**Table 6. The judgment scale to assess the support of techniques and method used**

<table>
<thead>
<tr>
<th>Scale point</th>
<th>Scale Definition</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>No support</td>
<td>The specified technique is not mentioned.</td>
<td>-</td>
</tr>
<tr>
<td>Implicitly discussed</td>
<td>The specified technique is mentioned.</td>
<td>+</td>
</tr>
<tr>
<td>Explicitly discussed</td>
<td>The specified technique is mentioned and discussed but no detailed information is given.</td>
<td>++</td>
</tr>
<tr>
<td>Explicitly discussed with evidence of use</td>
<td>The specified technique is mentioned, discussed and there is empirical evidence of its usability.</td>
<td>+++</td>
</tr>
</tbody>
</table>