A Structured Legacy to SOA Migration Process and its Evaluation in Practice

Ravi Khadka, Amir Saeidi, Slinger Jansen, Jurriaan Hage
Department of Information and Computing Sciences, Utrecht University
{r.khadka, a.m.saeidi, slinger.jansen, j.hage}@uu.nl

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

I. Introduction

One of the IT challenges faced by many enterprises is the maintenance of their legacy applications and migration of those applications to modern and flexible platforms. Legacy applications inherit various problems such as lack of up-to-date documentation, skilled manpower, resources of the legacy applications, and high maintenance costs [1]. Despite such problems, enterprises cannot simply remove/replace those applications as they are mission critical, implement the core business logic, and their failure can have a significant impact on business[1]. Thus, legacy applications present a dilemma: they are vitally important to the business, however maintaining them incurs unjustifiable expenses [2]. A viable solution to this dilemma is to migrate those systems into new technological environments in which the legacy features can be re-used. Service-Oriented Architecture (SOA) has gained significant attention from academic and industry as a promising architectural style enabling legacy applications to expose and reuse their functionalities [3]. A legacy to SOA migration not only aims at reducing maintenance costs [4], but also enables a higher Return on Investment (ROI) [5] and promotes flexibility to changing business needs [6]. With all these aforementioned benefits of SOA, several approaches to migrate legacy applications to SOA have been reported in academia [7], [8] and in industry [9], [10]. These approaches can be basically categorized into two aspects: (i) migration planning: to determine the migration feasibility based on technological and economical assessments, and (ii) migration execution: to develop a supporting technology so as to expose legacy applications as a service and to provide service provisioning upon exposing the service. However, a legacy to SOA migration approach requires the consolidation of both the aforementioned aspects (i.e., migration planning and migration execution) [11]. Furthermore, legacy to SOA migration is not only a complex technical endeavor, but it also involves various organizational and business perspectives [5]. So, any legacy to SOA migration requires a structured process that can address these technical, organizational and business issues. The need of such a structured process has been argued by various researchers such as Kontogiannis et al. [6] and Lewis et al. [12], [13].

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

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

The rest of the paper is structured as follows: Section II describes the structured process with its phases and elaborates rationale, best practices and challenges of each phase. Section III presents two case studies to evaluate the proposed process and provides mapping of activities reported in the literature. In Section IV, we analyze and discuss our findings and finally, Section V concludes our research with an outlook to future work.

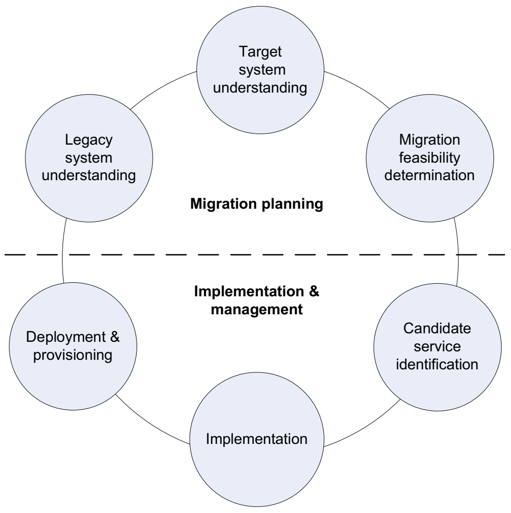
II. The Structured Process

Fig. 1 depicts the proposed structured legacy to SOA migration process that includes six phases divided over two aspects:

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

978-1-4673-4889-8/13/$31.00 ©2013 IEEE

migration planning and *implementation & management*. The *migration planning* aims at answering the questions "*how to plan the migration?*" and "*is migration feasible in the given context?* while *Implementation & Management* addresses the question "*how to execute & manage the migration process?* In the following subsections, we explain each phase.



Target system understanding
Migration feasibility determination
Legacy system understanding
Migration planning
Implementation & management
Deployment & provisioning
Candidate service identification
Implementation

Fig. 1. Legacy to SOA Migration Process

A. Legacy System Understanding

Legacy system understanding (LSU) is a deductive process of acquiring knowledge about the "as-is" situation of legacy applications. The goal of this activity is to find the answers to questions such as "*What does the legacy application do?*", "*How does the legacy application do it?*" *LSU* aims at acquiring information including source code characteristics, identifying dependencies, recovering "as-is" legacy system architecture. Techniques to obtain the legacy information range from manual inspection of development history, interviewing developers (if any) and current users to automated re-engineering techniques [14].

Rationale: In a legacy application, factors such as scarcity of knowledge, lack of resources and up-to-date documentation make the migration process expensive, complex and error prone. In such a context, understanding the existing capabilities of the legacy application is essential. The *LSU* phase does not only assist at creating an inventory of the existing features within the legacy applications, but also facilitates the decomposition of the legacy applications with the aim to maximize reusability. Hence, LSU is essential to the success of legacy to SOA migration [15], [16].

Current Practices: LSU has been extensively investigated, primarily using reverse engineering techniques. Nevertheless, the soft knowledge [17] (i.e., knowledge in the form of skills and experiences within technical staff) is one of the main sources of understanding the legacy applications because these systems are developed and/or maintained by staff familiar with existing legacy systems [5]. Fig. 2 depicts the techniques that are used (not intending to be exhaustive) for *LSU*. We have

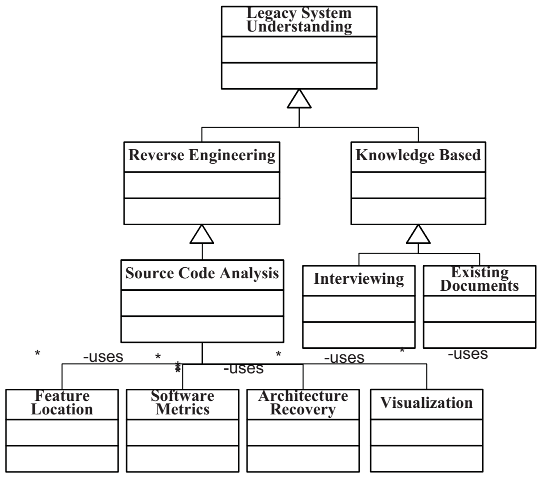


Fig. 2. Legacy system understanding techniques

categorized the techniques into two: (i) reverse engineering using source code analysis and (ii) knowledge-based. By knowledge-based techniques, we mean the knowledge and experience of initial developers and/or maintainer, the end-user experiences obtained via interviewing, and using existing documentation to understand the legacy application. Murer et al. [17] refer to such knowledge as soft knowledge and several authors argue that soft knowledge is of utmost importance to perform a successful legacy to SOA migration [5], [3], [18], [19]. Nevertheless, resources of the legacy applications are scarce and hence, reverse engineering techniques are widely used to further understand legacy application. In particular, researchers have used Source Code Analysis (SCA) to extract information from the legacy applications. Binkley [20] defines source code analysis as "*as a process of extracting information about a program from its source code or artifacts (e.g., from Java byte code or execution traces) generated from the source code using automatic tools"*. In legacy to SOA migration, SCA is used to locate and extract features, extract various software metrics, to recover the architecture of the legacy application and to visualize the dynamic behavior of the legacy application. Feature location, used in identifying functional units in a source code, has been used to understand the legacy application [21], [22], [23]. Similarly, software metrics have been extensively used; Sneed [24], [25] measured the size, complexity and quality of legacy programs in terms of modularity, reusability, maintainability metrics; Perepletchikov et al. [26] used coupling metrics to determine maintainability. Several authors such as Lewis et al. [19], [3], O'Brien et al. [16], Cuadrado et al. [27], Zhang et al., [28] have used architectural recovery- a technique to extract architectural information/views of a software system from the lower-level artifacts such as source code- to facilitate legacy system understanding. Source code visualization, a technique to visualize static and animated forms of software artifacts such as source code and their dependencies, is also used to understand the legacy application [29], [27], [30].

Challenges: Despite the extensive use of techniques to understand the legacy application, various challenges still persist. Being core to the business of the enterprises, legacy applications tend to stay around much longer than the IT staffs who built or maintained them, so the soft knowledge about those legacy applications, their inter-relationships with other components, and their design decisions will eventually become scarce if not carefully preserved. This process is known as the knowledge erosion [31] problem, one of the main issues of enterprises with legacy applications. A viable solution to this problem is to keep the documentation up-to-date and also to initiate technology transfer & training programs where the experienced skilled IT staffs transfer their knowledge. Knowledge erosion is an organizational challenge. One of the technical challenges often encountered in the *LSU* is the heterogeneity of the legacy application landscape with multiple programming languages spanning over different hardware platforms and operating systems. Developing generic tools and techniques for understanding a heterogeneous IT landscape results in high cost. Furthermore, most of the reverse engineering techniques are semi-automatic and require human expertise to complement or correct the information extracted from the legacy application. Maximizing the process of automated reverse engineering techniques to understand the legacy applications is also a challenge.

B. Target System Understanding

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

Rationale: One of the key benefits of legacy to SOA migration is leveraging existing assets [19] and the specification of the target architecture has impact on the level of reusability. Lewis et al. [3] argue that the target architecture largely determine the reusability of the existing legacy components. The other crucial factor that indicates the importance of the TSU phase is the fact that legacy applications have undergone numerous bug fixes and over the years they have been efficient, reliable and responsive to the daily business of the enterprise [1]. Hence, while migrating to SOA, considerable attention should be given to preserve those non-functional characteristics and the target architecture should facilitate to preserve such non-functional characteristics.

Current Practices: Despite its importance, the *TSU* phase is not given sufficient attention. The authors of the SMART method [19], [3] provide guidelines for developing the target architecture based on the legacy components and to assess them with the stakeholder by taking into account various functional and non-functional characteristics of the target system. The SOAMIG method [30] describes the importance of service design as a part of target system understanding, which is the result of forward engineering (design of the target architecture and the orchestration of services) and reverse engineering (potential features from the *LSU*). Cuadrado et al. [27] explain the selection of using the OSGi specification and service platform to preserve maintainability and interoperability non-functional characteristics.

Challenges: For a successful legacy to SOA migration, various business issues have to be considered. From a business perspective, a legacy to SOA migration is generally triggered by new business needs and goals. To fulfill those new business needs and goals, a target architecture needs to ensure a balanced support to the business and to the IT, often termed business-IT alignment. One of the challenges of legacy to SOA migration is to define an appropriate scope for "business-IT alignment" [17]. Key to overcome this challenge is componentization- a process to deconstruct, analyze and identify business components contributing towards business goals of the enterprise [32]. From a technical perspective, achieving and maintaining the non-functional characteristics of the legacy applications is a key challenge while developing a target architecture.

C. Migration Feasibility Determination (MFD)

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

Rationale: A legacy to SOA migration is a multifaceted complex process that involves technical, organizational and business perspectives. Predicting the feasibility of such a complex process to mitigate the risk of failure can contribute to the success. Needless to say, any failure can threaten the success and fortune of an enterprise [33]. With such associated risk of failure, determining the migration feasibility becomes inevitable from all three perspectives (i.e., technical, organizational and business perspectives).

Current Practices: One of the widely used techniques for determining migration feasibility is the Cost-Benefit Analysis (CBA), proposed by Sneed [34]. The CBA technique is used by Khadka et al. [18] and Sneed [24], [25]. Umar & Zordan [35] extended the CBA model to include the migration costs, which facilitates decision making in choosing a migration strategy. The SMART [3] method uses Options Analysis for Re-engineering (OAR) to determine the so called migration feasibility decision point.

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

D. Candidate Service Identification (CSI)

Legacy applications are subjected to evolutionary development and bug fixing. These activities leads to so-called "spaghetti code" [36]. Furthermore, lack of up-to-date documentation and resources make the understanding of the code inevitably hard. In such a scenario, identifying candidate services is a challenging task [6], [30].

Rationale: Identifying candidate services is an important activity in the context of legacy to SOA migration as this activity enables reusability and leveraging the existing legacy features [3]. A plethora of methods are reported (Gu & Lago [37], Arsanjani et al. [38]) to identify potential services.

Current Practices: Current practices of CSI is broadly categorized into two: (i) *top-down* and (ii) *bottom-up* approaches. In the *top-down* approach, initially a business process is modeled based on the requirements and then the process is subdivided into sub-processes until these can be mapped to legacy functions. The *top-down* approach is used by Alahmari et al. [39], Fuhr et al. [40], Ricca & Marchetto [41], and Zillmann et al. [30]. In contrast, the *bottom-up* approach utilizes the legacy code to identify services using various techniques such as information retrieval [42], concept analysis [43], business rule recovery [41], source code visualization [29]. *Challenges:* Several researchers argue that locating and identifying service-rich areas in legacy applications is not only a challenging task, but also an open problem [37], [44]. The functionalities are embedded in legacy applications in such a way that it is difficult to isolate the business functionality from the complex user interfaces and data access logic. Equally challenging is the determination of the optimal granularity of the candidate services such that they contribute to the business goals. Zhang et al. [28] argue that (i) service rationalization- an analysis process to identify the least frequently accessed component as a candidate than that of more frequently accessed ones; and (ii) service consolidation- an iterative process for redefining all the similar service instances into a consolidated version that supports a superset of all the functions exposed by the individual functions, can improve the candidate service identification.

E. Implementation

The *implementation* phase is related to the execution of the migration of the legacy applications to SOA. Needless to say that the implementation depends on factors such as proper migration strategies, assessments of available tools and techniques while executing the migration process. In our approach, we classify migration strategies into four categories: (i) *replacement* in which a legacy application is replaced entirely with a commercial off-the-shelf (COTS) product; (ii) *integration* in which the existing legacy application is accessible via an interface; (iii) *redevelopment* in which the entire legacy application is re-developed into SOA, and (iv) *migration* in which a legacy application is gradually moved to SOA with reusing the legacy components. Fig. 3 depicts the four strategies in a quadrant against "*business value & cost*" versus "*technical value*". Details of these strategies, their merits and demerits are presented by Almonaies et al. [45].

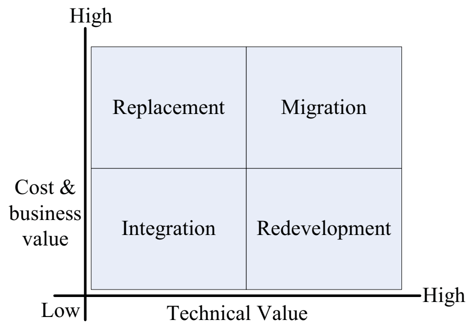


Fig. 3. Realization strategy

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

Current Practices: The *implementation* techniques used in legacy to SOA migration can be broadly grouped into *code level* and *architecture level*. Fig. 4 depicts various implementation techniques (non-exhaustive) that are used in legacy to SOA migration. The *code level* group is further divided into various techniques that have been used in legacy to SOA migration such as slicing [18], [43], [41], [46], wrapping [25], [24], [41], refactoring [27], code transformation [30]. In general, wrapping is by far the most extensively used technique. A plausible explanation of the predominant use of wrapping is due to the fact that it is fast, less risky, economical and easy. At the *architecture level*, graph transformation techniques are used by Heckel et al. [36] and Fuhr et al. [40]. Some of the other techniques being used in legacy to SOA migration are inspired by model-driven engineering [40], [39].

Challenges: One of the important challenges in the "*implementation*" phase is the selection of an appropriate migration strategy. There are no proper guidelines for deciding which factors (e.g., business priority, technical qualities, business value, non-functional characteristics) have to be considered while selecting a strategy. Furthermore, legacy applications are

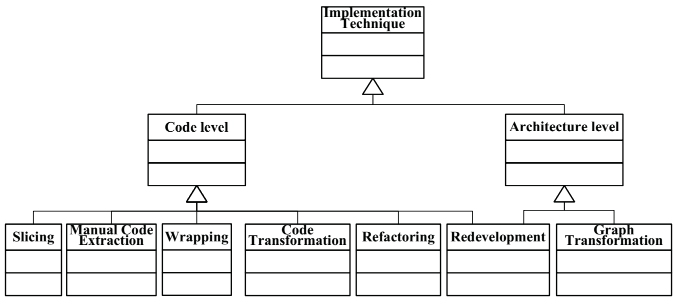


Fig. 4. Implementation Techniques

in a heterogeneous IT landscape and developing tools for each variation of the legacy languages tends to be very expensive. This leads to the following research challenge: can legacy to SOA migration be realized in a language independent manner?

F. Deployment & Provisioning (D&P)

This phase is related with the with deployment and management of the services after exposing the legacy application as a service. The exposed service is then deployed in the service infrastructure and tested to determine if the expected functionality is indeed exposed correctly as a service. A successful deployment then requires service provisioning that includes activities such as publishing and discovering services in a catalog, maintaining Quality of Services (QoS), versioning, testing, and evolution of services [47]. Furthermore, the user support materials such as documentation are created. *Rationale:* This phase includes post migration activities that are crucial to the SOA environment. Services are loosely coupled computation entities [48] and proper management of these entities throughout their life cycle is an absolute requirement [4]. Activities such as service discovery, maintaining QoS of services, testing and evolution of services that lead to the proper functioning of the services ensure that the SOA environment operates reliably and efficiently.

Current Practices: A plethora of research work are reported on service discovery domain [49] in which the authors present categories of service discovery approaches and compare those approaches. Whilst service testing is a relatively a new research domain, various traditional testing approaches are used to test services. A survey of these approaches has been reported by Canfora & Di Penta [50]. With respect to service evolution, various approaches have been reported for managing the evolution of services, e.g., Papazoglou [51] present a theoretical approach for addressing the service evolution problem; Andrikopoulos et al. [52] presents a service evolution management framework to identify changes and introduce version control mechanism for services; and Fang et al. [53] describe a service versioning mechanism to assist service evolution.

Challenges: One of the challenges of the *D&P* phase is automated service discovery with minimal user involvement. Most of the approaches, reported in literature, are semi-automated [49]. The use of semantics markup languages is a step towards automated service discovery [4]. With respect to service testing, various traditional testing approaches have been used. Nevertheless, a key challenge in service testing is to develop a testing approach combined with run-time verification [50]. In terms of service evolution, service versioning plays an important role. One of the challenges in service versioning is to determine the service compatibility between a new service version and the old one [53]. Apart from service compatibility, service versioning potentially introduces overlapping between the service functionalities. Hence, determining the proper level of service commonality in terms of overlapping functionality is also a challenge in service evolution [51].

III. Evaluation

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

A. SrnaCalc application

SrnaCalc[2] is an open-source and simple command-line calculator with basic mathematical functions and scripting capabilities. The goal of this migration project is to extract the function that evaluates any expression. As a part of the *LSU*, we used the reverse engineering tool Understand[3] to explore the features, functional dependencies and compute various metrics of the calculator program. This activity resulted in identifying the following features: "eval" to evaluate the expression, "operator" to display the operators used, "getPrecision" and "setPrecision" to manage precision, "memory" to list the contents of memory and functions to add, find, change, delete, and append a variable. Furthermore, the dependency graph, as depicted in Fig. 5, helped us to understand the structure of the program. As to *TSU*, a SOAP based web service is used in addition to WSO2/C++[4] web service framework as it supports C++. Due to the experimental case, determining economical feasibility of the migration was not relevant. Nevertheless, various software metrics such as coupling, cohesion and Mc-Cabe complexity were derived to determine the technical feasibility of migration. Furthermore, the *CSI* phase was also not relevant due to the small code base. As for *Implementation* phase, we used CodeSurfer[5] tool as a program slicing tool to extract the "eval" feature. Finally, the extracted service was deployed and was successfully tested.

B. Java calculator suite

The second case study is a Java-based mathematical calculator, called Java Calculator Suite[6]. It is an open-source and has Graphical User Interface (GUI) with basic mathematical functions. The motivation of this case study is to migrate

[2]http://sourceforge.net/projects/srnacalc/
[3]http://www.scitools.com/
[4]http://wso2.com/products/web-services-framework/cpp/
[5]http://www.grammatech.com/products/codesurfer/academic.html
[6]http://sourceforge.net/projects/bfegler/

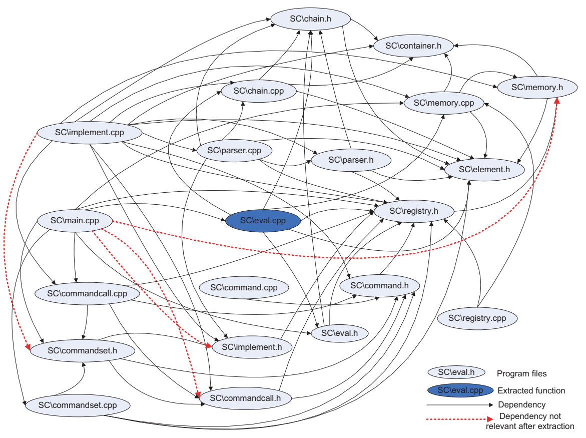


Fig. 5. Dependency Graph

and expose the evaluation function that calculates any given expression. Unlike SrnaCalc, the Java calculator does not have scripting and memory related operations. To perform *LSU*, we used STAN[7] to generate the dependency graph, as shown in Fig. 6. The dependency graph shows how the GUI is related with the calculator feature. Regarding the *TSU* phase, a SOAP based web service is used, and Apache Axis2 web service container[8] is used as infrastructure. The Java Calculator suite is relatively small in terms of size and is an experimental case study. Because of this, the *MFD* and *CSI* are not relevant. For "Implementation" phase, again program slicing was used. Indus[9], a program slicer for Java programs, was used to extract the evaluate function and then the service was deployed and tested.

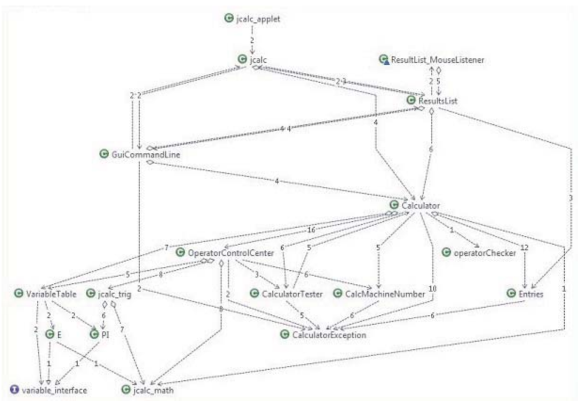


Fig. 6. Dependency Graph

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

The both case studies that we performed to evaluate the applicability of our proposed structured process were relatively small in terms of LOC and were of experimental nature. Because of this, some of the phases were not completely relevant such as in both of the cases we did not perform the *MFD* phase and the *CSI* was done manually. Thus, to further evaluate the applicability of the structured process, we selected academic papers reporting legacy to SOA migration from 2000 to 2011 and mapped the phases/activities from those literature to the phases of the structured process. The papers are selected from our previous work [7] in which we conducted a systematic literature review of legacy to SOA migration. Out of 121 papers, we selected 17 papers from 2000 to 2011 based on the following inclusion criteria:

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

Table I depicts the selected papers and the mapping between the activities reported in the papers with the phases of our structured process. The full citation report is available here[10]. Due to the two inclusion criteria, there were no papers selected from 2000-2002.

From Table I, we can observe that all the phases of our proposed structured process can be mapped to the activities found in the literature. Given this observation, we believe that the proposed structured process is sufficiently extensive enough to capture the essence of the activities that are performed within the legacy to SOA migration domain. It is interesting to observe that most of the literature (50%) from Table I have the *LSU*, *TSU*, *CSI* and *Imp.* as common phases. This indicates that legacy to SOA migration is perceived as a technical endeavor. Out of 17 publications, 15 papers address the implementation aspect of the migration followed by 15 papers addressing the *LSU* phase. Furthermore, much less attention is given to the *MFD* phase, which is a crucial phase for deciding if a migration process is to be performed. The *MFD* phase is an organizational prespective in which the concerned stakeholder decide on whether the migration project is to be executed based on economical, technical and business values. The low frequency of the occurance of the *MFD* phase further confirms the claim that legacy to SOA migration is largely perceived as a technical endeavor [5], [17]. In the

[7]http://stan4j.com/
[8]http://axis.apache.org/axis2/java/core/
[9]http://indus.projects.cis.ksu.edu/

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

TABLE I
ACTIVITY MAPPING BETWEEN THE SELECTED PAPERS AND THE STRUCTURED PROCESS

Paper	Year	Evaluation	LSU	TSU	MFD	CSI	Imp.	D&P
Sneed et al. [54]	2003	Industrial	X	X		X	X	
Zhang & Yang [55]	2004	Industrial	X	X		X	X	X
Jiang & Stroulia [56]	2004	Industrial				X	X	
Chen et al. [21]	2005	Preliminary	X	X		X	X	
Zhang et al. [28]	2005	Industrial	X	X		X	X	X
Sneed [57]	2006	Industrial				X	X	
Canfora [58]	2006	Preliminary	X				X	X
Cetin et al. [59]	2007	Industrial	X	X		X	X	X
Chung et al. [60]	2007	Industrial	X	X			X	
Canfora et al. [61]	2008	Industrial	X			X	X	X
Nakamura et al. [62]	2008	Industrial		X		X	X	
Umar & Zordan [35]	2009	Industrial	X		X			
Chen et al. [46]	2009	Industrial	X	X	X	X	X	
Alahmari et al. [39]	2010	Preliminary	X			X		
Bissyandé et al. [63]	2010	Preliminary	X				X	
Fuhr et al. [40]	2011	Industrial	X			X	X	
Zillmann et al. [30]	2011	Industrial	X	X	X	X	X	X

mapping process, Zillmann et al. [30] is the only paper in Table I that addresses all the phases of our structured process.

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

IV. Analysis and Discussion

We acknowledge the findings and challenges put forward by the following literature [12], [6], [5], [4]. In this section we analyze and discuss the findings of our research. Initially, we present and explain the findings focusing on each of the phase of the structured process. We then discuss the findings of the structured process in terms of research issues.

Table II summarizes the findings of current practices, challenges and possible solutions (future research directions) of each phase of the structured process. The *LSU* phase has a crucial role in the legacy to SOA migration because of the fact that knowledge about the legacy applications is scarce, particularly, documentation and resources are limited. Also, the original vendors of the programming languages or the hardware on which the legacy applications run may not be supported anymore. In such a context, discovering the existing capabilities of the legacy applications is inevitably hard. Due to this, understanding the legacy applications and its existing capabilities is crucial for a legacy to SOA migration. As seen from Table II and Fig. 2, the *LSU* phase leverages reverse engineering techniques to understand the legacy applications and its features and the knowledge residing within the legacy applications. Various reverse engineering techniques are employed to understand legacy applications such as feature location, using software metrics to determine the technical qualities, architecture recovery to extract the high level diagrammatic representation and software visualization to identify dependencies among the legacy applications. The reverse engineering techniques that are presented are not-exhaustive. Equally important as the reverse engineering techniques is the soft knowledge within the original developers, maintainers and users of the legacy applications because the systems are developed and/or maintained by such staff familiar with the existing legacy applications. Over the years, such knowledge and skills become scarce resulting in knowledge erosion due to factors such as ageing and retirements of the technical staff. Despite the wide use of reverse engineering techniques, several challenges still persist in the *LSU* phase. One of the important challenges is the development of generic toolsets for understanding the legacy application as these applications are heterogeneous in terms of programming languages, hardware and operating system on which they run. A viable approach to this challenge is leveraging Model-Driven Engineering (MDE) based techniques as they facilitate computation at a language independent level. Currently, the reverse engineering techniques are semi-automated and require human expertise to complement or correct the extracted information. One of the solutions to this challenge is to integrate the knowledge from human expertise as feedback to improve the reverse engineering process [64]. Finally, prevention of the (soft-)knowledge erosion is also another challenge which can be mitigated by conducting knowledge transfer program within the enterprises.

The *TSU* phase aims at developing a future "to-be" SOA environment not only based on standards and technologies but also considering the non-functional characteristics of the legacy applications. This phase also provides a blueprint for service design [30] that enables the reusability of the existing legacy features as services and orchestration of those services. To develop a future target architecture, various techniques are employed: using specific standards such as messaging and

TABLE II
OVERVIEW OF THE CURRENT PRACTICES, CHALLENGES AND THE POSSIBLE SOLUTIONS

Phases	Current Practices	Challenges	Possible Solutions
Legacy System Understanding	Feature location	Preventing knowledge erosion	Knowledge transfer programs
	Software Metrics	Developing generic tooling for heterogeneous legacy understanding	Model-Driven engineering
	Architecture recovery	Maximizing automation in reverse engineering process	Utilizing the human feedback
	Software visualization		
	Soft knowledge		
Target System Understanding	Specific standards	Identifying optimal business-IT alignment	Componentization
	Specific technology	Maintaining non-functional characteristics	Use of proper standards & technologies
	Functional specification		
Migration Feasibility Determination	Cost-Benefit Analysis	Automating migration feasibility determining toolset	Technical, economical & business value information based toolset
	OAR		
	Code complexity		
	Reusability assessment		
Candidate Service Identification	Modeling legacy process	Identifying functional areas in source code	Feature location
	Information retrieval		Trace visualization
	Concept analysis		Source code search
	Business rule recovery		
	Code visualization		
Implementation	Slicing	Selecting appropriate migration strategies	
	Code extraction	Tooling for developing generic toolset	Model-Driven engineering
	Wrapping		
	Code transformation		
	Refactoring		
	Redevelopment		
	Graph transformation		
Deployment & Provisioning	Discovery	Automated service discovery	Use of semantic markup languages
	Testing	Testing with run-time verification	Techniques to combine testing with run-time verification
	Evolution	Addressing service versioning	Usage of service compatibility
	Publication	Addressing service commonality	Self-adaptive services [51]

communication protocol for SOA based development; specific technology such as SOAP or Rest-based, service discovery mechanism, and functional specification to specify and preserve the existing functional and non-functional characteristics of the legacy applications in the future SOA environment. One of the challenges of the *TSU* is finding an optimal business-IT alignment of the future SOA with the business goals of the enterprise as the *TSU* phase intends to enable service design [17]. A solution to this challenge is to use a componentization process: a process to deconstruct, analyze and identify business component contributing to the business goals of the enterprise [32]. Equally important in the *TSU* phase is to maintain the existing functional and non-functional characteristics of the legacy applications in the future SOA environment. Lewis et al. [3] and Cuadrado et al. [27] argue to use SOA-based standards and technologies as countermeasures to this challenge.

The *MFD* phase involves business and organizational perspectives of the migration by allowing the organization to decide if migration is necessary and feasible. The need of migration is determined based on whether the future SOA environment fulfills or contributes to the business goals expected from the migration and the migration feasibility is determined by assessing the technical characteristics and the cost of the migration. Furthermore, this phase can assist at determining which implementation strategy is to be considered. Some of the widely used techniques in the *MFD* phase are: Cost-Benefit Analysis to determine the economic value of the migration; and, Option Analysis for Re-engineering, reusability assessment and code complexity techniques aim at assisting on determining migration feasibility in terms of technical characteristics such as mining existing components for reusability, calculating legacy code complexity to decide on which implementation strategy to follow. One of the challenges of the *MFD* phase is to automate the migration feasibility with a toolset that allows the stakeholder to provide information about the business value of a legacy component and the tool then extracts information about technical characteristics of the legacy application and the economic feasibility of the migration. Finally, the migration feasibility is determined based on those information. A representative framework is developed by Salama et al. [65] that facilitates the decision making process of selecting an appropriate migration strategy for SOA migration considering the migration feasibility determination.

Identifying service-rich areas in a huge chunk of legacy code has been a challenging task in legacy to SOA migration. Various techniques are currently used such as modeling business process and mapping these to the features within legacy code, information retrieval, concept analysis, business rule recovery, code visualization and so on. Despite the availability of many techniques still the candidate service identification remains a challenge. A potential research area to assist on locating candidate services can be feature location, source code searching and trace visualization techniques. Feature location and trace visualization have been already used in legacy to SOA migration. Source code searching [66], a technique to search for relevant code within source code, can be an interesting area to investigate.

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

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

Based on the findings of the Evaluation (Sec. III), the role of each phase and the structured process itself is applicable in any legacy to SOA migration method. However, there are some challenges remaining within the structured process. One of the challenges is the automation of the structured process with a toolset in which the tools and techniques for each phase can be suitably integrated. Such an integrated toolset can include tools and techniques ranging from reverse engineering tools, fact extractors, transformation tools, and code generators tools as per requirement. The need for such automation through the development of tools and techniques to assist phases of legacy to SOA migration is advocated by various researchers (e.g., Lewis et al. [12], Nasr et al. [5]). Additionally, legacy to SOA migration is not only about a successful technical transformation from a legacy application to a SOA, but also to determine whether an enterprise benefits from the claimed benefits of SOA and achieves its business goals. Currently, few case studies are reported in literature about the post migration experiences. Hence, we urge that more case studies in collaboration with industries are conducted and that they report on these experiences.

V. Conclusion

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

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

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

References

[1] K. Bennett, "Legacy systems: Coping with success," *IEEE Soft.*, vol. 12, no. 1, pp. 19–23, 1995.

[2] J. Bisbal, D. Lawless, B. Wu, and J. Grimson, "Legacy information systems: Issues and directions," *IEEE Soft.*, vol. 16, no. 5, pp. 103–111, 1999.

[3] G. Lewis, E. Morris, L. O'Brien, D. Smith, and L. Wrage, "SMART: The service-oriented migration and reuse technique," CMU/SEI, Tech. Rep. CMU/SEI-2005-TN-029, Sept 2005.

[4] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-oriented computing: State of the art and research challenges," *Computer*, vol. 40, no. 11, pp. 38–45, 2007.

[5] K. A. Nasr, H.-G. Gross, and A. van Deursen, "Realizing service migration in industry: lessons learned," *JSME*, 2011.

[6] K. Kontogiannis, G. Lewis, and D. Smith, "A research agenda for service-oriented architecture," in *SDSOA'08*. ACM, 2008, pp. 1–6.

[7] R. Khadka, A. Saeidi, A. Idu, J. Hage, and S. Jansen, "Legacy to SOA evolution- a systematic literature review," in *Migrating Legacy Applications: Challenges in Service Oriented Architecture and Cloud Computing Environments*, A. D. Ionita, M. Litoiu, and G. Lewis, Eds. IGI Global, 2012, pp. 40–71.

[8] M. Razavian and P. Lago, "A frame of reference for SOA migration," in *Towards a Service-Based Internet*. Springer, 2010, pp. 150–162.

[9] M. Razavian and P. Lago, "A survey of SOA migration in industry," in *Service-Oriented Computing*. Springer, 2011, pp. 618–626.

[10] M. Razavian and P. Lago, "A lean and mean strategy for migration to services," in *WICSA/ECSA'12*. ACM, 2012, pp. 61–68.

[11] A. De Lucia, R. Francese, G. Scanniello, and G. Tortora, "Developing legacy system migration methods and tools for technology transfer," *SPE*, vol. 38, no. 13, pp. 1333–1364, 2008.

[12] G. Lewis, D. Smith, N. Chapin, and K. Kontogiannis, "MESOA'09," SEI, Tech. Rep. 1424448972, 2009.

[13] G. Lewis and D. Smith, "Service-oriented architecture and its implications for software maintenance and evolution," in *FoSM'08*. IEEE, 2008, pp. 1–10.

[14] G. Canfora and M. Di Penta, "New frontiers of reverse engineering," in *FSE*. IEEE, 2007, pp. 326–341.

[15] R. C. Seacord, D. Plakosh, and G. A. Lewis, *Modernizing legacy systems: software technologies, engineering processes, and business practices*. Addison-Wesley Professional, 2003.

[16] L. O'Brien, D. Smith, and G. Lewis, "Supporting migration to services using software architecture reconstruction," in *STeP'05*. IEEE, 2005, pp. 81–91.

[17] S. Murer, B. Bonati, and F. J. Furrer, *Managed Evolution*. Springer, 2011.

[18] R. Khadka, G. Reijnders, A. Saeidi, S. Jansen, and J. Hage, "A method engineering based legacy to SOA migration method," in *ICSM'11*. IEEE, 2011, pp. 163–172.

[19] G. Lewis, E. Morris, and D. Smith, "Analyzing the reuse potential of migrating legacy components to a service-oriented architecture," in *CSMR'06*. IEEE, pp. 9–18.

[20] D. Binkley, "Source code analysis: A road map," in *ICSE'00*. IEEE, 2007, pp. 104–119.

[21] F. Chen, S. Li, H. Yang, C.-H. Wang, and W. Cheng-Chung Chu, "Feature analysis for service-oriented reengineering," in *APSEC'05*. IEEE, 2005, pp. 8–pp.

[22] R. Millham, "Migration of a legacy procedural system to service-oriented computing using feature analysis," in *CISIS'10*. IEEE, 2010, pp. 538–543.

[23] P. Vemuri, "Modernizing a legacy system to SOA-feature analysis approach," in *TENCON'08*. IEEE, 2008, pp. 1–6.

[24] H. Sneed, "A pilot project for migrating COBOL code to web services," *STTT*, vol. 11, no. 6, pp. 441–451, 2009.

[25] H. Sneed, "COB2WEB a toolset for migrating to web services," in *WSE'08*. IEEE, 2008, pp. 19–25.

[26] M. Perepletchikov, C. Ryan, K. Frampton, and Z. Tari, "Coupling metrics for predicting maintainability in service-oriented designs," in *ASWEC'07*. IEEE, 2007, pp. 329–340.

[27] F. Cuadrado, B. García, J. Dueas, and H. Parada, "A case study on software evolution towards service-oriented architecture," in *AINAW'08*. IEEE, 2008, pp. 1399–1404.

[28] Z. Zhang, R. Liu, and H. Yang, "Service identification and packaging in service oriented reengineering," in *SEKE'05*, 2005, pp. 219–26.

[29] J. Van Geet and S. Demeyer, "Lightweight visualisations of COBOL code for supporting migration to SOA," in *Soft Evol'07*, 2007.

[30] C. Zillmann, A. Winter, A. Herget, W. Teppe, M. Theurer, A. Fuhr, T. Horn, V. Riediger, U. Erdmenger, U. Kaiser *et al.*, "The SOAMIG Process Model in Industrial Applications," in *CMSR'11*. IEEE, 2011, pp. 339–342.

[31] A. Jansen and J. Bosch, "Software architecture as a set of architectural design decisions," in *WICSA'05*. IEEE, 2005, pp. 109–120.

[32] L. Cherbakov, G. Galambos, R. Harishankar, S. Kalyana, and G. Rackham, "Impact of service orientation at the business level," *IBM Sys. J.*, vol. 44, no. 4, pp. 653–668, 2005.

[33] R. N. Charette, "Why software fails [software failure]," *Spectrum, IEEE*, vol. 42, no. 9, pp. 42–49, 2005.

[34] H. Sneed, "Planning the reengineering of legacy systems," *IEEE Soft.*, vol. 12, no. 1, pp. 24–34, 1995.

[35] A. Umar and A. Zordan, "Reengineering for service oriented architectures: A strategic decision model for integration versus migration," *JSS*, vol. 82, no. 3, pp. 448–462, 2009.

[36] R. Heckel, R. Correia, C. Matos, M. El-Ramly, G. Koutsoukos, and L. Andrade, "Architectural transformations: From legacy to three-tier and services," in *Soft. Evol.* Springer, 2008, pp. 139–170.

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

[38] A. Arsanjani, S. Ghosh, A. Allam, T. Abdollah, S. Ganapathy, and K. Holley, "SOMA: A method for developing service-oriented solutions," *IBM Sys. J.*, vol. 47, no. 3, pp. 377–396, 2008.

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

[40] A. Fuhr, T. Horn, V. Riediger, and A. Winter, "Model-driven software migration into service-oriented architectures," *CSRD*, vol. 28, no. 1, pp. 65–84, 2011.

[41] A. Marchetto and F. Ricca, "Transforming a java application in an equivalent web-services based application: toward a tool supported stepwise approach," in *WSE'08*. IEEE, 2008, pp. 27–36.

[42] L. Aversano, L. Cerulo, and C. Palumbo, "Mining candidate web services from legacy code," in *WSE'08*. IEEE, 2008, pp. 37–40.

[43] Z. Zhang, H. Yang, and W. Chu, "Extracting reusable object-oriented legacy code segments with combined formal concept analysis and slicing techniques for service integration," in *QSIC'06*. IEEE, 2006, pp. 385–392.

[44] K. Kontogiannis, G. A. Lewis, D. B. Smith, M. Litoiu, H. Muller, S. Schuster, and E. Stroulia, "The landscape of service-oriented systems: A research perspective," in *SDSOA'07*. IEEE, 2007, pp. 1–1.

[45] A. Almonaies, J. Cordy, and T. Dean, "Legacy system evolution towards Service-Oriented Architecture," in *SOAME'10*. IEEE, 2010, pp. 53–62.

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

[47] R. Khadka, A. Saeidi, R. Jansen, J. Hage, and R. Helms, "An evaluation of service frameworks for the management of service ecosystems," in *PACIS'11*. AIS, 2011, p. 10.

[48] M. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-oriented computing: a research roadmap," *IJCIS*, vol. 17, no. 2, pp. 223–255, 2008.

[49] M. Rambold, H. Kasinger, F. Lautenbacher, and B. Bauer, "Towards autonomic service discovery: a survey and comparison," in *SCC'09*. IEEE, 2009, pp. 192–201.

[50] G. Canfora and M. Di Penta, "Service-oriented architectures testing: A survey," in *Software Engineering*. Springer, 2009, pp. 78–105.

[51] M. P. Papazoglou, "The challenges of service evolution," in *AISE*. Springer, 2008, pp. 1–15.

[52] V. Andrikopoulos, S. Benbernou, and M. P. Papazoglou, "Managing the evolution of service specifications," in *AISE*. Springer, 2008, pp. 359–374.

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

[54] H. M. Sneed and S. H. Sneed, "Creating web services from legacy host programs," in *WSE'03*. IEEE, 2003, pp. 59–65.

[55] Z. Zhang and H. Yang, "Incubating services in legacy systems for architectural migration," in *APSEC'04*. IEEE, 2004, pp. 196–203.

[56] Y. Jiang and E. Stroulia, "Towards reengineering web sites to web-services providers," in *CSMR'04*. IEEE, 2004, pp. 296–305.

[57] H. M. Sneed, "Integrating legacy software into a service oriented architecture," in *CSMR'06*. IEEE, 2006, pp. 3–14.

[58] G. Canfora, A. R. Fasolino, G. Frattolillo, and P. Tramontana, "Migrating interactive legacy systems to web services," in *CSMR'06*. IEEE, 2006, pp. 24–36.

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

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

[61] G. Canfora, A. R. Fasolino, G. Frattolillo, and P. Tramontana, "A wrapping approach for migrating legacy system interactive functionalities to service oriented architectures," *JSS*, vol. 81, no. 4, pp. 463–480, 2008.

[62] M. Nakamura, A. Tanaka, H. Igaki, H. Tamada, and K.-i. Matsumoto, "Constructing home network systems and integrated services using legacy home appliances and web services," *IJWSR*, vol. 5, no. 1, pp. 82–98, 2008.

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

[64] G. Canfora, M. Di Penta, and L. Cerulo, "Achievements and challenges in software reverse engineering," *CACM*, vol. 54, no. 4, pp. 142–151, 2011.

[65] R. Salama and S. G. Aly, "A decision making tool for the selection of service oriented-based legacy systems modernization strategies," in *ICSERP'08*, 2008.

[66] S. Bajracharya, T. Ngo, E. Linstead, Y. Dou, P. Rigor, P. Baldi, and C. Lopes, "Sourcerer: a search engine for open source code supporting structure-based search," in *OOPSLA'06*. ACM, 2006, pp. 681–682.