Service Identification Based on Quality Metrics

Object-Oriented Legacy System Migration Towards SOA

Seza Adjoyan*, Abdelhak- Djamel Seriai*, Anas Shatnawi*
* LIRMM, CNRS and University of Montpellier 2
161 rue Ada, Montpellier, France
{adjoyan, seriai, shatnawi}@lirmm.fr

***Abstract*—Migrating towards Service Oriented Architecture SOA has become a major topic of interest during the recent years. Since emerging service technologies have forced non-service based software systems to become legacy, many efforts and researches have been carried out to enable these legacy systems to survive. In this context, several service identification solutions have been proposed. These approaches are either manual, thus considered expensive, or rely on ad-hoc criteria that fail to identify relevant services. In this paper, within the big picture of migrating object-oriented legacy systems towards SOA, we address automatic service identification from source code based on service quality characteristics. To achieve this, we propose a quality measurement model where characteristics of services are refined to measurable metrics.**

Keywords- SOA; reengineering; migration; reverse engineering; Object-Oriented; service identification; quality; software reuse; legacy system.

I. INTRODUCTION

Service Oriented Architecture SOA, whose main bricks are services [7], has become a trend [3, 5] of computing paradigm to describe business functionalities and application logics. In SOA, a system is structured into a set of loosely coupled [6, 13, 20] and interoperable business services that can be easily composed [7], reused [7] and shared [8] regardless of their physical location. Services could either be all implemented on a single machine, residing on several machines of company's internal network, or even distributed on several systems over internet [2]. Moreover, having solid service oriented architecture in place will provide the infrastructure needed to successfully deploy services in cloud environment.

The evolution of service technologies in recent years has led non-service based software systems to become legacy software [3, 5]. Any software which has been developed using outdated technology [1], but still brings great value to the organization that uses it, is considered as a legacy software [18, 1]. In order to follow new technological advances and yet to conserve existing business value of current systems, a migration, which is considered as a variation of wrapping methodology [18], of legacy system should be carried out. Several approaches for legacy system migration towards SOA have been reported in literature [1, 2, 3, 4, 7, 9, 10, 14, 18, 19, 21]. SOA migration is achieved through two major phases: (1) legacy analysis, where available software artifacts are analyzed to identify provided services and (2) service implementation, that leverages extracted legacy code as usable services, wraps them by interfaces and orchestrates their operations. The first phase (i.e. service identification) is crucial in this process, especially with the unavailability of certain resources (e.g. developers, architects) and poor documentation [4, 7]. Even more, it is a challenging task, since legacy systems are not necessarily built with the vision of service. Therefore many approaches have been proposed to identify services by analyzing legacy software artifacts. The majority of them are carried out manually [1, 7, 9]. These solutions are considered as expensive in terms of expertise. Thus, some automatic or quasi-automatic approaches were proposed [3, 5, 6, 19, 21]. Most of these approaches assume the existence of large range of information about legacy systems such as their documentation, architecture and design documents [7, 21]. Therefore they are specific to systems where such information is available. They cannot be applied to a large number of systems where only the source code is available [13]. In addition, these approaches rely on ad-hoc criteria for evaluating candidate services, hence a gap between identified services and expected ones.

Our contribution in this paper is to automatically identify services from object-oriented source code. Unlike existing approaches, our service identification process is based on a quality function that measures the semantic correctness of identified services. We introduce a semantic correctness model in order to refine well-known service characteristics to measurable metrics.

The rest of the paper is structured as follows: In section 2, we outline the related works for service identification within migration towards SOA approaches. In section 3, we present our approach of service identification from object-oriented source code by defining quality metrics to evaluate services. In section 4, we evaluate our approach on two case studies. Finally, section 5 concludes the paper and provides some future directions.

II. RELATED WORK

Most of the approaches proposing migration of legacy systems to service-based ones offer only guidelines to identify services [4, 9, 10, 14]. Few of them propose technical steps. In [7], authors present a migration approach called Service-Oriented Migration and Reuse Technique (SMART). It defines five steps to achieve the migration of legacy system towards SOA. However, the proposed approach requires several sources of information (e.g. documentation) to support the analysis of the legacy system. Besides, the approach largely relies on human interaction (e.g. system analyst, maintenance programmer, etc.) that gathers information through

interviewing stakeholders in order to fill the gap between existing legacy system and target architecture. In [5], an architecture-based and requirement-driven service-oriented reengineering method is discussed. Services are identified by domain analysis and business function identification. The approach is based on both requirements abstraction and source code levels. This approach needs architectural and requirement information to be avalaible. [1] proposes an automatic approach to evaluate candidate services. Candidate services are considered as groups of object-oriented classes evaluated in terms of development, maintenance and estimated replacement costs. Other approaches propose to evaluate services either by code pattern matching and graph transformation [19], feature location [3] or formal concept analysis [6]. A detailed survey of all service identification methods is discussed in [11].

Services and software components have several characteristics in common, in particular, those related to their quality, nature, structure and behavior. For that obvious reason, component identification techniques from object oriented legacy system could be considered as related to this paper. One of the previous works in our team identifies components from object-oriented source code based on quality-centric metrics [22].

As to SOA quality metrics, diverse studies have been proposed in literature for measuring qualitative properties of SOA systems. Most of these works either assess systems that are already service based or evaluate systems only after their implementation. Unfortunately, such researches are not adapted to the context of reengineering an object-oriented system towards service oriented system. [23] proposes a framework to measure the degree of service orientation in SOA systems. It focuses on the internal SOA attribute, decomposes selected attribute to a set of factors and maps each factor to a set of measurable criteria. Each criterion is typically evaluated by a set of software metrics, though no dedicated metrics are defined for each criterion in the paper.

III. PROPOSED APPROACH

We propose a migration technique that identifies services as groups of classes in the legacy software source code. We base our legacy system analysis on the source code, since it is the only resource that is always available, while other resources such as documentation or architecture could often be missing. Unlike other approaches that identify candidate services in source code manually, we propose an automatic identification method of candidate services. Our approach is based on the definition of a fitness function that measures semantic correctness of each group of source code elements to be considered as a service.

A. Object-to-service mapping model

In order to be capable to identify services from object-oriented source code, we define a mapping between object oriented and SOA concepts (see Figure 1). We consider a service as a group of classes defined in object-oriented source code. Among these classes, some define the operations provided by the service, whereas others are inner classes. Inner classes are those which only have internal connections to other classes of the same service. Classes that define the operations

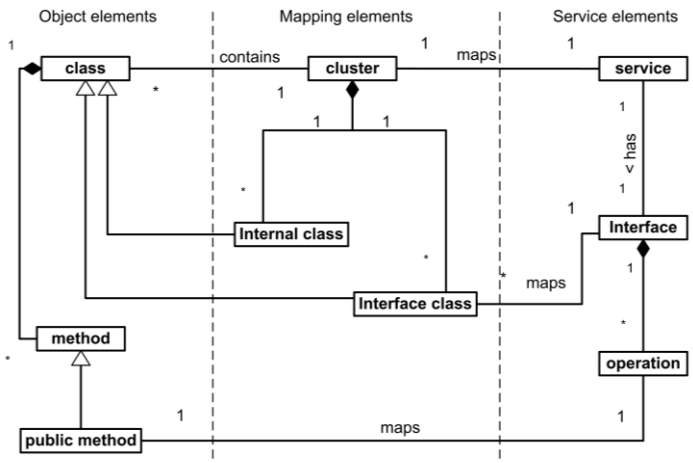

Figure 1. Object-service mapping model.

provided by the service are the classes that define its interface. Inner classes do not define operations provided by the service. Operations provided by the service are class's public methods.

B. Quality Measurement Model of Services

As we have mentioned earlier, a service is identified from a group of object-oriented classes. Initially, each group of classes is considered as a candidate service. A qualified service is selected from candidate ones based on a function that measures its quality. Similar to the standard for the evaluation of software quality ISO/IEC 25010:2011 [12], we define this quality function of services based on a set of characteristics that are mapped to a set of properties. Each property is later measured using a set of metrics.

1) Characteristics of Services

We deduct the quality characteristics of services based on the analysis of the most commonly used definitions of services in literature.

In literature, there are several definitions of services [2, 5, 13]. According to [5], a service is an abstract resource that performs a coherent and functional task. [13] considers a service as a process that has an open interface, self-containedness and coarse granularity. It can be easily composed and decomposed to implement various business workflows. [2] defines the service in terms of its characteristics: A service is a coarse-grained and discoverable software entity that interacts with applications and other services through a loosely coupled, often asynchronous, message-based communication model. Coarse-grained means that services implement more than one functionality and operate on larger data sets. Discoverable means that services can be found at both design time and run time, not only by unique identity but also by interface identity and by service kind. Self-contained refers to the self-sufficiency a service has, where context or state information is not required from other services. For loosely coupled, services are connected to other services and clients using standard, dependency-reducing, decoupled message-based methods such as XML document exchanges.

TABLE I. CHARACTERISTICS OF SERVICES

Characteristic	Type	
	Structural and Behavioral	***SOA platform***
coarse-grained = functionality	✓	
discoverable		✓
self-contained = loosely-coupled	✓	
dynamic-binding		✓
composable	✓	
message-based		✓
asynchronous		✓

Table I lists the characteristics of services as mentioned in the definitions above. We have categorized them into two categories: those related to the structure and behavior of services and others related to the SOA platform. In order to measure the semantic correctness of candidate services, we select from the aforementioned characteristics the ones that define service structure and behavior: self-containment, composability and coarse-grained (functionality).

2) Refinement of Service Characteristics

The former selected characteristics are refined to measurable quality properties.

- A service can be completely self-contained if it does not require any interface, i.e. it can be deployed as a single unit without depending on other services [13]. Thus, the property number of interfaces the service requires gives us a good indication on the self-containment of the service.

  The higher the number of required interfaces is, the less the service is self-contained.

- A service is subject to composition with other services. This composition is realized without internal modifications but through service interface. A decomposition of the legacy system will be effective with the principle of composing those services with high cohesion and loose coupling, i.e. two services are composed with each other if their interfaces are cohesive. Thus, the average of services' cohesion within an interface gives us a good indication on the composability of the service.

- A service is more likely to be coarse-grained and hence represent complex, rich and high-level business functionality. However, it may sometimes be fine-grained and hence represent low-level primitive functionality [14]. Choosing the right level of granularity is the key for a successful service reuse. The bigger the service grains are, the less the service becomes reusable. It is relatively difficult to determine from source code the exact number of functionalities that the service provides. However, several factors can help measuring the functionality of a service. (1) A service that provides several interfaces may provide numerous functionalities, thus the higher the number of interfaces is, the more the service provides functionalities. (2) An interface whose services are highly cohesive probably provide single functionality. (3) A group of interfaces with high cohesion are most favorable to provide single or limited number of functionalities. (4) When the extracted code of candidate service is highly coupled, this means that the service probably provides very few or single functionality. (5) When the extracted code of candidate service is highly cohesive, this means that the service probably provides very few or single functionality. Thus, we suggest binding the functionality characteristic to properties as indicated in Table II.

TABLE II. BINDING FUNCTIONALITY CHARACTERISTIC TO PROPERTIES

Functionality Characteristic	Property
A service that provides several interfaces may provide numerous functionalities, thus the higher the number of interfaces is, the more the service provides functionality.	Number of provided interfaces
An interface whose services are highly cohesive probably provide single functionality.	Average of service's interface cohesion within the interface
A group of interfaces with high cohesion are most favorable to provide single or limited number of functionality.	Cohesion between interfaces
When the extracted code of candidate service is highly coupled, this means that the service probably provides very few or single functionality.	Coupling inside a service
When the extracted code of candidate service is highly cohesive, this means that the service probably provides very few or single functionality.	Cohesion inside a service

3) The Quality Metrics

In our approach, according to the characteristics and properties of services we have chosen above, we build our quality metrics to evaluate the quality of candidate services. This quality will be the factor in distinguishing the extracted candidate services. The property functionality requires coupling and cohesion measurements, while composability only requires a cohesion measurement (see Figure 2). As to [15], cohesion of a service measures how strong the elements within this service are related to each other. A service is considered as highly cohesive, if it performs a set of closely related functions and cannot be split into finer elements. The metric LCC Loose Class Cohesion proposed by [16] measures the overall connectedness of the class. It is calculated by:

$$LCC = \frac{number\ of\ direct\ and\ indirect\ connections}{maximum\ number\ of\ possible\ connections}$$

Coupling means the degree of direct and indirect dependence of a class on other classes in the system. Here, two measures are counted: method calls and parameter use, i.e. two classes are considered coupled to each other if the methods of one class use the methods or attributes of the other class. In our approach, $Coupl(E)$ measures the internal coupling of the

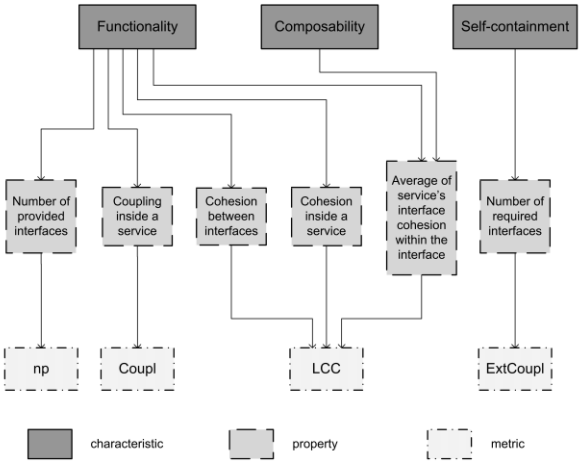

Figure 2. Refinement model of service characteristics.

candidate service E and is calculated by the ratio between number of classes inside the service that are internally called to the total number of classes within the candidate service E. $ExtCoupl(E)$ measures the coupling of the candidate service E with other services. It is calculated as $ExtCoupl(E) = 1 - Coupl(E)$.

4) Fitness Function Definition

We define a fitness function $FF(E)$ for an identified candidate service E as a linear combination between the 3 characteristics of services previously defined, $F(E)$ for functionality, $C(E)$ for composability and $S(E)$ for self-containment as follows:

$$FF(E) = \frac{\alpha F(E) + \beta C(E) + \gamma S(E)}{n}$$

Where α, β, γ are coefficient weights for each characteristic that are determined by software architect and $n = \sum(\alpha, \beta, \gamma)$.

The characteristics functionality $F(E)$, composability $C(E)$ and self-containment $S(E)$ are measured according to their definition as follows:

$$F(E) = \frac{1}{5}\left(np(E) + \frac{1}{I}\sum_{i \in I} LCC(i) + LCC(I) + Coupl(E) + LCC(E)\right)$$

Where $np(E)$ refers to number of provided interfaces, $LCC(i)$ refers to the average of service's interface cohesion within the interface, $LCC(I)$ refers to the cohesion between interfaces, $Coupl(E)$ refers to the coupling inside a service, and $LCC(E)$ refers to the cohesion inside a service.

$$C(E) = \frac{1}{I}\sum_{i \in I} LCC(i); \text{ where } i \text{ refers to interface}$$

$$S(E) = ExtCoupl(E)$$

PSEUDO CODE 1: AGGLOMERATIVE HIERARCHICAL CLUSTERING

Input: OO source code classes;
Output: A hierarchy of clusters (dendrogram);

```
1: let each class be a cluster;
2: compute fitness function of pair classes;
3: repeat
4:     merge two "closest" clusters based fitness function value;
5:     update list of clusters;
6: until only one cluster remains
7: return dendrogram
```

C. Clustering Process

In order to recover services from OO legacy code, we group classes based on their dependencies. For that purpose, we propose a hierarchical agglomerative clustering algorithm. This algorithm groups together the classes with the maximized value of the fitness function. At the outset, every class is considered as a single cluster. Next, we measure the fitness function between all pairs of clusters. The algorithm merges the pair of clusters with the highest fitness function value into a new cluster. Then, we measure the fitness function between the new formed cluster and all other clusters and successively merge the pair with the highest fitness function value. These steps are repeated as long as the number of clusters is bigger than one, as illustrated in Pseudo code 1. As a result, the legacy system is expressed in hierarchical view presented in a dendrogram, as illustrated in Figure 3.

To obtain a partition of disjoint clusters, the resulting hierarchy needs to be cut at some point. To determine the best cutting point we employ the standard depth first search (DFS) algorithm. Initially on the root node, we compare the similarity of the current node to the similarity of it child nodes. If the current node's similarity value exceeds the average of similarity value of its children, then the current node is a cutting point, otherwise, the algorithm continues recursively through its children.

By applying the aforementioned clustering algorithm, we evaluate the legacy system and represent its classes in coarse-grained and loose-coupled disjoint set of services. An example of partitioning legacy system's classes to services is illustrated in Figure 3.

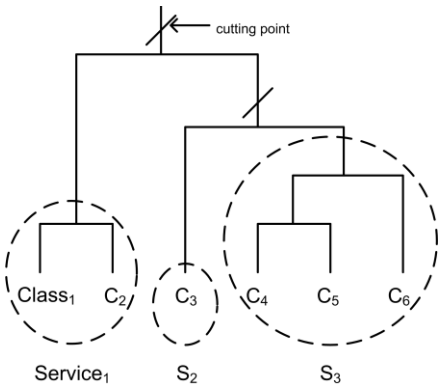

Figure 3. Dendrogram with set of services.

IV. EVALUATION

The proposed approach has been evaluated on two realistic case studies: Java Calculator Suite[1] which is a small system with 17 classes and MobileMedia[2] which is a medium sized system with 51 classes. Table III gives the number of classes and LOC (Line of Code) of these two case studies.

TABLE III. CASE STUDIES INFORMATION

Case study	Number of classes	LOC
Java Calculator Suite	17	2360
MobileMedia	51	3016

Java Calculator Suite is an open-source calculator implemented in Java. It performs basic mathematical operations, has a graphic interface and supports Booleans, large numbers, machine numbers, and about 25 different operations. MobileMedia is an open-source Java application used for managing media (photo, music, and video) on mobile devices.

A. Service Identification

1) Results

In this phase, we partition the source code of each case study into a set of clusters. Each cluster is composed of one or more classes. Each resulting cluster corresponds to one service. Table IV shows the results in terms of number of obtained services for each case study and the corresponding average service quality value for each of the three characteristics: functionality, composability and self-containment. The distribution rate of classes to services is 17/7= 2.4 classes per service for Java Calculator Suite and 3.9 for MobileMedia. Even more, we notice that almost all classes of same service are grouped to offer single functionality. For example, "Entries", "GuiCommandLine" and "ResultList" handle I/O issues.

TABLE IV. SERVICE IDENTIFICATION RESULTS

Case study	Number of services	Functionality	Composability	Self-containment
Java Calculator Suite	7	0.73	0.88	0.41
MobileMedia	13	0.60	0.79	0.59

2) Validation and Discussion

We validate the consistency of our proposed approach either by comparing resulting services with the known architectural design or by analyzing the relevance of the identified services.

In Java Calculator Suite, we notice that lexically relevant classes were grouped in one cluster. We document the resulting components by assigning a name based on the most frequent tokens in their classes' names. In Table V, we display service identification results in terms of clusters' names and their composing classes.

TABLE V. SERVICE IDENTIFICATION RESULTS

Cluster Number	Cluster Name	Composing Classes
1	Calc Machine Number	CalcMachineNumber
2	Operator Center Control	OperatorControlCenter
3	Calculator jcalc_applet Gui Results jcalc Entries Command	Calculator CalculatorException, CalculatorTester Jcalc jcalc_applet
4	E variable_interface jcalc_math jcalc_trig	E jcalc_math jcalc_trig variable_interface
5	Variable operator Checker Table	VariableTable, operatorChecker
6	PI	PI
7	Gui Line Results Entries Command List	Entries GuiCommandLine ResultsList

MobileMedia has a known architecture model. In [17], the authors presented aspect oriented architecture for MobileMedia. We manually compare our extracted services with the modules of this design, after excluding aspect modules. We have found out that some services were directly mapped to one corresponding module in the architecture, such as the service that includes two classes "MediaListScreen" and "MediaData" was mapped to the module named "MediaListScreen". In total, 5 services were successfully mapped to 5 modules. Some other extracted services could be mapped to more than one module. This category can be divided to two types. The first type is one module with closely related functionalities such as the service named "Video Media Util Screen Play Capture Music" was mapped to three modules "PlayMediaScreen", "VideoAccessor" and "VideoAlbumData". These three modules are in fact functionally related and the resulted service was more coarse-grained than the architecture design. The second type is modules that are weakly related. For this case, we have found two services that each of them was mapped to respectively 3 and 4 modules of the architecture. Some services that are functionally closely related (in our case study, 4 services related to the functionality of transferring media via SMS) were mapped to many modules of the architecture (in our case study, 2 modules related to media transfer functionality). These extracted services were finer-grained than their corresponding modules. Finally, one service that groups exception classes is missing from the architectural design since in the architecture, non-functional modules are not represented.

The results show that 77% (10/13) of extracted services were successfully mapped in the architectural design.

B. Example of Service Deployment

We deployed identified services as Web services using Apache Axis2 on Apache Tomcat Web server and then wrapped these Web services by generating their WSDL interfaces. For example, in MobileMedia case study, the configuration file services.xml (Figure. 4) describes the "mapping" between Web service "Video Media Util Screen

[1] http://sourceforge.net/projects/bfegler/

[2] http://homepages.dcc.ufmg.br/~figueiredo/spl/icse08/

Play Capture Music" and the Java classes composing this service.

```
<service name="VideoMediaUtilScreenPlayCaptureMusic"
scope="application">
    <description>
        Video Media Util Screen Play Capture Music
    </description>
    <messageReceivers>
        <messageReceiver
            mep="http://www.w3.org/2004/08/wsdl/in-only"
class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiv
er"/>
        <messageReceiver
            mep="http://www.w3.org/2004/08/wsdl/in-out"
 class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
    </messageReceivers>
    <parameter name="ServiceClass">
        sample.pojo.service.VideoMediaUtilScreenPlayCapture
Music
    </parameter>
</service>
```

Figure 4. Services.xml file.

V. CONCLUSION

The main contribution of the work presented in this paper is the extraction of services from legacy source code based on service quality characteristics. For this purpose, we first set a mapping model between object and service concepts. Then, unlike most ad-hoc identification approaches, we introduced a fitness function that measures the quality of identified services. The measurement metrics of fitness function are based on a refinement model of service's semantic characteristics. It is worthy to note that this approach is especially applicable to modernize legacy systems for which no software assets but the source code is available. Finally, to demonstrate the applicability of our proposed approach, we have applied it on two Java OO applications and obtained satisfying results. As a part of future work, we plan to apply our proposed on more complex case studies.

REFERENCES

[1] Sneed, H.M., "Integrating legacy software into a service oriented architecture," *Proceedings of the 10th European Conference on Software Maintenance and Reengineering, 2006. CSMR 2006.* pp.11 pp.,14, 22-24 March 2006.

[2] Brown, A; Johnston, S.; & Kelly, K. "Using Service-Oriented Architecture and Component-Based Development to Build Web Service Applications". Santa Clara, CA: Rational Software Corporation, 2002.

[3] Feng Chen; Shaoyun Li; Yang, H.; Ching-Huey Wang; Chu, W.C.-C., "Feature analysis for service-oriented reengineering," *12th Asia-Pacific Software Engineering Conference, 2005. APSEC '05*, pp. 201-208, 15-17 Dec. 2005.

[4] Khadka, R., Saeidi, A., Jansen, S., Hage, J, "A structured legacy to SOA migration process and its evaluation in practice", Proceedings of the *7th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA 2013)*.

[5] Zhang, Z., Liu, R., Yang, H., "Service identification and packaging in service oriented reengineering". *Proceedings of the Seventeenth International Conference on Software Engineering and Knowledge Engineering SEKE'2005*, Taipei, Taiwan, Republic of China, July 14-16, 2005.

[6] Feng Chen; Zhuopeng Zhang; Jianzhi Li; Jian Kang; Yang, H., "Service Identification via Ontology Mapping," *33rd Annual IEEE International Computer Software and Applications Conference COMPSAC '09*, vol.1, pp.486,491, 20-24 July 2009.

[7] G. Lewis, E. Morris, L. O'Brien, D. Smith, L. Wrage, "Smart: The service-oriented migration and reuse technique," CMU/SEI, Tech. Rep. CMU/SEI-2005-TN-029, Sept 2005, Available from: http://www.sei.cmu.edu/reports/05tn029.pdf

[8] Oracle Corporation, 2008. Business process management, service-oriented architecture, and web 2.0: Business transformation or train wreck? Oracle Corporation, White Paper, available online from http://viewer.media.bitpipe.com /934318651-120/1252521184-411/SOA-US-EN-WP-BPMSOA2.0.pdf.

[9] Khadka, R.; Reijnders, G.; Saeidi, A.; Jansen, S.; Hage, J., "A method engineering based legacy to SOA migration method," 2011 27th IEEE International Conference on Software Maintenance (ICSM), pp.163,172, 25-30 Sept. 2011 doi: 10.1109/ ICSM.2011.6080783

[10] Cetin, S.; Ilker Altintas, N.; Oguztuzun, H.; Dogru, A.H.; Tufekci, O.; Suloglu, S., "Legacy Migration to Service-Oriented Computing with Mashups," *International Conference on Software Engineering Advances ICSEA 2007*, pp.21, 25-31 Aug. 2007.

[11] R. Khadka, A. Saeidi, A. Idu, J. Hage, 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.

[12] ISO/IEC 25010:2011, Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models, 2011.

[13] Nakamura, M.; Igaki, H.; Kimura, T.; Matsumoto, K.-i., "Extracting service candidates from procedural programs based on process dependency analysis," *IEEE Asia-Pacific Services Computing Conference, APSCC 2009*, pp.484,491, 7-11 Dec. 2009.

[14] Channabasavaiah, K., Holley, K., Edward M. Tuggle, Jr., "Migrating to a service-oriented architecture", White Paper, IBM Corporation - Software Group, April 2004.

[15] Patidar, M. K., Gupta, R., & Chandel, G. S., "Coupling and Cohesion Measures in Object Oriented Programming". International Journal of Advanced Research in Computer Science and Software Engineering, Volume 3, Issue 3, March 2013, ISSN: 2277 128X.

[16] Bieman, James M.; Kang, Byung-Kyoo: Cohesion and Reuse in an Object-Oriented System. In *Proc. Int'l Symp. Software Reusability*, 1995, S. 259-262.

[17] E. Figueiredo et al., "Evolving software product lines with aspects: an empirical study on design stability", *Proc. of ICSE*, pp. 261-270, 2008.

[18] Stehle, E., Piles, B., Max-Sohmer, J., & Lynch, K. "Migration of Legacy Software to Service Oriented Architecture" *Department of Computer Science Drexel University Philadelphia, PA* 19104 (2008): 2-5.

[19] Matos, Carlos M. P. ; Heckel, Reiko, "Migrating Legacy Systems to Service-Oriented Architectures" In *ECEASST'16* .2008.

[20] Mike P. Papazoglou ; Willem-Jan Heuvel, "Service oriented architectures: approaches, technologies and research issues". *The VLDB Journal*, vol.16, n.3, July 2007, pp. 389-415.

[21] O'Brien, L.; Smith, D.; Lewis, G., "Supporting Migration to Services using Software Architecture Reconstruction," *13th IEEE International Workshop on Software Technology and Engineering Practice*, 2005, pp.81,91.

[22] Kebir, S.; Seriai, A.-D.; Chardigny, S.; Chaoui, A., "Quality-Centric Approach for Software Component Identification from Object-Oriented Code," *Joint Working IEEE/IFIP Conference on Software Architecture (WICSA) and European Conference on Software Architecture (ECSA)*, pp.181,190, 20-24 Aug. 2012

[23] Aldris, A.; Nugroho, A.; Lago, P.; Visser, J., "Measuring the Degree of Service Orientation in Proprietary SOA Systems," *IEEE 7th International Symposium on Service Oriented System Engineering (SOSE)*, pp.233,244, 25-28 March 2013