20th International Conference on Advances in ICT for Emerging Regions (ICTer 2020): 095 - 100

Transforming Monolithic Systems to Microservices - An Analysis Toolkit for Legacy Code Evaluation

Chamika Bandara, Indika Perera

Dept. of Computer Science and Engineering, University of Moratuwa
Sri Lanka

Chamika.10@cse.mrt.ac.lk

indika@cse.mrt.ac.lk

2020 20th International Conference on Advances in ICT for Emerging Regions (ICTer) | 978-1-7281-8655-9/20/$31.00 ©2020 IEEE | DOI: 10.1109/ICTer51097.2020.9325443

Abstract—**Microservices has become one of the mainstream approaches for system architecture; industry accept the importance of migrating existing monolithic systems into microservices-based architecture to gain benefits. Often it is a challenging task to achieve as the monolithic systems are tightly coupled from their implementation perspectives. To migrate existing monolithic systems, it may require the architecture diagrams, system artefacts and people who know the system well etc. yet not all the time we find these resources. With suitable tool support, we can get insight into the possible services which can be found in the existing monolithic system at code level without worrying about the existence of architecture diagrams, experts etc. It allows us to make efficient decisions on what to move as microservices from the monolithic system. This research introduced a toolkit to analyse monolithic systems and propose the best ways to decompose the functionality into a set of microservices. The evaluation shows accurate revamping of the system architecture with suitable microservices suite.**

Keywords— **microservices, monolithic, architecture revamp, legacy system transformation, service identification**

I. INTRODUCTION

Maintaining large scale monolithic legacy systems is a challenge due to their complexities. Many researchers look for well-defined architectural solutions to cope with such systems. Among those converting the existing system into microservices hold a specific place due to simplicity and related other advantages [1][2]. Though some concern that microservices is nothing much other than a simple extension to Service Oriented Architecture (SOA), microservices provide considerable advantages.

Normally at the core of a monolithic system lies the core business logic of the system realized by modules with services, domain objects, and events. Adapters that interface outside surrounds the core. Though it seems like a modular service architecture, the whole system packaged and deployed as one system. Monolithic systems have their disadvantages. Even a successful system is subject to modification and evolves. Once the system is quite large and complex it becomes extremely challenging to modify. Thus, it is not possible to understand the whole system by source code itself. Also, small changes in code cause larger deployment delays and system instability. That is where the microservices can be useful.

In a microservices system, idea is to build a set of a smaller and interconnected layer of services to provide the required functionality thus reducing the complexities and the other pitfalls in the monolithic systems. Applications typically use three types of scaling together: 1) decomposing the application into microservices, 2) multiple instances of each service for throughput and availability, and 3) partition of the services. In a microservice architecture, each of the services has its database, yet it adds more complexity to the system for controlling the distributed transaction and failovers.

Software engineers consider converting legacy monolithic systems into microservices to gain advantages. However, the question for the expert architects is to decide first to build a monolithic solution then convert it to a microservices architecture or to build a microservices architecture from scratch. Building microservice from scratch could be a high risk and require a lot of resources. Hence, practitioners try to move the low-risk part of the system into microservices initially.

Identifying the less risky services in a very complex and large-scale monolithic system can be challenging and extremely resource consuming. There can be many dependencies and complexities on each service. As pointed out earlier it is quite a challenging and complex task to identify and classify the existing service layer into microservices. There can be different service-related metrics to consider when we are separating from the existing services. Such as business values, dependencies, requirements, risks, and deadlines etc. It would be useful to have a tool that helps identify services from the existing monolithic system which are more feasible to convert into microservices. It is not feasible to go through each line of the code or architectural documents to identify the services to convert.

This research tries to help identify the services which can be converted as a microservices with low risk and effort. A tool for such work must consider different aspects that need to take into account when identifying those services. Those could be the architectural constraints, system limitations, business requirements, architectural complexities and dependencies. Among those using a tool will be limited to finding a solution for problems like architectural complexities, and dependencies. Because those are the factors that only can be measurable theoretically as well as practically using a simple tool. Other factors may require even more complex logics and research areas to accomplish (i.e. AI).

Contribution of this research paper is to automatically identify the microservices from the existing monolithic systems unlike the other manual approach as the source code be the only resource which will be available any given time. Service will be identified using its characteristics for being a microservice.

This paper is structured as follows: Section 2 summarizes existing migration strategies and the approaches and methodologies on identifying services from the legacy systems. Section 3 describes the approach, which this research uses for identifying the microservices lies in the existing monolithic system by using characteristics of being a

978-1-7281-8655-9/20/$31.00 ©2020 IEEE

microservice. Section 4 describes the approach to evaluating the suggested method using two approaches as a manual approach and a semi-automated approach. Finally, Section 5 concludes the paper.

II. Background and Related Work

The legacy system migration is classified into four categories as in [3] such as:

- **Replacement** - Legacy application is replaced entirely using COTS (commercial off-the-shelf) product.
- **Integration** - Existing legacy application is accessible via an interface.
- **Redevelopment** - Entire legacy application is re-developed.
- **Migration** - Legacy application is gradually moved to MSA with reusing the legacy components.

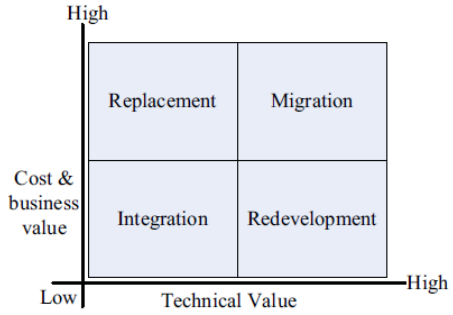

Fig. 1 Migration Techniques.

Among all the above four methods the migration has higher technical and business value. This research also follows an automated migration technique to extract the microservices from the existing legacy system. To identify the candidate services from the existing legacy system literature suggest many methods. In [4], [5] several methods are reported to identify potential services: Candidate Service Identification (CSI) is categorized into two methods as top-down and bottom-up. In the top-down approach, user requirements leading to system requirements are used to define process and sub-processes which are mapped to functions, [6] [7] and [8]. In contrast, the bottom-up approach utilizes the legacy code to identify services using techniques such as information retrieval [9], concept analysis [10], business rule recovery [7] and source code visualization [11].

The above techniques can be further divided into two categories such as Code level techniques, and Architecture level techniques. Example of code-level techniques are slicing ([12], [13], [14], [15]) wrapping ([16], [17], [18]) refactoring [19], code transformation [8]. Legacy to SOA migration uses model-driven engineering as well [6]. In this research, we focus on the bottom-up, code level technique to identify the candidate servicers which lies in the legacy monolithic software systems.

III. Proposed Solution Architecture

We propose a migration method by identifying microservices as groups of classes in the legacy system code. The source code is the only guaranteed resource which is available anytime whereas other service identifiable resources like architecture documents could often be missing. Several studies have used manual approaches to identify the services in the legacy code; here we propose an automatic identification method of candidate services based on a fitness function that measures characteristic similarity between each group of classes which can be a potential service. To identify the microservices from the existing legacy system, will follow 3 main steps. Such as:

1. Analyse monolithic codebase to identify potential candidate services.
2. Qualified service identification and clustering.
3. Format and present result to the user.

A. *Analyse monolithic codebase to identify potential candidate service*

To identify potential candidate services from the existing monolithic codebase, define a mapping between OOP concepts and the microservices concepts. It considers service as a group of classes defined in object-oriented code.

B. *Qualified service identification and clustering.*

This step identifies candidate services from a group of object-oriented classes. A qualified service is selected from candidate ones based on a function that measures its characteristics of being a microservice.

1) Characteristics of Microservices.

In the microservices literature there are many definitions found to define a microservice; among those [20] defined a strong definition of being a microservice. By referencing other definitions the following definition is derived: Microservice Is a coarse-grained, loosely-coupled, discoverable entity that interacts with the application and other services through an often asynchronous, has message-based communication model.

2) Refining the Characteristics.

Among the identified characteristics from being a microservices, there can only be some characteristics which can be measured using automatic code analyzing. Following software quality metrics with measurable parameters of being a microservice was identified: Functionality by granularity (Coarse-Grained), usage by discoverability, self-containment by loosely coupled, and composability by composable level. Using this set of measurable characteristics, we define a fitness function to cluster them into similar clusters based on the fitness function value.

- Functionality(Coarse-Grained)
  - Coupling and Cohesion measurement.
- Composability(Composable)
  - Cohesion measurement.
- Self-Containment(Loose-Coupled)
  - External Coupling measurement.
- Usage(Discoverable)
  - Inheritance(IS-A) and Composition(Has-A) factor measurement.

Using the above main characteristics, the fitness function value for a given candidate service is calculated as follows:

3) Fitness Function value model

For the identified candidate services, let us first define a fitness function as a linear combination of the aforementioned characteristics as follows; where F(E), C(E), S(E) and U(E)

be the identified characteristics of being a microservice respectively.

$$FF(E) = \frac{\alpha F(E) + \beta C(E) + \gamma S(E) + \delta U(E)}{n} \quad (1)$$

where $\alpha, \beta, \gamma, \delta$ are coefficient weights set by the expert (architect). As well as n will be calculated as follows:

$$n = \sum(\alpha + \beta + \gamma + \delta) \quad (2)$$

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

According to the definition in ([14], [20]) it defines the functionality with the combination of the following aspects.

1. A service supporting several interfaces can include many functionalities, thus the higher the number of interfaces is, the higher the service functionality – measured by the number of interfaces
2. An interface with highly cohesive services provides single functionality – measured by avg. of service interface cohesion within the interface
3. A group of interfaces with high cohesion are best to provide a selected functionality – measured by cohesion between interfaces
4. If the candidate service is highly coupled, the code implements one task – measured by coupling withing a service
5. If the candidate service is highly cohesive, that code provides single functionality – measured by cohesion within a service

Based on the above definition for the functionality F(E), it derives the following equation to calculate its value.

$$F(E) = \frac{1}{5}\Big(np(E) + \frac{1}{I}\sum_{i \in I} LCC(i) + LCC(I) + Couple\ (E) + LCC\ (E)\Big) \quad (3)$$

Where $np(E)$ refers to the 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, $Couple\ (E)$ refers to the coupling inside a service, and $LCC\ (E)$ refers to the cohesion inside a service.

From the definition for Composability C(E) [22] it can be calculated as follows;

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

As well as [23] Self-Containment has defined as the external coupling of the service.

$$S(E) = ExtCouple\ (E) \quad (4)$$

Further from the definition for Usage [24] can be calculated as follows:

$$U(E) = H(E) + P(E) \quad (5)$$

Where P(E) and H(E) be the inheritance (is-a) factor and P(E) be the composition(has-a) factor respectively. Further to the above calculations, it requires to calculate the Coupling of services to fulfil above calculations. From the definition for the Coupling [15], it takes the degree of direct and indirect dependence of a class on other classes in the system. It calculates the Coupling as follows:

$$Coupling(E) = \frac{Direct\ Dependencies}{(\ Direct + Indirect\ )\ Dependencies} \quad (6)$$

And External Coupling will calculate as follows:

$$ExtCoupling(E) = 1 - Coupling(E) \quad (7)$$

Using the above-defined equations, it can calculate the fitness function value for each of the candidate services identified from the code base, once identifies the candidate services by applying the clustering algorithm it can cluster the services into similar clusters based on the fitness function value. This clustering process will continue until the desired length is met.

4) Clustering Process.

For the service clustering, it calculates the fitness function value for each of those services. Initially, each of the services identifies will consider as a single microservice or a cluster. Then again based on the calculated fitness function value, those will be merge and process will continue until the defined depth is met. This depth should define by an Architect based on the business requirements and the experience. In this research, it uses defined cluster depth just prove the concept. Here it uses hierarchical agglomerative clustering [25] with single-linkage strategy [26] to cluster each of identified services based on the fitness function values. Once fitness values are calculated at each iteration of this grouping process it mergers minimum values services together using the single-linkage strategy based on the Euclidian matrix formed using the fitness function value and formed next cluster and then again calculates the fitness function value for those merged services as one service and continues the process. The following shows the pseudo-code for this process.

```
let each class be a group.
compute fitness function of pair classes;
repeat
  merge two "closest" groups based on
        fitness function value;
  update list of groups;
until reaching the defined depth.
return group map;
```

5) The Tool.

To upload and process the Java code and to view the result, it uses the following architecture tool consists of frontend and backend components of different technologies. I.e., for the frontend it uses angular2 and for the backend, it uses Java with spring boot. It follows client-server architecture; angular frontend communicates with spring Java backend using REST.

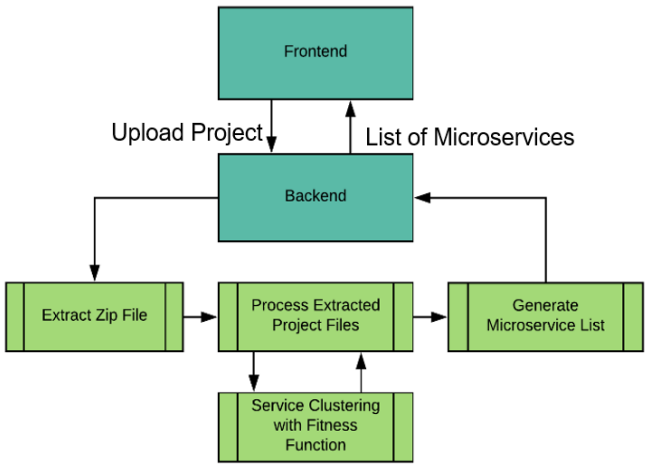

Fig. 2 Tool Architecture.

As per the above architecture of the tool implementation users can upload their source code as a zip file to process and the results will be presented to the users (Figure 3) with the list of name and associated risk value. For the risk value, it has taken self-containment value which tells how much it self-sufficient to do their jobs without relying on other services. Based on the final value obtained for the services risk value also be displayed to the end-user (High Risk<0.75; Medium Risk [0.75,0.8); Low Risk [0.80,0.95]; No Risk>0.95).

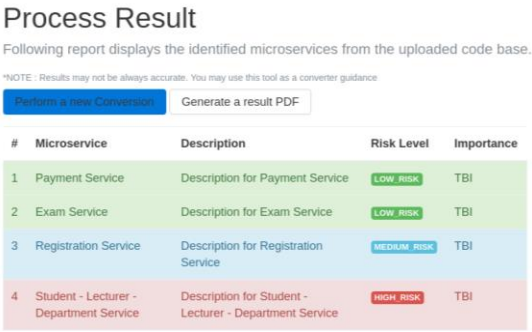

Fig. 3 Results view from the user end.

IV. Evaluation

The evaluation of this carried out by a project with different coefficient values. To test the behaviour, created a sample project with MVC architecture including different layers such as Service layer, Repository layer, Manager layer. It was developed using Java language and by heavily relying on the object-orientated programming concepts. This project is aimed to provide services for Student data management. It does CRUD operations for Student, Lecturer, Department, Payment, and Registration. The API is a REST API, which exposes those functionalities to the end-user.

The created tool was used to extract the microservices using the different values for its coefficients in the fitness function. It ran for several iterations. Tables 1,2, and 3 show some of the highlighted irritation results to clarify the tool results obtained from the process. The experts in the project team recommended different coefficient values as per their preferred views on each functional area implementation.

A. Results

1. Iteration 1
   Coefficient values: $\alpha = 1, \beta = 2,\ \gamma = 3,\ \delta = 4$

TABLE I FINAL CLUSTER VALUES FOR ITERATION 1

Service	Final Cluster	FF(F)
1	Payment Service	0.842
2	Exam Service	0.806
	Register Service	
	Student Service	
3	Lecturer Service	0. 710
	Department Service	

When increasing the bias for the usage of initial services the results from the tool according to the implementation of Student Management API it has given a result more bias to the service usages. In this case, it is harder to remove those as services as they have much more dependencies.

2. Iteration 2
   Coefficient values: $\alpha = 4, \beta = 2,\ \gamma = 3,\ \delta = 1$

TABLE II FINAL CLUSTER VALUES FOR ITERATION 2

Service	Final Cluster	FF(F)
1	Payment Service	0. 827
	Department Service	
2	Exam Service	0.713
	Register Service	
3	Student Service	0.764
	Lecturer Service	

When increasing the bias for the functionality of initial services the results from the tool according to the implementation of Student Management API it has given a result more bias to the service functionality.

3. Iteration 3
   Coefficient values: $\alpha = 2, \beta = 3,\ \gamma = 4,\ \delta = 1$

TABLE III. FINAL CLUSTER VALUES FOR ITERATION 3

Service	Final Cluster	FF(F)
1	Payment Service	0. 901
2	Exam Service	0. 831
3	Register Service	0. 786
4	Student Service	0.712
	Lecturer Service	
	Department Service	

When increasing the bias for the Self-Containment of initial services the results from the tool according to the implementation of Student Management API it has given a result more bias to the service self-containment. It means these services can be isolated easily.

B. Validation

To validate the results from the tool two methods were used

i. Manual Approach

This approach is the most suitable and reliable since the domain experts can give correct prediction to the outcome. In the context of service identification methodologies, many experts find this as the most convenient and accurate approach to identify the microservices lies in the code.

As explained about this method used to identify the microservice from the existing monolithic system, will manually predict the microservices in the monolithic system by looking at the current system architecture and the implementation. In the system architecture, it consists of 4 layers such as Controller, Service, Repository, and the Manager. Each will work together to accomplish some tasks. For an instance to add a new Student into the database it will create new registration record and will update department records, and lecturer records with the help of relevant services and manager functions. Likewise, there will be many dependencies exist when looking at the system architecture.

According to the system architecture and applying the knowledge on microservices, it can say that this system can be developed as a microservice architecture as follows.

- MS1: Student Service + Lecture Service
- MS2: Department Service
- MS3: Exam Service
- MS4: Registration Service
- MS5: Payment Service

Student Service and Lecture Service have similarity when looking at the architecture and the code. Other services can be developed as separate services as they do not depend on the rest. Hence according to the manual validation process, the results obtained on iteration 3 is more relevant and accurate. That is because in iteration 3 it has given more weight to the Self-Containment and Cohesion, which are the main aspects of the qualities when developing the microservice architecture. By looking at those results it can notice that there is a direct influence from those qualities when identifying the microservices from the monolithic system by using its quality attributes. By looking at the results show that close to 80% of extracted microservices from the tool are accurate. It can be assumed that by feeding context-related knowledge to the tool it can be more accurate than this, but it is too hard to implement for this research with the given time frame.

ii. Semi-Automated Approach

This approach is not the most reliable since this also a tool or a method which will be used to validate another tool or a method. This method can validate the outcome to some level.

The idea of this method is to validate the results we obtained from the tool by using the logs that were added to the Student Data Management API when executing a few common functions. After gathering those logs new tool which created to analyze the logs from the Student Data Management API will analyze how frequent those services and related manager functions were used to accomplish those tasks. To analyze the usages of each of the tasks, it has selected few tasks, such as Add a Student, Add a Lecturer, Add a Department, Add an Exam, and Make payments for an exam.

For an instance, according to the collected logs Adding a Student task produced usage percentages as follows: Registration Service 60.0%, Student Service 100%, Lecturer Service 64.9%, and Department Service 55.3%. By looking at the above results it says that when performing a Student service task, the other services such as Registration, Lecturer and Department has contributed as the above as an approximated percentage. Here Student service percentage was taken as 100% since it performs task relevant to that service. Other percentage values are calculated based on the number of services used in a particular service to the total number of service instances. Similarly, for the other selected tasks also the calculated results were obtained. It indicates how much each service is required to perform certain tasks and which are coupled together when performing those tasks.

By looking at the obtained results it can assume Student Service has dependencies to many of the actions performed. Whereas Payment and Exam have fewer interactions to other actions comparing to the Student and Lecturer Services. Moreover, the Department Service and Registration Service have some impact when performing above tasks but those are not significant as the Student or the Lecturer Services interactions (percentage-wise). Hence as a summary, it can be categorized the services as follows:

- MS1: Student Service + Lecture Service
- MS2: Department Service + Registration Service
- MS3: Exam Service
- MS4: Payment Service

1) Evaluation Summary

From manual and semi-automated validation methods, both suggest that the Student Service and the Lecture Service should be merged as one microservice. Additionally, the manual validation approach suggests that all other should be a one microservice whereas automated approach suggests having merge of Department Service and Registration Service. But looking at the code and the architecture of the Student Data Management API it is clear that Department Service and the Registration Service should keep as separate services as they do not much depend on each other and easy to separate due to having larger self-containment.

The proposed tool has suggested a different combination of services for iteration 1, iteration 2 and iteration 3 based on the differentiated prominence of quality attributes. Iteration 3 has obtained the most accurate service method combination, which also has high Cohesion and Self-Containment. As per the observations and many researchers the key aspect to separate microservices from an existing service or to build such from the scratch the service which chosen should have high cohesion and self-containment.

V. Conclusion

A. Contribution

The contribution of this research is the identification of microservices in legacy or monolithic system source code considering service quality characteristics. The tool first set a mapping model between Java classes and microservice concepts. Then, it proposes a fitness function to measure service quality. Initially, it assumed that all the service classes are as one single microservice then it improves the service identification iteratively using the fitness function. Then

based on the fitness function value of each of the identified services those will be clustered into a set to microservices at each of the iteration. A simple REST API system, which has the basic functionality to manage the student data for an institute is used as the case system and it has proven that this method has a significant impact on filtering out microservice from the existing monolithic system code.

The results have proven that the most applicable values to identify microservices are the self-containment and the composability. That is because when using higher values for the composability and the self-containment when extracting microservices from the monolithic system by using the tool has produced close results as to the manual identification process.

When designing a microservice-based system architecture it should design in a way that each service in the architecture should;

- Be able to independently develop and deployed.
- Have minimal dependence on other services.
- Closely related functionality must be together within a single service.
- Be loosely-coupled, self-contained and should have high cohesion.

This research has proven that converting existing monolithic systems into microservices can be done through an automated tool with a well-defined approach. This research has proven that we can improve this methodology to have more accurate results with minimum effort.

B. Study Limitations

The study selected the MVC pattern and Java as the programming language as the scope. With the limited time availability only a shallow level, yet comprehensive analysis was carried out. Advanced analysis such as service descriptions including its usages, dependency graphs was excluded, which can be carried out as future research. Being one of the pioneering studies on the research topic it was a challenging task complete the research work with limited literature and experimental models, which constrained the overall output of the research study.

C. Future Work

The results shown by the tool can be enhanced; its suggestions can relate with the context knowledge and the previous analysis knowledge. For the future work to improve this tool, it can leverage some of the aspects of context knowledge and the previous analysis knowledge.

The description is meant to show the depended classes for each of the services and its current usages. As well Importance is mean to show that how much that particular identified services are going to be important to extract as a microservice but to implement such functionality the tool should leverage some of the aspects of context knowledge about the monolithic system which has fed to the tool to extract the microservices.

Moreover, the current implementation of the tool only supports for the Java programs which have the MVC architectural pattern, as a future work it can extend to accept more languages with more architectural patterns and can incorporate machine learning models to improve accuracy.

REFERENCES

[1] A. Selmadji, A. Seriai, H. L. Bouziane, R. O. Mahamane, P. Zaragoza and C. Dony, "From Monolithic Architecture Style to Microservice one Based on a Semi-Automatic Approach," *IEEE ICSA2020*, Brazil, 2020, pp. 157-168

[2] A. Levcovitz, R. Terra, M. Valente. "Towards a technique for extracting microservices from monolithic enterprise systems" *arXiv preprint arXiv:1605.03175*. 2016 May 10.

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

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

[5] 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.

[6] 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.

[7] 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.

[8] C. Zillmann, A. Winter, A. Herget, et al., "The SOAMIG Process Model in Industrial Applications," *in CMSR'11. IEEE*, 2011, pp. 339–342.

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

[10] 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.

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

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

[13] 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.

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

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

[16] 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.

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

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

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

[20] I. Nadareishvili, R. Mitra, M. McLarty, and M. Amundsen, *Microservice Architecture: Aligning Principles, Practices, and Culture*", 2016 O'Reilly Media, Inc.".

[21] K. Channabasavaiah, K. Holley, and E. Tuggle, "Migrating to a service-oriented architecture", *IBM DeveloperWorks*, 16, pp.727-728, 2003

[22] J.M. Bieman, B.K. Kang, "Cohesion and reuse in an object-oriented system", *ACM SIGSOFT Software Engineering Notes*, 20(SI), pp.259-262, 1995

[23] B. Souley, and B. Bata, "A class coupling analyzer for Java programs", *West African Journal of Industrial and Academic Research*, vol. 7, No.1, pp.3-13, 2013

[24] A. B. Nasser, A. Rahman, A. Alsewari and K. Z. Zamli, "Tuning Of Cuckoo Search Based Strategy For T-Way Testing", *ARPN Journal of Engineering and Applied Sciences*, vol. 10, no.19, pp.8948-8954, 2015

[25] Hierarchical agglomerative clustering algorithm [Online]. Available: https://nlp.stanford.edu/IR-book/html/htmledition/hierarchical-agglomerative-clustering-1.html (2008)

[26] Linkage Strategies [Online]. Available: https://nlp.stanford.edu/IR-book/html/htmledition/single-link-and-complete-link-clustering-1.html (2018)