From Monolith to Microservices: A Semi-Automated Approach for Legacy to Modern Architecture Transition using Static Analysis

Mohd Hafeez Osman[1], Cheikh Saadbouh[2], Khaironi Yatim Sharif[3], Novia Admodisastro[4] and Muhammad Hadri Basri[5]
Faculty of Computer Science and Information Technology
University Putra Malaysia
43400 Serdang, Malaysia[1,2,3,4]
Malaysian Administrative Modernisation and Management Planning Unit
Prime Minister Department, Malaysia[5]

***Abstract*—Modern system architecture may increase the maintainability of the system and promote the sustainability of the system. Nowadays, more and more organizations are looking towards microservice due to its positive impact on the business which can be translated into delivering quality products to the market faster than ever before. On top of that, native support of DevOps is also desirable. However, transforming legacy system architecture to modern architecture is challenging. As manual modernization is inefficient due to its time-intensive and the significant amount of effort required, the software architect is looking for an automated or semi-automated approach for easy and smooth transformation. Hence, this work proposed a semi-automated approach to transform legacy architecture to modern system architecture based on static analysis techniques. This bottom-up approach utilized legacy source code to adhere to the modern architecture framework. We studied the manual transformation pattern for architectural conversion and explore the possibility of providing transformation rules and guidelines. A task-based experiment was conducted to evaluate the correctness and efficiency of the approach. Two open-source projects were selected and several software architects participated in an architectural transformation task as well as in the survey. We found that the new approach promotes an efficient migration process and produces correct software artifacts with minimum errors rates.**

***Keywords*—Static analysis; software architecture; software modernisation; microservices**

I. Introduction

Monolithic architecture is a typical cohesive paradigm for the construction of a software system. In this perspective, Monolithic means that it's all composed in one piece. Monolithic technology is developed to be self-contained; program components are strongly interconnected instead of decoupled, as is the case with modular software programs. Each element and its related accessories must be available in a tightly linked design for the software to be functional.

Microservices are a type of architecture of systems in which a huge and complex system is designed as a series of lightweight services (i.e. loosely coupled). Every module serves a common business purpose and provides a simple, excellently defined endpoint to connect with all other resources. Modern architecture such as microservices is introduced to increase the maintainability of a system. However, the transition from monolith to modern architecture is challenging as the manual migration process is inefficient due to its Time intensive and the significant amount of effort required. Researches [1] [2] [3] [4] have confirmed that the manual migration process is complex and lacks automated (or semi-automated) tools supported. Nowadays, more and more organizations are looking towards microservice due to its positive impact on the business which can be translated into delivering quality products to the market faster than ever before. On top of that native support of DevOps is also desirable.

The study aims at providing a semi-automated platform to guide software architects in transforming legacy architecture to modern system architecture. Furthermore, we aimed to fulfil the research gaps in software modernization by developing a semi-automated tool that may increase the efficiency of the software modernization process which may satisfy the need of software companies that are looking forward to such a tool to minimize the resources of the software modernization and deliver more value to their prospects. Thus, we perceive that providing a practical solution is crucial. The refactoring rules should be formulated based on the practitioner's and experts' practice. Hence, the refactoring rules are extracted from the refactoring practice that was discussed by the microservices experts from online discussion forums. Several possibilities on (semi-)automating the refactoring task in modernising legacy systems to microservices are investigated.

The main contributions of this study are the following: (i) Refactoring rules to transform legacy to microservices; (ii) Legacy to microservices refactoring framework and metamodel and (iii) Semi-automated legacy to microservices tool.

The remainder of this paper is structured as follow: Section II discusses the related research. Section III explains the research methodology while Section IV describes the correctness and efficiency evaluation. Section V discusses our findings and Section VI presents the conclusions and future work.

II. Related Work

This section discusses the transformation of the monolith to microservices architecture from the perspective of techniques

that were used such as model-driven, static analysis, dynamic analysis. We also discuss the other works that are related.

A. Model-Driven

There are several work that used model-driven technique in migrating monolith to microservices, such as [1], [4], [10], [23] and [24]. We detail some of these works in this paper.

Fritzsch et al. [1] has presented 10 refactoring methods in scientific literature to migrate monolithic systems into microservice. The methods are subdivided into four classes. The result has shown that most of these techniques shall only apply under several situations. The limitations have been identified such as input data size and the need for an implementation tool.

Kamimura et al. [4] discussed the efforts to distinguish candidates from monolithic systems to microservices, i.e. endpoints or sometimes called webservices resources which could be turned into coherent, work-alone smaller services. This is a long and complicated manual endeavor that involves the review of several aspects of information engineering and always relies heavily on the professional performer's knowledge and expertise. To solve this issue, they established a method that distinguishes microservice applicants from the codebase using the SARF technology clustering algorithm with the given application classes and information's to be able to generate microservice candidates. The approach also captures the candidates derived to demonstrate the connection between the candidates being extracted with the whole system.

B. Static Analysis

Several works have used static analysis as the main technique for the monolith to microservices migration. The work includes [5], [6], [2], [3] and [11]. We discuss some of this work in more detail.

Mazlami et al. [2] focus on the failure of automated support instruments and formal models in the area of software migration. They proposed a formal extraction model as a form of an algorithm suggestion to identify the microservices candidates as a web application prototype. They applied their proposed algorithm to 21 open-source projects (developed using different programming languages, e.g Java, C++, Python) and showed that the generated microservice can decrease the size of the development team to half.

Gysel et al. [5] introduced Service Cutter, a service decomposition based on 16 coupling criteria distilled from the literature and industry experience. Service Cutter offers a service extractor framework that implements graph aggregating algorithms along with includes ranking based on scoring beginning from building blocks and structured documents such as domain model with its use-cases.

Li et al. [3] proposed a semi-automatic decomposition approach that used data flow to extract services from the legacy monolith that relates to business logic. The decomposition has been specified into three stages: (i) define the use case along with business logic is assessed as a baseline of specifications; (ii) comprehensive DFD over various rates and its process-datastore are constructed through the business logic based on functional requirements; (iii) developed an algorithm to instantly bind the DFD to an undependable DFD. A comparative analysis that focused on relevant cohesion metrics and coupling metrics showed that a data flow-driven methodology is ideal in providing fair, repeatable, and readable microservice applicants. However, two (2) major limitations have been identified: (i) the dataflow method largely relies on the precise DFDs at all levels, and (ii) secondly do not put into consideration the non-functional requirements (NFR) of the microservices.

Levcovits et al. [6] outlined a strategy for describing and distinguishing microservices on a monolithic enterprise application. The assessment indicated that their methodology may recognize successful microservice candidates on a 750 KLOC financial system in which decreased the size of the code base modules and took full advantage of the architecture of microservices, such as the independent development and deployment of services and technological freedom. However, in certain situations, extra effort would be necessary to move mutual subsystems to a collection of microservices.

C. Dynamic Analysis

This subsection discussed the related work that uses dynamic analysis and also the combination of dynamic and static analysis.

Mayer and Weinreich [14] proposed an approach to constantly extracting services from a legacy software system based on REST microservices. Their approach relies on static and dynamic data collection, gathering input at runtime execution, and merging static and dynamic analysis. This customizable analysis technique captures runtime information to one dimension, allowing an analysis of the architectural design transformation over a longer period. Evaluated the proposed technique of a system, by establishing an empirical test that included several communicating services. The findings demonstrated the feasibility of the process of collecting and aggregating data.

Carrasco et al. [12] have discussed nine (9) potential mistakes in terms of specific bad smells. Such mistakes can be noticed and fixed along the way while transitioning to the new architectural style. As an illustration of a common mistake, when team members are separated across layers, for example, a front-end, code business Logic, and operation department, basic changes can require resources and time between the members of the team in authorizing the required action. From that point of view, a team member may propose a change on which level of the application they possess direct exposure to which can be limited by dividing responsibilities within specific services only. This work offers a strong foundation for relevant insights on the effective transition of monolithic architecture to the new architectural style.

ToLambda is a tool created by Kaplunovich [13] that dynamically allows the generation of system base code, turning the existing systems into healthier architectures, and continually improving. The tool converts existing java code into Nodejs code and changing the code structure to more modular using a microservices architecture. No experiment has been conducted to evaluate the correctness of the conversion and the architectural change.

D. Other Related Work

There are several works that related such as [7], [8], [9], [15], [16], [17], [18], [19], [20], [21] and [22]. We discuss some of these works in more detail.

Ahmadvand and Ibrahim [8] introduced a scientific method for breaking up monolithic applications into microservice applicants, aimed at an optimal level in safety and scalability as a quality attribute. Using this methodology, requirements engineers should carefully compare security and scalability criteria. This initiates design decisions at the requirement engineering activity and thereby extends the vision of the software architect's point of view regarding the system to be.

Meanwhile, Ren et al. [9] introduced a program-based research approach for moving the traditional monolith code to cloud applications. They suggested merging dynamic and static analysis of the binary code to derive the properties of the existing monolith program. The reactive tracing has been used to trace the functionality of the application, the user input data, the called methods and the accessed database information, and the rate of the method invocation throughout tracing. The experimental findings indicated that the new approach was successfully addressed the re-designing issue of the legacy system.

Tyszberowicz [15] proposed a systemic and rational approach to identify Microservices based on requirements in use and functional breakdown of those specifications. This method offered highly cohesive and tight decomposition coupled. The evaluation was conducted by implementing the approach on three different systems.

Meanwhile, Jin et al. [16] suggested a feature-oriented microservice retrieval approach to automatically create microservices candidates through a monolithic legacy system. Execution traces are used to facilitate the classification of features. Compared to the conventional approaches (which use static analysis), traces of execution can better represent software functionalities.

III. Methodology

This section provides the study design and methodology of every stage and ethical considerations, in order to develop a supporting tool that can assist software architects while modernizing applications. The methodology of this study consists of the following four stages.

A. Stage 1 - Framework Development

At this stage, we develop the overall architecture of the framework that consists of several components that define the framework which will define the rules of software modernization. The relationship between the framework components will be depicted and describe briefly. Each component of the framework will follow the single responsibility principle so that components reuse is easy for further enhancement. The overall framework processes will be summarized in short and consist of an algorithm written in pseudo-code. Furthermore, the refactoring rules that will be applied on the web layer will be listed in detail as a table along with the UML class diagram that depicts how rules work.

B. Stage 2 - Tool Design and Development

For this first step, we will concentrate on developing our framework and the overall architecture to better understand the system elements, and we will use UML diagrams and IntelliJ as an integrated development environment platform to implements these design architectures. The tool would include features that allow users to import the legacy application to our current tool, then be able to re-architect the legacy applications using basic drag-and-drop features and automatically refactor rules to modernize the legacy features on the web layer and create compatible cloud artifacts.

C. Stage 3 - Evaluating Correctness and Efficiency of the Tool

We will conduct an experiment that is favorable in software engineering research to enable us to verify the proposed tool's correctness and efficiency. We will use 2 open source projects in this experimental research as a laboratory for this study. In this case, for a broader view of the problem anatomy, 2 open-source projects were both built-in java with distinct requirements. Selections will be made for the two open-source initiatives and further information will be given. In addition, with our latest proposed tool, we are also preparing to conduct an expert assessment and the participants will first be told of the general purpose of the analysis and asked to perform software migration using the semi-automated built tool and manual migration, then metrics will be guided by the outcomes of migration to compare both the manual and semi-automated solution. In addition, a structured questionnaire will be used to invite the participants selected for the research to participate in the survey and to share their views on the effectiveness of the process in the industry.

D. Stage 4 - Findings and Conclusion

Results from stage three (Experiment and expert assessment of the correctness and efficiency of the tool) will be addressed in this segment to see how the outcome fits the targets based on our established tool and eventually, conclusions will be drawn and potential work progress will be explored next.

E. Framework Architecture

Fig. 1 illustrates the framework components and how they interact to achieve a common goal which is modernizing legacy java application that is built on top of spring framework. A brief explanation of the Fig. 1 is as follows:

- **Legacy Code Container** is responsible for holding the legacy code and passing it to the filter component to be filtered.
- **Filter** is responsible for filtering incoming legacy codebase on the web Layer where controllers are located. Once controllers are identified they will be passed to the Refactoring component to be refactored.
- **Refactoring** is responsible for modernized legacy code to a new modern way it consists of four sub-components which are:
    - *Refactoring Rules* contains all the rules which will be applied to the legacy code

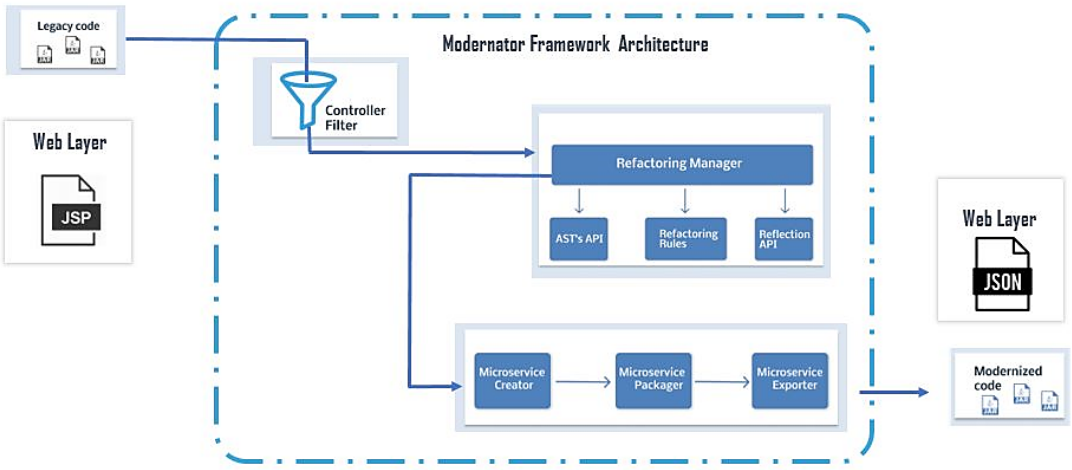

Fig. 1. Modernator Framework.

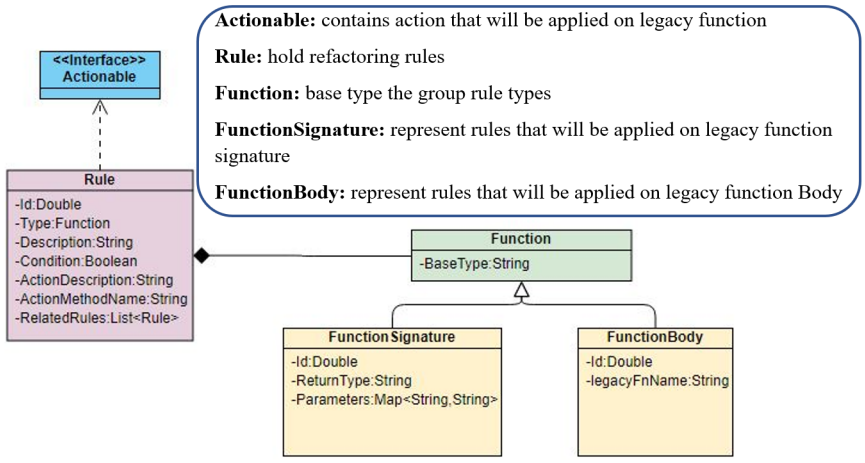

Fig. 2. Refactoring Rules Metamodel.

- *Refactoring Manager* process and takes care of the refactoring processes.
- *AST's API* contains needed API to understand the structure of the legacy code.
- *Reflection API* contains needed API to call rules that will be applied at runtime.

- **Microservices Creator** is responsible for taking refactored code and organize it and auto-generate the new microservices.
- **Microservices Packager** is responsible for taking newly created service and package it as a Docker image or jar/war.
- **Microservices Exporter** is responsible for exporting ready service to the user.

F. Framework Refactoring Rules Metamodel

Fig. 2 illustrates the metamodel of refactoring rules that are used in the framework. This figure shows that each rule is associated with a function. Either it is associated with a function signature or a function body. Each rule should have an Id, Type, Description, Condition, Action Description, Action Method Name and Related Rules.

G. Framework Refactoring Rules Table

The below table contains all the rules that will be applied to a legacy code. The rules are categorized into two types as follow:

- **Function Signature** rules that will be applied on the legacy function signature which is either on the function parameter or function return type.
- **Function Body** is rules which will be applied in the body of the legacy function.

The main target of this work is designing and developing a tool that can modernize java applications. The legacy application will be used as input to the tool. The refactoring task will be performed at the controller's level in the web layer.

TABLE I. SET OF REFACTORING RULES

Rule ID	Rule-Condition	Related - Rules	Before Refactoring	After Refactoring
Rule Type: Function Signture				
R1	Applied when : ModelMap object is presented as function parameter	R [2, 3, 4, 5, 6, 7, 8, 9, 10]	*String FunX (ModelMap map)*	*String FunX ()*{}
R11	Applied when: ModelAndView object is presented as function parameter	R [12 – 36]	String FunX(ModelAndView MAV){}	String FunX(){}
R37	Applied when: ConcurrentModel object is presented as function parameter		String FunX(ConcurrentModel CM){}	String FunX(){}
R38	Applied when: ExtendedMod-elMap object is presented as function parameter		String FunX(ExtendedModelMap EMM){}	String FunX(){}
R39	Applied when: RedirectView object is presented as function parameter		String FunX(RedirectView rv){}	String FunX(){}
Rule Type: Function Signture				
R40	Applied when: Map object is presented as function parameter		String FunX(Map m){}	String FunX(){}
R2	Applied when: addAttribute function is called	R1	String FunX(ModelMap map) map.addAttribute(“msg”,”hello”)	String FunX(ModelMap map) ArrayList<String,String>data = new ArrayList<>(); data.add(“msg”,”hello)
R3	Applied when: ModelMap function is called	R1	ModelMap md = new ModelMap (attributeName, attributeValue);	ArrayList<String,String>md = new ArrayList<>(); md.add (attributeName, attributeValue)
R4	Applied when: ModelMap function is called	R1	ModelMap md = new ModelMap(attributeValue);# attributeValue is of type Object	ArrayList<String,String>md = new ArrayList<>(); md.add(attributeValue)
R5	Applied when: addAtribute function is called	R1	addAttribute(attributeValue); # attributeValue is of type Object	ArrayList<String,String>md = new ArrayList<>(); md.add(attributeValue)

R6	Applied when: addAllAttributes function is called	R1	addAllAttributes(attributeValues); # attributeValue is of type Collection<?>	ArrayList <String,String>md = new ArrayList<>(); md. add(attributeValues)
R7	Applied when: addAllAttributes function is called	R1	addAllAttributes(attributes); # attributes is of type Map<String,?>	ArrayList<String,String>md = new ArrayList<>(); md.add(attributes)
R8	Applied when: mergeAttributes function is called	R1	mergeAttributes(attributes); # attributes is of type Map<String,?>	ArrayList<String,String>md = new ArrayList<>(); md.add(attributes)
R9	Applied when: containsAttribute function is called	R1	containsAttribute(attribute); # attribute is of type String	# deleted
R10	Applied when: getAttribute function is called	R1	getAttribute(attribute); # attribute is of type String	# deleted
R12	Applied when: ModelAndView function is called	R11	ModelAndView mav = new ModelAndView()	# look for mav references move its data then delete it
R13	Applied when: ModelAndView function is called	R11	ModelAndView mav = new ModelAndView(viewName)	# look for mav references move its data then delete it
R14	Applied when: ModelAndView function is called	R11	ModelAndView mav = new ModelAndView(view)	# look for mav references move its data then delete it
R15	Applied when: ModelAndView function is called	R11	ModelAndView mav = new ModelAndView (viewName,Model) #model is type of Map<String,?>	HashMap<String,?>model = new HashMap<>(); Model.putAll(Model);
R16	Applied when: ModelAndView function is called	R11	ModelAndView mav = new ModelAndView (viewName,status) #model is type of HttpStatus	HttpStatus hs = status ;
R17	Applied when: ModelAndView function is called	R11	ModelAndView mav = new ModelAndView (viewName,model,status) #model is type of HttpStatus	HashMap<String,?>model = new HashMap<>(); Model.putAll(Model); HttpStatus hs = status ;
R18	Applied when: ModelAndView function is called	R11	ModelAndView mav = new ModelAndView (modelName,ModelObject) #ModelObject is of type Object	HashMap<String,Object>model= new HashMap<>(); Model.put(ModelName, ModelObject);
R19	Applied when: setViewName function is called	R11	setViewName(viewname)	# delete
R20	Applied when: getViewName function is called	R11	getViewName()	# delete
R21	Applied when: setView function is called	R11	setView(view)	# delete
R22	Applied when: getView function is called	R11	getView()	# delete
R23	Applied when: hasView function is called	R11	hasView()	# delete

R24	Applied when: isReference function is called	R11	isRefrence()	# delete
R25	Applied when: getModelInternal function is called	R11	getModelInternal()	# delete
R26	Applied when: getModelMap function is called	R11	getModelMap() #return Model (need review)	# delete
R27	Applied when: getModel function is called	R11	getModel() #return Model (need review)	# delete
R28	Applied when: setStatus function is called	R11	setStatus(status)	# HttpStatus hs = status ;
R29	Applied when: getStatus function is called	R11	getStatus()	# delete
R30	Applied when: addObject function is called	R11	addObject(attributeName, attributeValue)	HashMap <String,Object>model = new HashMap<>(); Model.put(attributelName, attributeVAlue);
R31	Applied when: addAllObjects function is called	R11	addAllObjects(modelMap) # attribute-Value is of type Map<String,?>	HashMap<String,?>model = new HashMap<>(); Model.putAll(modelMap);
R32	Applied when: clear function is called	R11	clear()	# delete
R33	Applied when: isEmpty function is called	R11	isEmpty()	# delete
R34	Applied when: wasCleared function is called	R11	wasCleared()	# delete
R35	Applied when: toString function is called	R11	toString()	# delete
R36	Applied when: formatView function is called	R11	formatView()	# delete

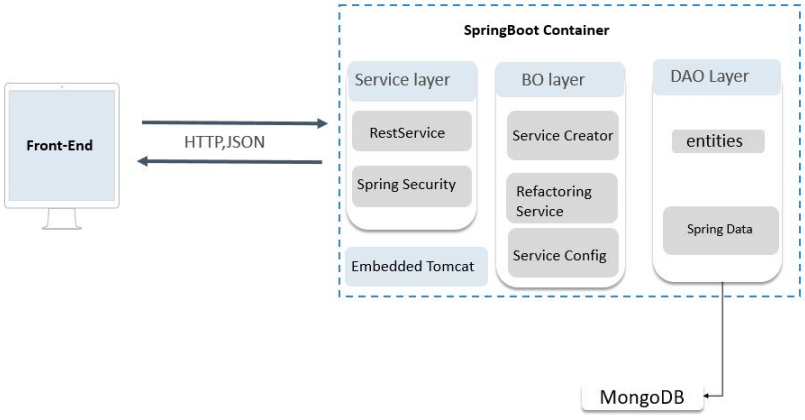

Fig. 3. Tool's Backend Architecture.

The refactoring task will be based on a set of rules (as shown in Table I) predefined by the tool to modernize the legacy application. Right after the refactoring, the tool can package the new application into a container which enables it to be cloud compatible, and in the final stage, the software architect will be able to download the newly modernized application.

The main target of this work is designing and developing a tool that can modernize java applications. The legacy application will be used as input to the tool. The refactoring task will be performed at the controller's level in the web layer. The refactoring task will be based on a set of rules (as shown in Table I) predefined by the tool to modernize the legacy application. Right after the refactoring, the tool can package the new application into a container which enables it to be cloud compatible, and in the final stage, the software architect will be able to download the newly modernized application. The proposed tool has been divided into two separate architectures: front-end architecture which represents the tool views and the Back-End architecture which represents the backbone of the developed tool and its functionalities. The front-end architecture is being developed using the Angular framework and comprises four (4) main modules:

- App Module: represent the starting point of the application such as index.html.
- Core Module: contains singleton components and services such as toolbar components.
- Feature Module: provides a feature that the application offers such as file upload.
- Shared Module: contains highly reusable components and services such as services that communicate with the back-end.

The core components (as illustrated in Fig. 3) of the back-end architecture are the following:

- Service Layer: handle incoming requests and security mechanisms such as authorization and authentication.
- Business Layer: concerned with service creation, refactoring, and packaging.
- Data access object layer: persist data into database.

IV. Correctness and Efficiency Evaluation

In this study, we have conducted a survey to conduct an initial evaluation of the proposed semi-automated approach. The subsequent questions are formulated based on the research problem:

- Q1: Does the semi-automated approach promote efficiency in transforming legacy architecture into modern system architecture?
- Q2: How is the correctness of the generated system using a semi-automated approach?

A. Evaluation Setting

Since this work is an initial work of microservice modernisation, we conducted an initial expert evaluation to evaluate the effectiveness and the correctness of our approach. We selected three (3) experts to evaluate our approach (based on our developed tool). All of the selected experts have at least five years of experience and have a software engineering background. The evaluation setting is the following:

- Step 1: The experts were asked to manually modernise a simple architecture module of software to microservice architecture.
- Step 2: The experts were introduced to the tool and allowed to use the tool.
- Step 3: The experts were asked to modernise a simple architecture module of software to microservice architecture by using the semi-automated tool.
- Step 4: The experts were required to answer several questions regarding the tool's correctness and efficiency in performing the modernisation task.

B. Results

As mentioned above, we aim at evaluating the correctness and the effectiveness of our approach (by using the semi-automated tool) in performing modernisation tasks. The result is based on the experts' answers to our survey.

1) Correctness: After performing all the required steps, we asked the experts about the error found (on average) when they use the manual approach and the error found after performing the modernisation task using our approach. In terms of the manual approach, one (1) expert mentioned that the 0 errors were found and the other two (2) experts stated that 1 – 5 errors were found in the manual approach. On the other hand, when we asked the experts on the correctness of our approach, all experts mentioned that (on average) they found 1 – 5 errors after they used our approach. From this result, we can summarize that the manual approach still produces better accuracy than using our approach. However, the result shows that the difference is not far (as illustrated in Fig. 4).

2) Efficiency: To evaluate the efficiency of our approach based on the semi-automated tool, after performing all the steps in Section IV, the experts were required to answer on the time to complete the tasks. By using a manual approach, all experts mentioned that they need to spend more than 4 hours to complete the microservice modernisation task. In contrast, by using the semi-automated approach, the experts stated that the modernisation task will be faster. One (1) expert mentioned that it takes 1-2 hours, one (1) expert mentioned that it takes 2-3 hours and another expert stated that it will take 3-4 hours. With this, we can summarize that by using our semi-automated approach, the modernisation task will be faster to complete compared to the manual approach (as illustrated in Fig. 5).

V. Discussion

This chapter discusses the implications of the practical and theoretical implications of the study as well as a list of the limitation of this approach.

A. Practical Implications

To better understand the practical implication of this study will compare it to previous studies from an enterprise's point of

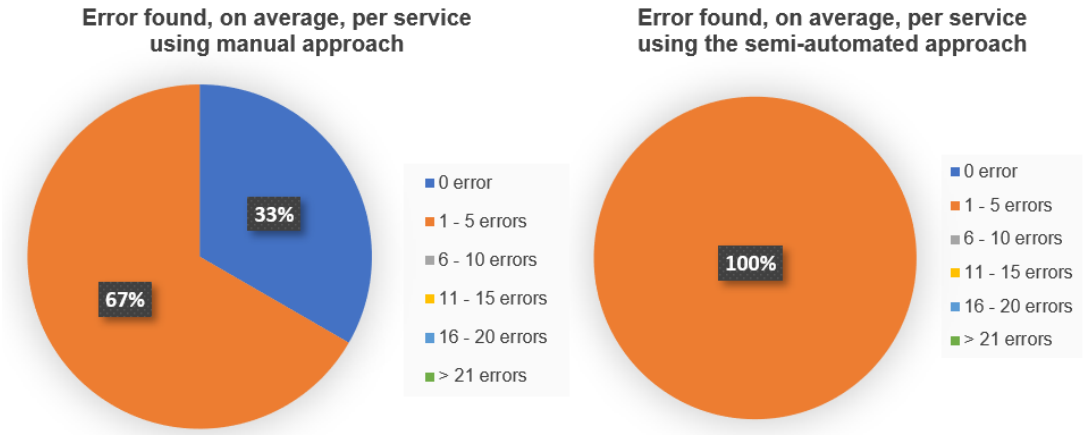

Fig. 4. Evaluation on the Tool's Correctness.

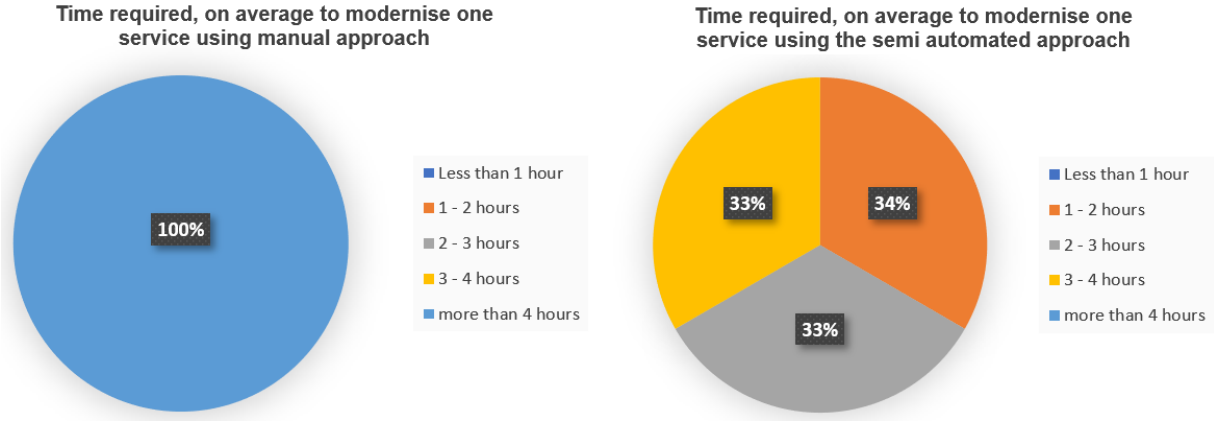

Fig. 5. Evaluation on the Tool's Efficiency.

view. Service cutter is a tool that assists in software transition from monolith to microservice by taking monolith domain model as input and suggested new decomposed domain model as microservices. On the other hand, our approach, Modernator is a tool that has been proposed and developed by this study which takes as an input monolith application(spring MVC) and does refactoring on the web layer, and produces microservice (spring boot) as an output.

We believe as organizations are looking to ease the transition from monolith to microservice while reducing the number of resources allocated to the transition in this case Modernator will have a positive impact on their transition as it produces production source code that can be alerted if needed comparing to service cutter that will produce domain model that needs to be implemented from scratch. However, Modernator will be useful only in the spring framework environment but since spring framework is widely used by enterprises as the backbone of their information systems that give Modernator a high rate to be widely used as well.

B. Theoretical Implications

This study has introduced a list of refactoring rules that can be used to refactor legacy spring MVC controllers that in nature returns JSP pages to modern spring boot controllers that return data in JSON format that is compatible with modern architecture such as microservice.

The construction of the rules has been made by following spring framework documentation and by identifying spring MVC legacy classes and mapping them to spring boot classes. The proposed rules can be dived into two type function signature which focuses of refactoring legacy function signature and function boy that find legacy function calls. We were able to construct around 40 rules, however; these rules do not include all possible cases there might be other legacy classes that we did not discover in this study.

C. Limitation

Since this work is an initial work for this study, we see a lot of possibility to improve this work based on the following limitations:

- Lack of boarder Evaluation.
- Support limited to java and more specifically application built with spring framework.
- Did not test all proposed refactoring rules.
- Refactoring focused only on Web Layer.
- Did not propose all possible refactoring rules.

VI. Conclusion and Future Work

The motivation behind this study is to fulfill the need of the industry to have a semi-automated approach for easy and smoothly transforming legacy system architecture to modern architecture. As the manual transition is inefficient due to its time-intensive and the significant amount of effort required. Therefore, this research proposed to develop a semi-automated tool to provide transformation rules and guidelines. An Expert Evaluation was conducted to evaluate the correctness and the efficiency of the approach. Lastly, this research has proved that the new approach promotes an efficient migration process and produces correct software to some extent.

For future work, we plan to conduct a Broader Evaluation of the proposed refactoring rules to further validated their correctness as well as constructing new rules for refactoring other layers such as the business layer and data access object layer.

References

[1] Fritzsch J, Bogner J, Zimmermann A, Wagner S. From monolith to microservices: A classification of refactoring approaches. InInternational Workshop on Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment 2018 Mar 5 (pp. 128-141). Springer, Cham.

[2] Mazlami G, Cito J, Leitner P. Extraction of microservices from monolithic software architectures. In2017 IEEE International Conference on Web Services (ICWS) 2017 Jun 25 (pp. 524-531). IEEE.

[3] Li S, Zhang H, Jia Z, Li Z, Zhang C, Li J, Gao Q, Ge J, and Shan Z, "A dataflow-driven approach to identifying microservices from monolithic applications," *J. Syst. Softw.*, vol. 157, p. 110380, Nov. 2019, doi: 10.1016/j.jss.2019.07.008.

[4] Kamimura M, Yano K, Hatano T, Matsuo A. Extracting candidates of microservices from monolithic application code. In2018 25th Asia-Pacific Software Engineering Conference (APSEC) 2018 Dec 4 (pp. 571-580). IEEE.

[5] Gysel M, Kölbener L, Giersche W, Zimmermann O. Service cutter: A systematic approach to service decomposition. InEuropean Conference on Service-Oriented and Cloud Computing 2016 Sep 5 (pp. 185-200). Springer, Cham..

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

[7] Knoche H, Hasselbring W. Using microservices for legacy software modernization. IEEE Software. 2018 May 4;35(3):44-9.

[8] Ahmadvand M, Ibrahim A. Requirements reconciliation for scalable and secure microservice (de) composition. In2016 IEEE 24th International Requirements Engineering Conference Workshops (REW) 2016 Sep 12 (pp. 68-73). IEEE.

[9] Ren Z, Wang W, Wu G, Gao C, Chen W, Wei J, Huang T. Migrating web applications from monolithic structure to microservices architecture. InProceedings of the Tenth Asia-Pacific Symposium on Internetware 2018 Sep 16 (pp. 1-10).

[10] Abdullah M, Iqbal W, Erradi A. Unsupervised learning approach for web application auto-decomposition into microservices. Journal of Systems and Software. 2019 May 1;151:243-57.

[11] Shimoda A, Sunada T. Priority order determination method for extracting services stepwise from monolithic system. In2018 7th International Congress on Advanced Applied Informatics (IIAI-AAI) 2018 Jul 8 (pp. 805-810). IEEE.

[12] Carrasco A, Bladel BV, Demeyer S. Migrating towards microservices: migration and architecture smells. InProceedings of the 2nd International Workshop on Refactoring 2018 Sep 4 (pp. 1-6).

[13] Kaplunovich A. ToLambda–Automatic Path to Serverless Architectures. In2019 IEEE/ACM 3rd International Workshop on Refactoring (IWoR) 2019 May 28 (pp. 1-8). IEEE.

[14] Mayer B, Weinreich R. An approach to extract the architecture of microservice-based software systems. In2018 IEEE symposium on service-oriented system engineering (SOSE) 2018 Mar 26 (pp. 21-30). IEEE.

[15] Tyszberowicz S, Heinrich R, Liu B, Liu Z. Identifying microservices using functional decomposition. InInternational Symposium on Dependable Software Engineering: Theories, Tools, and Applications 2018 Sep 4 (pp. 50-65). Springer, Cham.

[16] Jin W, Liu T, Zheng Q, Cui D, Cai Y. Functionality-oriented microservice extraction based on execution trace clustering. In2018 IEEE International Conference on Web Services (ICWS) 2018 Jul 2 (pp. 211-218). IEEE.

[17] Balalaie A, Heydarnoori A, Jamshidi P. Migrating to cloud-native architectures using microservices: an experience report. InEuropean Conference on Service-Oriented and Cloud Computing 2015 Sep 15 (pp. 201-215). Springer, Cham.

[18] Gouigoux JP, Tamzalit D. "Functional-First" Recommendations for Beneficial Microservices Migration and Integration Lessons Learned from an Industrial Experience. In2019 IEEE International Conference on Software Architecture Companion (ICSA-C) 2019 Mar 25 (pp. 182-186). IEEE.

[19] Mazzara M, Dragoni N, Bucchiarone A, Giaretta A, Larsen ST, Dustdar S. Microservices: Migration of a mission critical system. IEEE Transactions on Services Computing. 2018 Dec 21.

[20] De Alwis AA, Barros A, Fidge C, Polyvyanyy A. Discovering microservices in enterprise systems using a business object containment heuristic. InOTM Confederated International Conferences" On the Move to Meaningful Internet Systems" 2018 Oct 22 (pp. 60-79). Springer, Cham.

[21] Eski S, Buzluca F. An automatic extraction approach: Transition to microservices architecture from monolithic application. InProceedings of the 19th International Conference on Agile Software Development: Companion 2018 May 21 (pp. 1-6).

[22] Sayara A, Towhid MS, Hossain MS. A probabilistic approach for obtaining an optimized number of services using weighted matrix and multidimensional scaling. In2017 20th International Conference of Computer and Information Technology (ICCIT) 2017 Dec 22 (pp. 1-6). IEEE.

[23] Baresi L, Garriga M, De Renzis A. Microservices identification through interface analysis. InEuropean Conference on Service-Oriented and Cloud Computing 2017 Sep 27 (pp. 19-33). Springer, Cham.

[24] Escobar D, Cárdenas D, Amarillo R, Castro E, Garcés K, Parra C, Casallas R. Towards the understanding and evolution of monolithic applications as microservices. In2016 XLII Latin American Computing Conference (CLEI) 2016 Oct 10 (pp. 1-11). IEEE.