Some Studies on Migration of SOA Based Applications to Microservices Architecture

Submitted in partial fulfillment of the requirements

for the award of the degree of

DOCTOR OF PHILOSOPHY

Submitted by

Vinay Raj

(Roll No. 716173)

Under the guidance of

Dr. S. Ravichandra

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY WARANGAL
TELANGANA - 506004, INDIA
September 2021

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY WARANGAL
TELANGANA - 506004, INDIA

THESIS APPROVAL FOR Ph.D.

This is to certify that the thesis entitled, Some Studies on Migration of SOA Based Applications to Microservices Architecture, submitted by Mr. Vinay Raj [Roll No. 716173] is approved for the degree of DOCTOR OF PHILOSOPHY at National Institute of Technology Warangal.

Examiner

Research Supervisor
Dr. S. Ravichandra
Dept. of Computer Science and Engg.
NIT Warangal
India

Chairman
Prof. P. Radha Krishna
Head, Dept. of Computer Science and Engg.
NIT Warangal
India

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY WARANGAL
TELANGANA - 506004, INDIA

CERTIFICATE

This is to certify that the thesis entitled, Some Studies on Migration of SOA Based Applications to Microservices Architecture, submitted in partial fulfillment of requirement for the award of degree of DOCTOR OF PHILOSOPHY to National Institute of Technology Warangal, is a bonafide research work done by Mr. Vinay Raj (Roll No. 716173) under my supervision. The contents of the thesis have not been submitted elsewhere for the award of any degree.

Research Supervisor
Dr. S. Ravichandra
Associate Professor
Dept. of CSE
NIT Warangal
India

Place: NIT Warangal
Date: 13 September, 2021

DECLARATION

This is to certify that the work presented in the thesis entitled "*Some Studies on Migration of SOA Based Applications to Microservices Architecture*" is a bonafide work done by me under the supervision of Dr. S. Ravichandra and was not submitted elsewhere for the award of any degree.

I declare that this written submission represents my ideas in my own words and where others ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/date/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Vinay Raj

(Roll No. 716173)

Date: 13-09-2021

ACKNOWLEDGMENTS

It is with great pleasure that I acknowledge my sincere thanks and a deep sense of gratitude to my supervisor Dr. S. Ravichandra for his invaluable guidance to complete the work. He always gave me ample time for discussions, reviewing my work and suggesting requisite corrections, which enabled me to attain my objective in time. He has provided me all kinds of inputs directly with his words, indirectly with his values not only to be a good teacher also to be a good human being. His sincerity and commitment to every aspect of the life has influenced me greatly. He also gave me the freedom of work and supported me in every aspect of life. I want to inculcate all the great qualities of him for the rest of my life.

I extend my gratitude to all my Doctoral Scrutiny Committee members Prof. D.V.L.N. Somayajulu, Prof. R.B.V. Subramaanyam, Prof. D. Srinivasacharya, Dr. Ch. Sudhakar and Dr. Rashmi Ranjan Rout for their insightful comments and suggestions during oral presentations.

I am also grateful to Dr. K. Ramesh, Dr. U.S.N. Raju, Dr. P.V. Subba Reddy and Dr. M Sreenivas, who have always been supportive and encouraging all through my tenure. I am immensely thankful to Dr. Ch. Sudhakar, Prof. R.B.V. Subramaanyam, and Prof. P. Radha Krishna, Heads of Dept. of CSE during my stay in the department, for providing adequate facilities to complete my research work. I convey my special thanks to D. Govinda Rao sir and K.S.S.S Padmasri madam, senior technical assistant in the Department of Computer Science, for their support during my tenure. I wish to express my thanks to faculty members of Computer Science and Engineering department. I also express my thanks to Prof. N.V. Ramana Rao, Director, NIT Warangal for his official support and encouragement.

I have a very special thanks to my friends Vukam Raviteja, Srikar Menneni, Maneesh Aucharla, Akash Makthabai, Vamshi Krishna Samala, Nekkala Sandeep, Byreddy Vamshi Krishna Reddy, Abdul Gaffar Sheik, Appani Harish, Bandari Rajeev, Bolli Sumanth and Bhukya Sharath Chandra Rathod for their unconditional love, support and readiness to help me at any time. They stood beside me both during the happy and difficult times of

my research work and also in my life. My Ph.D. would have been difficult without these people and I will be forever grateful to have such friends in my life.

I would like to express my gratitude to my friends Sandeep Patlolla, Kiran Patrudu Gopalasetty, Vasav Nayak, Vemula Narender Reddy, Yasa Srikanth, Yada Sai Kumar, Veleti Sujith, Battini Vivek Goud, Baki Ravinder, Rahul Dubey, Kanishk Vohra, Sachin Nalluri and B.Tech 2008-2012 CSE batchmates for supporting me financially without which it would have been difficult for me to complete Ph.D and I will always be grateful to them for trusting me.

I would also like to specially thank K. Sandhya Reddy, Technical Director @ Talentio Solutions India Pvt Ltd for giving me good opportunities to explore and improve my teaching skills which in turn helped me financially. I would be grateful to her for the encouragement and motivation in handling challenges.

I wish to express my profound thanks to Dr. J. Pavan Kumar @ ICFAI University Hyderabad, Dr. M. Sai Krishna @ NIT Trichy, Dr. A. Sudharshan Chakravarthy @ BVRIT Hyderabad and Nalluri Sesha Kumar @ RGUKT Srikakulam for their valuable suggestions, constructive feedback in completing my research work. Their contribution to me has enriched my life both personally and professionally.

I would also like to thank my co-scholars of Computer Science department, Uma Sankara Rao, Hemkumar D, Umamaheswara Sharma, Punnam Chander, Preethi, Manoj, Abhilash, K Suresh Kumar, Satish Babu, Murukessan, Sachin Sood, and Vijay Chekravarthy for their valuable suggestions and for extending selfless cooperation. Special thanks to my friend and co-scholar Punnam Chander for his time and moral support.

I would like to express my sincere thanks to P.G students Nikhil Tarte, Nilesh Sinha, Issa Baddour and Amandeep Singh, B.Tech students Suraj Baradhi, Anirudh Nanduri and Anirudh Avirewho have completed their project work under my guidance from CSE Dept, NIT Warangal.

I extend my heartfelt thanks to scholars of other departments, Dr. Purushotham, Dr. Sasidhar, M Shekar, Katti Bharath, Dr. Sai Mahesh Yadav, Vemoori Raju, Oggu Praveen, Siliveri Suresh and Gandamalla Ambedkar for their time and cooperation for the completion of my research work.

I would also like to express my love to some of my good friends Katikala Karthikeya, P Sandeep Reddy, Kothagatla Sai Srikar, Chintala Sanjay Kumar, Dhreeravath Srinivas, K Siva Koti Reddy, Gadwal Narendra Kumar, Cheruku Hemanth Reddy, Chitikeshi Rohith, B Praveen Kumar, Arun Kumar Gandhi, D Benny, L Shivateja Reddy, Srinivas Polisetti, and 2006 SSC batchmates of Montessori High School, Yellandu who have always encouraged me positively during my hard times.

I would like to express my gratitude to my parents for their invaluable sacrifice for my education and trust they have. My gratitude to my family for their unconditional love, support and prayers for my success in achieving the goal. I am very much thankful to my family for all the support during the time I carried out my thesis work. My mother and father have made so many sacrifices in their life by having a deep trust in me to see where I am today. My brother Arun Kumar, who always wish for my best, was more like a best friend in helping me make better decisions and supporting me always in whatever help I need. I would like to express thanks to my sisters Santhoshi, Matheshwari & Vigneshwari for the love and care they have for my and my well being. I would also express thanks to my brother-in-law Mr. Deepak Kumar for this timely support and keeping us happy always with his presence. Lots of love for my nephew Rushikesh for fighting with me always and making me happy. Lastly, my deep sense of gratitude to my god, Lord Tirupati Venkateshwara Swamy for all the things that come into my life.

Vinay Raj

Dedicated to

My Family & Friends

ABSTRACT

Distributed systems have evolved rapidly as the demand for independent design, and deployment of software applications has increased. It has emerged from the monolithic style of client-server architecture to service-oriented architecture, and then to the trending microservices. Monolithic applications are difficult to update, maintain, and deploy as it makes the application code very complex to understand. To overcome the design and deployment challenges in monolithic applications, service oriented architecture has emerged as a style of decomposing the entire application into loosely coupled, scalable, and interoperable services. Though SOA has become popular in the integration of multiple applications using the enterprise service bus, there are few challenges related to delivery, deployment, governance, and interoperability of services. Additionally, the services in SOA applications are tending towards monolithic in size with the increase in changing user requirements. To overcome the design and maintenance challenges in SOA, microservices has emerged as a new architectural style of designing applications with loose coupling, independent deployment, and scalability as key features. Due to this paradigm shift in software development, many existing SOA applications are being migrated to microservices. However, some architects are in chaos, whether to migrate the application from SOA to microservices or not as they are unaware of the pros and cons of the migration. Also, there is no proper mechanism or strategy to migrate existing SOA applications to microservices.

The main objectives of this thesis work include: (i) to find the aspects of an SOA-based application that makes it suitable for migration to microservices architecture, (ii) to propose a framework for extraction of microservices and evaluate the QoS values of generated microservices, (iii) proposing an effort estimation technique and find its effectiveness, and (iv) present patterns for challenges which occur during the migration process.

In this thesis, to achieve the above mentioned objectives, we conducted some studies on migration of SOA based applications to microservices architecture. Firstly, we compare both SOA based services and microservices architectures with two different parameters, (i) *Complexity* with architectural metrics and (ii) *Performance* with load testing. We propose a formal model called as *Service Graph (SG)* which plays a significant role in the compar-

ison. It is clear from the results that though the complexity of microservices architecture is high, the response time for processing the requests is very fast compared to SOA services. Secondly, as it is clear that microservices have better QoS values, algorithms for the construction of service graphs and extraction of microservices from SOA applications are proposed. In particular, four algorithms are defined: i) construction of *Service Graph* (SG), ii) construction of *Task Graph* (TG) for each service of the SOA application, iii) extraction of candidate microservices using the service graph of SOA application, and iv) construction of service graph for microservices application to retain the dependencies between the generated microservices. Thirdly, a new effort estimation model called *Service Points* recasted from the use case points method is proposed. We also apply multiple regression analysis on the proposed approach with the Leave-N-Out policy. The proposed model is compatible with the design principles of microservices and provides a systematic and formal way of estimating the effort. Finally, we aim to provide patterns for the most recurring problems which occur during the migration process, i.e., the decomposition of SOA services, the size of each microservice, and the detection of anomalies in microservices. All the proposed algorithms and techniques are demonstrated and evaluated using a standard web-based application.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Applications developed to fulfill distributed systems needs have been growing rapidly as the demand for independent design, and deployment of software applications has increased [1]. Today's world demands the timely delivery of business needs and all that needs to be online. Distributed systems play a key role in the timely delivery of services with continuous integration and continuous development. It has emerged from the monolithic style of client-server architecture to service-oriented architecture, and then to the trending microservices.

Monolithic applications are built as a large block of code and deployed as a single archive file. Monolithic applications are difficult to update, maintain, and deploy as it makes the application code very complex to understand [2]. To overcome the design and deployment challenges in monolithic applications, service oriented architecture has emerged as a style of decomposing the entire application into loosely coupled, scalable, and interoperable services. The Service Oriented Architecture (SOA) is an architectural style of distributed applications with service as the main design component where service is a reusable software code that performs various business tasks that can be simple or complex based on the business requirements. Though SOA has become popular in the integration of multiple applications using the enterprise service bus, there are still few challenges related to delivery, deployment, governance, and interoperability of services. Additionally, the services in SOA applications are tending towards monolithic with the increase in changing user requirements. Monolithic services make the application complex, and it becomes dif-

ficult to maintain. Also, since SOA services are tightly coupled with ESB for the exchange of messages, and if there is a need to update a particular service, it requires redeploying the dependent components in the system. Therefore, the deployment process of SOA is still seen as a monolith. Further, the on-demand services can be scaled horizontally, but the hardware cost increases as it requires additional infrastructure. To overcome these design, deployment and maintenance challenges which are stated above in SOA, microservices have emerged as a new architectural style of designing applications with loose coupling, independent deployment, and scalability as key features.

Microservices is defined as an architectural style for developing applications as a suite of small and independent components. Each microservice runs in its own process and communicates with other services using lightweight protocols such as Representational State Transfer (REST) and HTTP [3]. Loose coupling and the freedom of choosing programming languages for the implementation of microservices are some of the major benefits. Microservices are deployed in docker containers which are lightweight, and they are best suitable for microservices as they start very quickly [4]. Container images consist of all required environmental configurations, and developers can easily access them from DockerHub. Microservices can be easily added or removed from the applications, and they can be easily migrated from one host to another. Independent deployment helps in auto-scaling of the microservices at a fast pace and can easily handle the load. Microservices enable continuous integration and continuous delivery and suit well with DevOps style, which acts as a framework for the complete Software Development Life Cycle (SDLC) of microservices [5]. Compared to traditional architectures, microservices are designed in short development time, reduces the inherent complexity, and increases the scalability [6].

1.1 Motivation

Because of the diverse benefits, IT companies have started designing their applications using microservices architecture, and few of them have started migrating their applications to microservices [7]. Due to this paradigm shift in software development, many existing SOA applications are being migrated to microservices. However, some architects are in chaos

whether to migrate the application from SOA to microservices or not as they are unaware of the pros and cons of the migration. As architects are unaware of the effort and cost required for designing the application from scratch, hence migrating is the best approach [8]. According to a systematic mapping study conducted by Di Francesco et al. [9], the research for the migration of legacy applications to microservices is at an early stage and it also highlights the multiple open challenges involved during migration process. Moreover, migrating applications to the cloud have also aroused for migrating to microservices as it suits better in the cloud environment [10]. Migrating the existing legacy application to microservices minimized the technical debt and improved the maintenance [11]. Because of these benefits, the industry is moving towards microservices, leaving SOA as a legacy system. Since the migration of the SOA based applications to microservices is an open challenge[12], we consider it as the major research problem in this thesis. However, the migration of systems towards microservices involves multiple difficulties [13] such as (i) not knowing the impact of migration, (ii) not having enough material on migration techniques, and (iii) not being aware of the migration effort. These challenges motivated us to study and propose possible solutions for the migration of SOA based applications to microservices architecture. The contributions in this thesis are as follows:

- **Evaluation and comparison of SOA and microservices architecture based applications:** In this work, the QoS attributes such as coupling, complexity, and performance are considered for comparing both SOA and microservices architectures. The complexity of the application is measured with architectural metrics, and performance is compared with load testing. A graph based model called *Service Graph* is designed in which each node is a service, and the edge between the node represents the dependency among the services. This service graph plays a vital role in the proposed comparison strategy. The results conclude that though the complexity of the microservices based application is high, it exhibits better QoS values compared to SOA application.

- **A service graph based extraction of microservices from monolith services of SOA:** Since it is observed from the above work that microservices architecture is

better, we migrate the SOA based applications to microservices architecture. This work presents a 3-step approach to extract the microservices from monolith services of SOA. The concept of *Task Graph* is introduced in this work which helps in choosing the monolith SOA services in the application. Four different algorithms, namely, Service Graph Construction for SOA, Task Graph Construction for each Service of SOA, Microservices Extraction Algorithm, and Service Graph Construction for Microservices Architecture are presented. A comparison of extracted microservices with SOA services with respect to coupling is also presented.

- **Effort estimation approach for migration of SOA applications to microservice architecture:** Before the actual migration of the application, estimating the effort required for migration helps the architects and project managers to better plan and execute the migration process. Hence, in this work, *Service Points* approach (based on the service graph concept) is defined, which is recasted from the use case points approach of effort estimation. Machine learning concepts such as multiple linear regression and Leave-N-Out policy are applied to the proposed service points approach for better prediction of the effort. Measures such as Magnitude of Relative Error (MRE), Mean of MREs (MMRE), Root Mean Square Error (RMSE), Mean Absolute Error (MAE) and Standardized Accuracy (SA), etc., are used to evaluate the accuracy of the proposed model.

- **Patterns for migration of SOA based applications to microservices architecture:** The migration of an application from one architecture to another poses many design challenges. Similarly, the migration of SOA based applications to microservices also exhibits many recurring challenges. This work presents patterns for commonly occurring problems during the migration of SOA based applications to microservices architecture. In particular, patterns for decomposition of an SOA service into microservice, size of the microservice, and bug detection in the complex microservices application are presented. All these patterns are based on the concept of service graph and task graph proposed in the above works.

1.2 Research Problems and Objectives

The aim of the research work done in this thesis is to help software architects in understanding the technical differences between SOA and microservices architectures. Also, it helps in the migration of SOA based applications to microservices architecture. The problems addressed in this thesis are:

- Is migration to microservices from SOA a good decision? If it's a good decision, how can we comment on it? It triggers to compare both the architectures, but how can we compare both SOA and microservices based applications?
- How to migrate SOA based applications to microservices architecture? How to identify and extract the microservices from the SOA applications? Do the extracted microservices exhibit better QoS values than the SOA services?
- How to estimate the effort for this new architectural style of microservices? Can we use the existing effort estimation models for microservices? How to estimate the effort required for migration of SOA application to microservices?
- How to split the monolithic services in the SOA based application to microservices? What should be the size of each microservice? On what basis can we measure the size of the microservices? How to trace the service which is responsible for the bug?

Based on the above research problems, the following objectives have been defined for the research work done in the thesis.

- **To define graph based models called as *Service Graph* and *Task Graph* which represents any service based application.**
- **To compare both SOA and microservices architecture based applications in terms of coupling, complexity and performance.**
- **To define algorithms for the construction of service graph & task graph and propose an approach for extraction of microservices from SOA based applications.**

- **To develop an effort estimation model for estimating the effort required for migration of SOA based applications to microservice architecture.**
- **To define new patterns for the challenges which occur during migration from SOA to microservices architecture.**

1.3 Summary of the contributions

To achieve the desired objectives listed above for migration of SOA based applications to microservices architecture, solutions for four different yet interrelated problems are designed in this thesis work. In this section, an overview of the chapter wise contributions is presented.

1.3.1 Evaluation and comparison of SOA and microservices architecture based applications.

As both SOA and microservices architectures have services as the main component, a graph based model called as Service Graph (SG) is formulated to represent any given service based application where each node represents a service and the edge between the nodes represent the dependency between services. Service graphs can be used in software engineering activities like effort and cost estimation, fault detection, and monitoring of the services.

We present a comparison of a web application that is designed using both SOA and microservices architectures. The comparison is presented with two different parameters, (i) *Complexity* with architectural metrics and (ii) *Performance* with load testing. We have adapted the metrics for measuring the complexity of the software architectures and extract the metric values from service graph representation. To measure the complexity of the architectures, two metrics, namely Total Complexity and Global Complexity are used. Additionally, metrics related to loose coupling such as Number of Services (NoS), Coupling of Services (CS) and Relative Coupling of Services (RCS) are also presented, which are used in evaluating the global complexity metric. Performance related features such as response

time are of more interest to determine the acceptability of software design. Load testing is performed using the JMeter tool with 500 & 1000 users of load on both the applications, and the average response time is captured for 500 and 1000 users separately for all services.

The values of both total and global complexities for both SOA and microservices based applications are compared and the results show that the microservices based application is more complex when compared with SOA based application. It is observed that a 50% increase in the number of services has lead to an 84% increase in the dependencies in the microservices application. In order to compare the performance of both the architectural styles, we define Business Requests (BR) according to the functionality of the case study application. The sequence of services invoked for processing a particular business functionality is stored as a sequence with the service numbers. By understanding the complete case study application, we identified seven major business requests. The time taken for each of these business requests in both SOA and microservices based applications is considered as criteria for comparison. The average response times of each of the services involved in the business requests are added to get the response time of the complete business request. The comparison results show that the average response time for completing the business requests in SOA based application is high compared to microservices application. For a detailed analysis of the performance, we consider two different scenarios w.r.t to the number of services involved in getting the response of a business request.

1.3.2 A service graph based extraction of microservices from monolith services of SOA

In this work, an approach to extract the candidate microservices from SOA based applications using graph based algorithms is presented. To migrate SOA based web services to microservices, a three-step approach is defined.

In the first step, the service graph construction algorithm is presented for the given SOA application, and for each service, task graphs are constructed using another algorithm. To construct the service graph for SOA application, an API document that contains the complete information about the services, operations in each service, and input/output parame-

ters of each operation are used. If the application is implemented using web services, then we will have a WSDL file as the API document, and if we implement the application as normal services, then we have XML format of the complete application. Considering the API document (WSDL or XML file) of the SOA based application as input, we extract the serviceNames, and then the inputs and outputs for each of the serviceNames (operations) are extracted. Each serviceName is marked as a node, and the edges between the nodes of the service graph are generated by mapping the inputs and outputs of services. After constructing the service graph, iteratively, we call the task graph construction algorithm for each of the services to generate the task graph inside each node of service graph using the API document of individual service. The input of Algorithm 1 is the API of the complete application, whereas the input for each task graph is the API of the individual service of the application.

Using the service graph representation (including task graphs in each node of service graph), a microservices extraction algorithm to generate the candidate microservices is presented in the second step. In the algorithm, we find out the order of each node of the service graph to find out the nodes which are monolithic in nature. If the order of the node is one, then it is directly considered as a microservice. For the services with order more than one will be treated as monolithic services. Each node of the task graph is iterated, and all the nodes of the task graph will be considered as microservices.

Finally, the service graph construction algorithm for the microservices application is proposed, which helps in retaining the dependencies between microservices. The service graph (along with the task graph in each node of the service graph) is considered as input for the algorithm. Considering the extracted microservices (from step 2) as nodes, the dependencies between the nodes of task graph of each service as broken and new dependencies are created between the extracted microservices. The newly formed dependencies are the edges for the service graph for microservices. The nodes in the graph will represent a microservice, and the edge between the node represents the dependency between the microservices.

Additionally, the extracted microservices are compared with SOA services in terms of QoS parameters such as loose coupling. The coupling intensity is calculated using Service

Coupling Factor (SCF) metric, and the results show that microservices have lesser coupling values compared to SOA services.

1.3.3 Effort estimation approach for migration of SOA applications to microservice architecture.

In this work, an effort estimation approach called Service Points which is recasted from the use case points model is proposed. Service graph plays an important role in this approach as the details of the services and edges are used are input parameters. At first, the types of services involved during the migration process, such as Available Service, New Service, Migrated Service and Composed Service are presented. However, only migrated services will be considered for estimating the effort in our proposed approach. The major steps involved in the proposed effort estimation approach using service graph as follows: (i) classification of the services, (ii) calculation of weights and points, (iii) calculation of TCF and EF, and (iv) final service point evaluation.

The services of the application are classified as simple, average and complex based on the interactions each service has with other services. A service is classified as simple if it interacts with less than four services, average if it interacts with less than eight services, and service is treated as complex if it interacts with more than or equal to eight services. The terms simple, average and complex are used by considering the impact of change requirements on other services. As mentioned, service is termed as simple, if it interacts with less than four services and to make a change in that particular service, it may not impact more than four services, and hence it is considered as simple. The next step is to calculate the unadjusted service points based on the weights assigned and it is calculated by summation of the number of services of each type multiplied by the weight assigned to the corresponding service type.

The list of 21 technical and environmental factors is updated by considering the characteristics of the microservices architecture. Each factor has a value assigned between 0 and 5 depending on the importance and impact that particular factor has on the system. An online survey is conducted to collect the inputs from different practitioners, software archi-

tects and developers working with microservices architecture. We have posted the online questionnaire on multiple social networking platforms including the groups on LinkedIn, Twitter, and Facebook etc. The rating of each factor between 0 and 5 for each factor is collected through this survey. Based on the data collected, the average of ratings is taken and assigned to all the factors which are further used to calculate the TCF and EF values. The final Service Points (SP) is calculated by multiplying the unadjusted service point with both technical and environmental factor values. The proposed estimation approach is applied to a case study application to demonstrate the process of estimating the effort.

Further, to validate the proposed approach, machine learning techniques are applied as it plays a significant role in software effort estimation. In order to validate the efficiency of the proposed method, N applications of SOA, which are migrated to microservices are chosen as datasets and regression analysis is performed on the datasets. To evaluate the accuracy of the estimated approach, several frequently used measures are considered such as magnitude of relative error (MRE), mean of MRE (MMRE), root mean square error (RMSE), prediction within 25% of the actual value, Mean Absolute Error (MAE) and Standardized accuracy (SA).

For ease of understanding and comparison, the proposed service points method is named as *SP-Proposed approach*, the approach proposed using Karner's default rating is named as *SP-Karner's approach* and the effort estimation model generated using the regression analysis is named as *SP-Regression approach*. The efforts calculated using these three approaches were compared with actual efforts and the results are presented with the estimation success parameter. The accuracy of the proposed methods is evaluated using the measures and the effort estimated using the SP-Regression approach is much closer to the actual efforts of the applications.

1.3.4 Patterns for migration of SOA based applications to microservices architecture.

In this work, the challenges which occur during the migration process are identified from the literature and also from our experience in migrating applications to microservices are

presented. In particular, we present the patterns for three major challenges which are discussed below. The solutions are presented in the form of patterns with different sections, including the criteria, context, problem, solution, and challenges. The service graph along with task graph representation are considered for providing the solutions for the challenges.

The first pattern presented is the *Decomposition of an SOA service into microservices* which helps in identifying the monolithic services in SOA. Using the service graph, the order of each task graph is calculated and the node which has its order of more than one will be decomposed by applying the extraction approach proposed in objective 2. Representing the entire SOA application as a service graph is difficult for large enterprise applications, as it includes a large number of services. The tools required for generating such large graphs are also a challenging task.

The second pattern is measuring the *Size of the microservice*. Every service in the SOA application consists of many tasks that perform simple or complex business requirements. Microservices follow single responsibility principle that requires each service to perform only one business function. We, therefore, use the service graph representation of the SOA based application and consider each task in the task graph to be microservices. In this case, the size of the microservice is not a measurable metric. When a specific service performs only one operation, it may be considered as a microservice and the size of the service is not considered. The assumption of considering each task in the task graph as microservices remains a challenge because not all tasks may perform one business requirement.

The third pattern is bug detection in the complex microservices application. When the number of services increases in microservices based application, it is difficult to detect the bug and identify the root cause of the bug. To solve this, the workflow of the business requirements is defined through the use of software artifacts mentioned in the software requirement specification documents. A mapping is created between the business requirement and the services it goes through to process the business requirement. These workflows help in easy and quick identification of the cause of the bug. We have demonstrated the proposed patterns using a case study application.

1.4 Organization of the thesis

The research work in this thesis mainly focuses on studies related to the migration of SOA based applications to microservices architecture. The contributions and findings of the research work are organized into chapters as below.

Chapter 1: This chapter provides the necessary background and motivation for the work reported in this thesis. It also presented the overview of the research contributions of the thesis with respect to the migration of SOA based applications to microservices architecture.

Chapter 2: In this chapter, a brief introduction to the distributed systems such as monolithic, SOA, and microservices architectures are presented. A detailed literature survey on the migration of SOA based applications to microservices is presented. Also, the related work done for each objective is presented.

Chapter 3: In this chapter, a comparison between SOA and microservices based applications with respect to QoS parameters such as complexity and performance is presented. A mathematical model called Service Graph is proposed, which helps in the comparison of both the architectures. Multiple criteria defined for comparison along with the results are also presented.

Chapter 4: The concept of Task Graph is introduced in this chapter, and a 3-step approach to partition and extraction of microservices from monolithic services of SOA based applications is presented. In particular, algorithms for the generation of service graph, task graph and microservices extraction are presented. Also, an algorithm for the generation of service graph for microservices application is presented. The comparison of the coupling intensity of the extracted microservices with SOA services is also presented.

Chapter 5: In this chapter, an effort estimation approach called Service Points for estimating the effort required for migration of SOA based applications to microservices is presented. The demonstration of the proposed service points approach using a case study application is presented. To measure the accuracy of the proposed approach, machine learning techniques such as multiple linear regression and Leave-N-Out policy are presented. The comparison of estimated efforts using three different approaches such as SP-Proposed

approach, SP-Karner's approach and SP-Regression approach are also presented. The accuracy for the proposed approaches is measured with frequently used measures such as MRE, RMSE, etc., and the results are presented.

Chapter 6: In this chapter, patterns for solving the recurring problems which occur during the migration process are presented. In particular, patterns for decomposition of SOA based services, measuring the size of microservices and identification of bugs are presented.

Chapter 7: This chapter summarizes the research outcomes of the work in this thesis. It also presents future directions and scope for extensions of the works done in this thesis.

Chapter 2

Literature Review

Distributed systems have evolved rapidly as the demand for quick design and deployment of business requirements has increased [1]. Distributed Systems provide many benefits to applications that include scalability, resiliency, resource sharing, flexibility, and concurrency. They also help reduce technical debt by allowing teams to use applications at so many levels, and these applications can operate continually even if parts of the applications fail. As today's world demands the timely delivery of business needs, distributed systems play a key role in the accelerated delivery of services with continuous integration and continuous development. The taxonomy of evolution of distributed systems is represented in Figure 2.1.

2.1 Monolithic Applications

Monolithic applications are built as a single large block of code, and the entire application is deployed as a single archive [14]. The server component of a client-server architecture is a monolithic product that handles HTTP requests and communicates with the database. It contains a single executable code that handles all of the server-side functions for an application.

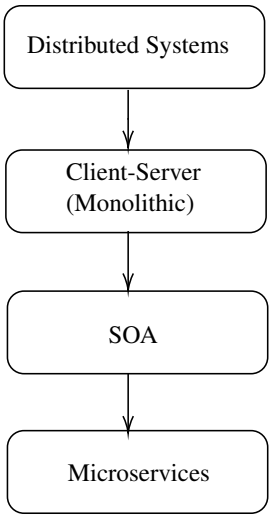

Figure 2.1: Distributed systems taxonomy evolution

2.1.1 Benefits of Monolithic applications

To design any new application, monolithic architecture is the best choice at the start of the project because development, testing, and deployment are very simple and easy [15]. Monolithic architecture is best suitable for small applications with less business functionality. When the load increases, applications can be easily scaled horizontally. Moreover, there will be no problem related to network latency and security in the applications.

2.1.2 Drawbacks of Monolithic applications

Monolithic applications are very tightly coupled and can evolve into a complex web of code, making it difficult for developers to maintain over time. Monolithic applications are difficult to update, maintain, and deploy as it makes the application code very complex to understand [2]. To make a single update in the system, the entire application needs to be shut down and redeploy every component [16]. Since monoliths must be developed and deployed as a single unit, it can be challenging to divide development efforts into independent teams. Each code change must be carefully planned, which slows down the development process. It is impossible to achieve operational agility when deploying monolithic application components repeatedly. Monolithic architecture has a limitation in the size and complexity of the application. To overcome the design and deployment challenges

in monolithic applications, service oriented architecture has emerged as a style of decomposing the entire application into loosely coupled, scalable, and interoperable services [17].

2.2 Service Oriented Architecture

Service oriented architecture (SOA) is the architectural style of distributed applications with service as the main design component. A service is a reusable software code that performs various business tasks that can be simple or complex based on the business requirements. SOA is primarily used for the integration of various components with the middleware feature using Enterprise Service Bus (ESB) [18]. ESB is the backbone of SOA, which helps in providing the features of the middleware system. ESB acts as a mediator between the service requestor & provider and provides a high performance and scalability platform. SOA follows several design principles such as loose coupling, interoperability, statelessness, etc., which are presented below.

2.2.1 Principles of SOA

There is no standard body that defines the principles of SOA, but many principles have originated from IT organizations with their experience. Below is the list of principles to be followed by any system that implements the concepts of SOA [19].

1. **Service Contract**: Service contract contains documents called service description documents which have the meta information of the services. Web Services Description Language (WSDL) is the most common document available in service contracts when services are implemented.

2. **Loose Coupling**: The relationship or dependency between two services is referred to as coupling. It states that there should be independence between design and business logic, and implementation details should be hidden from customers.

3. **Service Reusability**: This principle states that services should be built such that they can be reused across business applications. The logic of the services should be independent of any business requirement and technology.

4. **Service Abstraction**: This principle states that only the required information should be present in the service contract and hide the underlying details of the services as much as possible.

5. **Service Statelessness**: This principle states that state information of the services should be separated to implement scalable services. By this, services can handle more requests and help in fast processing.

6. **Service Composability**: This principle helps in the design of new services by composing already existing services to achieve the business requirements. This also satisfies the concept of reusability by reusing the existing services.

7. **Service Discoverability**: This principle states that service description of the services present in the service contract should contain communicative data such that services are easily discoverable on the web or internet.

8. **Service Interoperability**: This principle states that services must be designed such that they can share data with other services. If the services are not interoperable, they need to be integrated to share data.

2.2.2 Web services

SOA gained more popularity with the evolution of web services, which is the popular implementation of SOA concepts. SOA is the concept, and web services are the implementation of the concept [18]. Web services are also services that can be designed, accessed, and discovered over the internet using communication protocols such as XML based SOAP and WSDL. Web services use HTTP and REST protocols for the transfer of messages through the internet. Web services use XML (WSDL) to access information and share messages between different services. The typical web service architecture, as shown in Figure 2.2, consists of three components, namely a service provider, a service requester, and a service registry, which maintains all the web services. Service provider and requester are web services where the latter requests for information and the former responds with the information. A single web service can be used by multiple clients simultaneously and can be

easily deployed.

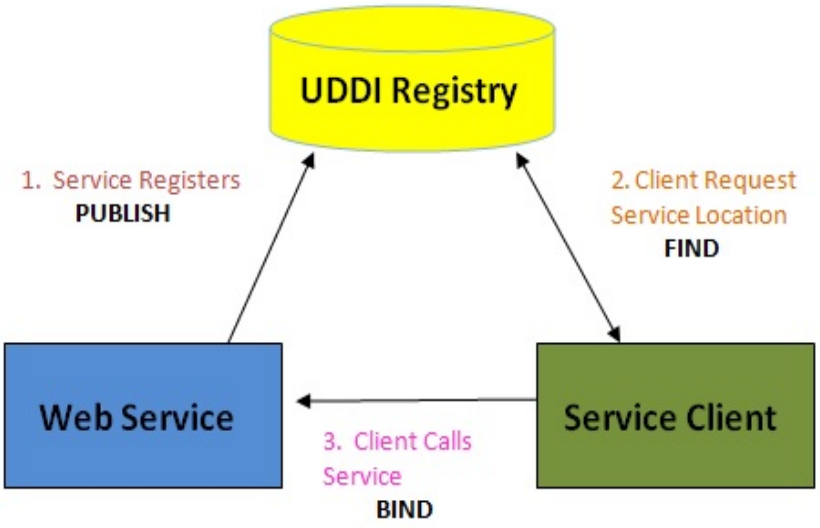

Figure 2.2: Web Services Architecture

2.2.3 Drawbacks of SOA

Though SOA has become popular in the integration of multiple applications using the enterprise service bus, there are still few challenges related to delivery, deployment, governance, and interoperability of services [20]. Additionally, the services in SOA applications are tending towards monolithic with the increase in changing user requirements. Also, since SOA services are tightly coupled with ESB for the exchange of messages and if there is a need to update a particular service, it requires redeploying the dependent components in the system. Therefore, the deployment process of SOA is still seen as a monolith [21]. Further, the on-demand services can be scaled horizontally, but the hardware cost increases as it requires additional infrastructure. Below are the few bottlenecks in the implementation of SOA.

Interoperability to some extent: An interoperable service is one that can work across platforms, languages, applications, and web services from different vendors. SOA Services do not implement interoperability to the full extent because it is very difficult to maintain the complexity of the services [22]. This problem arises because of incompatible data types,

other entities like pointers, structures, database connectivity, etc., supported by different programming languages like C++, Java, and C#. We may face challenges while implementing Quality of Service (QoS) parameters like security, reliability, etc.

Scalability is not achieved: Web services are stateful as they use SOAP protocol for the transfer of messages among different services. Also, both provider and consumer should share the same message data in stateful services. This could reduce the overall scalability if the provider needs to store the message sent to all consumers. It makes switching between services difficult and binds the services with tight coupling. Hence loose coupling is not completely achieved in web services [23].

Services are less reusable: The more business-oriented the service is, the less reusable it is. Services that are anchored to specific applications cannot be reused, and thus the reusability feature is limited in SOA services [24].

Middleware dependency of Services: Service based architectures are attached to heavy-weight middleware using ESB as it allows business services to integrate applications without coding. Dependency on middleware makes it difficult to inculcate new business needs and reduces application flexibility in the future [25]. When failure occurs, it is difficult to replace the service without impacting other services.

Orchestration vs Choreography: SOA services use orchestration for communication between services which involves a point-to-point connection between the services. These connections create many different communication paths and it is difficult to update any service as a developer should be aware of each connection [26].

To overcome these design, deployment, and maintenance challenges which are stated above in SOA, microservices have emerged as a new architectural style of designing applications with loose coupling, independent deployment, and scalability as key features.

2.3 Microservices architecture

Microservices is a new style of designing enterprise applications that is based on SOA principles with additional features. It is a way of designing applications where each component is designed using a lightweight protocol and deployed independently [27]. Microservices

uses the REST communication protocol and the JSON data exchange format for the exchange of messages between services. It follows the concept of the Single Responsibility Principle (SRP), where only one business function should be performed by each service. Continuous Integration (CI) and Continuous Delivery (CD) are the two core principles of microservice architecture. Applications designed with microservices are loosely coupled, scalable, and designed independently. Microservices are well suited to the cloud environment as containers are used for the deployment of the services. The main advantage of using microservices over other architectural styles is that only the required service is deployed independently without having an impact on the other services of the application [28]. The use of containers renders the services auto-scalable. Each service has its own database and configuration environment for the processing of business requests. Moreover, applications are migrating towards cloud [29], and because of the diverse benefits, companies have started migrating their existing legacy applications to microservices architecture. Netflix, Amazon, and Google have started developing their applications with this new style [30].

2.3.1 Definition

Microservices is best defined by Fowler and Lewis ,
"Microservices is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business requirements and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies "[3].

2.3.2 Characteristics of microservices

The main characteristics of microservices are [31]:

- **Single purpose:** Microservices follow Single Responsibility Principle (SRP), which states that each service should perform only one business task and it should do it

well. Generally, the size of the code increases over time to include additional business requirements. Therefore this principle helps in avoiding the complexity of the application.

- **Encapsulation:** Each microservice should have its own database, and data should be accessed only via the defined APIs. The implementation part of the microservice should be hidden and should be kept private.

- **Flexibility:** Microservices are flexible enough to support all the features necessary in the dynamic business environment to remain competitive.

- **Modularity:** Each service is self-contained and focuses on specific business functions that contribute to overall system behavior rather than the full functionality of a single service.

- **Evolution:** Each service can be expanded with new features and is easily maintained. And if a particular service is down, it does not have an effect on the other services in the system.

2.3.3 Benefits of using microservices

- Loose coupling is perfectly implemented in microservices as the definition says that services should be small, lightweight, and should be built for satisfying a single business requirement [32]. Every microservice has its own database, and there is no data sharing via the database. This reduces coupling between services. Also, encapsulation reduces the coupling between services and consumers.

- Microservices are more scalable as we can scale only the required services instead of scaling the entire application [33]. This will help in minimizing performance issues.

- Services built using microservices architecture can be reusable as the services perform unique business requests, and we cannot add more functionality to existing services. So all the services can be reused for other business requirements using service composition [5].

- Microservices are stateless services as they do not store anything. They handle requests and submit the responses [30]. They use transport protocols like HTTP as it is stateless when compared to SOAP which is used in web services. Being stateless, scalability is also increased. If the services are dependent on the state, they should be moved into separate containers.

- Microservices specify endpoints with associated business logic. With changing business needs, services can be updated independently without affecting the existing application. As microservices deal with small functionality, it is easy to change or update the service when a failure occurs. Deployment can be done independently without affecting the rest of the application [34].

- Microservices support service choreography over service orchestration because microservices architecture does not require middleware support [35]. Every service has its own decision logic, and they are not dependent on other services. This independent feature helps to achieve loose coupling.

- DevOps is a software engineering practice that combines development with operations. DevOps focuses on developing applications as small modules, testing independently, and frequent deployment. Continuous Integration and Continuous Delivery are the main features of DevOps, and hence microservices fits well in this structure [5].

- Microservices aligns well with the agile software development process. The agile process focuses on the incremental and iterative model of delivery to cope with change requests quickly. For this reason, applications are divided into smaller modules in the agile process. As microservices are designed as small and independent services, it suits well with the agile process [36].

- In microservices, we can use any programming language to design the services. As each service is deployed in different containers, communication between services will take place with standard protocols like XML/JSON. Whereas in SOA services, we cannot use multiple programming languages as the exchange of information is

difficult. Different services can be built in different technologies, and it is easier for developers to choose their own choice of technology for development [37].

- The architectural requirements for Internet of Things (IoT) suitable for microservices architectural principles. Few tasks in IoT involve just read the data from sensors and update the status of objects from zero to one or vice versa. Since microservices support single service single task policy, these are applicable for IoT. Also, microservices communicate through protocols like HTTP, REST or MQTT; these technologies are best supported in IoT systems as well [38].

2.3.4 Technical differences between SOA and microservices

The primary difference of microservices style with SOA is the emphasis on scalability, independence, and semantic cohesiveness of each component in the system [39]. In contrast with SOA, microservices are required to be self-contained with data, user interface, and databases.

Technically, there are certain differences between the design and implementation strategies of both SOA and microservices applications. SOA is based on the concept of sharing as much as possible, whereas microservices architecture is based on the idea of sharing as little as possible [40]. SOA depends on heavyweight middleware and enterprise service bus for communication between the services, whereas microservices rely only on lightweight technologies. SOA is related to protocols and formats such as WSDL, SOAP, etc., and microservices use REST and HTTP for communication, and JavaScript Object Notation (JSON) as data exchange format [37]. Microservices follow the concept of smart endpoints, dumb pipes and follow the strategy of choreography over orchestration [19].

2.3.5 Why use microservices?

Loose coupling and the freedom of choosing programming languages for the implementation of microservices are some of the major benefits. Microservices are deployed in docker containers which are lightweight, and they are best suitable for microservices as

they start very quickly [4]. Container images consist of all required environmental configurations, and developers can easily access them from DockerHub. Microservices can be easily added or removed from the applications, and they can be easily migrated from one host to another. Independent deployment helps in auto-scaling of the microservices at a fast pace and can easily handle the load. Microservices enable continuous integration and continuous delivery and suit well with DevOps style, which acts as a framework for the complete Software Development Life Cycle (SDLC) of microservices [5]. Compared to traditional architectures, microservices are designed in short development time, reduces the inherent complexity, and increases the scalability [6]. Software architects may believe that microservice is SOA done right, but microservices architecture is about designing isolated services with a strong focus on data isolation [10].

2.4 Characteristics of the SOA system to be suitable for migration

To migrate an application from one architecture to another, it is necessary to study the characteristics of the existing system which is being migrated. Along with the drawbacks highlighted in section 2.2.3, migrating to microservices architecture adds many technical and business benefits. For any system to become suitable for migration, the QoS parameters play a major role in deciding whether to migrate the application or not. In chapter 3, we present the QoS values for both SOA and microservices architecture. Additionally, the applications built using the concepts of SOA as normal services or web services are suitable to our proposed framework. As we are proposing a new graph based model called service graph, which is generic to any SOA system, this framework can be used for migration.

2.5 Study on migration of SOA applications to microservices architecture

Since the migration of the SOA based applications to microservices is an open challenge[12], we consider it as the major research problem in this thesis. However, the migration of systems towards microservices involves multiple difficulties [13] such as (i) not knowing the impact of migration, (ii) not having enough material on migration techniques, and (iii) not being aware of the migration effort. These challenges motivated us to study and propose possible solutions for the migration of SOA based applications to microservices architecture. The motivation and related work behind each objective are discussed below.

2.5.1 Comparison of SOA and Microservices architecture based applications

One of the goals in the Architecture-Level Modifiability Analysis (ALMA) model presented by Lassing et al. [41] considers comparing two or more architectures to find the better one. Here, we consider SOA and microservices architectures for comparison to find the appropriate one to use in the design of enterprise applications. The number of services will be more in microservices and the applications will be more complex compared to SOA. Moreover, the deployment strategy of microservices is completely different from that of SOA, and hence it motivates us to select the comparison criteria as complexity analysis and performance testing of the service-based architectures.

2.5.1.1 Complexity

In the context of software architectures, complexity can be defined as the complexities of services or components that make up the complete architecture and their dependencies [42]. Continuous upgrades and enhancements of software applications make the services larger and hence the design, implementation, and deployment of such applications become more complex [43]. The results of the survey presented in [44] state that despite having low service coupling in microservices, the complexity of the application increases. Microser-

vices system consists of fine-grained services and interactions are very complex including the configurations of the environments. Multiple service calls occur for a simple business task to process and give the result [45]. Also, the cost and time for development increases with the increase in the complexity of the application [3]. Though it has been stated in the literature that the complexity of the microservices application is high, it is required to compare the same with SOA and check the behavior of the application with the increase in complexity.

2.5.1.2 Performance testing

Performance related features such as response time are of more interest to determine the acceptability of software design [46]. Response time is the time taken for a particular business request from initiating to the successful completion of the task. Though the chosen architectures are service-based, the implementation style and deployment environment are completely different, and the impact of the cloud can be analyzed with load testing. JMeter tool is best suitable for performing the load testing, and it has been successfully used to evaluate the SOA based web services [47]. In a comparative study, [48], container-based services perform better when compared to Virtual Machine (VM) based services. Hence, we deploy microservices in cloud containers and verify the response times of both applications.

2.5.2 Extraction of microservices from SOA based applications

With the increase in user requirements, few services in SOA are tending towards monolithic in size and makes it difficult to maintain the application. With the dependency on ESB and the use of heavyweight protocols for the exchange of messages, SOA applications become less scalable, and the complexity increases with the increase in change requirements. SOA is still seen as monolithic from a deployment perspective [49]. The ability of independent service deployment and elastic scalability in microservices makes SOA applications as legacy [12]. Also, as mentioned in the introduction, many design challenges exist with the implementation of SOA using web services. To the best of our knowledge, very few

works have been proposed in the literature to migrate SOA based web services to microservices. One such approach is proposed by Tusjunt M et al. to migrate web services based on business capabilities, and scenario base analysis [50]. It identifies vocabularies and their relationships and generates microservices. However, the proposed approach is domain-specific and cannot be applied to other domains, and identifying a complete set of vocabulary is difficult. Therefore, we consider the migration of SOA based applications to microservices architecture in this work.

2.5.2.1 Need for migration to microservices

Because of the diverse benefits, IT companies have started designing their applications using microservices architecture, and few of them have started migrating their applications to microservices [11]. There are numerous reasons which trigger migration towards microservices. As architects are unaware of the effort and cost estimation required for designing the application from scratch, migrating is the best approach [51]. A systematic mapping study conducted by Di Francesco et al. states that research for the migration to microservices is at an early stage [52]. Migrating applications to the cloud have also aroused for migrating to microservices as it suits better in the cloud environment [44]. Technical debt has reduced by migrating the existing legacy application to microservices, and maintenance has improved [53].

2.5.2.2 Challenges in migration

A major challenge in migration is identifying the appropriate partition of the system into microservices [54]. An overview of the lessons learned and challenges while migration to microservices is discussed by da Silva Filho HC et al. [55]. Few challenges include decoupling of services, effort estimation for migration, identification of service boundaries, and the effort to analyze every part of the system and decide what should be converted to microservice. Multi-tenancy, statefulness, and data consistency are a few other challenges of microservices migration [56]. However, a feedback study conducted by Henry A et al. identified that more than 50% of the responses state that finding the right way to break the legacy applications is the major difficulty, and 49% of the responses state that the complex

task during migration is to overcome tight coupling [10]. Therefore, we strive to propose a solution for extracting the candidate microservices from existing legacy service-oriented applications.

2.5.2.3 Existing migration techniques

Considering the major challenge of extracting microservices, some researchers have contributed to the solution in the past few years. Tyszberowicz S et al. in [54] have proposed an approach to identify microservices using functional decomposition. This approach is not applicable for extracting from existing applications; rather, it is applicable to extract microservices from requirement specifications. Similar work for the extraction of candidate microservices from application code using a clustering algorithm is proposed by Kamimura M et al. [57]. The relation between extracted candidates and the whole structure of the software is also visualized. However, the proposed approach can be used just to analyze the system before actual migration. An exploratory study conducted by Carvalho L et al. finds that customization and variability are needed after the extraction of microservices from legacy systems [9]. As our approach extracts microservices from SOA based systems, they can be used directly without any customization of the services. Gysel M et al. have proposed a service cutter approach for service decomposition [58]. In this approach, a tool that supports structured service decomposition through graph cutting is designed where internal structure is decomposed based on coupling criteria. Also, the user has to provide the software artifacts as input to extract services. Mazlami G et al. have proposed a clustering algorithm to extract the microservices from monolithic applications [59]. This approach considers classes as an atomic unit of computation, and not all monolithic will be based on classes. Baresi L et al. have proposed a technique for microservices identification through interface analysis [60]. In this approach, decomposition is based on reference vocabulary and open API. However, this approach cannot suit service-based architectures as the decomposition of artifacts is based on vocabulary. A functionality oriented microservice extraction method is proposed by Jin W et al., which identifies the dependencies using the execution traces [61]. There are many limitations in their method as it is not fully automated, and the coverage of test cases may not be accu-

rate. A recent study conducted by Ponce F et al. states that 90% of the proposed techniques use the design element as input and applicable only for object-oriented software [62]. All the aforementioned approaches directly or indirectly depend on user inputs and are either manual or semi-automatic approaches. Also, the above approaches discussed focus mostly on the migration of monolithic applications to microservices architecture.

Few efforts have been contributed to overcome the challenges in SOA based applications. Martha VS & Lenglart M have proposed an approach for web services engineering named Web Service Development Life Cycle (WSDLC), where each web service can be developed independently from other services in the enterprise [63]. It is merely a streamlining approach for the existing application but not migration to microservices. This approach reduces the problems related to the performance and availability of web services and has few limitations. *Verb-based* and *Noun-based* decomposition techniques are used to partition the web services from a large service in the enterprise. It is not always possible to divide the services uniformly, as mentioned in the approach. As microservices are said to be SOA done well, migrating existing applications to microservices architecture is the best solution.

2.5.3 Effort estimation for microservices architecture

With the various benefits of microservices, software architects have started migrating their existing legacy applications to microservices architecture [3]. Many companies, including Netflix, Amazon, and Twitter, have started building their new applications with this style of architecture [4]. However, the effort required for migration and designing the microservices based applications is the major challenge. Effort estimation helps software architects in the proper execution and management of the project. Effective estimation helps in the proper scheduling of the software engineering activities. Software effort is given by the formula effort = people * time [64]. It has to be done during the early stage of the application design as it gives insights into the effort and cost required to complete the application. Software effort estimation techniques are divided into four types, namely, empirical, regression, theory-based, and machine learning techniques based estimation [65]. The empirical

way of estimating is very popular as it gives a clear picture of the effort required numerically, and a few of the models include function point, use case point, and analogy based techniques. These techniques are not suitable for measuring the effort for service-based systems as they are designed for procedural object-oriented systems [66].

Use Case Points (UCP) is a commonly used technique because of its simplicity, fastness, and accuracy to a certain extent [67]. UCP approach is based on the use case diagrams for calculating the effort. Though use case points approach is presented as a flaw for estimating the efforts, there are many successful implementations of use case points for estimating the effort of object oriented systems. Many variations and enhancements have been published in the literature to improve the accuracy of the approach [68, 69, 70]. Though the use case point approach is based on the use case diagrams of object-oriented concepts, attempts have been made for estimating the effort for service-oriented architectures [71]. All the traditional approaches available for effort estimation cannot be used directly for service-based systems.

2.5.4 Patterns for microservices architecture

Many design and environmental challenges occur during the migration process of one architecture to another architecture [72, 73]. Similarly, many challenges occur during the migration process and post-migration. A few of the challenges which occur during migration to microservices are: Identification of candidate microservices from legacy source code, testing of services designed with different programming languages, integration of polyglot services, debugging and RCA for issues in the migrated services and setting up the configuration environment for newly generated microservices [54]. In software engineering, patterns are used to solve the commonly occurring problems that occur during the SDLC phases of the application [74]. In addition, migration patterns can also be used to support issues that occur during the migration from one architecture to another [75]. The exploitation of design patterns helps in mitigating/solving some pains of microservices [76]. There are broader advantages of using migration patterns during migration, as it is the new architectural style. There are very few or no design patterns defined for the problems occurring

in the design of microservices in the literature [77].

2.6 Summary

In this chapter, distributed systems such as monolithic, SOA, and microservices architectures are discussed. The benefits and drawbacks of monolithic and SOA applications are presented. The definition, characteristics, and benefits of using microservices architecture are also presented. A detailed literature survey of studies conducted in this thesis is presented. In the next chapter, the comparison of both SOA and microservices architecture applications using complexity and performance parameter is presented.

Chapter 3

Comparison of Service Oriented Architecture and Microservices Based Applications

Microservices architecture has gained a lot of attention since its inception in the year 2014 by Martin Flower [3]. With the evolution of microservices as a new style for designing enterprise applications, it has attracted many researchers to contribute their insights on this new style. Some proponents of microservices claim it as a new style, whereas the advocates of SOA claim it as an implementation of SOA [78]. Vural H et al. [79] has presented the current trends and emerging standards in the research of microservices. One of the major possible research gaps highlighted was the comparison of SOA with microservices architecture and migration of legacy monolithic & SOA applications to microservices.

One of the goals in the Architecture-Level Modifiability Analysis (ALMA) model presented by Lassing et al. [41] considers comparing two or more architectures to find the better one. In several studies conducted by Rademacher F et al. [80], Pahl C et al. [81], and Cerny T et al. [21], it is mentioned that there is a need to compare and investigate both SOA and microservices architecture in terms of performance, development effort, and maintenance. Cerny T et al. [12] discusses the theoretical differences between both the styles in terms of different architectural parameters. The differences are presented in both research and industry perspectives. Similar work of comparing distributed systems such

as client/server, mobile agents, SOA, and microservices is presented by Salah T et al. [1]. However, there has been no empirical work done in comparing these two architectural styles. Hence, we consider complexity and performance as the parameters for comparison of both the architectures. The main contributions of this chapter are described below:

- Proposed a formal model called as Service Graph (SG) which resembles any service based application. It acts as a blueprint of the service based application, which helps in identifying the metrics required for comparison of both the architectures.
- Presented a comparison of a web based application which is built using both SOA and microservices architectures. The comparison is presented with two different parameters:
    1. Complexity with architectural metrics.
    2. Performance with load testing.

3.1 Service Graph

We define a formal model called service graph, which resembles any service-based application. By considering the inputs and outputs from the Application Programming Interfaces (APIs) of the application, we create a service graph. As both SOA and microservices architectures have services as the main component, we use this service graph for comparison of both the architectural styles.

Service graph (SG) is a standard graph created for the visual analysis of communication and dependence between the services of the application. It helps in extracting the values for metrics such as number of services and the complexity of each service. It also helps in software engineering tasks such as effort estimation, complexity analysis and design patterns, etc. The generalized form of a service graph is shown in Figure 3.1.

Service Definition

Let a graph G(V,E) be a service graph with **n** nodes, where the nodes of the graph represent a set of services in the application, and edges between the nodes represent the interactions or dependency each service has with other services in the application. Let V=$\{s_1,s_2,s_3,...\}$

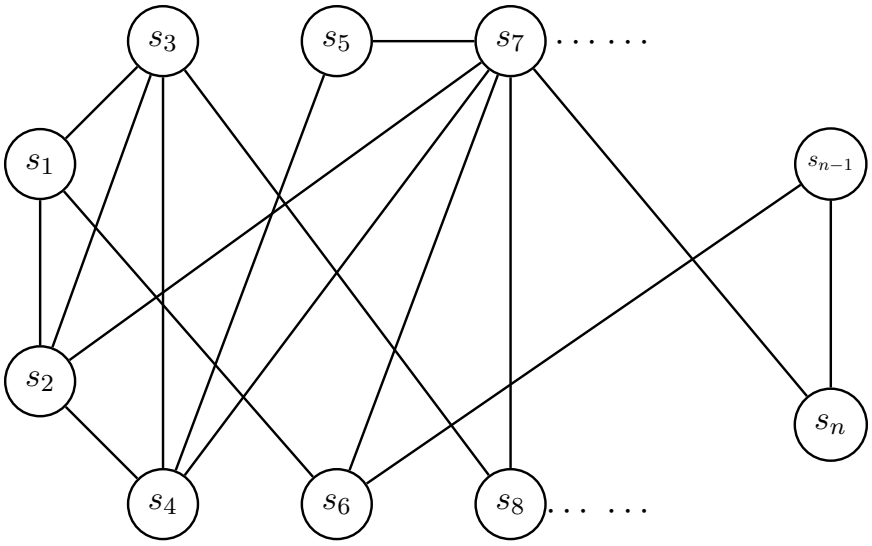

Figure 3.1: Formal representation of service based application

be the nodes of the service graph where s_1,s_2,s_3,... are services and E= $\{(s_1,s_2)$, (s_1,s_3), (s_2,s_4), ...$\}$ be the edges between the nodes which represent the dependency between the services. A service can be represented as a set of coordinating and interacting processes as defined in equation (1).

$$S_i =< {P_1}^i, {P_2}^i, {P_3}^i, \cdots\cdots, {P_n}^i, \Lambda > \tag{3.1}$$

where S_i is the logical service instance, ${P_k}^i$ indicates k^{th} process implementing logical service functionality f_i through the programmatic interface I_i and Λ represents network communication function between individual processes [82].

3.2 Case Study: Vehicle Management System

We use a standard web-based application, Vehicle Management System (VMS) [83] which is used to select, customize, and purchase vehicles and its parts using a web interface. The goal of this application is to help customers to select, customize, compare vehicles, locate dealers, and request a quote. All the details of the vehicles, their parts, and prices are configured in the database and help customers with details using the user interface. Customers can select the vehicle of choice and the dealer for the selected vehicle from the

inventory data. Customers can even select the part and the product type of the vehicle from the interface. The selected information is generated as a lead and sent to the dealer, who helps the customer in purchasing the vehicle and its parts.

3.2.1 SOA based application

The chosen case study application has eight services in the SOA implementation. The details of the SOA services are listed in Table 3.1 and the service graph representation denoted as SG_SOA is presented as shown in Figure 3.2. We implemented the SOA based application using The Information Bus Company (TIBCO) Business Works (BW) and deployed using TIBCO administrator. TIBCO BW is used to create, orchestrate and integrate services with graphical user interface environment. It is widely used for designing SOA based enterprise, web and mobile applications. For data storage, oracle database is considered and database palettes of TIBCO BW help in connecting to the database. The communication between the services is using REST protocol via HTTP. Each service is deployed as an independent archive in a single server.

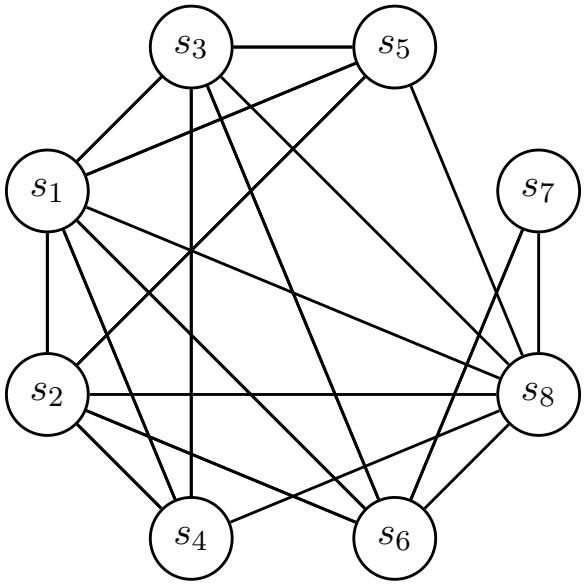

Figure 3.2: SG_SOA : Service graph representation of SOA based web application

The service graph of the SOA based application for the case study application is presented in Figure 3.2. The services of the application are represented as nodes of the graph

and the dependencies and interaction between the services are presented as edges between the nodes. In the graph, node s_1 represents the config service given in Table 3.1. Since the config service is required for all the services to perform the business operations, many services have communication with the s_1 and the edges from s_1 represents the communication between the services.

Notation in SG_SOA	**SOA services**	**Microservices**	**Notation in SG_MSA**
s_1	Config Service	Config Service	ms_1
s_2	Part Service	Part Service	ms_2
s_3	Product Service	Product Service	ms_3
s_4	Compare Service	Compare Service	ms_4
s_5	Incentives & Pricing Service	Incentives Service	ms_5
		Pricing Service	ms_6
s_6	Dealer & Inventory Service	Dealer Service	ms_7
		Dealer Locator Service	ms_8
		Inventory Service	ms_9
s_7	Lead service	Get-A-Quote Service	ms_{10}
		Lead Processor Service	ms_{11}
s_8	User Interface Client	User Interface Client	ms_{12}

Table 3.1: Details of services of both SOA and Microservices based applications

3.2.2 Microservices based application

In order to develop microservices based application, we adopt the microservices extraction approach from SOA based application [84] and generate the service graph, which helps in identifying the candidate microservices. Considering the set of microservices, the chosen case study application is implemented using the spring boot framework, and REST/JSON formats are used for communication among the services in the network. Eureka service is used as a service registry to store all the services. To store the data, MYSQL database is considered and spring boot connector retrieves data using the JPA connector. Each microservice is deployed with containers using docker in the cloud. The docker image of the

application is created, deployed in the docker hub, and containers are created from docker images. The details of the generated microservices are presented in Table 3.1 and the service graph (SG_MSA) is presented in Figure 3.3. The nodes of the service graph represent the microservices and their dependencies on other services. Since, we have many microservices, the number of services communicating among themselves is also high, which is represented as edges in the graph.

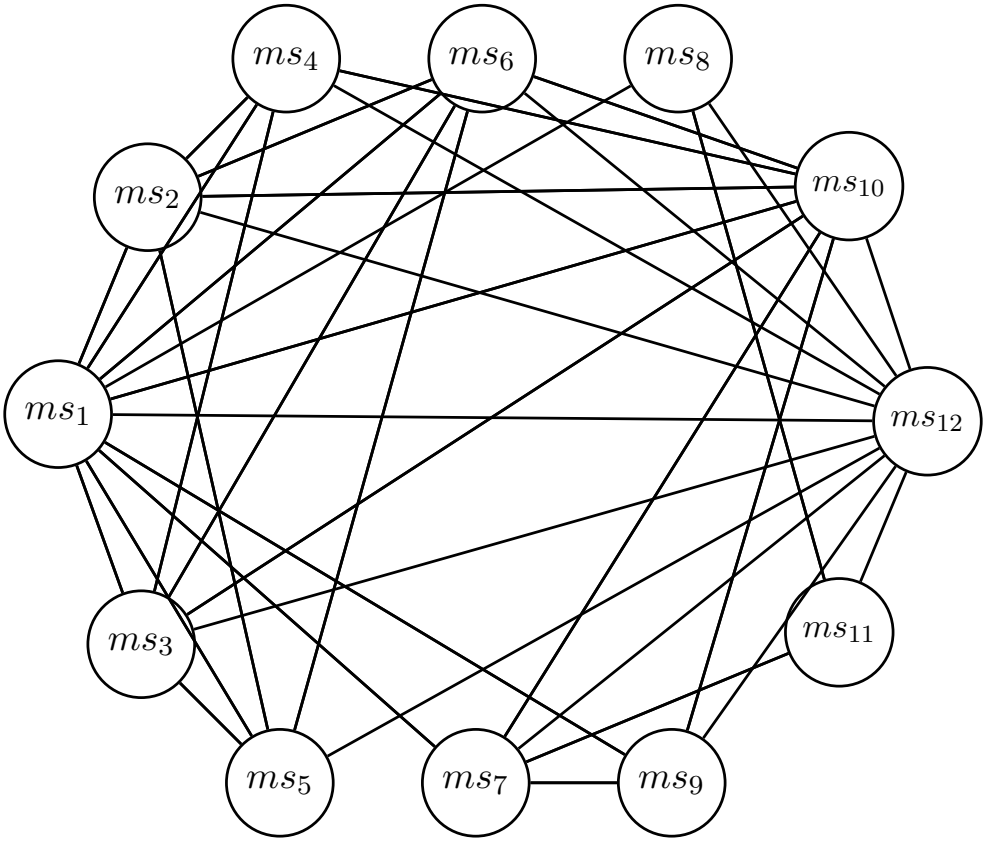

Figure 3.3: SG_MSA: Service graph representation of microservices based application

The detailed information of the service graph generated using the APIs is presented in Chapter 4. The procedure and algorithms for constructing the service graphs of both the systems are presented in Chapter 4. In this chapter, we have intuitively considered the service graph representations for both architectures. The actual migration and the challenges faced during the migration of the chosen vehicle management system are also presented in the next chapter.

3.3 Complexity Analysis

As discussed, there is a strong need to compare both SOA and microservices architecture as software architects are in chaos whether to continue the applications in SOA or to migrate them to microservices style. We consider the metrics for measuring the complexity of the software architectures [85] and extract the metric values from service graph representation. The metric values are considered to compare the complexity of both the architectures. Below, the metric definitions and evaluation of both the architectures are discussed.

Total Complexity of the software architecture: This metric is used to calculate the total complexity of the application by considering the dependencies between the services. Let D_t be the set of all dependencies in the service graph of the software architecture, the total complexity of M_T of the architecture can be measured as

$$M_T =\mid D_t \mid \tag{3.2}$$

The dependencies between the services are extracted from the service graph representation.

Global Complexity of the software architecture: Each service may contain multiple processes and each process also adds to the complexity of the system. By considering the individual complexities of the services, we consider another metric for calculating the global complexity. Let M_T be the total complexity and $M_1, M_2, \cdots, M_k$ be the individual service complexities, then the global complexity M_G is calculated as

$$M_G = M_T + \sum_{i=1}^{k} M_i. \tag{3.3}$$

To calculate the individual complexities of each service in the application, the coupling factor of the service is considered.

Service Coupling: The coupling between the services indicates the dependencies it has on other services, and it should always be low. The more coupling intensity between the services, the high is the complexity of the system. Hence, we consider the metrics related to coupling to calculate the complexity of individual services.

Metrics related to coupling: Service graph provides the details of basic metrics that are

used to determine other metric values. **Number of Services (NoS)** value is given by the count of nodes in the service graph,

$$NoS = n \tag{3.4}$$

Coupling of Services (CS) value is given by the degree of each node as given in equation 3.5.

$$CS_i = deg(s_i) \tag{3.5}$$

Relative Coupling of Services (RCS) denotes the degree of coupling in a particular service [86]. RCS of service is calculated using the formula in equation 3.6.

$$RCS[s] = \frac{CS[s]}{NoS} \tag{3.6}$$

The complexity of the application can be measured with this metric. The coupling intensity of a service is directly proportional to the value of the RCS. Using the above metrics, we evaluate the chosen case study application and compare both the architectural styles.

3.3.1 SOA based application

In order to compare both styles, we extract the metric values from the service graph of both the styles. From the service graph representation of SOA style as shown in Figure 3.2, we identify the dependencies and also calculate the CS and RCS values, presented in Table 3.2.
Total Complexity: The total complexity of the application is calculated by the summation of all dependencies among the services. By considering the CS values from Table 3.2, we calculate the total complexity.

$$M_T = \mid D_t \mid = 38$$

Global Complexity: The global complexity metrics require the complexities of individual services and hence, we consider the RCS values for measuring the individual service complexity.

$$M_G = M_T + \sum_{i=1}^{8} M_i. = 38 + 4.73 = 42.73$$

Service #	Interacting Services	CS value	RCS value
s_1	2,3,4,5,6,8	6	0.75
s_2	1,4,5,6,8	5	0.62
s_3	1,4,5,6,8	5	0.62
s_4	1,2,3,8	4	0.5
s_5	1,2,3,8	4	0.5
s_6	1,2,3,7,8	5	0.62
s_7	6,8	2	0.25
s_8	1,2,3,4,5,6,7	7	0.87

Table 3.2: List of services with CS & RCS values of SOA based application

3.3.2 Microservices based application

Similarly, we extract the dependencies from the service graph as shown in Figure 3.3 and calculate the CS and RCS values, presented in Table 3.3. **Total Complexity:** The total complexity of the application is calculated by the summation of all dependencies in the service graph of microservices based application. By considering the CS values from Table 3.3,

$$M_T = \mid D_t \mid = 70$$

Global Complexity: The global complexity metrics require the complexities of individual services and hence, we consider the RCS values for measuring the individual service complexity. From the RCS values in Table 3.3,

$$M_G = M_T + \sum_{i=1}^{12} M_i. = 70 + 5.8 = 75.8$$

3.3.3 Comparison of Complexities

The values of both total and global complexities for both SOA and microservices based applications are plotted as a graph, as shown in Figure 3.4. The results show that the microservices based application is more complex when compared with SOA based application. By

Service #	Interacting Services	CS value	RCS value
ms_1	2,3,4,5,6,7,9,10,12	9	0.75
ms_2	1,4,5,6,10,12	6	0.5
ms_3	1,4,5,6,10,12	6	0.5
ms_4	1,2,3,10,12	5	0.41
ms_5	1,2,3,6,12	5	0.41
ms_6	1,2,3,5,10,12	6	0.5
ms_7	1,9,10,11,12	5	0.41
ms_8	11,12	2	0.16
ms_9	1,7,10,12	4	0.33
ms_{10}	1,2,3,4,6,7,9,12	8	0.67
ms_{11}	7,8,12	3	0.25
ms_{12}	1,2,3,4,5,6,7,8,9,10,11	11	0.91

Table 3.3: List of services with CS & RCS values of microservices based application

the definition of microservices, the number of services will be more in microservices than in SOA. For the chosen case study application, the number of services in SOA application is 8 and in the microservices based application, it is 12, whereas the sum of total dependencies is 38 in SOA application and 70 in microservices based application. We observe that a 50% increase in the number of services has lead to an 84% increase in the dependencies in the microservices application. The more services, the more will be the complexity of the application. From the service graphs, as shown in Figure 3.2 and Figure 3.3, we can observe that the dependencies among the services in microservices application are very high compared to SOA based application.

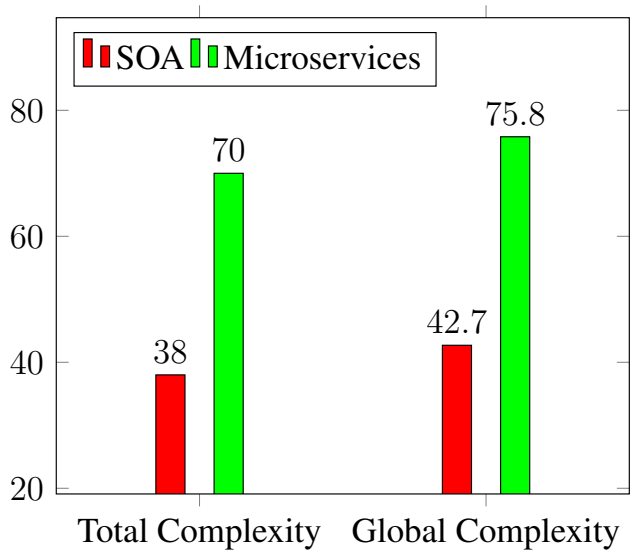

Figure 3.4: Comparison of complexities

3.4 Performance Testing

In order to compare the performance of both the architectures, we consider load testing on both the applications. As JMeter can be used to carry out performance tests for SOAP and REST based web services, we configured JMeter to perform the load testing by considering 500 and 1000 users of load on both the applications. All the services in both SOA and microservices based applications are given load using the JMeter tool, and we capture the average response time for 500 and 1000 users separately for all services. The time taken from sending the request to getting the first response is treated as response time, and it is measured in millisecs. The average response time of both SOA based services and microservices are presented in Table 3.4 and Table 3.5, respectively.

3.4.1 Criteria for performance comparison

In order to compare the performance of both the architectural styles, we define Business Requests (BR) according to the functionality of the case study application. The sequence of services invoked for processing a particular business functionality is stored as a sequence with the service numbers. By understanding the complete case study application, we identified seven major business requests and listed them in Table 3.6 for SOA and microservices

SOA services	Response Time (millisecs)	
	500 users	1000 users
s_1	3412.76	6850.68
s_2	4519.34	8080.58
s_3	6127.12	10870.84
s_4	5923.67	10381.83
s_5	7534.18	15127.50
s_6	8316.41	16199.60
s_7	5681.75	13887.48
s_8	6120.67	11812.99

Table 3.4: Response time values of SOA services

microservices	Response Time (millisecs)	
	500 users	1000 users
ms_1	2691.47	7353.89
ms_2	4245.85	14590.98
ms_3	5381.52	6712.26
ms_4	4818.51	14296.40
ms_5	3133.11	3481.54
ms_6	4914.38	9608.29
ms_7	4608.70	5398.62
ms_8	4037.72	5218.80
ms_9	4403.81	3783.48
ms_{10}	2308.32	4839.70
ms_{11}	3671.6	10303.73
ms_{12}	4866.44	9376.96

Table 3.5: Response time values of microservices

based applications respectively. The time taken for each of these business requests in both SOA and microservices based applications is considered as criteria for comparison.

To better understand the business requests, let us consider the business functionality of comparing two different vehicle models and their parts. The comparing of products and parts of the vehicles is configured as business request BR2. The user, through the web interface client, selects the product and parts of the vehicles from inventory and chooses the compare option in the application. The results of the comparison are displayed on the screen and the user decides on the vehicle to select. The services which get invoked in this process are represented as a sequence, shown in Table 3.6.

Business Requests	**Sequence of SOA services**	**Sequence of microservices**
BR1	$s_8 - s_1 - s_3 - s_2 - s_5 - s_6$	$ms_{12} - ms_1 - ms_3 - ms_2 - ms_6 - ms_5 - ms_9$
BR2	$s_8 - s_6 - s_3 - s_2 - s_4$	$ms_{12} - ms_9 - ms_3 - ms_2 - ms_4$
BR3	$s_8 - s_6 - s_3 - s_2 - s_5 - s_4$	$ms_{12} - ms_9 - ms_3 - ms_2 - ms_6 - ms_5 - ms_4$
BR4	$s_8 - s_6$	$ms_{12} - ms_8 - ms_9$
BR5	$s_8 - s_6 - s_7$	$ms_{12} - ms_9 - ms_{10} - ms_{11} - ms_8 - ms_7$
BR6	$s_8 - s_1 - s_6$	$ms_{12} - ms_1 - ms_7 - ms_8$
BR7	$s_8 - s_1 - s_5$	$ms_{12} - ms_1 - ms_5 - ms_6$

Table 3.6: Mapping of business requests with workflows

3.4.2 Performance comparison results

The average response times of each of the services involved in the business requests are added to get the response time of the complete business request. The time taken for all the requests under 500 and 1000 users for both SOA and microservices applications are plotted as a tornado graph, as shown in Figure 3.5 and Figure 3.6. It is clear from the graph that the average response time for completing the business requests in SOA based application is high compared to microservices application. For a detailed analysis of the performance load testing, we consider two different scenarios w.r.t to the number of services involved in getting the response of a business request.

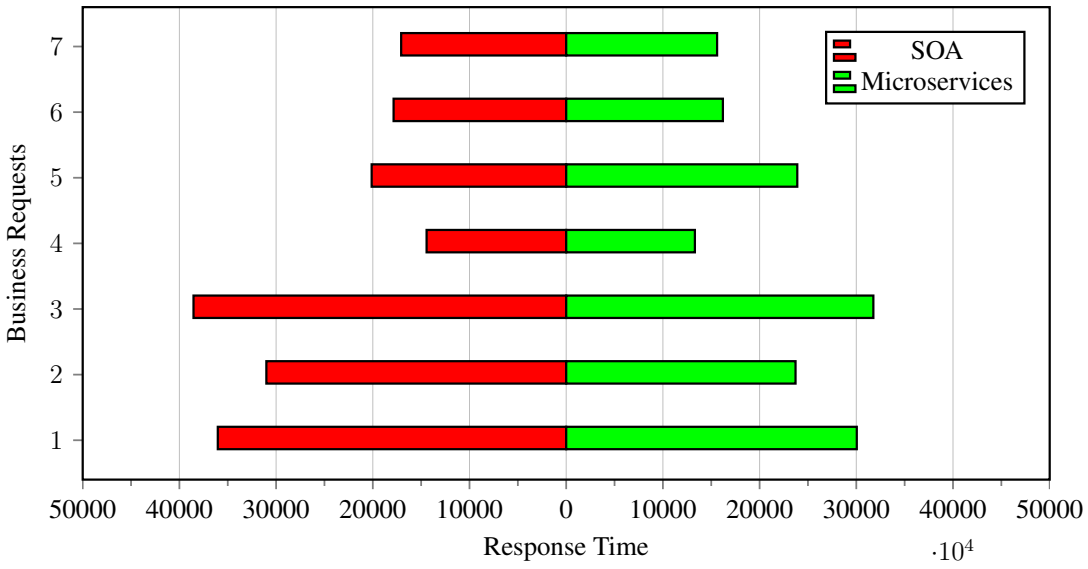

Figure 3.5: Response time of business requests for 500 users

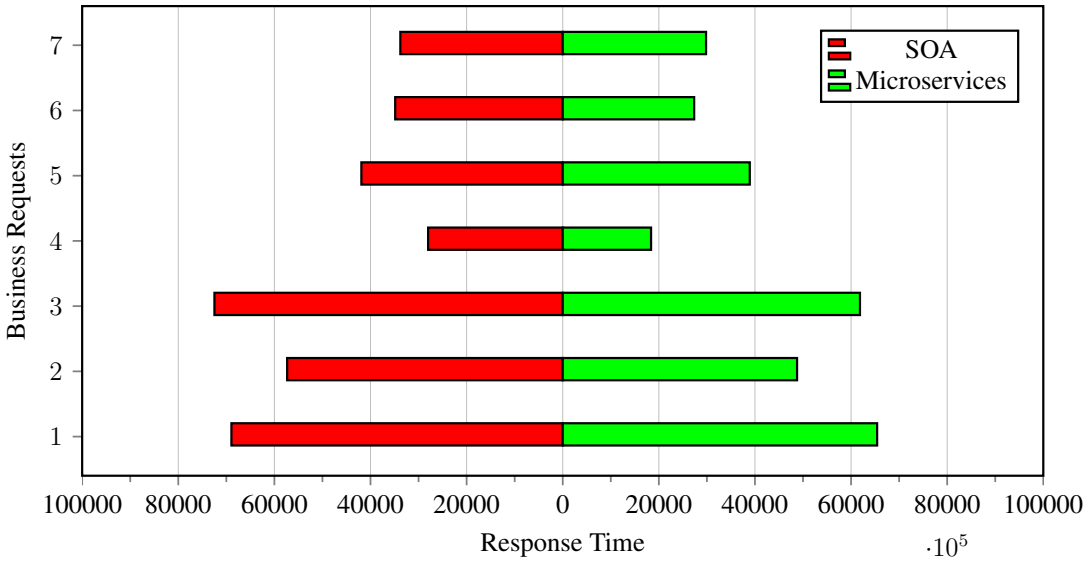

Figure 3.6: Response time of business requests for 1000 users

3.4.2.1 Business request having the same NoSs

From the details of business requests and corresponding sequences in Table 3.6, we can observe that BR2 has the same number of services and the functionality of the service also remains the same but designed with different architectural styles. We consider this scenario to test the response time for BR2, which clearly presents the impact of architecture and its environmental factors. The business request BR2 is verified using 500 and 1000 users and the average response times are plotted as a bar graph as shown in Figure 3.7. The results from the graph for the business request BR2 of our case study show that the microservices based application has better response time when compared to SOA based application even though the number of services and the business functionality remains the same.

3.4.2.2 Business request having different NoSs

To identify how both the architectures behave with a different number of services, we choose the business request BR5 as the number of services involved in SOA based application is three and in microservices based application is six. The business request BR5 is also tested with 500 and 1000 users for both the applications and results are plotted as a bar graph as shown in Figure 3.8. The results show that microservices have a high response time when compared to SOA with 500 users and the response time is better for microser-

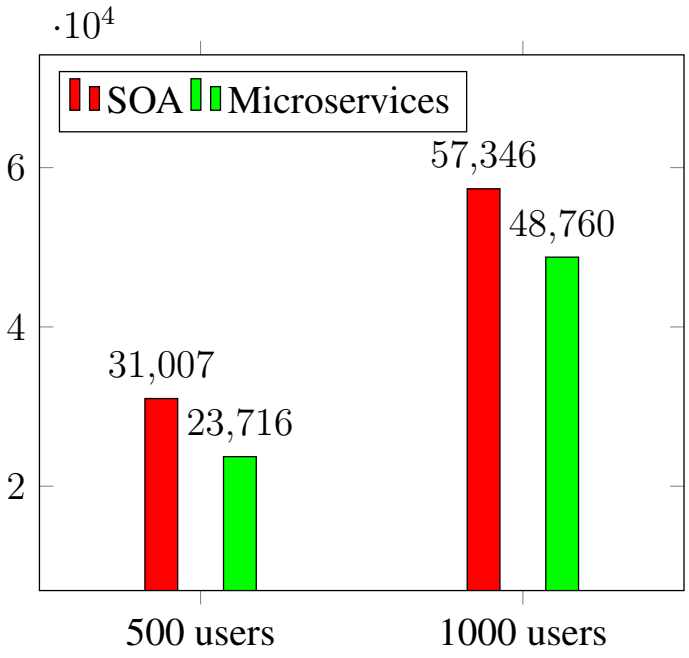

Figure 3.7: Average response time for BR2

vices when the number of users is increased to 1000. It shows that though the number of services is high in microservices, because of its cloud-based deployment environments, the average response time of microservices-based application is low.

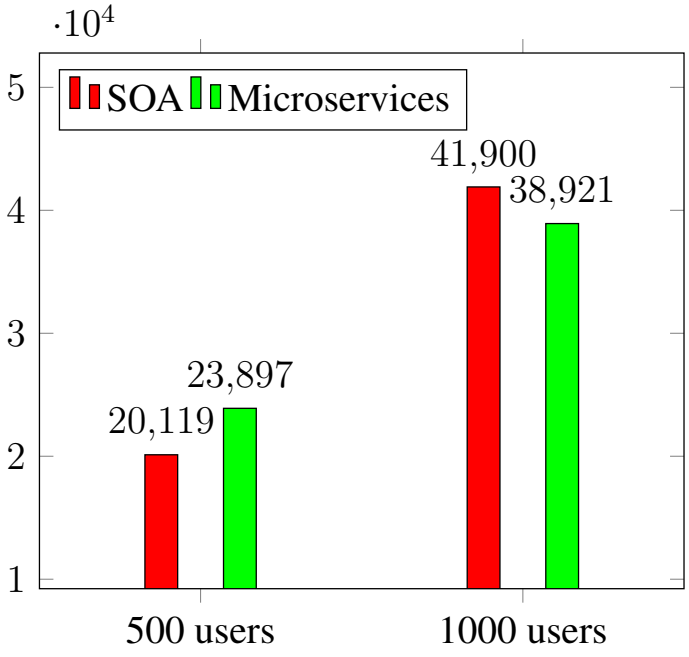

Figure 3.8: Average response time for BR5

3.5 Summary

In this chapter, empirical evaluation and comparison of both SOA and microservice architectures are presented. A service graph model for representing any given service based application is proposed. This chapter helps in understanding the differences in terms of complexity and performance of both styles. Based on the analysis done for the complexity of the applications, it is clear that microservices application has more number of services and application is more complex than SOA based application. The coupling between the services is also less in microservices architecture. However, the performance testing shows better results for microservices with quick response times for 500 and 1000 users. Furthermore, the chosen case study application exhibits better results when chosen the business requests with the same number of services in both styles. After this experimental study, we conclude that microservices architecture exhibits better performance results with the use of cloud-based environments and can be used in the design of enterprise applications.

In the direction to perform research in migration of SOA based application to microservices architecture, this is the initial study conducted. Hence, we have chosen only one case study application to demonstrate and evaluate the performance and complexity of the applications built using both SOA and microservices architecture. However, based on this initial study, we cannot comment whether the response time will be better for any given microservices based application.

Chapter 4

A service graph based extraction of microservices from monolith services of SOA

In the previous chapter, we compared both SOA and microservices architecture, and the results motivate us to migrate the existing applications to microservices style. However, most of the works presented in the literature focus on migrating monolithic applications to microservices but not from SOA to microservices. To the best of our knowledge, there has been no or very limited work done in proposing approaches for the migration of SOA based applications to microservices. Few efforts have been contributed to overcome the challenges in SOA based applications by Mazlami G et al. [59]. One such approach is proposed by Tusjunt M et al. to migrate web services based on business capabilities, and scenario base analysis [50]. The proposed approach is domain-specific and cannot be applied to other domains, and identifying a complete set of vocabulary is difficult. A major challenge in migration is identifying the appropriate partition of the system into microservices [54]. Di Francesco P et al. [52] present challenges which occur while migrating to microservices. Few challenges include decoupling of services, effort estimation for migration, identification of service boundaries, and the effort to analyze every part of the system and decide what should be converted to microservice. However, a feedback study conducted by Henry A et al. [10] identified that more than 50% of the responses state that

finding the right way to break the legacy applications is the major difficulty. In this chapter, we attempt to propose a graph based microservices extraction approach from legacy SOA based applications. The main contributions of this chapter are described below:

- Presented the concept of Task Graph (TG) and how each service in the Service Graph (SG) contains this task graph in it.
- Algorithms for construction of service graph and task graphs for any given service based application are proposed.
- Algorithms for extraction of microservices from SOA based applications and generation of service graph for the microservices based applications are also proposed.
- An SOA based web application is chosen for the demonstration of the proposed algorithms, and the extracted microservices are compared with SOA services in terms of loose coupling.

4.1 Service graph construction

Given an SOA application, we need to construct the service graph representation, which helps in the extraction of the microservices. Every SOA based application has an API document that contains the complete information about the services, operations in each service, and input/output parameters of each operation. If the application is implemented using web services, then we will have a WSDL file as the API document, and if we implement the application as normal services, then we have XML format of the complete application. Operations in each service are termed as a process. Using the API as input, we present algorithm 4.1 and algorithm 4.2 to construct the service graph and the task graphs for the SOA application.

In the algorithm 4.1, we can skip the first two steps if the application is not built using web services. For the normal implementation of SOA as services, the XML file can be directly generated, and we need not convert it again to XML format. In the service graph, we are adding an undirected path between the services as the entry point can be from any

Algorithm 4.1 Service_Graph_Construction_for_SOA

Input: API (WSDL) file of SOA based application
Output: Service Graph $G = (V, E)$

Begin
Read API file
Convert API to XML format
Parse the generated XML
Extract serviceNames from parsed file
▷ Each serviceName represents a service
V=$\{s_1, s_2, \cdots s_n\}$
where $s_i \leftarrow$ service and n $\leftarrow$ number of services
for each service s_i, i $\leftarrow 1, n$ **do**
    Extract inputs and outputs of each service s_i
end for
s_k.input $\leftarrow$ input set of service s_k
s_k.output $\leftarrow$ output set of service s_k
for $i \leftarrow 1, n$ **do**
    for $j \leftarrow 1, n$ **do**
        if i $\neq$ j and s_i.output $\cap$ s_j.input $\neq \emptyset$ **then**
            ▷ Add edge from s_i to s_j in G
            E = E $\cup$ $\{(s_i, s_j)\}$
        end if
    end for
end for
for each service $s_i \in V$ **do**
    Call Task_Graph_Construction(s_i)
end for
return G
End

services in the application based on the business requirements. As each service is a set of processes, the dependency among the processes inside each service is represented as a task graph discussed in the next section.

4.2 Task Graph

Task graph is a directed acyclic graph where each node represents the process, and the edge between the node represents the dependency of one node on another. Each service in SOA may contain one or more processes performing different tasks (based on the defini-

tion of service from equation 1), and therefore we generate a task graph for processes in each service. The task graph represents the application with a directed acyclic graph G(V, E), where V is the set of nodes, each node representing a process, and E is the set of arcs between communicating processes. Service graph with the task graphs inside each SOA service is represented as shown in Figure 4.1 where s_1,s_2,s_3,... are services and p_1,p_2,p_3,... are processes inside each service.

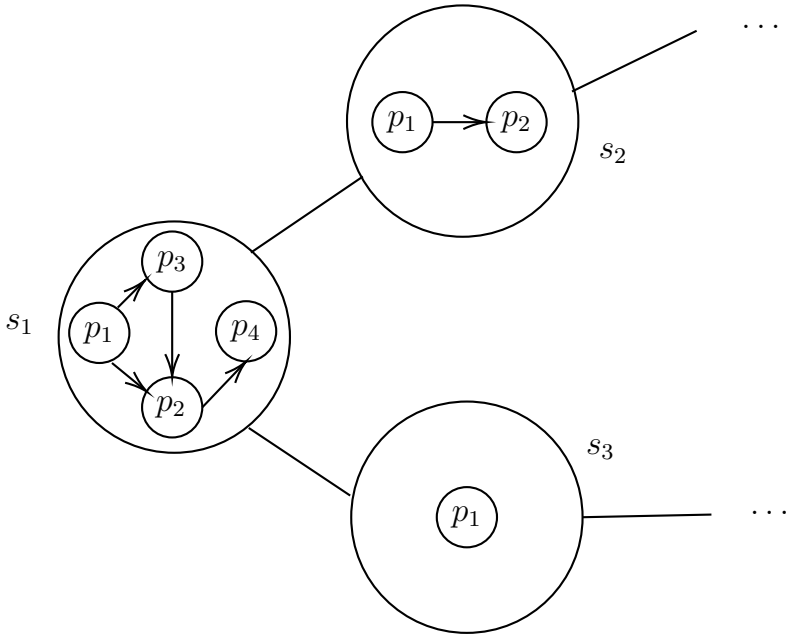

Figure 4.1: Service graph containing task graphs

Task graph construction

The processes in each service as represented as a task graph as shown in Figure 4.1. Each service that is built using the concepts of SOA will have an API file representing the operations performed in the service. As mentioned in section 4.1, we may have a WSDL file or XML file based on the design approach of the application. For generating the service graph, the complete API of the application is used, whereas to generate the task graph for a single service, we consider the API of the particular service only. The API document comprises the set of operations along with the input and output parameters involved in the operation. The sequence of the tasks is determined by the inputs and outputs. If any of the services have independent operation, they are connected with the input of the entire

service in which the task is performed. Services in SOA do not have constraints like microservices to perform only one business task. Therefore, they can be multiple operations in each service, and we represent the processes as task graph using the algorithm 4.2. In the task graph, the processes execute the business requests in some particular order. Hence, we have represented the edges as directed paths.

Algorithm 4.2 Task_Graph_Construction(s_i)

Input: API (WSDL) file of a SOA service
Output: Task graph G_t=(V_t,E_t)

Begin
Read API file
Convert API to XML format
Parse the generated XML
Extract operations from parsed file
▷ Operation is designed as a process p_i
$V_t = \{p_1, p_2, \cdots, p_n\}$
where $p_i \leftarrow$ process and n $\leftarrow$ number of processes
for each process p_i, i $\leftarrow 1, n$ **do**
    Extract inputs and outputs of each process p_i
    V_t= $V_t \cup \{p_i\}$
end for
p_k.input $\leftarrow$ input set of service p_k
p_k.output $\leftarrow$ the output set of service p_k
for $i \leftarrow 1, n$ **do**
    for $j \leftarrow 1, n$ **do**
        if i $\neq$ j and p_i.output $\cap$ p_j.input $\neq \emptyset$ **then**
▷ Add edge from p_i to p_j in G_t
            E_t = $E_t \cup \{(p_i,p_j)\}$
        end if
    end for
end for
return G_t
End

4.3 Microservices extraction algorithm

Now, we extract the microservices from the service graph using algorithm 4.3, and it generates a set of microservices as the output. The variable Set_m in the algorithm indicates the set of candidate microservices.

Algorithm 4.3 Microservices_Extraction

Input: Service graph $G = (V, E)$
Output: Set of candidate microservices Set_m

Begin
$V = \{s_1, s_2, \cdots, s_n\}$ where s_i is a task graph.
for $i \leftarrow 1, n$ **do**
    visit the node $s_i \in$ V, if not visited before
    calculate the order of node s_i
    if $|s_i|$=1 **then**
        Set_m = $Set_m \cup s_i$
    else
        for $j \leftarrow 1, n$ **do**
            i. visit node p_j where $p_j \in s_i$
            ii. Set_m = $Set_m \cup p_j$
        end for
    end if
end for
return Set_m
End

4.4 Service graph generation for microservices

After extracting the services, we retain the interactions of each service by constructing the service graph for microservices based application.

Algorithm 4.4 Service_Graph_for_Microservices

Input: Service graph of the SOA application, G = (V,E) with V = $\{S_1, S_2, \ldots, S_n\}$ where S_1,S_2,. . .,S_n are task graphs.

Output: Service graph of microservices application, G' = (V',E')

Begin
Let the given service graph vertex S_i = (V_i,E_i) is a directed acyclic graph (DAG) of order k_i where, V_i = $\{p_i^1, p_i^2, p_i^3, \ldots p_i^{k_i}\}$.
Compute V' = $\cup_{i=1}^{n} V_i$
Compute E' = $\{(p_i^s, p_j^t) : p_i^s \in V_i, p_j^t \in V_j, (S_i, S_j) \in E, 1 \leq \text{s} \leq k_i, 1 \leq \text{t} \leq k_j\} \cup E_1 \cup E_2 \ldots \cup E_n$.
return G'
End

Thus, the generated graph G' = (V', E') represents the service graph for the microservices application where V' represents the services in the microservices application, and E' represents the dependency among different microservices. The generation of service graph and extraction of microservices using the proposed approach is demonstrated through a case study application.

4.5 Case Study: Vehicle Management System

We applied our proposed algorithms to the same web-based application considered in Chapter 3. However, the service graph representation in this chapter for the chosen application contains the task graphs in each service (node) of the service graph. The service graph, along with its internal task graphs of the VMS application, is illustrated in Figure 4.2.

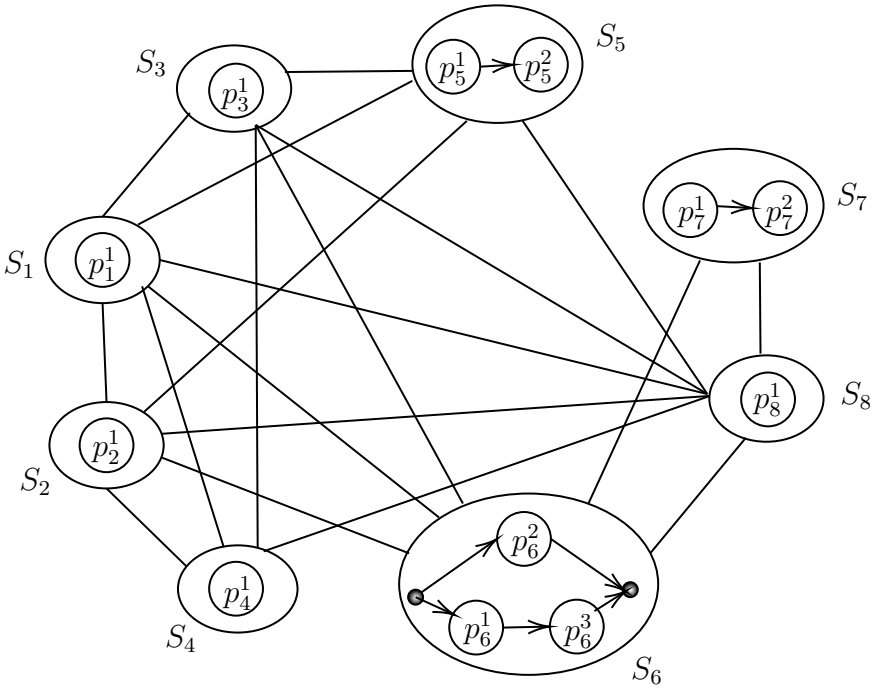

Figure 4.2: Service graph representation of SOA based application

The details of the services of the SOA based VMS application are given below.

S_1**:** This service is used for configuring the details of the vehicle's parts, products,

price information, dealer and to manage the inventory. The details are configured through the user interface.

S_2**:** This service provides information related to different parts of the vehicle available for the model. The details are configured through service S_1.

S_3: This service provides information related to different models of the available vehicles.

S_4: This service is used to compare different models or parts of the vehicles.

S_5: This service is used to get the information related to price of each vehicle and price of each part. It also searches for applicable incentives for the chosen vehicle or part.

S_6: The details of the vehicle and dealers who provide quotations to customers are extracted using this service.

S_7: The details entered through user interface are read and converted into a lead using this service. These leads are sent to dealers for the business.

S_8: This is the front-end part of the application through which all the business operations are performed.

4.5.1 Extraction of microservices

Applying the algorithm 4.3, we now extract the set of candidate microservices from the constructed service graph as shown in Figure 4.2. Let the Set_m=$\emptyset$ initially. We need to visit each node of the graph and calculate the order of task graph in that particular node. Therefore, the order of each service in graph G is as given below.

$| S_1 |= 1$, $| S_2 |= 1$, $| S_3 |= 1$, $| S_4 |= 1$, $| S_5 |= 2$, $| S_6 |= 3$, $| S_7 |= 2$, $| S_8 |= 1$

For services S_5, S_6, S_7, We need to visit each node of the task graph and add it to the Set_m. Therefore, the final set of candidate microservices is given as Set_m=$\{p_1^1, p_2^1, p_3^1, p_4^1, p_5^1, p_5^2, p_6^1, p_6^2, p_6^3, p_7^1, p_7^2, p_8^1\}$

4.5.2 Service graph construction

The proposed approach to extract microservices described in this section is based on the eight services in the VMS application. We construct the service graph of the application using the proposed approach. Let us consider the service graph of SOA application as shown in Figure 4.2 as input graph G = (V, E). As there are eight services in the given service graph, it represents the eight vertices of the graph. From the service graph,

$V = \{S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8\}$ and

$E = \{(S_1, S_2), (S_1, S_3), (S_1, S_4), (S_1, S_5), (S_1, S_6), (S_1, S_8), (S_2, S_4), (S_2, S_5), (S_2, S_6),$
$(S_2, S_8), (S_3, S_4), (S_3, S_5), (S_3, S_6), (S_3, S_8), (S_4, S_8), (S_5, S_8), (S_6, S_7), (S_6, S_8), (S_7, S_8)\}$

where $S_1, S_2, \ldots, S_8$ are the services of the application which internally consists of directed acyclic graphs (DAG). Now we represent vertices and edges of DAG in each service. The first service G_1 is represented as $S_1 = (V_1, E_1)$ where $V_1 = \{p_1^1\}$ as it consists of only one process in the service S_1 and $E_1 = \emptyset$ as it has no dependency with other processes. Similarly, other services are represented as below.

$S_2 = (V_2, E_2)$: where $V_2 = \{p_2^1\}$ and $E_2 = \emptyset$,

$S_3 = (V_3, E_3)$: where $V_3 = \{p_3^1\}$ and $E_3 = \emptyset$,

$S_4 = (V_4, E_4)$: where $V_4 = \{p_4^1\}$ and $E_4 = \emptyset$,

$S_5 = (V_5, E_5)$: where $V_5 = \{p_5^1, p_5^2\}$ and $E_5 = \{(p_5^1, p_5^2)\}$

$S_6 = (V_6, E_6)$: where $V_6 = \{p_6^1, p_6^2, p_6^3\}$ and $E_6 = \{(p_6^1, p_6^3)\}$

$S_7 = (V_7, E_7)$: where $V_7 = \{p_7^1, p_7^2\}$ and $E_7 = \{(p_7^1, p_7^2)\}$

$S_8 = (V_8, E_8)$: where $V_8 = \{p_8^1\}$ and $E_8 = \emptyset$.

We need to determine the target output graph $G' = (V', E')$ which represents the service graph of microservices application.

As $V' = \cup_{i=1}^{8} V_i$,

$V' = \{p_1^1, p_2^1, p_3^1, p_4^1, p_5^1, p_5^2, p_6^1, p_6^2, p_6^3, p_7^1, p_7^2, p_8^1\}$

We compute edge set E' by considering the edges in the input graph.

$E' = \{(p_1^1, p_2^1), (p_1^1, p_3^1), (p_1^1, p_4^1), (p_1^1, p_5^1), (p_1^1, p_5^2), (p_1^1, p_6^1), (p_1^1, p_6^2), (p_1^1, p_6^3), (p_1^1, p_8^1),$
$(p_2^1, p_4^1), (p_2^1, p_5^1), (p_2^1, p_5^2), (p_2^1, p_8^1), (p_3^1, p_4^1), (p_3^1, p_5^1), (p_3^1, p_5^2), (p_3^1, p_7^1), (p_3^1, p_8^1), (p_4^1, p_7^1),$
$(p_4^1, p_8^1), (p_5^1, p_5^2), (p_5^1, p_8^1), (p_5^2, p_8^1), (p_6^1, p_6^3), (p_6^1, p_7^1), (p_6^1, p_7^2), (p_6^1, p_8^1), (p_6^2, p_7^2), (p_6^2, p_8^1),$

$(p_6^3, p_7^1), (p_6^3, p_7^2), (p_6^3, p_8^1), (p_7^1, p_7^2), (p_7^1, p_8^1), (p_7^2, p_8^1)\}$

Thus the service graph for microservices is represented using the graph G' = (V',E'). For better understanding of the application in generated graph G', nodes are renamed as services ms_1, ms_2, ms_3, ..., ms_{12}. We have generated the service graph for microservices application using the V' and E' as shown in Figure 4.3.

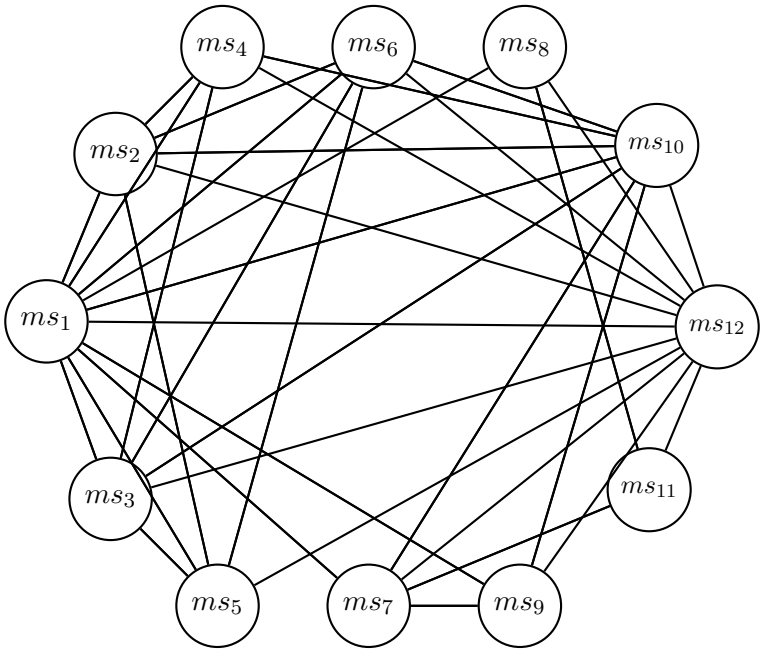

Figure 4.3: Service graph representation of microservices based application

The details of the functionality of each microservice extracted from SOA based applications are as given below:

p_1 This service is used for configuring the details of the vehicle's parts, products, price information, dealer and to manage the inventory. The details are configured through the user interface.

p_2 This service provides information related to different parts of the vehicle available for the model. The details are configured through service p_1.

p_3 This service provides information related to different models of the available vehicles.

p_4 This service is used to compare different models or parts of the vehicles.

p_5 This service fetches the incentives applicable to the parts or models of vehicles selected.

p_6 The price of each part of the vehicle and different models of vehicles are extracted from this service.

p_7 The list of dealers available are presented with this service.

p_8 Once the user selects the parts and models of vehicles, this service generates a Quote based on the inputs of the user.

p_9 The dealer information is fetched once the lead is generated.

p_{10} All the details of the available parts and models are presented through the service.

p_{11} This service generates the lead with the user information.

p_{12} This is the front-end part of the application through which all the business operations are performed.

4.5.3 Discussion on proposed approach

The service graph constructed with the proposed approach represents the microservices and their dependencies with other services in the application. Microservices architecture follows the principle of single responsibility, where each service should accomplish only one business task. The VMS application has few services which perform only a single task, and few services are loaded with multiple business requirements. We applied the proposed approach, partitioned the services, and generated individual microservices that perform only a single task. The benefit of our approach is communication between the services in the chosen SOA application hold intact in the service graph of microservices application. We also get to know the services which have to be redesigned as microservices as few existing services are performing only one task. Therefore, it helps the developer in easy migration and saves time for migration. The number of services can be identified from the service graph of microservices. It helps in doing other software engineering activities like effort and cost estimation for migration of the application.

Apart from the benefits of our approach, it has a few limitations in the extraction of microservices. We have taken a simple application to demonstrate our approach and representation of large enterprise applications as service graphs may be complex. To overcome this, architects should carefully analyze the system and generate the service graph automatically using graph generation tools. In the future, we plan to consider the database also for partition as it is suggested to have an individual database for each microservice.

4.6 Evaluation of the extracted microservices

One of the main reasons for migrating towards microservices is that it exhibits better QoS compared to SOA based services. However, to the best of our knowledge, very few works have been done in comparing both SOA based services and microservices. In one of our earlier works, we have compared the services of both the architectures with respect to performance, complexity, and scalability [87]. The results conclude that though the complexity of microservices is higher, it has better response time and throughput compared to SOA based services.

4.6.1 Evaluation criteria

In addition to the performance and complexity parameters, we consider loose coupling as evaluation criteria for comparing both microservices and SOA based services. Loose coupling is one of the essential characteristics of the service-oriented design. Coupling is measured by the level of dependency each service has on other services in the system. Coupling between the services should be minimal such that it holds the standards of service design principles. If the coupling between the services increases, the complexity of the architecture also increases. Hence coupling is a crucial factor among all the principles of SOA. Other SOA principles, including statelessness, scalability, etc., are directly or indirectly related to loose coupling which are discussed below. The relation between coupling and other principles is represented in Figure 4.4.

- Nature of loose coupling minimizes cross-service dependencies, by this service au-

tonomy is achieved [88].

- Loose coupling frees the tight dependencies on other components. This increases their availability for reuse opportunities [89, 90].
- Service statelessness is achieved by designing loosely coupled services [91].
- Scalability is achieved if the services are loosely coupled [92].

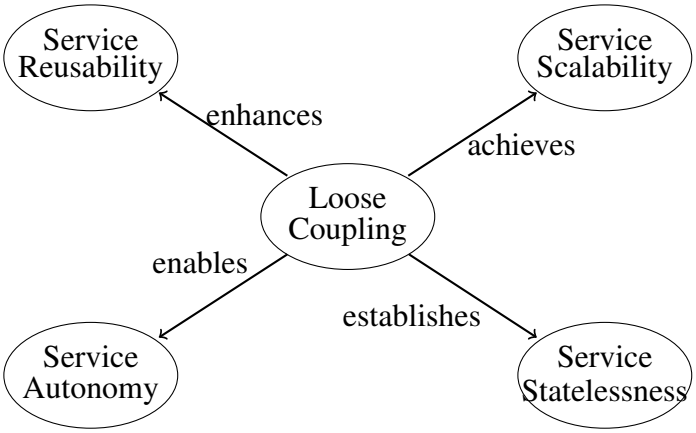

Figure 4.4: Relation between coupling and other SOA principles

4.6.2 Extraction of metric values from service graph

Service graph provides the details of basic metrics which are used to determine other metric values. Number of Services (NoS) value is given by the count of nodes in the service graph, NoS= n. Coupling of Services (CS) value is given by the degree of each node as given in equation 4.1.

$$CS_i = deg(s_i) \tag{4.1}$$

Relative Coupling of Services (RCS) denotes the degree of coupling in a particular service [86]. RCS of service is calculated using the formula in equation 4.2.

$$RCS[s] = \frac{CS[s]}{NoS} \tag{4.2}$$

The complexity of the application can be measured with this metric. Coupling intensity of a service is directly proportional to the value of the RCS. The complexity of the service-oriented system is also indicated with another metric, Service Coupling Factor (SCF). It is calculated as given in equation 4.3.

$$SCF = \frac{\sum_{s \in S[*]} CS[s]}{NoS^2 - NoS} \tag{4.3}$$

SCF metric is used to indicate the overall coupling of the application. The value of SCF ranges between 0 and 1. The lower the SCF value, the better is the system. Moreover, any service in a service-oriented system cannot have the values of SCF as 0 or 1.

4.6.3 Evaluation of SOA based application

Using the service graph for SOA based application as given in Figure 4.2, metric values are calculated and presented in Table 4.1. The number of services (NoS) in SOA based application is eight as we have eight nodes in the service graph. Coupling Value (CS) of services is the number of interactions each service has with other services, and Relative Coupling of services (RCS) value depends on CS value.

For example, the CS value of *Incentives and Pricing Service* is four as it has communication with *Config Service, Part Service, Product Service* and *User Interface Client* services. The corresponding RCS is calculated by $\frac{4}{8} = 0.5$, where 8 is the NoS value. Similarly, CS and RCS values are calculated for all services. Services are assigned a number from 1 to 8, coupled services corresponding to each service are also given with the assigned service numbers in Table 4.1. The interacting services field in the table indicates the services with which the given service has communication in the application.

4.6.4 Evaluation of microservices based application

We calculate the CS and RCS values for the services of the application built using the microservices style. Using the service graph generated for microservices based application, we extract the values for the metrics defined. The number of services (NoS) in this style

Service #	Service Name	Interacting Services	CS value	RCS value
1.	Config Service	2,3,4,5,6,8	6	0.75
2.	Part Service	1,4,5,6,8	5	0.62
3.	Product Service	1,4,5,6,8	5	0.62
4.	Compare Service	1,2,3,8	4	0.5
5.	Incentives and Pricing Service	1,2,3,8	4	0.5
6.	Dealer and Inventory Service	1,2,3,7,8	5	0.62
7.	Lead service	6,8	2	0.25
8.	User Interface Client	1,2,3,4,5,6,7	7	0.87

Table 4.1: List of services with CS & RCS values of SOA based application

is 12. As like the SOA application, the Coupling of Services (CS) and Relative Coupling of Services (RCS) are calculated for microservices based application. Each service, its CS and RCS value are presented in Table 4.2.

4.6.5 Results

We evaluated both the systems using the metrics related to coupling derived from the service graphs. The results are presented in terms of coupling as it is an important principle of concentration in this work. The impact of coupling values on other principles is also discussed.

4.6.5.1 Comparison based on RCS values

The RCS values of both the applications are calculated as shown in Tables 4.1 & 4.2. A graph is plotted with the values of CS and RCS values as shown in Figure 4.5 in which Coupling of Services (CS) values are represented in the X-axis, and Relative Coupling of Services (RCS) values are represented in the Y-axis. It is observed from the graph that microservices architecture has low RCS values. If the coupling values are less, then the architecture is good for designing enterprise applications.

Service #	Service Name	Interacting Services	CS value	RCS value
1.	Config Service	2,3,4,5,6,7,9,10,12	9	0.75
2.	Part Service	1,4,5,6,10,12	6	0.5
3.	Product Service	1,4,5,6,10,12	6	0.5
4.	Compare Service	1,2,3,10,12	5	0.41
5.	Incentives Service	1,2,3,6,12	5	0.41
6.	Pricing Service	1,2,3,5,10,12	6	0.5
7.	Dealer Service	1,9,10,11,12	5	0.41
8.	Get-A-Quote Service	11,12	2	0.16
9.	Dealer Locator Service	1,7,10,12	4	0.33
10.	Inventory Service	1,2,3,4,6,7,9,12	8	0.67
11.	Lead Processor Service	7,8,12	3	0.25
12.	User Interface Client	1,2,3,4,5,6,7,8,9,10,11	11	0.91

Table 4.2: List of services with CS & RCS values of microservices based application

4.6.5.2 Comparison based on SCF

- The SCF value can be calculated using the Number of Services (NoS) value and the total sum of CS values. From Table 4.1 of web services based application, the total of CS values is 38, and the NoS value is 8. Using the metric to calculate SCF value, SCF for web services=$\frac{38}{8^2 - 8}$ = 0.67
- Similarly for microservices application, from Table 4.2, the sum of CS values is 70, and NoS value is 12.
  SCF for microservices style = $\frac{70}{12^2 - 12} = 0.53$.
- Any system with less SCF score has low coupling between the services. The SCF score for web services based application is 0.67, and for microservices-based application, it is 0.53. Therefore, the overall SCF is better for microservices when comparing with web services. Microservices architecture is 20% less complex when comparing the SCF scores with web services architecture for the chosen web-based application.

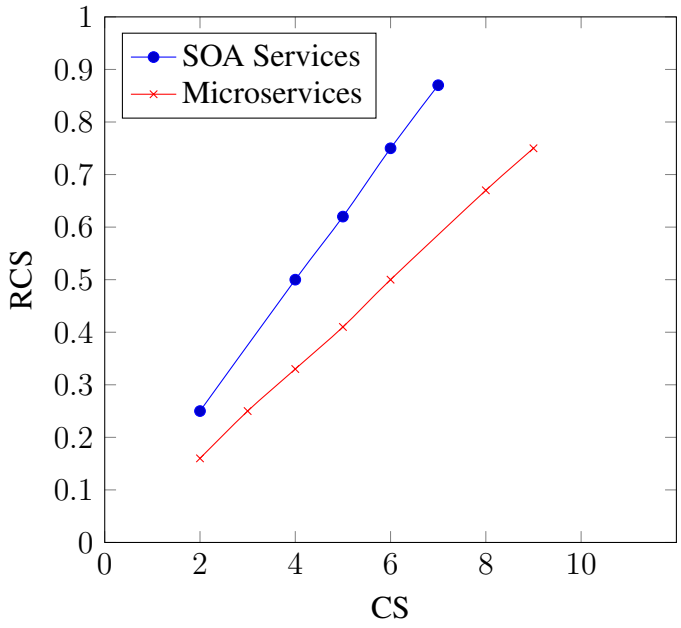

Figure 4.5: Comparison of coupling intensity

4.6.6 Discussion on comparison

From the above evaluation and comparison of extracted microservices with SOA services, it is clear that microservices have low coupling. Though the number of services is more in the microservices application, the overall coupling intensity is less compared to SOA based application. It makes the developers easily handle additional change requirements, and the time taken to deploy the services is reduced. Moreover, as the microservices are extracted from SOA services, it becomes very easy to configure and deploy the extracted services. As already discussed, many features and QoS parameters are dependent on loose coupling, and from the results, it can be concluded that microservices have better QoS values compared to SOA services.

4.7 Summary

To identify the candidate microservices from SOA applications, we proposed a new approach using the service graph. In this approach, we use the service graph representation of SOA based application along with the task graph in each node of the service graph. We presented algorithms for the construction of the service graph of a given SOA application,

microservices extraction, and for constructing the service graph of the microservices application, which is to be designed. The generated service graph acts as a blueprint for the new application to be designed, and it helps in easy and fast migration to microservices. Additionally, the dependencies among the services are also represented in the service graph of the microservices application. We have evaluated the extracted microservices w.r.t loose coupling and compared them with existing SOA services. It is clear from the results that microservices have low coupling compared to SOA services.

Chapter 5

A novel effort estimation approach for migration of SOA applications to microservices

As the pros and cons of using microservices are not known, some of the architects are hesitant to migrate the applications to microservices architecture. The major challenge is estimating the effort required to migrate the existing applications to microservices [51, 93].

To the best of our knowledge, there has been no work or very little work done in estimating the effort required for migration of SOA based applications to microservices architecture. In this chapter, we attempt to propose an approach for effort estimation by recasting the existing use case point model by enhancing it to suit appropriately for microservices. Generally, effort estimation requires knowing about the system before the design phase, and it is difficult. Service graph representation of the microservices application which is generated by the migration approach is used [84] and it gives detailed information about the number of services and dependency it has on other services. The main contributions of this chapter are described below:

- Proposed different types of services involved during the migration process.
- Proposed an approach for estimating the effort required for migration considering the service graph. The technical and environmental factors are updated such that they are

suitable for microservices architecture.

- Demonstrated the proposed approach on an SOA based web application.
- We also apply multiple regression analysis on the proposed approach with the Leave-N-Out policy. To evaluate and compare the proposed techniques, seven SOA based applications migrated to microservices are considered as the dataset.

5.1 Types of services involved in migration process

To migrate SOA based applications to a microservices architecture, the monolithic services need to be broken into small and independent services. However, there may exist few services in SOA based application which perform a single business task and can be directly considered as microservices. For systematic estimation of the effort, business services are classified into available, migrated, new, or composed services [94]. However, many other types of services are involved in achieving the business requirements, such as utility services, process services, proxy services, integration services and, suspended services, etc. Here, we discuss the significance of each service in the migration of SOA to microservices architecture.

- **Available service:** Services that can be used directly in the new architecture are treated as available services. Service, which does a single business task and is independent of other services, can be directly considered as microservice. It requires no development effort, and hence it is considered as available service.
- **Migrated service:** Service, which is extracted from legacy applications and generated by applying different migration strategies, is considered as migrated service. Here, the services in SOA which are partitioned to form microservices will be considered as migrated service. These services require an effort to redesign the new application.
- **New service:** Service, which is built from scratch and required for achieving business needs, is considered a new service. It requires effort, and it is very easy to calculate

the effort for a new service. However, as both SOA and microservices architectures are service-based systems, no new services will be required while migration from SOA to microservices. Therefore, we will not consider this kind of service in effort estimation.

- **Composed service:** Service, which is formed by combining one or more services, is considered as composed service. By the definition of microservices, each service should perform only a single task and independent from other services. Therefore, there will be no composed services in the new architecture.

It is inferred from the above that only the migrated services need to be considered in the effort estimation of the migration process. So the proposed model considers only the migrated services in the effort estimation.

5.2 Proposed approach

Our approach is stimulated from the use case points model of effort estimation. The use case point method depends on the use case diagram, and our model depends on the service graph as we estimate the effort for service-based architectures. The service graph is a blueprint for the application to be designed, and it gives complete information regarding the number of services and complexity of the services based on the dependencies on other services. Similar to the use case point method, we propose a service point (SP) model to estimate the effort required for migration to microservices. We classify the services and then calculate the weights and points using the classification of the services. Technical and environmental factors are two important factors that play a major role in effort estimation. The factors accessed for the existing use case point method do not suit well for microservices architecture. Therefore, we have updated the technical and environmental factors considering the principles of service-oriented systems. The steps for effort estimation using the service point technique are illustrated in Figure 5.1.

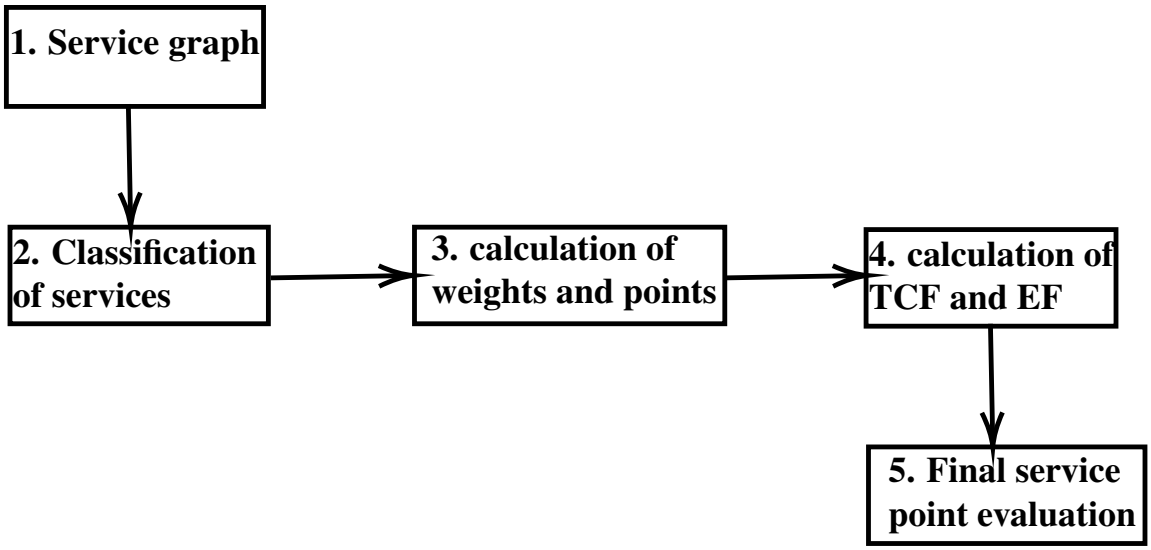

Figure 5.1: Service point calculation steps

5.2.1 Classification of services

The first step of the service point approach is to classify the services based on the interactions it has with other services. Unlike the use case point, we don't have actors entity here in the proposed approach, instead, some of the services act as actors for other services. So, we consider each service's dependencies on other services and classify them as simple, average, and complex. The service graph helps in the identification of services and their dependencies. A service is classified as simple if it interacts with less than four services, average if it interacts with less than eight services, and service is treated as complex if it interacts with more than or equal to eight services [95]. We use the terms simple, average and complex considering the impact of change requirements on other services. As mentioned, service is termed as simple, if it interacts with less than four services and to make a change in that particular service, it may not impact more than four services and hence it is considered as simple. Similarly, based on the number of interactions, we defined average as well as complex service types. Based on the complexity, different weights are assigned to each service, which is used in the calculation of service weights. The classification of services and the weights assigned are given in Table 5.1.

Service complexity	Number of interacting services	Weight
Simple	Less than or equal to 3	1
Average	4 to 7	2
Complex	More than 7	3

Table 5.1: Classification of services with weights

5.2.2 Calculation of weights and points

The next step is to calculate the unadjusted service points based on the weights assigned in Table 5.1. It is calculated by summation of number of services of each type multiplied by weight assigned to corresponding service type. Unadjusted Service Points (USP) is calculated as shown in equation (2).

$$USP = \sum_{i=1}^{3} S_i \times W_i \tag{5.1}$$

Where S_i is the number of services of type i and W_i is the corresponding weight of the service of type i where i={simple, average, complex}.

5.2.3 Technical and Environmental factors

We calculated the unadjusted service point value from equation 2 and the final value of the service point depends on technical and environmental factors. The 21 factors [95] relates to the factors which contribute to the complexity and the efficiency of the system. However, most of the factors included in existing works presented in the literature are related to object oriented systems and are not suitable for both service oriented architecture and microservices. Therefore, we have removed few factors and added new factors relevant to microservices architecture.

Each factor has a value assigned between 0 and 5 depending on the importance and impact the factor has on the system. In the existing use case points approaches, weights have been assigned based on the experience in their projects [95]. However, we have conducted an online survey to collect the inputs from different practitioners working on SOA and microservices architectures, software architects involved in the migration process and

developers working with microservices architecture. We have posted the online questionnaire on multiple social networking platforms, including the groups in LinkedIn, Twitter, and Facebook, etc. The questionnaire included the following questions.

1. What is the current role/designation of the participant?
2. How much work experience the participant has in SOA and microservices projects?
3. Does the participant has real time experience in migration projects?
4. How much rating does the participant would like to rate for each of the 21 factors?

The rating of each factor between 0 and 5 for each factor is collected through this survey. Based on the data collected, we have taken the average of ratings and assigned them to all the factors. The weights assigned and ratings of technical and environmental factors are indicated in Table 5.2 and Table 5.3.

Fi	Factors contributing to complexity	Wi	Rating
F1	Distributed systems	2	5
F2	Application performance objectives	1	4
F3	End-user efficiency	1	2
F4	Complex internal processing	1	2
F5	Reusability	1	3
F6	Easy installation	0.5	1
F7	Interoperability	0.5	2
F8	Portability	0.5	1
F9	Changeability	1	1
F10	Coupling	1.5	5
F11	Scalability	2	4
F12	Statelessness	1	3
F13	Independent deployment	1	4

Table 5.2: Technical factors

Fi	Factors contributing to the efficiency	Wi	Rating
F1	Familiar with cloud container	1.5	3
F2	Service configurations	1	2
F3	Analyst capability	0.5	4
F4	Application experience	0.5	2
F5	Cloud computing experience	1	2
F6	Motivation	1	5
F7	Polyglot	1.5	2
F8	Stable requirements	1	4

Table 5.3: Environmental factors

5.2.3.1 Calculation of Technical Complexity Factor(TCF)

To calculate the TCF, total weight of the 13 factors is calculated which is obtained by multiplying the value assiged to each factor between 0 to 5 and weights assigned to each factor. Calculation of TFactor is given by equation (3).

$$TFactor = \sum_{i=1}^{13} TF_i \times W_i \tag{5.2}$$

where TF_i is the rating of the technical factor i and W_i is the weight assigned to corresponding factor. Technical Complexity Factor (TCF) is calculated by the below equation (4).

$$TCF = 0.6 + (0.01 \times TFactor) \tag{5.3}$$

5.2.3.2 Calculcation of Environmental Factor (EF)

Similarly, the impact of environmental factors in the final service point is evaluated by finding the EF score. To calculate the EF value, the weight of each factor is multiplied with the rating assigned to each factor. It is given by equation (5).

$$EFactor = \sum_{i=1}^{8} EF_i \times W_i \tag{5.4}$$

where EF_i is the rating of the environmental factor i and W_i is the weight assigned to the corresponding factor. Environmental Factor (EF) is calculated by the below equation (6).

$$EF = 1.4 + (-0.03 \times EFactor) \tag{5.5}$$

5.2.4 Final service point evaluation

The final Service Points (SP) is calculated by multiplying the unadjusted service point with both technical and environmental factor values. It is given by the below equation (7).

$$SP = USP \times TCF \times EF \tag{5.6}$$

According to Karner, [95], the effort required to implement each use case point is 20 hours. Hence, we do consider the same 20 hours for each service point. Therefore, to estimate the final man-hours, the calculated service point should be multiplied by 20 to get the effort required for migration. Moreover, it is observed that the effort required for migrating and designing a microservices application is more compared to designing existing legacy applications [51].

We define the naming convention for different approaches proposed in this chapter. The service points approach with the ratings collected through the online survey is denoted as *SP-Proposed Approach*; the same service points approach with the Karner's default value as *SP-Karner's Approach* and the SP-Proposed approach with regression analysis as *SP-Regression Approach*. These notations are used throughout this chapter.

5.3 Empirical evaluation of the proposed approach

To evaluate the proposed approach, we choose a standard web application that is built based on SOA. In [83], the author has chosen a Vehicle Management System (VMS) application for the migration of the legacy application to SOA style. Taking the SOA based VMS application as input and applying the microservices extraction approach proposed by Raj, V. *et al.* [84], we have generated the service graph for the corresponding microservices based VMS application. The service graph of the microservices application is represented in Figure 5.2. The service graph is the prototype of a microservices application that has to be built through the migration process. From the service graph represented in Figure 5.2, it is clear that there are 12 services in the migrated system. The details of the SOA services, extracted microservices, and the type of services are mentioned in Table 5.4. As mentioned in Section 5.1, we will consider only the migrated service for estimating the effort as few services in SOA based applications can be directly considered as microservices. The calculation of service points, according to the proposed approach, is presented in the next section.

SOA services	**Microservices**	**Notation**	**Type**
Config Service	Config Service	S_1	Available
Part Service	Part Service	S_2	Available
Product Service	Product Service	S_3	Available
Compare Service	Compare Service	S_4	Available
Incentives & Pricing Service	Incentives Service	S_5	Migrated
	Pricing Service	S_6	Migrated
Dealer & Inventory Service	Dealer Service	S_7	Migrated
	Dealer Locator Service	S_8	Migrated
	Inventory Service	S_9	Migrated
Lead service	Get-A-Quote Service	S_{10}	Migrated
	Lead Processor Service	S_{11}	Migrated
User Interface Client	User Interface Client	S_{12}	Available

Table 5.4: Details of extracted microservices from SOA application

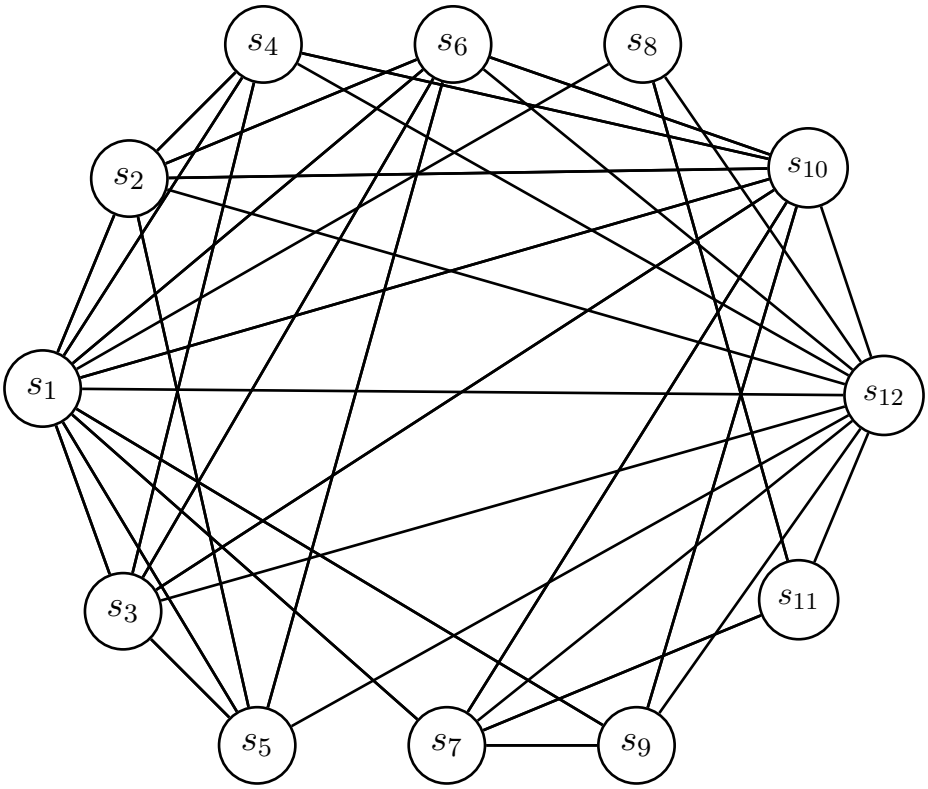

Figure 5.2: Service graph representation of microservices based application

5.3.1 Classification of services

The details of the services along with classification are presented in Table 5.5. Based on the classification and the weights and ratings of technical and environmental factors, we calculate the service point value used for migration of SOA based application to microservices architecture. Only the services with tick marks will be considered for effort estimation as they are migrated services. Efforts for available services will be considered zero.

5.3.2 Calculcation of USP

Unadjusted service point value is calculated by multiplying the number of services based on each classification and the weights assigned to each type. From the information from Table 5.5, there are 2 simple, 4 average and 1 complex services. Therefore, the value of USP is

$$USP = (2 \times 1) + (4 \times 2) + (1 \times 3) = 2 + 8 + 3 = 13.$$

Service #	Interacting Services	Classification	Services considered in estimation
S_1	2,3,4,5,6,7,9,10,12	Complex	
S_2	1,4,5,6,10,12	Average	
S_3	1,4,5,6,10,12	Average	
S_4	1,2,3,10,12	Average	
S_5	1,2,3,6,12	Average	✓
S_6	1,2,3,5,10,12	Average	✓
S_7	1,9,10,11,12	Average	✓
S_8	11,12	Simple	✓
S_9	1,7,10,12	Average	✓
S_{10}	1,2,3,4,6,7,9,12	Complex	✓
S_{11}	7,8,12	Simple	✓
S_{12}	1,2,3,4,5,6,7,8,9,10,11	Complex	

Table 5.5: Services along with classification for microservices based application

5.3.3 Effort estimation using SP Proposed Approach

The values of TCF and EF are calculated by considering the ratings collected through online survey. We take the average of ratings for each factor given by different software associates and use them in this approach.

Technical Complexity Factor: First, we need to calculate the TFactor using the information from Table 5.2. TFactor value is calculated as given below

$$TFactor = \sum_{i=1}^{13} TF_i \times W_i = 46.5$$

Now, we calculate the TCF value.

$$TCF = 0.6 + (0.01 \times TFactor) = 0.6 + (0.01 \times 46.5) = 1.065$$

Environmental Factor: Similarly, we calculate the EFactor using the information from Table 5.3 and then use this value of EFactor to calculate the EF value.

$$EFactor = \sum_{i=1}^{8} EF_i \times W_i = 23.5$$

Environmental Factor (EF) is calculated by the below equation

$$EF = 1.4 + (-0.03 \times EFactor) = 1.4 + (-0.03 \times 23.5) = 0.695$$

Final service point calculation: The service point is given as the product of USP, TCF, and EF. It is calculated as below.

$$SP = USP \times TCF \times EF = 13 \times 1.065 \times 0.695 = 9.62$$

The total effort required for migrating the SOA based VMS application to microservices is calculated by multiplying the number of services points with 20 hours.
Total estimated effort = 9.62 $\times$ 20 $\approx$ 193 hours.

5.3.4 Effort estimation using SP-Karner's Approach

Karner suggests that if we cannot fill the values for the factors for any reason, we can use the default value as 3 for all the factors [95]. Considering this default value for all factors, we calculated the TCF, EF and service points values.

Technical Complexity Factor

$$TFactor = \sum_{i=1}^{13} TF_i \times W_i = 42.$$

$$TCF = 0.6 + (0.01 \times TFactor) = 0.6 + (0.01 \times 42) = 1.02.$$

Environmental Factor:

$$EFactor = \sum_{i=1}^{8} EF_i \times W_i = 24.$$

$$EF = 1.4 + (-0.03 \times EFactor) = 1.4 + (-0.03 \times 23.5) = 0.68.$$

Final service point calculation:

$$SP = USP \times TCF \times EF = 13 \times 1.02 \times 0.68 = 9.0.$$

Total estimated effort = 9.0 $\times$ 20 $\approx$ 180 hours.

5.3.5 Observation

By considering the TCF, EF and SP values of both the approaches, the values are very close to each other. As we have used the rating collected through online survey in SP-Proposed approach and Karner's default value in SP-Karner's approach, the ratings of factors collected by online survey can be used as reference for estimating the effort of other projects as well.

Approach	TCF	EF	SP
SP Proposed Approach	1.065	0.695	9.62
SP-Karner's Approach	1.02	0.68	9.01

Table 5.6: Comparison of TCF, EF and SP values

5.4 Experimental Study

Machine learning models have been widely applied in software effort estimation and it has given promising benefits [96, 97]. In order to validate the efficiency of the proposed method, we choose seven applications of SOA which are migrated to microservices and we perform the regression analysis on the chosen datasets. In this section, the regression approach, description of the datasets and the measures to predict the accuracy of the proposed models are discussed.

5.4.1 Regression Analysis

It is one of the popular analysis methods to study the relationship between dependent and independent variables and present the relationship in the form of a model [98]. If we have more than two variables, then it is referred to as multiple regression, and it is the most preferred and applied method for cost estimation [99]. Since the proposed service point approach is based on multiple factors, including USP, TCF and EF, we define the multiple regression based effort estimation which is represented based on the below equation.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon \tag{5.7}$$

where y represents the effort caculated, x_1 represents the size metric calculated with USP of the chosen application and TCF, x_2 represents the adjustment factor (AF) considered as an independent variable in this multiple regression and the coefficients β_1, β_2, and β_0 represents the constant values. Here the additional ϵ represents the error induced during the calculation of the effort.

$$Size(x_1) = USP \times TCF \tag{5.8}$$

Adjustment Factor is calculated as the product of environment factor (EF) and the productivity factor (PF). The value of PF can be considered as 20 hours as proposed by Karner, if the projects do not have any historical data. We consider only the EF in the calculation of x_2 as TCF is already included in the first variable x_1.

$$AdjustmentFactor(x_2) = {}_pPF \times EF \tag{5.9}$$

However, we calculate the productivity factor ${}_pPF$ by dividing the actual effort by the SP as it gives the accurate effort required to implement each service point.

$${}_pPF = \frac{ActualEffort}{SP} \tag{5.10}$$

5.4.2 Datasets

The effort estimation for migration to microservices architecture is based on the service graph representation of the microservices application migrated from the SOA based application. Moreover, the use of microservices architecture has just started, and there are very few projects which are migrated from SOA. The authors in [100] state that data collection is more important for the validation of effort estimation techniques. Due to the inability to access SOA projects developed in the industry and the unavailability of datasets based on UML artifacts in the industry, the study research investigation is collected from [66]. The dataset is represented a dataframe and the size of the dataset is 7 rows with 5 columns. Out of the 7 applications we gathered, 5 applications were collected from Indian IT organizations and one application from a research centre in UK where a team is working on the best approaches for migration of SOA based applications to microservices. The remaining one application is developed at our university by a team of Post Graduate (PG) students, which is related to the online exam portal during the pandemic time. The details of the data collected are presented in Table 5.7. As per the proposed approach, we are considering only the services which are marked as migrated for estimating the effort required for migration. We generated the service graphs for the chosen applications and identified the number of services along with the complexity of the services. The Unadjusted Service Points (USP) is also calculated for the applications and is presented in Table 5.7.

Application	Total No.of Services			USP
	Simple	Average	Complex	
A1	6	5	2	22
A2	2	2	2	12
A3	6	10	2	32
A4	23	15	4	65
A5	0	7	3	23
A6	3	14	0	31
A7	20	15	21	113

Table 5.7: Characteristics of applications in dataset

5.4.3 Evaluation criteria

To evaluate the accuracy of the estimated approach, several frequently used measures [101] are considered such as Magnitude Relative Error (MRE), Mean of MRE (MMRE), Root Mean Square Error (RMSE), and prediction within 25% of the actual value. The definitions of the measures are discussed below.

- The magnitude of relative error (MRE) is calculated as given below.

$$MRE = \frac{ActualEffort - EstimatedEffort}{ActualEffort} \tag{5.11}$$

- MMRE is the mean of the MREs of all the applications and it is used to evaluate the prediction performance of the model. The Mean of MRE is calculated as

$$MMRE = \frac{1}{n}(\sum_{i=1}^{n} MRE_i) \tag{5.12}$$

- The Root Mean Square Error (RMSE) evaluates the difference between actual effort and the estimated effort. It is used to find the standard deviation of the errors which occur after applying the proposed method on the datasets. RMSE is calculated as given below.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(ActualEffort - EstimatedEffort)^2} \tag{5.13}$$

- To verify whether the predicted values are within m% of the actual values, we use a measure and in software engineering, the value of m is typically set to 25%. The measure PRED(25) is calculated as

$$PRED(25) = \frac{k}{n} \tag{5.14}$$

where k is the number of observations for those whose MRE is less than or equal to

0.25 and n is the total number of applications.

- The above discussed measures are criticised and behave differently when evaluating prediction models, hence two other measures are suggested [102]. Mean Absolute Error (MAE) is unbaised and it is calculated as given below.

$$MAE = \frac{1}{n}\sum_{i=1}^{n} \mid ActualEffort_i - EstimatedEffort_i \mid \tag{5.15}$$

where n is the number of applications chosen for evaluation of performance.

- Standardized Accuracy (SA) is one the recommended measures for comparing the performance of prediction models which is based on MAE. It is defined as follows:

$$SA = 1 - \frac{MAE_{Pj}}{MAE_{guess}} \times 100 \tag{5.16}$$

where MAE_{Pj} is the MAE of the proposed approach and MAE_{guess} is the value of MAE of a random guess. The standard accuracy represents the impact of MAE_{Pj} when compared to any random guess. For effort estimation techniques, the value of MAE should be less and SA should be maximum.

5.5 Experimental Results

The SP-Proposed method with the collected ratings, SP-Karner's method by considering Karner's default value and the proposed SP-Regression methods are applied on the 7 applications and the results obtained by comparing there three methods are presented in this section.

5.5.1 Application of SP-Proposed and SP-Karner's methods

The proposed service points approach is applied to the 7 applications, and the effort is calculated by considering the TCF and EF values. The effort is calculated by considering the collected ratings and also with Karner's default value. The PF value for calculating the

effort is taken as 20 man-hours. The magnitude of relative error (MRE) is also calculated with the help of the actual efforts of the applications. The results of the efforts calculated are presented in Table 5.8. The actual efforts of the chosen applications where gathered from the documentations of the projects managed by the project managers. From the data

Application	Actual Effort [h]	SP-Proposed Approach		SP-Karner's Approach	
		Estimated Effort [h]	MRE	Estimated Effort [h]	MRE
A1	396	326	0.176	305	0.229
A2	140	178	-0.271	166	-0.185
A3	587	474	0.192	444	0.243
A4	1205	962	0.201	902	0.251
A5	518	340	0.343	319	0.384
A6	502	459	0.08	430	0.143
A7	2034	1673	0.177	1567	0.229

Table 5.8: Applying service point approach to applications.

of Table 5.7, we can observe that if the number of complex services is high, then the value of USP is also high, and hence the effort is also high for such applications. All three types of services are combined to calculate the effort in this proposed approach. It is clear from Table 5.8 that the estimated effort is more accurate when using the ratings collected through an online survey when compared to the effort estimated by considering Karner's default value for factors while calculating the TCF and EF values. The results obtained by the ratings collected through online survey are better because the collected data is given by the professionals working or have good experience in microservices and SOA systems.

5.5.2 Application of SP-Regression model

The proposed regression method is applied on the same dataset and the function for effort estimation is obtained. First, the variables x_1 and x_2 are calculated with the equations (9) and (10). The calculated values of the variables are presented in Table 5.9. These values x_1 and x_2 will be used in the calculation of the effort using the regression model. We consider the TCF value which is calculated using the ratings collected through an online survey in

the regression analysis.

Using the calculated values of variables, we applied multiple linear regression on the applications. However, to improve the accuracy, we apply Leave-N-Out policy where we train the model only on the first 5 applications to calculate the coefficients and test with the remaining two applications. The values of the coefficients calculated are presented in the effort estimation function, given in the below equation.

$$\hat{y} = -274.10 + 15.017x_1 + 17.136x_2 \tag{5.17}$$

Application	**Size** (x_1)	**AF** (x_2)
A1	23.43	16.8
A2	12.78	10.9
A3	34.08	17.1
A4	69.22	17.3
A5	24.49	21.1
A6	33.01	15.1
A7	120.34	16.8

Table 5.9: Variable values for multiple regression analysis.

Using the calculated coefficient values β_0= -274.10, β_1=15.017, and β_2 = 17.136, we test the remaining two applications. The efforts calculated using the generated coefficients are presented in Table 5.10. It is clear from the table that the values are very close to the actual efforts of the applications. The proposed regression model works as a recommendation system for estimating the effort required for the migration of SOA applications to microservices. The coefficients generated are used to calculate the efforts for all the other applications which are used for training the model as well.

Application	Actual Effort [h]	Estimated Effort (Regression) [h]	MRE
A6	502	480.36	0.043
A7	2034	1940.99	0.045

Table 5.10: Application of regression model to testing data

5.5.3 Comparison

The estimation function is calculated for all the 7 applications by regression analysis on the 6 applications and leaving one application every time. The values of the coefficients are calculated and the effort by regression analysis is generated for all the applications. The results of the comparison of all the proposed methods are presented in Table 5.11. For each approach, the estimated effort in hours and estimation success values are given in the table. It is clear that the efforts calculated through the regression analysis give better values closer to the actual efforts required for migration. For application A2, the efforts estimated through SP-Proposed approach and SP-Karner's approach are more than the actual efforts. Hence the estimation success percentage is more than 100 which is marked as *. Generally, success percentage of more than 100 does not make sense. So, only the SP-Regression model gives better results for the application A2. The efforts estimated by the SP-Proposed approach, SP-Karner's approach, SP-Regression approach and the actual efforts are compared and presented as a bar graph as shown in Figure 5.3. The x-axis represents the 7 applications and the y-axis represents the efforts in man-hours for each application. From the graph, it is clear that the effort estimated with the regression model is close to the actual efforts of all the applications.

The accuracy of the proposed methods is evaluated using the measures discussed in section 5.4.3 and the values are presented in Table 5.12. From the results, the MAE value is very less and SA value is also better for the effort estimated through the regression model. Though the MMRE and PRED values are close to each other, the RMSE value is better only for the regression model.

Application	Actual Effort [h]	SP-Proposed Approach		SP-Karner's Approach		SP-Regression Approach	
		Estimated Effort [h]	Estimation Success (%)	Estimated Effort [h]	Estimation Success (%)	Estimated Effort [h]	Estimation Success (%)
A1	396	326	82.32	305	77.02	366	92.42
A2	140	178	127*	166	118*	105	75.0
A3	587	474	80.74	444	75.63	531	90.45
A4	1205	962	79.83	902	74.85	1062	88.13
A5	518	340	65.63	319	61.58	455	87.83
A6	502	459	91.43	430	85.65	480	95.61
A7	2034	1673	82.25	1567	77.04	1941	95.42

Table 5.11: Comparison of proposed effort estimation techniques

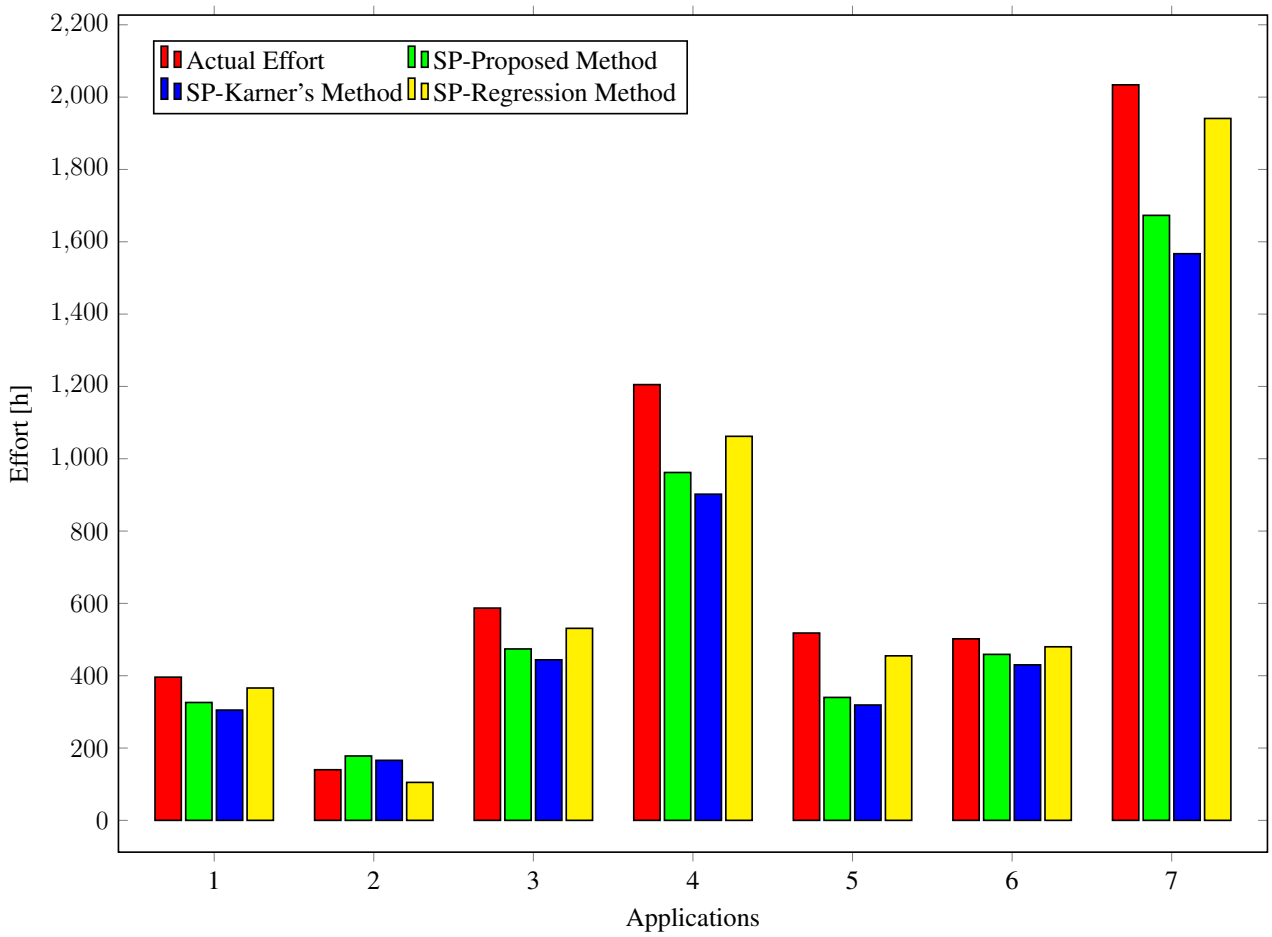

Figure 5.3: Comparison of efforts estimated by proposed methods.

5.5.4 Threats to validity

There are few threats that might affect the validity of the proposed service point approach.

1. Though it is one of the first attempts to propose an effort estimation technique for microservices architecture, we recast it from the use case point and proposed as a new approach considering the service graph. The first step in the proposed approach differs from the existing use case point as we consider the service graph instead of the use case diagram. Moreover, the technical and environmental factors are updated according to the SOA and microservices architectures. In some of the research articles in the literature have highlighted the disadvantages of the traditional use case point approach [103, 104].

Approach	MMRE	RMSE	PRED(25)	MAE	SA
SP-Proposed Approach	0.128	185.95	0.857	138.57	86.143
SP-Karner's Approach	0.184	234.24	0.714	178.42	82.158
SP-Regression Approach	0.107	74.55	0.857	63.24	93.67

Table 5.12: Accuracy of the proposed methods using different measures.

2. One of the threats is the dataset considered for validation of the proposed service point approach. We used only 7 projects as it was very difficult to get the projects which were migrated from SOA to microservices. Though there are other works published by considering a small number of projects, we cannot validate the proposed model with less number of projects.

5.6 Summary

In this chapter, we propose a new technique that is recasted from the well-known use case points technique to estimate the effort required for the migration of SOA based applications to microservices architecture. We have demonstrated the new technique through a case study application and calculated the effort required for migration. We have compared the results with effort calculated by considering the default values for factors suggested by Karner. Additionally, we propose a machine learning model using multiple linear regression with Leave-N-Out policy, where we train the model with N applications and test with the remaining all-N applications. The comparison results show that the efforts estimated using the SP-Regression model are close to the actual efforts, and the error rate is very low compared to the SP-Proposed approach and SP-Karner's approach.

Chapter 6

Patterns for migration of SOA based applications to microservices architecture

Many design and environmental challenges occur during the migration of one architecture to another architecture. Similarly, the migration of SOA based applications to microservices architecture also exhibits many challenges that occur during the migration process and post-migration. We identified the most common recurring problems such as extraction of microservices from monolithic applications, evaluating the size of microservices, bug detection in the complex microservices applications, etc. We proposed patterns for the mentioned recurring problems in this chapter.

The use of patterns for providing recursive solutions in the migration of architectures is less explored. Few researchers have contributed the patterns related to the migration of monolithic applications in the literature. Lessons learned while migrating legacy monolithic applications using the patterns are reported in [105]. Several patterns for migration of monolithic applications to microservices and how each pattern benefits the migration process are also addressed in [5]. The migration patterns needed for complete migration and its related challenges are presented in [106]. The patterns for rearchitecting the existing monolithic applications are also discussed but we are considering patterns for migration and decomposition of monolithic SOA services. Also, a new pattern called strangler pattern

is introduced that adds new microservices to the monolithic code [51]. In the long run, the monolithic services will gradually be replaced with microservices. However, the migration of SOA applications to microservices is our topic of interest.

Patterns for migration of SOA applications to microservices are proposed [107], but these patterns do not support the independent development and deployment of services. Under these scenarios, ESB is still used for communication between services. Legacy software modernization using microservices is proposed with the help of patterns proposed in [108]. However, the approach is not appropriate for migrating service oriented applications. The main contributions of this chapter are described below:

- Presented the patterns for the most commonly occurring problems highlighted in the literature.
- An SOA based web application is chosen for the demonstration of the proposed patterns. We have evaluated those patterns on the chosen case study application.

6.1 Patterns

Patterns act as a framework for presenting tested solutions to issues that are suitable in any given context. Every pattern has few sections which helps in understanding the problem as well as the solution. This includes the criteria/requirement, context, problem, solution, challenges, and the illustration or a diagram of the solution. The structure of the any pattern consists of:

1. **Name of the pattern**: Pattern name is used to define the problem, its solution and consequences. It increases the design vocabulary of the pattern and helps in understanding it.
2. **Requirement**: The problem space for the recurring problem and its technical, environmental and configurational properties are discussed. The need for the pattern is also highlighted in this step.

3. **Problem**: It describes the situation for applying the pattern. It also explains the actual problem and its context. Any additional conditions under which the problem may occur will also be discussed in this step.

4. **Solution**: It explains all the entities which make up the solution for the problem discussed. It does not give any implementation or design of the problem. The solution is like a template that can be applied in different situations.

5. **Challenges**: These are the consequences and trade-offs that appear after applying the pattern. It is important to list the challenges as it helps in the evaluation of the patterns.

The below section discusses the proposed patterns.

6.1.1 Pattern 1: Decomposition of an SOA service to Microservices

This pattern is used to address the recurring challenge faced by software architects while migrating legacy SOA based applications to microservices.

Requirement: There is a complex SOA based application and the software architects plan to migrate the application to the microservices architecture which will reduce the scalability issues and increase application performance. Also, the effort required for migration should be reduced. There are few monolithic services in the application and few basic services that can be called microservices. The presence of monolithic services makes the application complex and the best approach is to migrate the application to microservices.

Problem: How to split the monolithic services in the SOA based application to microservices? How to extract the microservices from the SOA services?

Solution: Graph theory has been widely used to solve many complex problems since it is easy to represent the components and their dependencies as a graph. The service graph, along with its internal task graphs, is a formal representation of every SOA application and it helps in identifying the monolithic services.

The architecture of microservices follows the single responsibility principle and states that each service should perform only one business function. SOA based applications can

contain few services that perform only one task and these services may be regarded as microservices directly. This reduces the migration effort as we do not need to consider all the services for migration to microservices.

Further, identification of monolithic services or the services which need to be migrated to microservices should be considered for the extraction of microservices. Using graph properties from the service graph, find out the order of each task graph and the service with the order as one can be directly considered as microservices and services with order more than one should be treated as monolithic services. Only such monolithic applications should be considered for the extraction of microservices.

Using the graph partition method [84], we can break the monolithic services into microservices and the independent task graph nodes represent the microservices. We use the service graph to identify complex services and generate microservices.

Challenges: Representing the SOA framework as a service graph is difficult for large enterprise applications, as it includes a large number of services and tools required for generating such large graphs is challenging.

6.1.2 Pattern 2: Size of each Microservice

This pattern is used when designing the microservices or extracting from service oriented based applications.

Requirement: There is a complex SOA based application and the architects want to migrate the application to microservices architecture. During migration, the SOA services are partitioned to generate the microservices. Yet one of the main issues faced by software architects, microservices developers, and practitioners is what the size of each microservice will be. By definition, microservices are small, independent, and scalable services that are deployed using cloud-based technologies. So, the question is how small each microservice should be.

Problem: What should be the size of each microservice? On what basis can we measure the size of the microservices?

Solution: SOA based applications have feature level services and microservices have

task level services. Every service in the SOA application consists of many tasks that perform the business requirements. Microservices follow single responsibility principle that requires each service to perform only one business function. We, therefore, use the service graph representation of the SOA based application and consider each task in the task graph to be microservices.

In this case, the size of the microservice is not a measurable metric. When a specific service performs only one operation, it may be considered as a microservice and the size of the service is not considered.

Challenges: While microservices perform a single task, there can be few services that perform multiple computations to meet the business requirements. These microservices can increase the complexity of the application.

6.1.3 Pattern 3: Bug Detection in Complex Microservices Application

This pattern is used to monitor and find bugs after migrating SOA-based applications to the microservice architecture.

Requirement: An SOA based application is migrated to microservices architecture with the help of Pattern 1 and Pattern 2. When the number of services increases in microservices based application, it is difficult to detect the bug if it happens. Among those multiple microservices, it is difficult to trace the bug and recognize the service that causes the bug in the entire application.

Problem: How to trace the service which is responsible for the bug? How to identify the location of the bug among all available microservices in the application?

Solution: Business requirements are specified prior to the design phase of the software development life cycle. The workflow of the business requirements is defined through the use of UML diagrams in the software requirement specifications document. Nonetheless, despite the nature of the specification, it is difficult to establish a connection between the business requirements and the services that execute the requirements. As a result, the service graph model allows us to map the criteria and the services that fulfill them.

Using the service graph representation, each workflow of the business requirements is

mapped with the nodes in the graph where each node has a service number. We need to define the workflow of each requirement and store the sequence of services it passes to fulfill the business request. Each sequence of service numbers is stored in the database and anytime a particular service fails, we need to check for all the sequences in which the service is involved. If a particular business request has failed, we get the corresponding sequence of the business requirement and trace only the services involved in the sequence.

Challenges: If the application is complex, it is difficult to create a service graph and store all the sequences in the database. Also, often a specific service can be involved in several sequences, so it is difficult to determine the appropriate sequence for a given bug.

6.2 Evaluation

The above mentioned patterns are used to solve the most common recurring problems which occur during the migration and after the migration of SOA based applications to microservices architecture. In Chapter 4, we have extracted the microservices from the SOA based Vehicle Management System (VMS) [83]. The challenges occurred during the actual migration process of this VMS application to microservices are resolved using the proposed patterns. The application of the proposed patterns on the vehicle management system is discussed in this section.

The VMS application is used to select, customize, and purchase vehicles and its parts using a web interface. This VMS web application is implemented and we created a service graph (SG_SOA) using the API documents as shown in Figure 6.1. There are eight services in the application and each service performs specific business tasks. Using the concept of task graph, the processes within each service are depicted as a task graph along with the edges between the processes. Using our earlier work of extracting microservices from SOA based applications [84], we have created the service graph for microservices as shown in Figure 6.2. The details of the services of both SOA and microservices based application along with the representations in service graphs are presented in Table 6.1. SG_SOA indicates the service graph of SOA based application and SG_MSA indicates the service graph of microservices based application. Both SG_SOA and SG_MSA are used in the validation

of the proposed patterns discussed in the below sections. Here, we present the application of proposed patterns on the case study application.

Notation in SG_SOA	**SOA services**	**Microservices**	**Notation in SG_MSA**
S_1	Config Service	Config Service	ms_1
S_2	Part Service	Part Service	ms_2
S_3	Product Service	Product Service	ms_3
S_4	Compare Service	Compare Service	ms_4
S_5	Incentives & Pricing Service	Incentives Service	ms_5
		Pricing Service	ms_6
S_6	Dealer & Inventory Service	Dealer Service	ms_7
		Dealer Locator Service	ms_8
		Inventory Service	ms_9
S_7	Lead service	Get-A-Quote Service	ms_{10}
		Lead Processor Service	ms_{11}
S_8	User Interface Client	User Interface Client	ms_{12}

Table 6.1: Details of services of both SOA and Microservices based applications

6.2.1 Pattern 1

From the given SOA based application, we need to extract the microservices by decomposing the monolithic services of SOA. There are eight services in the application and they are named S1, S2, S3, ... S8 in the service graph and each service has an internal task graph. Applying the pattern, the degree of each graph indicates the number of nodes in the graph. So, we need to calculate the order for each subgraph. Using the service graph information, order of S_1, $\mid S_1 \mid$= 1, as we have only one node in the service S_1. Similarly, for all other services, $\mid S_2 \mid = 1$, $\mid S_3 \mid = 1$, $\mid S_4 \mid$ =1, $\mid S_5 \mid$ =2, $\mid S_6 \mid$ =3, $\mid S_7 \mid$ =2, and $\mid S_8 \mid$ =1.

Now, we need to consider only the services with orders greater than one, so we have three services S_5, S_6, and S_7 which should be migrated to microservices. Using the graph partition approach, we can break the monolithic services to generate the microservices. The service graph provides an overview of the services and their interaction with other services and helps in easy migration for software architects.

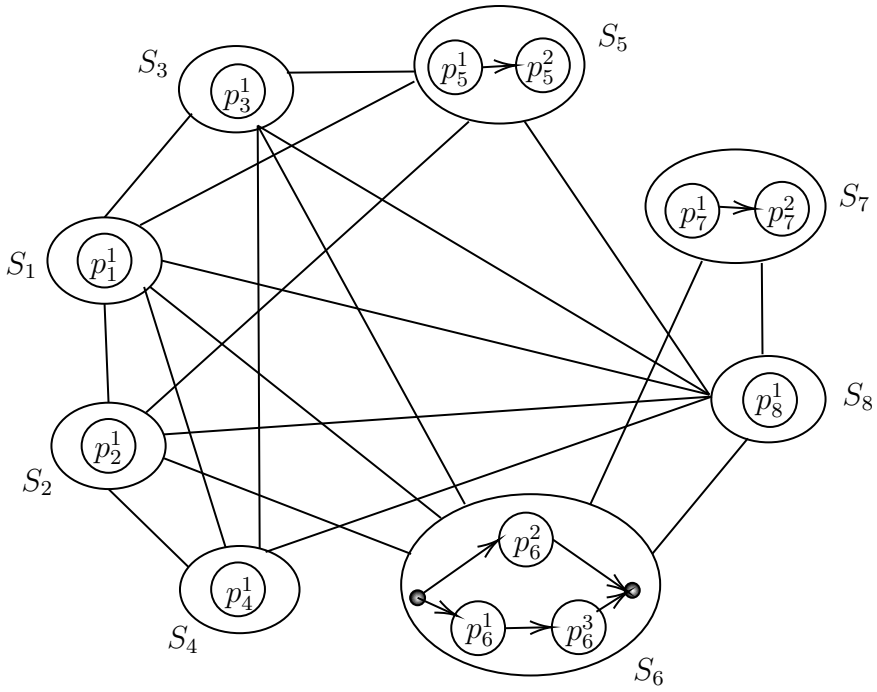

Figure 6.1: SG_SOA: Service graph representation of SOA based application

6.2.2 Pattern 2

The size of the microservices is immeasurable. Designing the service graph along with task graphs helps us in identifying the size of the microservice. By the definition of microservices, each service should perform only one business task and there is no specific metric to determine the size of the microservice. As a consequence, nodes in the task graph itself represent the microservices and each service performs only one task.

6.2.3 Pattern 3

In the complex application of microservices, if a service fails or a bug occurs, we need to trace the route cause of the failure. The applications designed using microservices generally have a complex network and it is difficult to trace and monitor the applications.

For the chosen application, the business requests should be stored in the database. From the service graph given in Figure 6.2, we extract the sequence of the services in the order of execution and map with the business request number. We have studied the chosen application, and a few of the requests are considered. The possible business requests of the

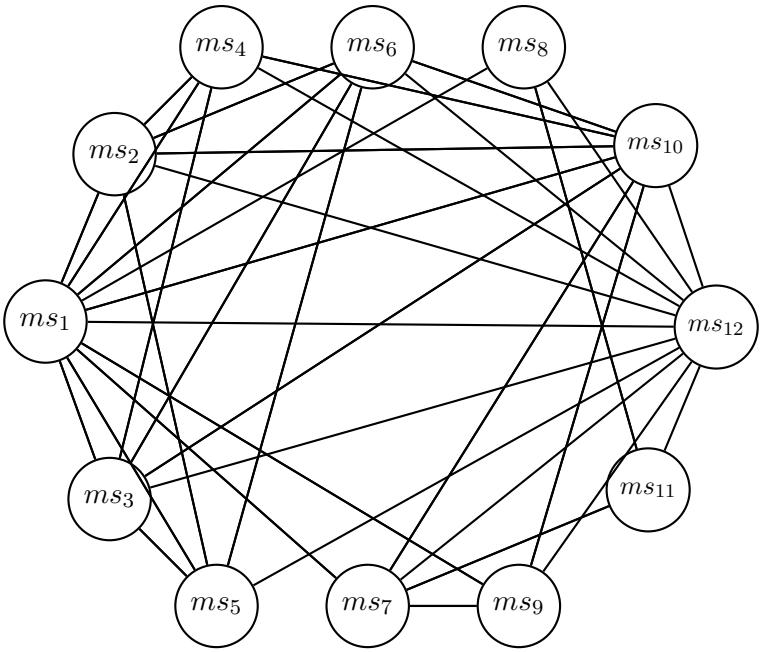

Figure 6.2: SG_MSA: Service graph representation of microservices based application

given application are stored as shown in Table 6.2.

Business Requests	**Sequence of services**
BR1	$ms_{12} - ms_1 - ms_3 - ms_2 - ms_6 - ms_5 - ms_9$
BR2	$ms_{12} - ms_9 - ms_3 - ms_2 - ms_4$
BR3	$ms_{12} - ms_9 - ms_3 - ms_2 - ms_6 - ms_5 - ms_4$
BR4	$ms_{12} - ms_8 - ms_9$
BR5	$ms_{12} - ms_9 - ms_{10} - ms_{11} - ms_8 - ms_7$
BR6	$ms_{12} - ms_1 - ms_7 - ms_8$
BR7	$ms_{12} - ms_1 - ms_5 - ms_6$

Table 6.2: Mapping of business requests with workflows

From the logs and monitoring data, if any service fails, the root cause of the failure for the particular business request can be traced. For example, if the service ms_7 fails, it is involved in business requests 5 and 6. We, therefore, need to analyze the error with the business requests. Business request 5 is to get the quote, generate the lead and select the

proper dealer for the vehicle and business request 6 is to configure the dealer along with the dealer's location. Based on the business request and the error, the bug can be easily traced and the problem can be quickly resolved.

6.3 Summary

In this chapter, we propose patterns for the most common recurring issues and these patterns help in easy migration. The problems addressed are identified in a literature study on the migration of legacy applications to microservices. The solutions provided are presented with our experience in migrating SOA based applications to microservices. We proposed solutions for some of the recurring problems which occur during the migration of SOA based applications to microservices architecture. The proposed patterns are best illustrated and evaluated using a standard case study application.

Chapter 7

Conclusion and Future Research

Here, we present the summary of contributions made in this thesis and mention the future scope for research. The overall conclusion of the thesis work and conclusion of each objective are discussed. The future scope of research from this thesis is also presented.

7.1 Conclusions

This thesis presents a framework for migration of service oriented applications to microservices style which includes the comparison of SOA & microservices applications, algorithms for extraction of microservices, the effort required for migration, and patterns for problems that occur during the migration process.

In chapter 3, a formal model to represent any given service based application called as service graph is proposed. To better understand the technical differences between the service oriented architecture and microservices, we presented a comparison between these two styles. The parameters used for comparison are complexity using the loose coupling and performance using the load testing. Based on the analysis done for the complexity of the applications, it is clear that microservices application has more number of services and application is more complex than SOA application. The coupling between the services is also less in microservices architecture. However, the performance testing shows better results for microservices with quick response times for 500 and 1000 users. Furthermore, the chosen case study application exhibits better results when chosen the business requests

with the same number of services in both styles. After this experimental study, we conclude that microservices architecture exhibits better performance results with the use of cloud-based environments and can be used in the design of large enterprise applications. This motivated us to migrate the SOA based applications to microservices architecture.

In chapter 4, to identify the candidate microservices from SOA applications, we proposed a new approach using the service graph. In this approach, we use the service graph representation of SOA based application along with the task graph in each node of the service graph. We presented algorithms for the construction of service graph of a given SOA application, microservices extraction, and for constructing the service graph of the microservices application, which is to be designed. Additionally, the dependencies among the services are also represented in the service graph of the microservices application. We have evaluated the extracted microservices w.r.t loose coupling and compared them with existing SOA services. It is clear from the results that microservices have low coupling intensity compared to SOA services.

In chapter 5, we propose a new technique which is recasted from the well-known use case points technique to estimate the effort required for the migration of SOA based applications to microservices architecture. We have revised the technical and environmental factors as the existing factors are not compatible with the microservices architecture. We have conducted an online survey to collect the ratings of each of these factors and used them in our effort estimation model. We have demonstrated the new technique through a case study application and calculated the effort required for migration. We have compared the results with effort calculated by considering the default values for factors suggested by Karner.

Additionally, we propose a machine learning model using multiple linear regression with Leave-N-Out policy, where we train the model with N-k applications and test with the remaining k applications. We have taken 7 applications designed with SOA and migrated to microservices, and we calculated the efforts using the regression function. The accuracy of the proposed models is evaluated using different metrics such as MRE, RMSE, PRED, MAE, and SA. The comparison results show that the efforts estimated using the SP-Regression approach are close to the actual efforts, and the error rate is very low compared

to the SP-Proposed approach and SP-Karner's approach.

In chapter 6, to address the most common recurring issues, we propose patterns that help with easy migration. The problems addressed are identified in a literature study on the migration of legacy applications to microservices. The solutions provided are presented with our experience in migrating SOA based applications to microservices.

7.2 Future Scope

- The comparison between the two architectures is presented in terms of coupling, complexity and performance. The other features such as scalability, maintenance, architecture stability, etc., can be considered for comparison.
- In chapter 4, algorithms for extraction of microservices are presented. However, the proposed approach can be enhanced by considering database in the migration approach as each microservice should have individual database.
- The effort estimation approach proposed in chapter 5 uses only a limited dataset of 7 projects. However, the machine learning techniques give better results with more number of projects in the dataset. The proposed approach is applied only on a single case study application, and it can be tested on applications of different domains and large enterprise applications.
- In this thesis, we have proposed a framework for migrating to microservices architecture. However, serverless computing is one of the recent trends and applications can be migrated to serverless computing.

Bibliography

[1] Tasneem Salah, M Jamal Zemerly, Chan Yeob Yeun, Mahmoud Al-Qutayri, and Yousof Al-Hammadi. The evolution of distributed systems towards microservices architecture. In *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 318–325. IEEE, 2016.

[2] Chen-Yuan Fan and Shang-Pin Ma. Migrating monolithic mobile application to microservice architecture: An experiment report. In *2017 IEEE International Conference on AI & Mobile Services (AIMS)*, pages 109–112. IEEE, 2017.

[3] James Lewis and Martin Fowler. Microservices: a definition of this new architectural term. *MartinFowler. com*, 25:14–26, 2014.

[4] Marcelo Amaral, Jorda Polo, David Carrera, Iqbal Mohomed, Merve Unuvar, and Malgorzata Steinder. Performance evaluation of microservices architectures using containers. In *2015 IEEE 14th International Symposium on Network Computing and Applications*, pages 27–34. IEEE, 2015.

[5] Armin Balalaie, Abbas Heydarnoori, and Pooyan Jamshidi. Microservices architecture enables devops: Migration to a cloud-native architecture. *Ieee Software*, 33(3):42–52, 2016.

[6] Sheng Wang, Zhijun Ding, and Changjun Jiang. Elastic scheduling for microservice applications in clouds. *IEEE Transactions on Parallel and Distributed Systems*, 32(1):98–115, 2020.

[7] Shanshan Li, He Zhang, Zijia Jia, Zheng Li, Cheng Zhang, Jiaqi Li, Qiuya Gao, Jidong Ge, and Zhihao Shan. A dataflow-driven approach to identifying microservices from monolithic applications. *Journal of Systems and Software*, 157:110380, 2019.

[8] Dmitry Namiot and Manfred Sneps-Sneppe. On micro-services architecture. *International Journal of Open Information Technologies*, 2(9):24–27, 2014.

[9] Luiz Carvalho, Alessandro Garcia, Wesley KG Assunção, Rodrigo Bonifácio, Leonardo P Tizzei, and Thelma Elita Colanzi. Extraction of configurable and reusable microservices from legacy systems: an exploratory study. In *Proceedings of the 23rd International Systems and Software Product Line Conference-Volume A*, pages 26–31, 2019.

[10] Alexis Henry and Youssef Ridene. Migrating to microservices. In *Microservices*, pages 45–72. Springer, 2020.

[11] Claus Pahl and Pooyan Jamshidi. Microservices: A systematic mapping study. In *CLOSER (1)*, pages 137–146, 2016.

[12] Tomas Cerny, Michael J Donahoo, and Michal Trnka. Contextual understanding of microservice architecture: current and future directions. *ACM SIGAPP Applied Computing Review*, 17(4):29–45, 2018.

[13] Paolo Di Francesco. Architecting microservices. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, pages 224–229. IEEE, 2017.

[14] Alan Megargel, Venky Shankararaman, and David K Walker. Migrating from monoliths to cloud-based microservices: A banking industry example. In *Software Engineering in the Era of Cloud Computing*, pages 85–108. Springer, 2020.

[15] Miika Kalske, Niko Mäkitalo, and Tommi Mikkonen. Challenges when moving from monolith to microservice architecture. In *International Conference on Web Engineering*, pages 32–47. Springer, 2017.

[16] Lucas Krause. *Microservices: patterns and applications*. Lucas Krause, 2015.

[17] Zhongxiang Xiao, Inji Wijegunaratne, and Xinjian Qiang. Reflections on soa and microservices. In *2016 4th International Conference on Enterprise Systems (ES)*, pages 60–67. IEEE, 2016.

[18] Vinay Raj and S Ravichandra. Microservices: A perfect soa based solution for enterprise applications compared to web services. In *2018 3rd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, pages 1531–1536. IEEE, 2018.

[19] Justus Bogner, Stefan Wagner, and Alfred Zimmermann. Automatically measuring the maintainability of service-and microservice-based systems: a literature review. In *Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement*, pages 107–115, 2017.

[20] Naghmeh Niknejad, Iraj Sadegh Amiri, et al. Literature review of service-oriented architecture (soa) adoption researches and the related significant factors. *The impact of service oriented architecture adoption on organizations*, pages 9–41, 2019.

[21] Tomas Cerny, Michael J Donahoo, and Jiri Pechanec. Disambiguation and comparison of soa, microservices and self-contained systems. In *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*, pages 228–235, 2017.

[22] José C Delgado. Structural interoperability as a basis for service adaptability. In *Adaptive web services for modular and reusable software development: Tactics and solutions*, pages 33–59. IGI Global, 2013.

[23] Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. Restful web services vs." big"'web services: making the right architectural decision. In *Proceedings of the 17th international conference on World Wide Web*, pages 805–814, 2008.

[24] Hasan Derhamy, Jens Eliasson, and Jerker Delsing. System of system composition based on decentralized service-oriented architecture. *IEEE Systems Journal*, 13(4):3675–3686, 2019.

[25] Hong-Mei Chen, Rick Kazman, and Opal Perry. From software architecture analysis to service engineering: An empirical study of methodology development for enterprise soa implementation. *IEEE Transactions on Services Computing*, 3(2):145–160, 2010.

[26] Cristina Paniagua, Jens Eliasson, and Jerker Delsing. Efficient device-to-device service invocation using arrowhead orchestration. *IEEE Internet of Things Journal*, 7(1):429–439, 2019.

[27] Johannes Thönes. Microservices. *IEEE software*, 32(1):116–116, 2015.

[28] Kyle Brown and Bobby Woolf. Implementation patterns for microservices architectures. In *Proceedings of the 23rd Conference on Pattern Languages of Programs*, pages 1–35, 2016.

[29] Saleem Durai MA et al. Rescue: Reputation based service for cloud user environment. *International Journal of Engineering*, 27(8):1179–1184, 2014.

[30] Mario Villamizar, Oscar Garcés, Harold Castro, Mauricio Verano, Lorena Salamanca, Rubby Casallas, and Santiago Gil. Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. In *2015 10th Computing Colombian Conference (10CCC)*, pages 583–590. IEEE, 2015.

[31] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. Microservices: yesterday, today, and tomorrow. *Present and ulterior software engineering*, pages 195–216, 2017.

[32] Nicola Dragoni, Ivan Lanese, Stephan Thordal Larsen, Manuel Mazzara, Ruslan Mustafin, and Larisa Safina. Microservices: How to make your application scale. In *International Andrei Ershov Memorial Conference on Perspectives of System Informatics*, pages 95–104. Springer, 2017.

[33] Wilhelm Hasselbring and Guido Steinacker. Microservice architectures for scalability, agility and reliability in e-commerce. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, pages 243–246. IEEE, 2017.

[34] Andy Singleton. The economics of microservices. *IEEE Cloud Computing*, 3(5):16–20, 2016.

[35] Pedro Valderas, Victoria Torres, and Vicente Pelechano. A microservice composition approach based on the choreography of bpmn fragments. *Information and Software Technology*, 127:106370, 2020.

[36] Rory V O'Connor, Peter Elger, and Paul M Clarke. Continuous software engineering—a microservices architecture perspective. *Journal of Software: Evolution and Process*, 29(11):e1866, 2017.

[37] Pooyan Jamshidi, Claus Pahl, Nabor C Mendonça, James Lewis, and Stefan Tilkov. Microservices: The journey so far and challenges ahead. *IEEE Software*, 35(3):24–35, 2018.

[38] Muhammad Aslam Jarwar, Sajjad Ali, Muhammad Golam Kibria, Sunil Kumar, and Ilyoung Chong. Exploiting interoperable microservices in web objects enabled internet of things. In *2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN)*, pages 49–54. IEEE, 2017.

[39] Manuel Mazzara, Antonio Bucchiarone, Nicola Dragoni, and Victor Rivera. Size matters: Microservices research and applications. In *Microservices*, pages 29–42. Springer, 2020.

[40] Norman Wilde, Bilal Gonen, Eman El-Sheikh, and Alfred Zimmermann. Approaches to the evolution of soa systems. In *Emerging Trends in the Evolution of Service-Oriented and Enterprise Architectures*, pages 5–21. Springer, 2016.

[41] Nico Lassing, PerOlof Bengtsson, Hans Van Vliet, and Jan Bosch. Experiences with alma: architecture-level modifiability analysis. *Journal of systems and software*, 61(1):47–57, 2002.

[42] Mohsen AlSharif, Walter P Bond, and Turky Al-Otaiby. Assessing the complexity of software architecture. In *Proceedings of the 42nd annual Southeast regional conference*, pages 98–103, 2004.

[43] Marco Gribaudo, Mauro Iacono, and Daniele Manini. Performance evaluation of massively distributed microservices based applications. In *31st European Conference on Modelling and Simulation, ECMS 2017*, pages 598–604. European Council for Modelling and Simulation, 2017.

[44] Paolo Di Francesco, Ivano Malavolta, and Patricia Lago. Research on architecting microservices: trends, focus, and potential for industrial adoption. In *2017 IEEE International Conference on Software Architecture (ICSA)*, pages 21–30. IEEE, 2017.

[45] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Wenhai Li, and Dan Ding. Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study. *IEEE Transactions on Software Engineering*, 2018.

[46] Bridget Spitznagel and David Garlan. Architecture-based performance analysis. 1998.

[47] Dmitri Nevedrov. Using jmeter to performance test web services. *Published on dev2dev*, pages 1–11, 2006.

[48] Tasneem Salah, M Jamal Zemerly, Chan Yeob Yeun, Mahmoud Al-Qutayri, and Yousof Al-Hammadi. Performance comparison between container-based and vm-based services. In *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, pages 185–190. IEEE, 2017.

[49] Eberhard Wolff. *Microservices: flexible software architecture*. Addison-Wesley Professional, 2016.

[50] Mathawee Tusjunt and Wiwat Vatanawood. Refactoring orchestrated web services into microservices using decomposition pattern. In *2018 IEEE 4th International Conference on Computer and Communications (ICCC)*, pages 609–613. IEEE, 2018.

[51] Davide Taibi, Valentina Lenarduzzi, and Claus Pahl. Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation. *IEEE Cloud Computing*, 4(5):22–32, 2017.

[52] Paolo Di Francesco, Patricia Lago, and Ivano Malavolta. Architecting with microservices: A systematic mapping study. *Journal of Systems and Software*, 150:77–97, 2019.

[53] Valentina Lenarduzzi, Francesco Lomio, Nyyti Saarimäki, and Davide Taibi. Does migrating a monolithic system to microservices decrease the technical debt? *Journal of Systems and Software*, page 110710, 2020.

[54] Shmuel Tyszberowicz, Robert Heinrich, Bo Liu, and Zhiming Liu. Identifying microservices using functional decomposition. In *International Symposium on Dependable Software Engineering: Theories, Tools, and Applications*, pages 50–65. Springer, 2018.

[55] Heleno Cardoso da Silva Filho and Glauco de Figueiredo Carneiro. Strategies reported in the literature to migrate to microservices based architecture. In *16th International Conference on Information Technology-New Generations (ITNG 2019)*, pages 575–580. Springer, 2019.

[56] Andrei Furda, Colin Fidge, Olaf Zimmermann, Wayne Kelly, and Alistair Barros. Migrating enterprise legacy source code to microservices: on multitenancy, statefulness, and data consistency. *IEEE Software*, 35(3):63–72, 2017.

[57] Manabu Kamimura, Keisuke Yano, Tomomi Hatano, and Akihiko Matsuo. Extracting candidates of microservices from monolithic application code. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, pages 571–580. IEEE, 2018.

[58] Michael Gysel, Lukas Kölbener, Wolfgang Giersche, and Olaf Zimmermann. Service cutter: A systematic approach to service decomposition. In *European Conference on Service-Oriented and Cloud Computing*, pages 185–200. Springer, 2016.

[59] Genc Mazlami, Jürgen Cito, and Philipp Leitner. Extraction of microservices from monolithic software architectures. In *2017 IEEE International Conference on Web Services (ICWS)*, pages 524–531. IEEE, 2017.

[60] Luciano Baresi, Martin Garriga, and Alan De Renzis. Microservices identification through interface analysis. In *European Conference on Service-Oriented and Cloud Computing*, pages 19–33. Springer, 2017.

[61] Wuxia Jin, Ting Liu, Qinghua Zheng, Di Cui, and Yuanfang Cai. Functionality-oriented microservice extraction based on execution trace clustering. In *2018 IEEE International Conference on Web Services (ICWS)*, pages 211–218. IEEE, 2018.

[62] Francisco Ponce, Gastón Márquez, and Hernán Astudillo. Migrating from monolithic architecture to microservices: A rapid review. In *2019 38th International Conference of the Chilean Computer Science Society (SCCC)*, pages 1–7. IEEE, 2019.

[63] Venkata Swamy Martha and Maurin Lenglart. Webservices engineering. In *Webservices*, pages 173–196. Springer, 2019.

[64] Parag C Pendharkar, Girish H Subramanian, and James A Rodger. A probabilistic model for predicting software development effort. *IEEE Transactions on software engineering*, 31(7):615–624, 2005.

[65] Girish H Subramanian, Parag C Pendharkar, and Mary Wallace. An empirical study of the effect of complexity, platform, and program type on software development effort of business applications. *Empirical Software Engineering*, 11(4):541–553, 2006.

[66] SAMSON WANJALA MUNIALO and SAMSON WANJALA. *A SIZE METRIC-BASED EFFORT ESTIMATION METHOD FOR SERVICE ORIENTED ARCHITECTURE SYSTEMS*. PhD thesis, MMUST, 2020.

[67] Shashank Mouli Satapathy, Barada Prasanna Acharya, and Santanu Kumar Rath. Early stage software effort estimation using random forest technique based on optimized class point approach. *INFOCOMP Journal of Computer Science*, 13(2):22–33, 2014.

[68] Marcio Rodrigo Braz and Silvia Regina Vergilio. Software effort estimation based on use cases. In *30th Annual International Computer Software and Applications Conference (COMPSAC'06)*, volume 1, pages 221–228. IEEE, 2006.

[69] Sergey Diev. Use cases modeling and software estimation: applying use case points. *ACM SIGSOFT Software Engineering Notes*, 31(6):1–4, 2006.

[70] Parastoo Mohagheghi, Bente Anda, and Reidar Conradi. Effort estimation of use cases for incremental large-scale software development. In *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.*, pages 303–311. IEEe, 2005.

[71] Gerardo Canfora, Anna Rita Fasolino, Gianni Frattolillo, and Porfirio Tramontana. A wrapping approach for migrating legacy system interactive functionalities to service oriented architectures. *Journal of Systems and Software*, 81(4):463–480, 2008.

[72] Barry Boehm and Richard Turner. Management challenges to implementing agile processes in traditional development organizations. *IEEE software*, 22(5):30–39, 2005.

[73] M Tajamolian and M Ghasemzadeh. A versioning approach to vm live migration. *International Journal of Engineering*, 31(11):1838–1845, 2018.

[74] Janusz Zalewski. Real-time software architectures and design patterns: Fundamental concepts and their consequences. *Annual Reviews in Control*, 25:133–146, 2001.

[75] Farah Lakhani and Michael J Pont. Using design patterns to support migration between different system architectures. In *2010 5th International Conference on System of Systems Engineering*, pages 1–6. IEEE, 2010.

[76] Jacopo Soldani, Damian Andrew Tamburri, and Willem-Jan Van Den Heuvel. The pains and gains of microservices: A systematic grey literature review. *Journal of Systems and Software*, 146:215–232, 2018.

[77] Sara Hassan and Rami Bahsoon. Microservices and their design trade-offs: A self-adaptive roadmap. In *2016 IEEE International Conference on Services Computing (SCC)*, pages 813–818. IEEE, 2016.

[78] Olaf Zimmermann. Microservices tenets. *Computer Science-Research and Development*, 32(3):301–310, 2017.

[79] Hulya Vural, Murat Koyuncu, and Sinem Guney. A systematic literature review on microservices. In *International Conference on Computational Science and Its Applications*, pages 203–217. Springer, 2017.

[80] Justus Bogner, Stefan Wagner, and Alfred Zimmermann. Using architectural modifiability tactics to examine evolution qualities of service-and microservice-based systems. *SICS Software-Intensive Cyber-Physical Systems*, 34(2):141–149, 2019.

[81] Claus Pahl, Pooyan Jamshidi, and Olaf Zimmermann. Architectural principles for cloud software. *ACM Transactions on Internet Technology (TOIT)*, 18(2):1–23, 2018.

[82] Aliaksei Yanchuk, Alexander Ivanyukovich, and Maurizio Marchese. Towards a mathematical foundation for service-oriented applications design. *Journal of Software*, 1(1):32–39, 2006.

[83] Pushparani Bhallamudi, Scott Tilley, and Arunesh Sinha. Migrating a web-based application to a service-based system-an experience report. In *2009 11th IEEE International Symposium on Web Systems Evolution*, pages 71–74. IEEE, 2009.

[84] Vinay Raj and Ravichandra Sadam. A service graph based extraction of microservices from monolith service of soa. *Software: Practice and experience*, 2021.

[85] Jianjun Zhao. On assessing the complexity of software architectures. In *Proceedings of the third international workshop on Software architecture*, pages 163–166, 1998.

[86] Zhang Qingqing and Li Xinke. Complexity metrics for service-oriented systems. In *2009 second international symposium on knowledge acquisition and modeling*, volume 3, pages 375–378. IEEE, 2009.

[87] Vinay Raj and Ravichandra Sadam. Performance and complexity comparison of service oriented architecture and microservices architecture. *International Journal of Communication Networks and Distributed Systems*, 26(3), 2021.

[88] François Jammes and Harm Smit. Service-oriented paradigms in industrial automation. *IEEE Transactions on industrial informatics*, 1(1):62–70, 2005.

[89] Hironori Washizaki, Hirokazu Yamamoto, and Yoshiaki Fukazawa. A metrics suite for measuring reusability of software components. In *Proceedings. 5th International Workshop on Enterprise Networking and Computing in Healthcare Industry (IEEE Cat. No. 03EX717)*, pages 211–223. IEEE, 2004.

[90] Renuka Sindhgatta, Bikram Sengupta, and Karthikeyan Ponnalagu. Measuring the quality of service oriented design. In *Service-Oriented Computing*, pages 485–499. Springer, 2009.

[91] Cesare Pautasso and Erik Wilde. Why is the web loosely coupled? a multi-faceted metric for service design. In *Proceedings of the 18th international conference on World wide web*, pages 911–920, 2009.

[92] Wilhelm Hasselbring. Microservices for scalability: Keynote talk abstract. In *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering*, pages 133–134, 2016.

[93] Xabier Larrucea, Izaskun Santamaria, Ricardo Colomo-Palacios, and Christof Ebert. Microservices. *IEEE Software*, 35(3):96–100, 2018.

[94] Esraa A Farrag, Ramadan Moawad, et al. An approach for effort estimation of service oriented architecture (soa) projects. 2015.

[95] Gustav Karner. Resource estimation for objectory projects. *Objective Systems SF AB*, 17:1–9, 1993.

[96] Tim Menzies, Ye Yang, George Mathew, Barry Boehm, and Jairus Hihn. Negative results for software effort estimation. *Empirical Software Engineering*, 22(5):2658–2683, 2017.

[97] Siba Mishra and Chiranjeev Kumar. Effort estimation for service-oriented computing environments. *Computing and Informatics*, 37(3):553–580, 2018.

[98] Jianfeng Wen, Shixian Li, Zhiyong Lin, Yong Hu, and Changqin Huang. Systematic literature review of machine learning based software development effort estimation models. *Information and Software Technology*, 54(1):41–59, 2012.

[99] Douglas C Montgomery, Elizabeth A Peck, and G Geoffrey Vining. *Introduction to linear regression analysis*. John Wiley & Sons, 2021.

[100] Federica Sarro, Alessio Petrozziello, and Mark Harman. Multi-objective software effort estimation. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 619–630. IEEE, 2016.

[101] Martin Shepperd and Chris Schofield. Estimating software project effort using analogies. *IEEE Transactions on software engineering*, 23(11):736–743, 1997.

[102] Justus Bogner, Alfred Zimmermann, and Stefan Wagner. Analyzing the relevance of soa patterns for microservice-based systems. *Zeus*, 9:9–16, 2018.

[103] Mohammad Azzeh, Ali Bou Nassif, and Imtinan Basem Attili. Predicting software effort from use case points: A systematic review. *Science of Computer Programming*, 204:102596, 2021.

[104] Ali Bou Nassif, Luiz Fernando Capretz, and Danny Ho. Enhancing use case points estimation method using soft computing techniques. *arXiv preprint arXiv:1612.01078*, 2016.

[105] Armin Balalaie, Abbas Heydarnoori, and Pooyan Jamshidi. Migrating to cloud-native architectures using microservices: an experience report. In *European Conference on Service-Oriented and Cloud Computing*, pages 201–215. Springer, 2015.

[106] Armin Balalaie, Abbas Heydarnoori, Pooyan Jamshidi, Damian A Tamburri, and Theo Lynn. Microservices migration patterns. *Software: Practice and Experience*, 48(11):2019–2042, 2018.

[107] Frank Leymann, Christoph Fehling, Sebastian Wagner, and Johannes Wettinger. Native cloud applications: Why virtual machines, images and containers miss. In *Proceedings of the 6th International Conference on Cloud Computing and*, pages 7–15. SciTePress.

[108] Holger Knoche and Wilhelm Hasselbring. Using microservices for legacy software modernization. *IEEE Software*, 35(3):44–49, 2018.

List of Publications

Journal Papers

1. **Vinay Raj** and Sadam Ravichandra, "Performance and complexity comparison of SOA and microservices architectures", *International Journal of Communication Networks and Distributed Systems, Inderscience*. (*Published*).
2. **Vinay Raj** and Sadam Ravichandra, "Patterns for migration of SOA based applications to microservices architecture", *Journal of Web Engineering, River Publishers* (*Published*).
3. **Vinay Raj** and Sadam Ravichandra, "Evaluation of SOA based web services and microservices architecture using complexity metrics", *SN Computer Science, Springer* (*Published*).
4. **Vinay Raj** and Sadam Ravichandra, "A service graph based extraction of microservices from monolith services of SOA", *Journal of Software: Practice and Experience, Wiley* (*Revisions Submitted*).
5. **Vinay Raj** and Sadam Ravichandra, "A novel effort estimation approach for migration of SOA applications to microservices", *Journal of Information Systems and Telecommunication, Advanced Information Systems (AIS) Research Group* (*Revisions Submitted*).
6. **Vinay Raj** and Sadam Ravichandra, ''A Framework for Migration of SOA based Applications to Microservices Architecture", *Journal of Computer Science and Technology, Springer* (*Under Review*).

Conference Papers

1. **Vinay Raj** and Sadam Ravichandra, "Microservices: A perfect SOA based solution for Enterprise Applications compared to Web Services", *3rd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)* May 18-19, 2018, held at Sri Venkateshwara College of Engineering, Bengaluru, India.