SN Computer Science (2023) 4:577
https://doi.org/10.1007/s42979-023-01971-2

ORIGINAL RESEARCH

Assessing the Impact of Migration from SOA to Microservices Architecture

Vinay Raj[1] · **Hanumanthu Bhukya**[2]

Received: 4 March 2023 / Accepted: 27 May 2023

Abstract

Microservices has become a buzzword in IT and large enterprise firms such as Netflix, Twitter, Spotify and others have started to design their applications by adopting this new architectural style. A few organizations have started migrating their traditional monolithic and SOA-based applications to microservices in order to benefit from the features of new style. Software architects, on the other hand, are in chaos whether to adopt this new style or not as they are unaware of the pros and cons of microservices architecture. Also, the impact of the migration on SOA-based applications in terms of performance and complexity is unknown, leading to dilemma on the migration process. In this paper, a study of the migration impact on the existing SOA based applications to microservices is presented. For this study, change propagation probability and architectural stability metrics are used to examine the effect of migrating a SOA-based application to microservices architecture. The proposed approach is illustrated on a case study application designed using SOA and then migrated to microservices. It is observed from the results that though the impact of migration is high, migrating to microservices has significant benefits and it is best suitable for large enterprise applications.

Keywords Service oriented architecture · Microservices · Migration · Change propagation · Stability

Introduction

The monolithic approach to application design was the beginning of the rapid evolution of distributed systems. The application components of a monolithic code are tightly connected and have a big codebase. The size and complexity of the application are both constrained by monolithic architecture. SOA has developed as a result of the growing complexity of enterprise systems, business objectives, and the necessity to create distributed applications [8]. Service oriented architecture (SOA) has been widely used in designing large enterprise applications in the last two decades. It was developed mainly to address the deployment and scaling problems with monolithic systems. All of the system's components are designed as services in the SOA design approach to developing applications [18]. A service is a reusable piece of software code that performs a range of business operations, which can be basic or complex depending on the needs of the company. SOA is a method of integrating numerous software components that use the Enterprise Service Bus (ESB) as a communication channel [32]. The backbone of SOA is the ESB, which aids in the provision of middleware system functions. The ESB operates as a mediator between the service requestor and the provider, providing a high-performance and scalable platform. SOA gained popularity as web services evolved, which is a common application of SOA ideas [19]. Web services are internet-based services that may be established, accessed, and found through the use of communication protocols such as XML-based SOAP and WSDL. HTTP and REST protocols are used by web services to transport messages over the internet. The three main components of the web services architecture are the

This article is part of the topical collection "Research Trends in Communication and Network Technologies" guest edited by Anshul Verma, Pradeepika Verma and Kiran Kumar Pattanaik.

✉ Vinay Raj
vinayraj@nitt.edu

Hanumanthu Bhukya
bhcsekits@gmail.com

[1] Department of Computer Applications, National Institute of Technology Tiruchirappalli, Tiruchirappalli, Tamil Nadu 620015, India

[2] Department of Computer Science and Engineering (Networks), Kakatiya Institute of Technology and Science, Warangal, Telangana 506015, India

Published online: 31 July 2023

service provider, service consumer, and service registry. A single web service may be utilised by several clients at the same time and is simple to implement.

Despite its popularity for application development, SOA has a few design and deployment issues [23]. Due of its dependency on other services and tight connection to the ESB, upgrading a single service necessitates the re-deployment of numerous components. Delivering many services leads to a monolithic deployment strategy, which has an influence on the company [9]. In addition, as the complexity of continuously changing business requirements rises, the application becomes complex and harder to manage as the size of SOA services becomes monolithic. Since SOA follows centralised control, scaling such monolithic applications is a bottleneck [30]. Overloaded services can be expanded horizontally by creating numerous copies of the same service, but the hardware cost rises. Moreover, web services exchange messages using complicated and heavyweight protocols such as SOAP.

Microservices emerged as a new architectural solution that employs cloud-based containers for deployment to address these shortcomings in traditional systems [15]. It is a design pattern in which each service is a tiny, loosely linked, scalable, and reusable service that can be built and deployed separately [28]. Each service should only do one task and have its own database and deployment infrastructure. Microservices exchange data using communication protocols such as HTTP/REST and JSON. Microservices, unlike SOA, may be implemented independently since there is no centralised governance and no reliance on middleware technology. Scaling on-demand microservices is simple when using cloud-based containers [14]. Microservices design fits well with the DevOps philosophy since each work is split down into little components and the SDLC is completed separately [27]. DevOps and agile approaches necessitate rapid application design and deployment to production.

To clearly understand the concepts of monolithic, SOA, and microservices architectures, a diagram is presented in Fig. 1. A monolithic system contains a single enormous unit of code, and when it comes to SOA, the massive codebase is partitioned into coarse-grained services. The services in SOA are further partitioned to generate fine-grained services in microservices.

Due of the multiple benefits of microservices architecture, software architects are starting to transition their old systems to this design [26]. Numerous companies, including Netflix, Amazon, and Twitter, have begun to use this architecture in new applications [24]. Because microservices have just recently evolved, there is a great deal of interest in both business and academics to investigate the tools, technologies, and programming languages utilised in this design. However, some software architects are uncertain about whether or not to adopt this new paradigm, as they are unfamiliar

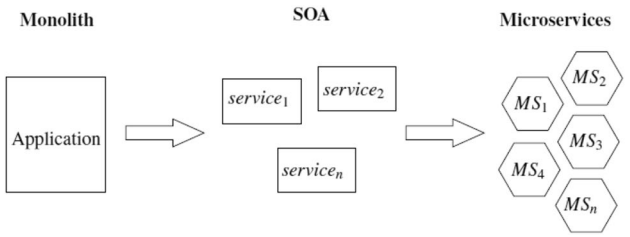

Fig. 1 Understanding of monolithic, SOA and microservices architectures

with the benefits and drawbacks of adopting microservices [26]. However, academic research into microservices is still in its early stages, with relatively little work comparing and evaluating microservices against SOA.

The remaining part of the paper is organized as follows. The necessary background information is discussed in "Background", and the preliminaries are presented in "Preliminaries". The details of the chosen application, along with the service graphs, are discussed in "Experimental study". Results and discussion are presented in "Results and discussion", and "Conclusion" concludes the paper.

Background

There are some technical variations between SOA and microservices application design and implementation methodologies. Microservices design is based on sharing as little as possible, and share as much as possible is the idea of SOA [29]. For communication between services, SOA relies on heavyweight middleware and enterprise service bus, but microservices depends solely on lightweight protocols. The REST and HTTP protocols are used for the exchange of messages and JSON as a data interchange format in microservices [11]. SOA uses WSDL and SOAP protocols for the exchange of messages. Microservices use the concepts of smart endpoints and dumb pipes, as well as the choreography over orchestration technique [5]. There is also a necessity to evaluate and study both the architectures with respect to performance, scalability, and deployment, according to various research [6, 12, 13]. The security of the applications is also a crucial factor to consider while comparing [25].

In the literature, the comparison of SOA with microservices has been addressed from several viewpoints. The authors of [7] presented the applicability of patterns proposed for SOA to microservices architecture. Microservices patterns are not new, and most patterns that are applicable to SOA are also applicable to microservices. This cannot, however, be called a comparison of the two architectures. In [8, 9], the theoretical contrasts between the two approaches in terms of characteristics are discussed. The distinctions are

discussed from both research and industry viewpoints. The authors in [16, 17] have proposed techniques for estimating the effort required for migration from SOA to microservices architecture. Their studies signifies that migration to microservices is inevitable. A study on the best practices for migrating legacy applications to microservices has been proposed in [22]. In [23], the author presents a similar comparison of distributed systems like client and server, SOA, and microservices. There has been a greater focus on comparing SOA with microservices in terms of communication protocols, message frameworks, and service discovery.

To the best of our knowledge, no analysis of the impact of upgrading an SOA-based application to a microservices design has been done. Hence, in this work, we use change propagation probability (CPP) and architectural stability metrics to assess the impact on the application while migrating it to the new style.

Preliminaries

Service Graph

We define the service graph as a formal model that reflects any service-based application. We design a service graph by examining the APIs to collect the inputs and outputs. Because services are the core component of both SOA and microservices designs, we utilize this service graph to compare the two architectural styles. The service graph (SG) is a formal graph that is used to visualise the interactions and relationships between the services in an application. Figure 2 depicts the generalised form of a service-based application as a service graph.

Definition 1 Let $G(V, E)$ be an service graph with **n** nodes, where each node represents a set of services in the application, and the edges between the nodes indicate the interactions or dependencies that each service has with other services in the application.

Let $V = \{s_1,s_2,s_3,...\}$ represent the nodes of the service graph where $s_1,s_2,s_3,...$ are services and $E = \{(s_1,s_2), (s_1,s_3), (s_2,s_4),....\}$ represent the edges between the nodes which indicates the dependency between the services. As stated in Eq. (1), a service is a collection of coordinating and interacting processes.

$$S_i = \langle {P_1}^i, {P_2}^i, {P_3}^i, \ldots, {P_n}^i, \Lambda \rangle \tag{1}$$

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

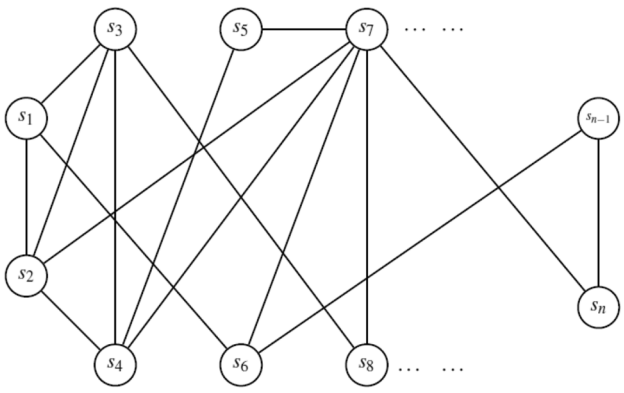

Fig. 2 Service graph representation

Change Propagation (CP)

The change in one component of the software architecture impacts the other components in the system. Therefore, the other components should also be updated and redeployed. However, to evaluate the impact of changing one component on other components of the architecture is given by change propagation probability. Let S be the application designed using architecture A which is to be migrated to form a new application S' of architecture B. We use this metric to find the probability, whether a change in one service s_1 of S requires a change in service s_2 while migrating the application from S to S'.

Definition 2 The conditional probability for the change propagation from service s_1 to s_2 in S is defined as [1]:

$$CP(s_1 : s_2) = P((s_2 \neq s_2') \mid (s_1 \neq s_1') \wedge (S = S')) \tag{2}$$

where S' is the application obtained from S by migrating s_1 to s_1' and s_2 to s_2'. Here, s_1 and s_2 are the services of application S of architecture A and s_1' and s_2' are the services of the migrated application S' designed with architecture B.

The service based architectures can be seen as a collection of services $s_i, i = 1, 2, \ldots, N$. Every service s_i has a set of processes V_i which provide the functionality for service s_i. Using Bernoulli random variable, we find the usage coefficient value π_v^{ij} for every process $p \in V_i$ and every other service s_j.

$$\pi_v^{ij} = \begin{cases} 1 & \text{the process } p \text{ of } s_i \text{ is required by } s_j \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

To estimate the change propagation probability CP($s_1 : s_2$), for every pair of services s_i and s_j, $i \neq j$, we use the values of random variable π_v^{ij}.

$$\mathrm{CP}(s_1 : s_2) = \frac{1}{|V_i|} \sum_{p \in V_i} \pi_v^{ij} \tag{4}$$

The CP is a matrix that contains the relation between all the services of the application. As matrices cannot be compared, we represent the values of the matrix as a scalar component which represents the architecture's potential to wrap its services from other's changes. We denote the scalar component as change propagation coefficient (CPC), and it is given as:

$$\mathrm{CPC} = \frac{\sum_j \sum_{j \neq i} \mathrm{CP}(s_i : s_j)}{N^2 - N} \tag{5}$$

Here N is the number of services in the architecture, and the CPC indicates whether changes in one service of the architecture propagates to other services or not. A low CPC value of the architecture indicates a good sign in the design of the application.

Architecture Stability

Stable software architecture has been a challenge for software architects due to the changes in environmental factors [31]. The major concern is the factors that influence architecture decisions, and stability of the architecture [10]. Stability is the capability of the application components to stay unchanged and remain intact while adding new changes or requirements [2]. Here, we consider the scenario of migrating service oriented architecture based applications to microservices, and as both the styles are service-based, we measure the stability of the application using the metrics. To measure the stability of the architectures, two metrics are defined based on the metrics proposed in [3].

Core Design Instability (CDI)

It is used to evaluate the change performed on the services of the architectural core when it is migrated. IT is defined as follows:

$$\mathrm{CDI} = \frac{n + d}{m}, \tag{6}$$

where

- n is the number of new services added after the migration of an application S from architecture A to S' in architecture B.
- d is the number of services deleted from the application S of A after migrating it to S' of B.
- m is the number of services of the application S of A before migrating it to the B.

Core Calls Instability (CCI)

It is used to evaluate the changes in the interactions between services and it is computed as:

$$\mathrm{CCI} = \frac{c + p}{t}, \tag{7}$$

where

- c is the total number of new calls between services belonging to the application S' of B and not present in the application S of A.
- p is the total number of calls between services of the application S and not present in the application S' after migration to B.
- t is the total number of calls between the services of the application S of architecture A.

Experimental Study

We use the vehicle management system (VMS) [4], a standard web-based tool for selecting, customizing, and purchasing automobiles and parts via a front-end web page. The application is used to assist clients in selecting, customizing, comparing vehicles, finding dealers, and requesting a quote. The database stores all of the information about the automobiles, their parts, and their costs, and the user interface assists clients with the details. Customers can use the inventory data to find the vehicle they want and the dealer that sells it. Customers may also select the part and product type for their car from the interface.

SOA Based Application

The SOA implementation of the VMS application has 8 services. Table 1 lists the details of the SOA services, and Fig. 3 shows the service graph representation, which is indicated as SG_SOA. We used TIBCO business works to develop the SOA-based application and TIBCO administrator to deploy it. An Oracle database is selected for data storage, and TIBCO BW database palettes assist in connecting to the database. The Representational State Transfer (REST) protocol is used to communicate between the services over HTTP. Each service is deployed on a single server as a stand-alone archive.

Microservices Based Application

Microservices-based application is designed using the extraction approach [21] proposed for extraction of microservices application from a SOA-based application to construct a service graph that helps in the identification

Table 1 Services of both the applications

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 and pricing service	Incentives service	ms_5
		Pricing service	ms_6
S_6	Dealer and 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}

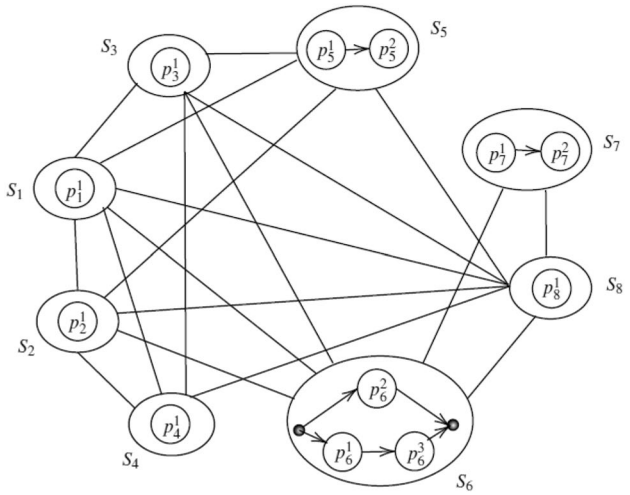

Fig. 3 SG_SOA: Service graph representation of SOA based application

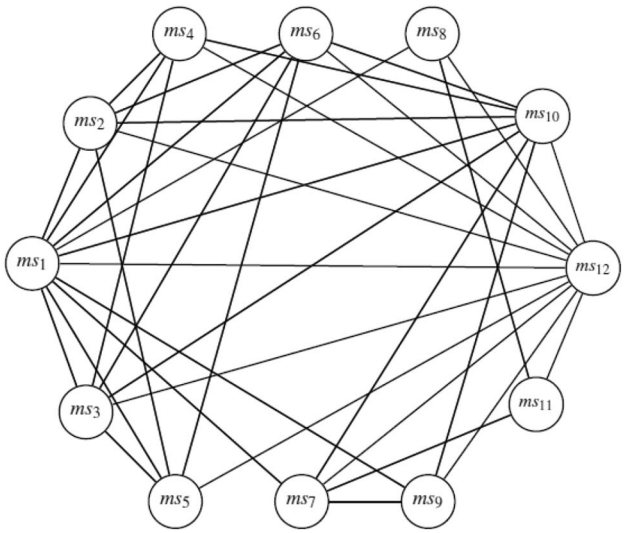

Fig. 4 SG_MSA: Service graph representation of microservices based web application

of candidate microservices. The VMS application is built with the spring boot framework and REST/JSON formats for communication between services in the network, taking into consideration the microservices. In a service register, the Eureka service is used to store all of the services. MYSQL database is used to store the data, while spring boot connector uses JPA connector to retrieve it. Each microservice is deployed in the cloud using Docker containers. The application's docker image is created, deployed to Docker Hub, and containers are built from docker images. Table 1 contains the information of the produced microservices, whereas Fig. 4 depicts the service graph (SG_MSA).

Results and Discussion

Let SOA be represented as A and microservices architecture as B. For the evaluation purpose, consider S and S' as the applications designed with both SOA and microservices styles respectively. We define CP_{SOA} and CP_{MSA} as the change propagation matrices of both SOA and microservices architectures. Similarly, we define CPC_{SOA} and CPC_{MSA} as change propagation coefficients for both SOA and microservices architectures.

Change Propagation Probability

From the service graph of the SOA application, the values of CP for each pair of services is calculated using the Equations (3) and (4). The values are presented in form of a matrix as given below.

$$CP_{SOA} = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \\ s_8 \end{array} \begin{array}{c} \begin{array}{cccccccc} s_1 & s_2 & s_3 & s_4 & s_5 & s_6 & s_7 & s_8 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0.5 & 0.5 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0.33 & 0.33 & 0 & 0 & 0 & 0.66 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

The generated CP matrix is converted to a scalar component using the change propagation coefficient (CPC). The CPC value indicates the impact of change in one particular service has on other services. As we have eight services in the SOA application, the N value is eight.

$$\text{CPC}_{\text{SOA}} = \frac{35.33}{8^2 - 8} = 0.63.$$

Similarly, from the service graph of microservices application given in Fig. 4, the CP values are calculated and presented as matrix given below.

$$CP_{MSA} = \begin{array}{c} \\ ms_1 \\ ms_2 \\ ms_3 \\ ms_4 \\ ms_5 \\ ms_6 \\ ms_7 \\ ms_8 \\ ms_9 \\ ms_{10} \\ ms_{11} \\ ms_{12} \end{array} \begin{array}{c} \begin{array}{cccccccccccc} ms_1 & ms_2 & ms_3 & ms_4 & ms_5 & ms_6 & ms_7 & ms_8 & ms_9 & ms_{10} & ms_{11} & ms_{12} \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

The change propagation coefficient of the microservices application is also calculated using the CP matrix.

$$\text{CPC}_{\text{MSA}} = \frac{72}{12^2 - 12} = 0.54.$$

From the above CPC values of both the styles, microservices based application exhibits low CPC value compared to applications built using SOA concepts. As the CPC value of microservices is low, it indicates a good sign of design, and it is best suitable for large enterprise applications compared to SOA based applications. This result of the chosen case study helps the software architects to assess the impact of migration. Though the other parameters such as effort required for migration and complexity of the applications are high for microservices [20], these metrics show that the use of microservices in design makes the application stable and maintainable.

Stability Evaluation

By considering the details of the services and service calls from the service graphs of the chosen application, the values of CDI and CCI are evaluated. From the services information in Table 1, the value of CDI is calculated as:

$$\text{CDI} = \frac{4 + 0}{8} = 0.5$$

Similarly, we calculate the value of CCI,

$$\text{CCI} = \frac{32 + 0}{38} = 0.84$$

The metric values CDI and CCI indicate a measure of how much the services of the application S of architecture A have changed after migrating to application S' of architecture B. The threshold value chosen for both CDI and CCI is 0.15. If the metric values are less than 0.15, then the architecture is said to be stable and otherwise unstable. By observing the calculated values, the metric values are greater than 0.15, and hence, it indicates that the services in SOA application have undergone a major change to form the services in microservices application. Also, the impact of migration is very high.

Conclusion

With the evolution of microservices architecture, there is a paradigm shift in designing software applications. With the advancement of new technologies and tools, every day the IT world is witnessing many improvements and benefits of using new things. Similarly, many IT giants are migrating their SOA applications to microservices architecture. However, the impact of migration is not assessed, and some architects are uncertain whether or not to migrate, as both SOA and microservices have their own set of benefits and drawbacks. Hence, in this work, we presented an assessment of the impact of migrating SOA based applications to microservices architecture. We observe that, to migrate an SOA application, the system needs to change and be updated drastically as the design and deployment environments are quite different for both styles. The effort required for complete migration and comparing both the architectures with QoS attributes such as performance, maintenance, scalability, etc., can be considered as future work.

Funding The authors did not receive support from any organization for the submitted work.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Abdelmoez W, Shereshevsky M, Gunnalan R, Ammar HH, Yu Bo, Bogazzi S, Korkmaz M, Mili A. Quantifying software architectures: an analysis of change propagation probabilities. In: The 3rd ACS/IEEE international conference on computer systems and applications. IEEE; 2005. p. 124.
2. Almousa H, Alenezi M. Measuring software architecture stability evolution in object-oriented open source systems. J Eng Appl Sci. 2017;12(2):353–62.
3. Aversano L, Molfetta M, Tortorella M. Evaluating architecture stability of software projects. In: 2013 20th working conference on reverse engineering (WCRE). IEEE; 2013. p. 417–24.
4. Bhallamudi P, Tilley S, Sinha A. Migrating a web-based application to a service-based system-an experience report. In: 2009 11th IEEE international symposium on web systems evolution. IEEE; 2009. p. 71–4.
5. Bogner J, Wagner S, Zimmermann A. 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; 2017. p. 107–15.
6. Bogner J, Wagner S, Zimmermann A. Using architectural modifiability tactics to examine evolution qualities of service-and microservice-based systems. SICS Softw Intensive Cyber Phys Syst. 2019;34(2):141–9.
7. Bogner J, Zimmermann A, Wagner S. Analyzing the relevance of SOA patterns for microservice-based systems. Zeus. 2018;9:9–16.
8. Cerny T, Donahoo MJ, Pechanec J. Disambiguation and comparison of SOA, microservices and self-contained systems. In: Proceedings of the international conference on research in adaptive and convergent systems; 2017. p. 228–35.
9. Cerny T, Donahoo MJ, Trnka M. Contextual understanding of microservice architecture: current and future directions. ACM SIGAPP Appl Comput Rev. 2018;17(4):29–45.
10. Figueiredo E, Cacho N, Sant'Anna C, Monteiro M, Kulesza U, Garcia A, Soares S, Ferrari F, Khan S, Filho FC, et al. Evolving software product lines with aspects. In: 2008 ACM/IEEE 30th international conference on software engineering. IEEE; 2008. p. 261–70.
11. Jamshidi P, Pahl C, Mendonça NC, Lewis J, Tilkov S. Microservices: the journey so far and challenges ahead. IEEE Softw. 2018;35(3):24–35.
12. Pahl C, Jamshidi P, Zimmermann O. Architectural principles for cloud software. ACM Trans Internet Technol (TOIT). 2018;18(2):1–23.
13. Rademacher F, Sachweh S, Zündorf A. Analysis of service-oriented modeling approaches for viewpoint-specific model-driven development of microservice architecture. 2018. arXiv preprint arXiv:1804.09946.
14. Raghavendran CV, Patil A, Satish GN, Shanmukhi M, Madhuravani B. Challenges and opportunities in extending cloud with fog computing. Int J Eng Technol. 2018;7(439):142–6.
15. Raj V, Ravichandra S. 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 and communication technology (RTEICT). IEEE; 2018. p. 1531–536.
16. Raj V, Ravichandra S. Enhanced service point approach for microservices based applications using machine learning techniques. In: Advanced informatics for computing research: 5th international conference, ICAICR 2021, Gurugram, India, December 18–19, 2021, Revised Selected Papers. Springer; 2022. p. 78–90.
17. Raj V, Ravichandra S. A novel effort estimation approach for migration of SOA applications to microservices. J Inf Syst Telecommun (JIST). 2022;2(38):80.
18. Raj V, Sadam R. Evaluation of SOA-based web services and microservices architecture using complexity metrics. SN Comput Sci. 2021;2(5):1–10.
19. Raj V, Sadam R. Patterns for migration of SOA based applications to microservices architecture. J Web Eng. 2021;25:1229–46.
20. Raj V, Sadam R. Performance and complexity comparison of service oriented architecture and microservices architecture. Int J Commun Netw Distrib Syst. 2021;27(1):100–17.
21. Raj V, Sadam R. A service graph based extraction of microservices from monolith services of SOA. Softw Pract Exp. 2021;51(3):489–502.
22. Raj V, Reddy KS. Best practices and strategy for the migration of service-oriented architecture-based applications to microservices architecture. In: Proceedings of second international conference on advances in computer engineering and communication systems: ICACECS 2021. Springer; 2022. p. 439–49.
23. Salah T, Zemerly MJ, Yeun CY, Al-Qutayri M, Al-Hammadi Y. The evolution of distributed systems towards microservices architecture. In: 2016 11th international conference for internet technology and secured transactions (ICITST). IEEE; 2016. p. 318–25.
24. Soldani J, Tamburri DA, Van Den Heuvel W-J. The pains and gains of microservices: a systematic grey literature review. J Syst Softw. 2018;146:215–32.
25. Sridevi M, Sunitha KVN. A hybrid framework for secure web applications. In: International conference on intelligent computing and communication technologies. Springer; 2019. p. 140–51.
26. Taibi D, Lenarduzzi V, Pahl C. Processes, motivations, and issues for migrating to microservices architectures: an empirical investigation. IEEE Cloud Comput. 2017;4(5):22–32.
27. Taibi D, Lenarduzzi V, Pahl C, Janes A. Microservices in agile software development: a workshop-based study into issues, advantages, and disadvantages. In: Proceedings of the XP2017 scientific workshops; 2017. p. 1–5.
28. Thönes J. Microservices. IEEE Softw. 2015;32(1):116–116.
29. Wilde N, Gonen B, El-Sheikh E, Zimmermann A. Approaches to the evolution of SOA systems. In: Emerging trends in the evolution of service-oriented and enterprise architectures. Springer; 2016. p. 5–21.
30. Xiao Z, Wijegunaratne I, Qiang X. Reflections on SOA and microservices. In: 2016 4th international conference on enterprise systems (ES). IEEE; 2016. p. 60–7.
31. Yau SS, Collofello JS. Design stability measures for software maintenance. IEEE Trans Softw Eng. 1985;9:849–56.
32. Yin J, Chen H, Deng S, Zhaohui W, Calton P. A dependable ESB framework for service integration. IEEE Internet Comput. 2009;13(2):26–34.