A Prescriptive Model for Migration to Microservices Based on SDLC Artifacts

Deepali Bajaj[1,*], Urmil Bharti[1], Anita Goel[2] and S. C. Gupta[3]

[1]*Department of Computer Science, Shaheed Rajguru College of Applied Sciences for Women, University of Delhi, Delhi, India*
[2]*Department of Computer Science, Dyal Singh College, University of Delhi, Delhi, India*
[3]*Department of Computer Science, Indian Institute of Technology, Delhi, India*
E-mail: deepali.bajaj@rajguru.du.ac.in; urmil.bharti@rajguru.du.ac.in; goel.anita@gmail.com; drsc.gupta@gmail.com
[*]*Corresponding Author*

Received 21 March 2020; Accepted 01 March 2021;
Publication 02 June 2021

Abstract

Microservices architectural style is gaining popularity in industry and is being widely adopted by large corporations like Amazon, Netflix, Spotify, eBay, and many more. Several other organizations are also preferring to migrate their existing enterprise scale applications to microservices architecture. Researchers have proposed various approaches for microservices decomposition to be used in migrating or rebuilding a monolithic application to microservices. Applying any available approach to an existing monolithic application is not a straightforward decision; thus, there is a need for guidelines that assist in the migration process. There are various challenges in a migration process because different migration approaches use different sets of input data to identify microservices. Since the available migration techniques are not structured, logically, selection of an appropriate migration strategy is a difficult decision for any system architect. So, it is a recurrent open research

Journal of Web Engineering, Vol. 20_3, 817–852.
doi: 10.13052/jwe1540-9589.20312

question – which migration technique should be adopted to get microservices for a legacy monolithic application? This paper addresses this research challenge by examining existing approaches for microservices migration and groups them based on software development life cycle (SDLC) artifacts. Our research also proposes a microservices prescriptive model (MPM) from the existing prominent microservice migration techniques. This model provides recommendation (1) for refactoring an existing legacy system to microservices, and (2) for new microservices development projects. Our study also helps in gaining more insight about greenfield and brownfield development approaches in microservices applications. Moreover, researchers and practitioners of the field can benefit from this model to further validate their migration approaches based on the available system artifacts.

Keywords: Microservices, decomposition, migration, extraction, slicing, identifying, brownfield development, greenfield development, microservices architecture (MSA).

1 Introduction

Microservices architecture (MSA) is gaining popularity in both domains of academia and industrial world as they overcome the limitations of traditional, centralized, and monolithic architecture. MSA can be defined as a programming approach for developing an application as a set of small modular/loosely coupled, autonomous services with a specific business goal [31]. As per its definition given by Fowler [32], each service has its own database and it has the flexibility to use a type of database that is best suited to its business needs.

A microservice runs in its own process space and communicates with other services either by synchronous protocols such as HTTP/REST or asynchronous protocols such as AMQP, STOMP, and MQTT. Each microservice can scale autonomously and can be easily deployed on the cloud which has been adopted as a *de-facto* platform for microservices. These services, due to small granularity, are more fault-tolerant and easier to maintain. If one of the services fails, it will not bring down the whole system that could happen in earlier monolithic systems. Independent deployment characteristic of microservices makes it helpful for continuous integration and delivery pipeline [25].

To adopt this new architectural style, technical managers and system architects have options to either rebuild (rewrite the capability as a new

service) or refactor the existing legacy application into a set of microservices [26]. Refactoring is a controlled technique wherein software designers perform behavior preserving transformations in order to improve the internal design structure of an existing code base. This restructuring of existing code improves its internal design but, at the same time, does not modify the external behavior of the software system [32].

Decision on rebuilding a legacy application is very critical and may be preferred over refactoring approach in many scenarios; a few of them are listed below.

- The new infrastructure for hosting a microservice is quite dissimilar from the old legacy application runtime and would need boilerplate code to be rewritten.
- Sometimes existing capabilities are not coded around clear domain-driven design (DDD) concepts. This result in major restructuring of the entire codebases, making code reuse options less attractive.
- Legacy code attains high code toxicity level as they go through much iteration of change and yield low value for reuse. Toxicity means weakening of internal structure of the application's code and high number of access points by client applications.

Thus, refactoring can be considered a viable option if the capability is relevant and defined in a clear domain driven context.

For designing and developing any application in MSA, different approaches that are available in research literature can be classified as brownfield or greenfield [25]. If a new software system is required to rebuild or refactor from an existing legacy software application, then it is known as brownfield development. Fresh development of an application from scratch is known as greenfield development.

Existing brownfield techniques proposed to migrate a legacy monolith application to MSA can be further classified as static or dynamic. In static approach, system artifacts and characteristics that are static in nature are analyzed to identify the candidate microservices. System artifacts like requirement engineering documents, design (high level or low level or both) documents, source code files, and revision history repository of a legacy system can be used to analyze and identify important implicit system characteristics like coupling (afferent, efferent, and evolutionary), cohesion (internal and external), and functional dependencies (class, module, and database) [2–4, 12, 13]. Migration techniques based on static analysis of source code are also known as white-box techniques. In dynamic approach, source code of

application is analyzed at run time along with the analysis of web access log/user usage log/execution traces for mining the behavioral system characteristics like workload, performance, and scalability of the legacy system. Migration techniques based on dynamic analysis of source code are also known as black-box techniques.

Adoption to MSA is not an effortless or smooth journey [26] and companies are spending a lot of resources and efforts in migrating their legacy applications to MSA. In literature, a number of migration approaches have been proposed, discussed, and validated. However, researchers have recommended a migration strategy considering only a subset of the available system artifacts and there is no "one size fits all" for all use cases. To build this gap, we propose a microservices prescriptive model (MPM) based on software development life cycle (SDLC) artifacts to adopt a microservice-based architectural style.

The key contributions of this paper are as follows.

- Collated key migration techniques to build microservices (1) from an existing legacy software system, and (2) for a new application development.
- Identified key parameters and analyzed the selected migration techniques for these parameters.
- Formulated an MPM for migration based on greenfield and brownfield microservice development. MPM depicts mapping of the SDLC artifacts for adoption of existing migration techniques while architecting with microservices.
- Presented evidence-based findings for state-of-the-art migration techniques, static and dynamic analysis tools for migration automation, validation methods, and quality measures adopted in these migration techniques.

Our research aims to provide a detailed informative knowledge base to understand and use the microservice migration process effectively. This could be a useful resource for software industries and companies that are planning to adopt MSA.

The remainder of this paper is organized as follows. Section 2 briefs about existing methodologies that are used to extract microservices from legacy monolithic applications. In Section 3, an outline of related work is presented and Section 4 provides an in-depth description of the research methodology and search design. Section 5 elucidates the parameters that are established to compare decomposition strategies discussed in selected primary papers. This

section also provides answers to research questions based on our extensive literature review. Then, Section 6 states the discussion and results.

2 Existing Migration Approaches

Migrating legacy applications to MSA implies breaking existing code into loosely coupled services [23, 31]. Here, we discuss papers about refactoring/migrating/decomposing/rebuilding legacy monolithic applications to microservices.

Taibi et al. [1] propose a framework for decomposition process that provides software architects with a set of decomposition options using process mining on access logs. They also propose a metric-based ranking mechanism to evaluate and compare the quality of decomposed microservices. In their approach, they have applied process mining tools on application log files to identify the most frequent execution path of the system and removal of branches and circular dependencies.

Algorithmic recommendation of microservice candidates in a refactoring and migration scenario is given by Mazlami et al. [2] using formal coupling criteria: logical coupling, semantic coupling, and contributor coupling. In logical coupling, software components that get changed for the same reason are gathered together as one microservice. In semantic coupling, information retrieval techniques are employed to identify microservices by investigating the semantics and contents of source code files. Contributor coupling is based on ownership architecture that reveals communication and team structuring patterns. In this coupling, the key idea is to maximize internal communication and reduce communication overhead among external teams. These three coupling criteria transform a given monolith code into graphical representation wherein weight of the edge indicates coupling strength between classes. Extraction of microservices is achieved by deleting edges from the graphical representation using a minimum spanning tree algorithm.

Dataflow-driven decomposition approaches are used in [3, 4] for identifying microservices from detailed business requirements. A top-down decomposition approach used in [3], in which authors have transformed traditional dataflow diagram (DFD) to get purified DFD. Authors designed a two-phase automation algorithm for decomposition: (1) generating a decomposable DFD from purified DFD; (2) identifying candidate microservice from decomposable DFD. A semi-automatic dataflow-driven microservice decomposition approach is suggested by [4]. They propose to generate a process-datastore version of DFD ($\mathrm{DFD_{PS}}$) from fine-grained DFDs. $\mathrm{DFD_{PS}}$

shows the connection between processes and related data stores and excludes information like external entities. Another condensed DFD called decomposable DFD is taken out from $\mathrm{DFD}_{\mathrm{PS}}$ by extracting the sentence sets in which a process reads or writes data to a data store. Last step of their proposed approach is to group modules of fine-grained processes and the related data stores to identify candidate microservices.

Amiri et al. [5] presents a microservice identification method using clustering technique based on a set of business processes. Authors have shown that it is possible to identify fine-grained microservices from a single business process or a set of business processes. Their works exploited the notions of structural and object dependency between business activities represented as business process model notation (BPMN). They introduced multiple approaches for microservices identification, i.e., user-driven, process-driven, object-driven, and an extended approach. They also studied the accuracy results of their approaches for different business processes.

Not just business entities of any application, but the relationships and database interaction of these entities are also important in deciding the entities that should be clubbed together and where to break into finer components. Static code analysis is performed to get this relationship and is depicted as a dependency graph. Levcovitz [8] proposed a decomposition technique based on client side, façade, and system database. Since a database is an important component in any application design, their approach uses a database schema for splitting the application.

Analysis done on the basis of business domain may not find highly loaded parts of the system [6, 9]. The load predictions done by the software architects may not be enough for complex use cases and should be supplemented with users' access records for the web pages. Mining web usage logs to dig out useful details regarding user behavior has been investigated in research. Muhammad et al. [6] devised a novel approach to automatically decompose a monolithic application into microservices based on a black-box approach using web access logs of the application for better performance and scalability and then applied an unsupervised clustering method to decompose microservices. They also proposed a method to dynamically select the appropriate resource type to deploy microservices so as to improve the overall performance of microservices-based applications.

Mustafa et al. [9] also suggested a black-box-based technique that extends utilization of web usage mining techniques and considered non-functional requirements like performance and scalability. Authors dig out web access logs of a web application to find workload distribution patterns and then apply

clustering technique to identify granular microservices. The rationale behind their approach is to discover pages that are attracting high loads within certain periods of time slots and those should be designed as microservices.

Ahmadvand [13] proposed a conceptual methodology that reconciles security and scalability requirements to be included in the requirements engineering phase and develops an improved view for the system architects. Their approach maps functional and non-functional requirements to identify more optimal system decomposition.

Ren et al. [14] suggested that static code analysis techniques do not include runtime dynamic characteristics; so they developed a combined technique of static and dynamic analysis to get static and runtime behavior of monolithic application. They use coupling among functions to estimate the degree of dependence between functions. They further employ function clustering to achieve the migration of monolithic applications to microservices.

Eski et al. [12] used the code repositories and evolutionary coupling information of software classes and apply graph clustering techniques to transform existing monolithic applications into microservices. They observed that identifying microservices using both static and evolutionary coupling achieves better results than just using either of the two approaches. To evaluate the quality of decomposition, they compared identified microservice and reference microservices that are designed by the developers of associated projects.

Selmadji et al. [13] proposed an auto-decomposition technique to identify microservices from Object Oriented source code. They group classes on the basis of their dependencies and then apply hierarchical agglomerative clustering algorithm to extract microservices. Their approach also assesses quality of microservices as a measure of structural and behavioral validity. Classes that maximize the microservice quality function are grouped together.

Gysel et al. [14] proposed a structured and repeatable approach for microservice decomposition. Their technique is based on 16 coupling criteria extracted from software engineering artifacts such as domain models and use cases. This coupling information is represented as an undirected weighted graph to extract densely connected clusters. They have developed an indigenous tool "service cutter" for service decomposition. The practical applicability of this tool is inadequate due to its dependency on detailed specification of the system.

Baresi et al. [15] proposed semantic similarity of available functionality evaluated through OpenAPI specifications. Their approach identifies

potential candidate microservices by matching the key terms in the specifications against a reference vocabulary and suggests possible decompositions. The success of their approach is dependent on well-defined Application Programming Interfaces that give meaningful names.

Sayara et al. [17] gave a probabilistic approach for slicing monolith application and getting microservices using multidimensional scaling. Their technique employs update and scaling probabilities for generating a weighted matrix which is used to generate optimal number of microservices.

Nakazawa et al. [18] suggested a runtime analysis of the monolithic system to produce a calling-context tree (CCT) that can be used to determine the strength of relationships between components. They developed a visualization tool that allows developers to design microservices based on source code and function calling behaviors from a monolithic prototype. Their tool constructs a compact CCT from profile data. They have applied two clustering algorithms: (1) semantic-based clustering and (2) calling-context-based clustering. They developed a visual interface for further refining microservice design.

Krause et al. [19] combined both static and dynamic analysis of a software system. They implement runtime behavior, i.e., live trace visualization to identify microservices. They also combine bounded context patterns of DDD to augment dynamic software visualization results to make out appropriate microservice boundaries.

Jin et al. [20] proposed a functionality-oriented method for microservice extraction. Rather than static source code analysis, they instrumented the test application by adding probes to collect program dynamic behavior. In the next step, clustering is performed on execution traces to achieve microservices candidates. They also validated their results by exploiting five metrics of service cohesion and coupling.

Stojanovic et al. [21] proposed a systematic system analysis based on functions, interfaces, dataflows, and data storage of the system to extract microservices.

3 Related Work

Though a few research articles exist in academic research that brief about available migration techniques, they are majorly confined to challenges regarding microservices adoption in industries, trends in microservices research, and microservices patterns and principle.

Fritzsch et al. [28] compared ten recent refactoring approaches proposed in academic literature and provided a visual guide for reference. Ponce et al. [29] presented a rapid review of some migration techniques available in literature. They highlighted the types of programming languages suitable for MSAs. Also, they briefed about challenges faced in the migration process. Kazanavièius et al. [30] reviewed five migration methods along with their benefits and drawbacks. Schmidt [33] investigates microservices identification proposals with the perspective of model-driven engineering (MDE) and DDD.

Francesco et al. [34] have applied a systematic mapping methodology to classify and evaluate research publications from the following perspectives: (1) publication trends (publication venue and research groups contributing in this field), (2) research aspects (performance analysis and monitoring, security concerns, and fault tolerance issues), and (3) potential for industrial adoption. Francesco et al. [35] performed an empirical study on available migration techniques. They conducted a survey on 18 industry practitioners, by means of interviews and questionnaires, engaged in the migrating process and collated quantitative information about MSA like (1) the set of activities performed during migration, and (2) challenges faced during the migration. Taibi et al. [36] carried out a mapping study to explore common microservices patterns and principles. They also prepared a catalog highlighting advantages and disadvantages in those patterns.

Our literature review exposed a lack of structured knowledge base regarding building new microservice applications and rebuilding/refactoring techniques to migrate existing monolithic legacy code. We found that there is a clear dearth of systematic study that will distinctly guide system architects and developers about the suitable migration technique that should be adopted to achieve MSAs for their monolithic application. A necessity of pragmatic guidance was realized that would concretely aid when migrating an existing legacy system (brownfield development) or building a new system as per microservice architectural design (greenfield development).

We have not come across any research article that compares available migration techniques at such great depth covering validation methods, automation tools used for migration, and microservice evaluation framework for result. This makes our work more comprehensive and complete and addresses inadequacies of the existing studies. To the best of our knowledge, there is no model currently available that structures migration techniques

on the basis of SDLC artifacts. Thus, our research study aims to fill the existing gap by providing a comprehensive MPM that can be used in selecting migration techniques for greenfield or brownfield development based on SDLC artifacts.

4 Research Methodology and Search Design

The primary objective in this section is to consolidate the decomposition approaches available in literature for migrating a monolithic application to MSA. We used the guiding principle of systematic literature review (SLR) for literature assimilation [22]. SLR facilitated us to organize the procedure of identifying and classifying relevant writings in our domain. Following subsections illustrate, in detail, our research methodology, search strategy, and design.

4.1 Research Questions

Our research goal is to gather, collate, and investigate the migration/extraction techniques proposed in literature. Based on these goals, we formulated the following research questions to be addressed, which are the major part of this study:

RQ1: What are the migration techniques proposed in the literature for MSA?
RQ2: How are existing microservice migration techniques associated with SDLC artifacts? What all microservice decomposition techniques are applicable to greenfield and brownfield development?
RQ3: How are results of decomposition techniques validated by authors?
RQ4: What are the significant measures for microservice evaluation framework?
RQ5: What all tools are used in migration techniques for refactoring a monolithic application?

In order to investigate and get answers to the above-formulated research questions, we design the search strategy discussed in the next subsection.

4.2 Search Strategy

We explored research articles indexed in ACM Digital Library, Scopus, IEEE Xplore, Science Direct, SpringerLink, and Google Scholar databases. The

search keywords used were:

("Service" OR "Microservice[s]" OR "Micro-service[s] architecture" OR "Cloud-native architecture[s]")
AND
(Migrat* OR Transform* OR Extract* OR [Auto-] Decompos* OR Slic* OR
Re-architect* OR Identificat* OR Discovery OR Modernizat*)

The search list includes articles extracted during 2016–2020 (inclusive). To make the list more relevant and complete, we applied snowballing [22], by checking the reference list of the articles found previously. This makes our selection work more comprehensive and aggregate.

"Microservices" being the latest and fertile area for research, we considered both journal articles and conference papers. After removing duplicate papers, 40 potentially relevant papers were identified in the first compiled list. To further narrow down and refine our collection of papers, we augment our selection procedure by supplementary criteria:

(a) Language of the manuscript must be English.
(b) Only peer reviewed articles to be included.
(c) Focus of the paper must be some migration mechanism or technique (refactoring/rebuilding).
(d) Papers must answer at least one of the research questions formulated above (RQ1-RQ5).

Papers satisfying above criteria were finally selected. From this complex analysis, we distilled 21 most recent and relevant papers pertinent to our study and we designate them as primary studies. We thoroughly investigated these primary studies to get answers for our research questions.

5 Study Results

There are various challenges in software migration from monolithic architecture to MSA. In this paper, we have studied various migration techniques and their unique strengths and weaknesses are also analyzed. We have reviewed decomposition approaches and tabulated on the following attributes: Technique of Decomposition, Approach Type, SDLC Artifact Used, Methodology, Tools used for Refactoring, Measures & Metrics, and Validation Method. Table 1 (Part A and Part B) gives a synopsis of the reviewed decomposition

Table 1 (Part A) Summary of migration strategies

S. No. No.	Paper Title (Year): Citation Number in Reference List	Technique of Decomposition	Approach Type	SDLC Artifact Used
1.	From monolithic systems to microservices: a decomposition framework based on process mining (2019):[1]	Process mining	Dynamic	Application logs
2.	Extraction of microservices from monolithic software architectures (2017):[2]	Graph-based clustering algorithm based on graph constructed using three formal coupling strategies: logical, semantic, and contributor coupling	Static	Change history of code obtained from version control system
3.	From monolith to microservices: a dataflow-driven approach (2017):[3]	Top-down analysis approach based on dataflow-driven decomposition algorithm	Static	Traditional DFD and business requirement
4.	A dataflow-driven approach to identifying microservices from monolithic applications (2019):[4]	Dataflow-driven semi-automatic decomposition approach	Static	Traditional DFD and business requirement
5.	Object-aware identification of microservices (2018):[5]	Genetic algorithm applied on structural and data object dependency	Static	Business process model
6.	Unsupervised learning approach for web application auto-decomposition into microservices(2019):[6]	Black-box decomposition using unsupervised scale weighted K-mean clustering (SWK)	Dynamic	Web access log
7.	Migrating monolithic mobile application to microservice architecture: an experiment report (2017):[7]	Systematic SDLC -based decomposition	Static	Domain-driven design

8.	Towards a technique for extracting microservices from monolithic enterprise systems (2016):[8]	Extraction technique based on client side, façade, and database design	Static	Requirement document based on system and sub-system classification
9.	Optimizing economics of microservices by planning for granularity level(2017):[9]	Black-box decomposition technique based on fuzzy C-mean clustering	Dynamic	Web access logs
10.	Requirements reconciliation for scalable and secure microservice (de)composition (2016): [10]	Conceptual decomposition	Static	Functional and non-functional (scalability and security) requirements
11.	Migrating web applications from monolithic structure to microservices architecture (2018):[11]	Semi-automatic migration approach based on K-means hierarchical clustering and Jaccard similarity index	Static + dynamic	Classes, function call relations, application code, and application log
12.	An automatic extraction approach-transition to microservices architecture from monolithic application (2018):[12]	Fast community graph clustering algorithm and MOJO similarity index	Static	Static code repositories
13.	Re-architecting OO software into microservices: a quality-centered approach (2018):[13]	Hierarchical agglomerative clustering	Static	Source code files
14.	Service cutter: a systematic approach to service decomposition(2016):[14]	Clustering algorithm based on coupling criteria catalog	Static	Entity relationship models (ERMs), use case, domain model in JSON
15.	Microservices identification through interface analysis(2017):[15]	Clustering of API specifications based on semantic similarity	Static	OpenAPI specifications

(*Continued*)

Table 1 (Part A) Continued

S. No. No.	Paper Title (Year): Citation Number in Reference List	Technique of Decomposition	Approach Type	SDLC Artifact Used
16.	Towards a microservices architecture for clouds (2016):[16]	Bottom-up data-driven migration	Static + dynamic	Source code files and application log
17.	A probabilistic approach for obtaining an optimized number of services using weighted matrix and multidimensional scaling (2017):[17]	Probabilistic technique based on multidimensional scaling	Static	Requirement document containing update rate, scaling rate and technology of business capabilities
18.	Visualization tool for designing microservices with the monolith-first approach (2018):[18]	Clustering based on semantic and calling-context tree clustering	Static	Source code files
19	Microservice decomposition via static and dynamic analysis of the monolith (2020): [19]	Combined static and dynamic analysis techniques	Static + Dynamic	Domain-driven design, source code files and application log
20	Functionality-oriented microservice extraction based on execution trace clustering (2018): [20]	Clustering of execution traces obtained by monitoring of program dynamic behavior	Dynamic	Application execution traces
21	Identifying microservices using structured system analysis (2020): [21]	Functions, interfaces, dataflows, and data storage of thesystem can be grouped into microservices	Static	Dataflow diagrams

Table 1 (Part B) Summary of migration strategies

S. No.	Paper Title (Year): Citation Number in Reference List	Methodology	Tools Used for Refactoring	Measures and Metrics	Validation Method
1	From monolithic systems to microservices: a decomposition framework based on process mining (2019):[1]	Six step decomposition framework to identify different slicing options	ElasticAPM, DISCO	Coupling between objects, coupling between microservice, number of classes/microservice, number of classes need to be duplicated	A document management system for bookkeeping, for Italian tax accountants developed in JAVA. Validated against static methods of expert software architectures
2	Extraction of microservices from monolithic software architectures (2017): [2]	Two-step process (construction and clustering) based on three extraction stages (monolith, graph and microservice stage)	Nil*	Average development team size, average domain redundancy	A proof of concept is developed in an open source Java project that takes sample set of 21 open source monolithic code bases using GIT VCS
3	From monolith to microservices: a dataflow-driven approach (2017): [3]	Three-step process: (1) business requirement analysis, construct a purified DFD, condensing the purified DFD into a decomposable DFD	Nil*	Cohesion, coupling, neutral development technique, fine granularity	Two use cases – (1) movie-information-crawling project (2) movie ticket price comparison Application and validated against service cutter tool

(*Continued*)

Table 1 (Part B) Continued

S. No.	Paper Title (Year): Citation Number in Reference List	Methodology	Tools Used for Refactoring	Measures and Metrics	Validation Method
4	A dataflow-driven approach to identifying microservices from monolithic applications (2019):[4]	Four-step decomposition procedure: construct the fine-grained DFD and the process-datastore version of DFD, extract the dependencies between processes and data stores, and identify candidate microservices	SonarGraph architect	Afferent coupling, efferent coupling, instability, relational cohesion	Cargo tracking system, an opensource GIT project, and validated against service cutter [17] and microservice identification using API interface [18]
5	Object-aware identification of microservices (2018):[5]	Identifying interconnections between business activities and their data dependencies	Nil*	Accuracy	Experiment based on number of activities, number of gateways, number of objects, and number of processes

6	Unsupervised learning approach for web application auto-decomposition into microservices (2019):[6]	URI space partitioning from historical access logs and mapping to appropriate VM types based on resource usage	Nil*	Scalability Performance	AcmeAir, opensource benchmark web application for a fictitious airline, available in both monolithic and microservices implementations.
7	Migrating monolithic mobile application to microservice architecture: an experiment report (2017):[7]	Gradual iterative approach based on bounded context and analysis of database structures	Nil*	Nil*	EasyLearn mobile application as monolithic application deployed on Google Play
8	Toward a technique for extracting microservices from monolithic enterprise systems (2016):[8]	Six-step procedure to identify microservices using facade, business functions, and database tables	Nil*	Nil*	750 KLOC real-world monolithic banking system
9	Optimizing economics of microservices by planning for granularity level (2017):[9]	Mining workload patterns based on web access logs	Nil*	CPU utilization, resource utilization	Micro-Book Shop Online application deployed on cloud Azure
10	Requirements reconciliation for scalable and secure microservice (de)composition (2016): [10]	Requirement reconciliation and elicitation	Nil*	Nil*	Imaginary application: MovStreamInc (online movie streaming system implemented as SaaS application)

(*Continued*)

Table 1 (Part B) Continued

S. No.	Paper Title (Year): Citation Number in Reference List	Methodology	Tools Used for Refactoring	Measures and Metrics	Validation Method
11	Migrating web applications from monolithic structure to microservices architecture (2018):[11]	Program analysis using static analysis (obtained fromabstract syntax tree, class hierarchical architecture, and call graph) and dynamic tracing using clustering	EasyAPM WALA	Quality	Open source and industrial projects (total 12) validated against 4 benchmark applications (DayTrader, JPetStore, TPC-W, RUBiS)
12	An automatic extraction approach-transition to microservices architecture from monolithic application (2018):[12]	Graph clustering obtained by static and evolutionary coupling between software classes	Nil*	Nil*	Two java projects: 1) eQuality and 2) Academics. Validate similarities between identified microservice candidates and reference microservices (obtained from authoritative decomposition)
13	Re-architecting OO Software into microservices, a quality-centered approach (2018):[13]	Classes are grouped based on their dependencies by using a tool developed in JAVA	Nil*	Quality, internal coupling, external coupling, internal cohesion, external cohesion	Three GitHub projects and microservice decomposition was validated against manual methods using precision and recall formula

14	Service cutter: a systematic approach to service decomposition (2016):[14]	Provides a tool framework for service decomposition and extract microservices from 16 coupling criteria	Service cutter	Nil*	1) Trading system (fictitious application) and 2) Cargo Tracking System (DDD sample application)
15	Microservices identification through interface analysis (2017):[15]	Decomposition on the basis of OpenAPI specifications and reference vocabulary as fitness function	DISCO	Nil*	Cargo tracking system (DDD sample application) and validated against service cutter[17]
16	Toward a microservices architecture for clouds (2016):[16]	Decompose microservices as functional properties and non-functional capabilities	Nil*	Nil*	Synthetic video processing application deployed on green cloud
17	A probabilistic approach for obtaining an optimized number of services using weighted matrix and multidimensional scaling (2017):[17]	Uses a weighted matrix obtained from update and scale probability followed by multidimensional scaling to group together similar business capabilities	Nil*	Nil*	Fictitious application: book subscription, feedback and recommendation system
18	Visualization tool for designing microservices with the monolith-first approach (2018):[18]	Generating calling-context tree after filtering library calls from profile data collected from a monolithic prototype	JProf - Java profiling agent, microservice visualization tool	Service granularity runtime performance	Two open-source web benchmark applications 1) Acme Air and 2) DayTrader, compared to their official design

(*Continued*)

Table 1 (Part B) Continued

S. No.	Paper Title (Year): Citation Number in Reference List	Methodology	Tools Used for Refactoring	Measures and Metrics	Validation Method
19	Microservice decomposition via static and dynamic analysis of the monolith (2020): [19]	Domain analysis involving three steps: familiarizing, modeling,and partitioning	Structure101, Explorviz, DBeaver	Nil*	JAVA real-word, legacy online lottery application iniFOCUS
20	Functionality-oriented microservice extraction based on execution trace clustering (2018): [20]	Three-step process to perform execution Trace analysis (class and method level execution traces) to group source code entities as microservices	Kieker 1.13	Cohesion at domain level, cohesion at message level, interface number, operation number, interaction number	Four Java applications: (1) JPetstore, (2) Springblog, (3) JForum, and (4) Roller. Validated against MEM [2]
21	Identifying microservices using structured system analysis (2020): [21]	Systematic approach based on grouping of primitive functions and their data stores	Nil*	Nil*	Nil*

*Nil indicates research paper does not discuss this attribute.

approaches. Below is a brief description of the attributes:

- *Technique of Decomposition* is the real core of the approach that authors have used for microservice identification.
- *Approach Type* points to static, dynamic, or hybrid approach depending on system artifacts considered as input.
- *SDLC Artifact Used* is a monolithic system's artifacts and characteristics that are used and analyzed to identify the candidate microservices.
- *Methodology* is systematic analysis of the approach chosen by the author. This illustrates the description of strategy used in each research article.
- *Tools used for Refactoring* describe the existing tools used in the paper or developed by the author for decomposition purposes. Tools can be used for web log collection, static code analysis, or coupling calculation.
- *Measures and Metrics* suggest empirical evaluation of the approaches through some metrics. Not all primary studies have shown evaluation of their microservices based on some evaluation framework.
- *Validation Method* is the practice used to assess the goodness of the decomposition approach employed for migration. Researchers have used various validation methods like industrial project, sample project, case study, experiment, or proof of concept (PoC).

5.1 Answer to Research Questions Based on Our Extensive Literature Review

Our systematic review of primary studies gave answers to initially phrased research questions. In this section, we will provide answers to all the above-formulated research questions.

RQ1: What are the existing migration techniques proposed in the literature for MSA?

After analyzing the selected primary studies, we can classify the existing microservices migration techniques into three broad categories as follows:

- Static: These techniques or approaches are based on software artifacts that are static in nature like requirement documents, design documents, source code files, and source code revision history repository.
- Dynamic: These approaches are based on software artifacts that are collected at execution or run time of an application like web access log and user usage log.

Table 2 Identified broad categories of microservices migration techniques

Approach Type	SDLC Artifacts Used	Paper Reference Number
Static	Requirement documents	8, 10, 14, 17
	Requirement models	3, 4, 5, 21
	Design documents	7, 14, 15 19
	Source code files	2, 11, 12, 13, 16, 18, 19
	Source code revision history repository	2, 12
Dynamic	Web access log	6, 9
	Application logs	1, 11, 16, 19, 20
Hybrid	Both static and dynamic documents	11, 16, 19

- Hybrid: In these approaches, researchers have exploited both the static characteristics and dynamic or run-time behavior of an application.

Table 2 shows Approach Type, SDLC Artifact Used, and paper reference number (as per Table 1) where the approach is being used. Mazlami et al. [2] have used change history of code and commit history from version control system to establish set of classes that were changed together or worked on by the same set of developers. Chen et al. [3], Li et al. [4], and Stojanovic et al. [21] have exploited DFD of the business logic to derive microservices. Amiri et al. [5] used business process models along with structural and data object dependencies for microservice identification. Fan et al. [7] have used DDD of the monolithic application to identify potential microservice candidates. A similar approach is used by Levcovitz et al. [8] by identifying requirement documents containing systems and subsystems information. Ahmadvand et al.'s [10] decomposition methodology is based on functional and non-functional requirements of the legacy system. Approaches discussed in [2, 11–13, 16, 18], and [19] are based on source code repositories of an existing application. Coupling and cohesion between classes are used for identification of microservices. Gysel [14] presents legacy application as nano-entities comprising data, its operation, and artifact elements. These nano-entities are connected on the basis of 16 coupling criteria to extract groups of services.

Techniques given by Taibi et al. [1], Muhammadet al. [6], Mustafa et al. [9], and Jinet al. [20] are based on web access log or application logs to derive microservices. Renet al. [11], Procaccianti et al. [16], and Krause et al. [19] use both static and dynamic approaches for migration.

RQ2: How are existing microservice migration techniques associated with SDLC artifacts? What all microservice decomposition techniques are applicable to greenfield and brownfield development?

To answer this question, we are proposing an MPM based on SDLC artifacts as shown in Figure 1. In our model, we are identifying microservices for three kinds of applications – (1) anew application, (2) an existing monolith application where the code may not be of much use, and (3) an existing monolith application where the code may be used.

After investigating the selected primary studies, we identified that decomposition techniques exist for both greenfield as well as brownfield development. A new application development from scratch is called greenfield development. Rebuilding a legacy application means re-architecting and developing the application again from its existing requirement documents and models. Refactoring a legacy software means restructuring the existing code into MSA without changing its external behavior, i.e., using the existing monolith code for microservices construction. Refactoring and rebuilding techniques come into the category of brownfield development.

We observed that researchers have not categorized the applicability of their approaches according to greenfield and brownfield development. This indistinctness may pose challenges in the best approach selection for migration.

The proposed MPM model defined on the basis of greenfield and brownfield is as follows.

- **New Application:** During greenfield development, system architects and developers may use requirement documents for the identification of the microservices. These documents may vary depending on the availability likeuse cases, functional, and non-functional requirements. The microservices can also be identified from the available requirement models like DFD and BPMN.
- **Rebuilding:** Occasionally, developers disregard the available code of legacy application and plan to rebuild the entire application as microservices due to (1) high maintenance cost of old code and (2) low value outcome in extracting and reusing the old code. It also gives flexibility to system architects to implement new services in more appropriate programming languages and emerging technology stacks. Thus, rebuilding offers technology refreshment for services. As a result, they retire the legacy application code and use requirement documents (use cases, functional, and non-functional requirements), requirement models (DFD

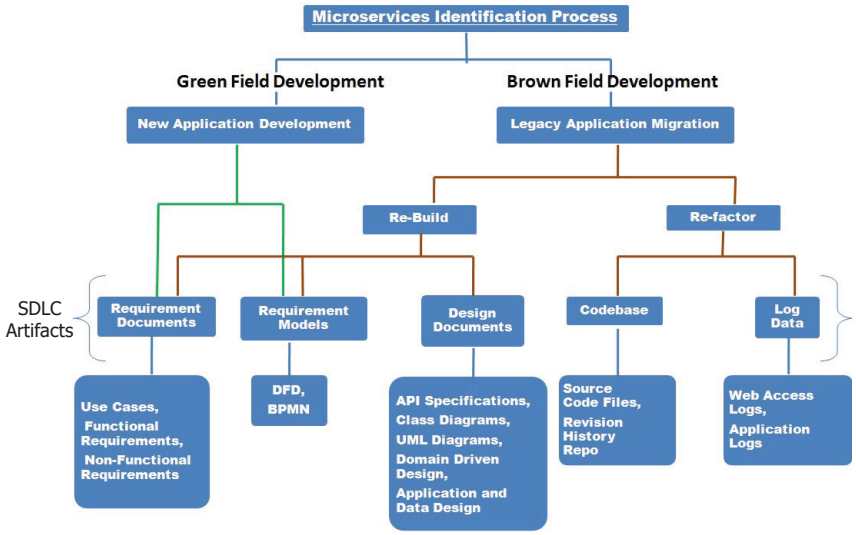

Figure 1 Microservices prescriptive model for migrating to microservices based on SDLC artifacts.

and BPMN), and design documents (API specifications, class diagrams, Unified Modeling Language diagrams, DDD, application and data design for rebuilding microservices.

- **Refactoring:** Code modules can be considered as good candidates for reuse and extraction, if (1) they are not toxic, (2) their functionality is based on single responsibility principle with clearly defined boundaries in terms of DDD, and (3) have high intellectual property. In such scenarios, developers may plan to restructure existing source code files. Code revision history, web access logs, and application logs provide supplementary information in the refactoring process.

Table 3 shows decomposition technique, documents that can be used, and paper reference number where the approach is being used. Techniques discussed in [3–5, 8, 10, 14, 17], and [21] make use of requirement documents and models which enable these approaches suitable for greenfield development and thus can be exploited for building a new microservice application. Requirement documents and models along with design documents discussed in [7, 14, 15], and [19] can be exploited in rebuilding microservices from an existing legacy application.

Similarly techniques discussed in [1, 2, 6, 9, 11–13, 16, 18, 19], and [20] can be used by software architects and designers for brownfield development

Table 3 Identified broad categories of microservices migration

Development Type	Decomposition Strategy	SDLC Artifact Used	Paper Reference
Greenfield development	New application development	Requirement documents	8, 10, 14, 17
		Requirement models	3, 4, 5, 21
Brownfield development	Rebuilding	Requirement documents	8, 10, 14, 17
		Requirement models	3, 4, 5, 21
		Design documents	7, 14, 15 19
	Refactoring	Source code files	2, 11, 12, 13, 16, 18, 19
		Source code revision history repository	2, 12
		Web access log	6, 9
		Application log	1, 11, 16, 19, 20

as they mainly employ source code files, source code revision history repository, web access log, and user usage log to extract microservices.

RQ3: How are results of decomposition techniques validated by authors?

For this question, we recognized various validation methods used by different researchers to demonstrate the correctness of their decomposition process. It is very essential to assess goodness and correctness for any migration technique to justify its usefulness. In literature, we observed that authors have validated their migration approaches by decomposing either an industrial project or sample project and presented their work as an experience report. Few researchers have demonstrated the feasibility of their approach through PoC and shown the validity of results. Others have used either a case study or experiment analysis to prove the correctness of their migration technique. Table 4 shows the different validation methods opted by researchers.

Some primary studies validated their results using sample open source web benchmark applications and compared their results with their official design. *JPetStore* and *Cargo Tracking System* are commonly used monolithic web applications used for validations, while *AcmeAir* and *SpringBlog* are benchmarking applications implemented both as monolithic and MSAs.

Table 4 Identified validation methods of microservices

Validation Approach	Paper Reference
Industrial project	1, 8, 11,19
Sample project	4, 7, 9, 11, 12, 13, 14, 15, 18, 20
Proof of concept	2, 16
Case study	3, 6, 10, 17
Experiment	5
Consultation from industry expert	1, 12, 13

Table 5 Identified quality measures of microservices

Measure	Paper Reference
Coupling	1, 3, 4, 13, 20
Cohesion	3, 4, 13, 20
Accuracy	5, 11, 13
Scalability	6
Performance	6, 18
Operations provided by the microservice	20
CPU utilization/resource utilization	9

Some researchers got their decomposition approach validated by consulting software industry experts who have identified microservices manually. Few have just laid down their approach without referring to any validating approaches [21].

Few researchers have also proved the goodness of their approach by comparing their results with the results presented in other research articles. For this, they have applied their migration approach on some benchmark web applications. Li et al. [4] have compared their results with Gysel et al. [14] and Baresi et al. [15]. Similarly, Nakazawa et al. [18] and Jin et al. [20] have compared their results with Mazlami et al. [2].

RQ4: What are the significant measures for a microservice evaluation framework?

To evaluate quality, accuracy, and goodness of identified microservices, it must be evaluated against some microservices measurement framework [21]. Our literature study reveals that not much work has been done in identifying the right measures for assessing the quality of microservices. Different

Table 6 Tools used in migration techniques for refactoring a monolithic application

Type of Tool	Tool Name	Brief Description of the Tool	Paper Ref. No.	License
Static	Structure101 [37]	Software structure analysis tool to analyze the dependencies and restructuring the source code packages (SCP).	31	Proprietary
	JPROF [38]	Java profiling agent which is used to capture execution flow in the form of call trees.	18	Open source
	SonarGraph Architect [39]	Analysis tool used for continuous inspection of code. It allows simulation of refactoring without touching the source code.	4	Proprietary
	WALA [40]	Code analyzer for Java to produce class hierarchy and inter-procedural dataflow analysis.	11	Open source
Dynamic	Kieker [41]	Provides dynamic analysis capabilities, i.e., monitoring and analyzing a software system's runtime behavior – enabling application performance monitoring and architecture discovery.	20	Open source
	ExplorViz [42]	Trace visualization tool that enables a live visualization and monitoring of large software applications. It reveals applications underlying architecture including package hierarchy and shows related communication.	19	Open source

(*Continued*)

Table 6 Continued

Type of Tool	Tool Name	Brief Description of the Tool	Paper Ref. No.	License
	Elastic APM [43]	Application performance monitoring (APM) tool built on the ElasticStack. It allows programmers to monitor applications in real time by gathering detailed performance statistics.	1	Open source
	DISCO [44]	Process mining tool that allows discovering visual maps from process data.	1, 15	Proprietary
Database Analysis	DBeaver [45]	Universal database management tool used to identify database tables used in a specific business context. It enables data and schema migrations.	20	Open source

measures proposed by researchers are coupling (afferent, efferent, and evolutionary), cohesion (internal or external), accuracy, scalability, performance, and resource utilization. Not all primary studies have shown evaluation of their microservices based on some evaluation framework. A metric-based ranking mechanism is proposed by [1] to assess identified decomposition options. They used coupling between microservices (CBM) which can be derived by number of external links divided by number of class in the microservice. Ahmadvand [13] uses internal cohesion and coupling and external cohesion and coupling for global evaluation of microservice characteristics. Primary studies [6] and [18] take into consideration scalability and performance respectively for evaluation. Table 5 shows identified quality measures of microservices.

RQ5: What tools are used in migration techniques for refactoring a monolithic application?

Refactoring a monolithic application can be done by examining dependencies in its codebase and finding out various parts that are interconnected. Several generic software quality, architectural analysis, and application performance analysis tools are available using which microservice specific

architectural patterns can be identified. These tools enable us to understand the linkage between closely related units which can serve as microservices. Broadly, these tools can be categorized as static analysis tools and dynamic analysis tools. Static analysis tools identify logically cohesive components from a wide range of system components before running an application. Dynamic analysis identifies cohesive components after running an application, i.e., observing runtime behavior from application log or application runtime traces. Table 6 shows the tools used by researchers in their migration techniques for refactoring a monolithic application along with its licensing options.

Researchers have used these tools in their migration strategies to refactor monoliths to get microservices. Table 6 shows the list of static and dynamic analysis tools. Majority of these tools support Java, C#, C, C++, and .Net programming languages.

6 Discussion and Conclusion

As monolithic systems are becoming monstrous to deal with, several organizations are breaking their legacy code to the MSA style. Microservices are relatively a new architectural style and old monoliths are very different from it in many aspects. Journey from monolith to MSA is not easy as there are various decomposition techniques available in literature, and identifying a technique to follow is a complex decision. Existing migration methods and techniques do not fit well for every use case.

This research study is based on SLR and it creates awareness about existing migration techniques from monolith to MSA. Existing migration strategies have been categorized as static, dynamic, or hybrid build upon the SDLC artifacts used and it answers our initially phrased first research question. We observed that majority of papers selected as primary studies in our review are based on static approaches capturing SDLC artifacts. Dynamic approaches make use of runtime data, i.e., application logs and web access logs to achieve optimally tailored services. Approaches involving both static and dynamic methods are categorized as hybrid.

In our study, we give a comprehensive literature review on various migration techniques and classify them. We have proposed an MPM based on SDLC artifacts which answers our second research question. This model can be used in new application development or rebuilding/refactoring an existing legacy application to microservices. Thus, our model will definitely help its readers in identifying a suitable migration approach for their projects.

Researchers have detailed about their migration approaches but never explicitly recorded its applicability for greenfield and brownfield developments. We have categorized existing migration strategies used in greenfield or brownfield developments. Our study revealed that most of the approaches exist for brownfield scenarios as it is relatively straightforward to partition an existing system. Identifying service boundaries in a greenfield system could be complex as it necessitates a sufficient level of technical maturity and complete understanding of the system at the beginning itself. This indicates that greenfield development would be a future research direction.

The third research question addresses the multiple validation approaches used by researchers in their migration strategies. We perceived that most scientific articles have validated their approach using sample projects to present their results. Some have compared their results with other established techniques as well. We suggest that any migration strategy should be objectively and quantitatively assessed as done by Li et al. [4] and Jin et al. [20] rather than just subjective assessment as done in Chen et al. [3]. The fourth question discusses microservice evaluation frameworks that are witnessed in literature. By and large, our findings revealed that there is a lack of standard metrics to verify the quality of decomposition. A few migration techniques compared their results against the quality metrics of coupling and cohesion. Other quality metrics like performance and accuracy are also used in few articles. The last question talks about tools available to support refactoring a monolithic application. We have described various tools that have been used to automate the static and dynamic migration approaches. This comprehensive list of tools will assist system architects in planning for existing system migration. To conclude, the findings of our research and proposed model are useful to practitioners and system architects to avoid experiencing the migration challenges already discussed by researchers.

References

[1] Taibi, D., & Systä, K. (2019). From Monolithic Systems to Microservices: A Decomposition Framework based on Process Mining. In 8th International Conference on Cloud Computing and Services Science, CLOSER.

[2] Mazlami, G., Cito, J., & Leitner, P. (2017, June). Extraction of microservices from monolithic software architectures. In 2017 IEEE International Conference on Web Services (ICWS) (pp. 524–531). IEEE.

[3] Chen, R., Li, S., & Li, Z. (2017, December). From monolith to microservices: a dataflow-driven approach. In 2017 24th Asia-Pacific Software Engineering Conference (APSEC) (pp. 466–475). IEEE.

[4] Li, S., Zhang, H., Jia, Z., Li, Z., Zhang, C., Li, J., ... & Shan, Z. (2019). A dataflow-driven approach to identifying microservices from monolithic applications. Journal of Systems and Software, 157, 110380.

[5] Amiri, M. J. (2018, July). Object-aware Identification of Microservices. In 2018 IEEE International Conference on Services Computing (SCC) (pp. 253–256). IEEE.

[6] Abdullah, M., Iqbal, W., & Erradi, A. (2019). Unsupervised learning approach for web application auto-decomposition into microservices. Journal of Systems and Software, 151, 243–257.

[7] Fan, C. Y., & Ma, S. P. (2017, June). Migrating monolithic mobile application to microservice architecture: An experiment report. In 2017 IEEE International Conference on AI & Mobile Services (AIMS) (pp. 109-112). IEEE.

[8] Levcovitz, A., Terra, R., & Valente, M. T. (2016). Towards a technique for extracting microservices from monolithic enterprise systems. arXiv preprint arXiv:1605.03175.

[9] Mustafa, O., & Gómez, J. M. (2017). Optimizing economics of microservices by planning for granularity level. Experience Report.

[10] Ahmadvand, M., & Ibrahim, A. (2016, September). Requirements reconciliation for scalable and secure microservice (de) composition. In 2016 IEEE 24th International Requirements Engineering Conference Workshops (REW) (pp. 68–73). IEEE.

[11] Ren, Z., Wang, W., Wu, G., Gao, C., Chen, W., Wei, J., & Huang, T. (2018, September). Migrating Web Applications from Monolithic Structure to Microservices Architecture. In Proceedings of the Tenth Asia-Pacific Symposium on Internetware (p. 7). ACM.

[12] Eski, S., & Buzluca, F. (2018, May). An automatic extraction approach: transition to microservices architecture from monolithic application. In Proceedings of the 19th International Conference on Agile Software Development: Companion (p. 25). ACM.

[13] Selmadji, A., Seriai, A. D., Bouziane, H. L., Dony, C., & Mahamane, R. O. (2018, September). Re-architecting OO Software into Microservices. In European Conference on Service-Oriented and Cloud Computing (pp. 65–73). Springer, Cham.

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

[15] Baresi, L., Garriga, M., & De Renzis, A. (2017, September). Microservices identification through interface analysis. In European Conference on Service-Oriented and Cloud Computing (pp. 19–33). Springer, Cham.

[16] UU, Z. L., Korpershoek, M., & VU, A. O. Towards a MicroServices Architecture for Clouds.

[17] Sayara, A., Towhid, M. S., & Hossain, M. S. (2017, December). A probabilistic approach for obtaining an optimized number of services using weighted matrix and multidimensional scaling. In *2017 20th International Conference of Computer and Information Technology (ICCIT)* (pp. 1–6). IEEE.

[18] Nakazawa, R., Ueda, T., Enoki, M., & Horii, H. (2018, September). Visualization tool for designing microservices with the monolith-first approach. In *2018 IEEE Working Conference on Software Visualization (VISSOFT)* (pp. 32–42). IEEE.

[19] Krause, A., Zirkelbach, C., Hasselbring, W., Lenga, S., & Kröger, D. (2020, March). Microservice Decomposition via Static and Dynamic Analysis of the Monolith. In *2020 IEEE International Conference on Software* Architecture Companion (ICSA-C) (pp. 9–16). IEEE.

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

[21] Stojanovic, T. D., Lazarevic, S. D., Milic, M., & Antovic, I. (2020, February). Identifying microservices using structured system analysis. In 2020 24th International Conference on Information Technology (IT) (pp. 1–4). IEEE.

[22] Keele, S. (2007). Guidelines for performing systematic literature reviews in software engineering (Vol. 5). Technical report, Ver. 2.3 EBSE Technical Report. EBSE.

[23] Taibi, D., Lenarduzzi, V., Pahl, C., & Janes, A. (2017, May). Microservices in agile software development: a workshop-based study into issues, advantages, and disadvantages. In Proceedings of the XP2017 Scientific Workshops (p. 23). ACM.

[24] Taibi, D., & Systä, K. (2019). A Decomposition and Metric-Based Evaluation Framework for Microservices. arXiv preprint arXiv:1908.08513.
[25] Garriga, M. (2017, September). Towards a taxonomy of microservices architectures. In International Conference on Software Engineering and Formal Methods (pp. 203–218). Springer, Cham.
[26] Dehghani, Z. (2018). How to break a Monolith into Microservices.
[27] J. Lewis and M. Fowler, "Microservices," 2014. http://martinfowler.com/articles/microservices.html. Accessed on 13th March 2020
[28] Fritzsch, J., Bogner, J., Zimmermann, A., & Wagner, S. (2018, March). From monolith to microservices: a classification of refactoring approaches. In International Workshop on Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment (pp. 128–141). Springer, Cham.
[29] Ponce, F., Márquez, G., & Astudillo, H. (2019, November). Migrating from monolithic architecture to microservices: A Rapid Review. In 2019 38th International Conference of the Chilean Computer Science Society (SCCC) (pp. 1–7). IEEE.
[30] Kazanavièius, J., & Mažeika, D. (2019, April). Migrating legacy software to microservices architecture. In 2019 Open Conference of Electrical, Electronic and Information Sciences (eStream) (pp. 1–5). IEEE.
[31] Pedreira, O., Silva-Coira, F., Places, Á. S., Luaces, M. R., & Folgueira, L. G. (2019). Applying Feature-Oriented Software Development in SaaS Systems: Real Experience, Measurements, and Findings. Journal of Web Engineering, 18(4), 447–476.
[32] https://martinfowler.com/books/refactoring.html, Access on 21.5.2020
[33] Schmidt, R. A., & Thiry, M. (2020, June). Microservices identification strategies: A review focused on Model-Driven Engineering and Domain Driven Design approaches. In *2020 15th Iberian Conference on Information Systems and Technologies (CISTI)* (pp. 1–6). IEEE.
[34] Di Francesco, P., Lago, P., & Malavolta, I. (2019). Architecting with microservices: A systematic mapping study. Journal of Systems and Software, 150, 77–97.
[35] Di Francesco, P., Lago, P., & Malavolta, I. (2018, April). Migrating towards microservice architectures: an industrial survey. In 2018 IEEE International Conference on Software Architecture (ICSA) (pp. 29–2909). IEEE.
[36] Taib, D., Lenarduzzi, V., & Pahl, C. (2018). Architectural patterns for microservices: a systematic mapping study. SCITEPRESS.

[37] https://structure101.com/legacy/structural-analysis/ Accessed on 18.11.2020
[38] http://perfinsp.sourceforge.net/jprof.html#Overview Accessed on 18.11.2020
[39] https://www.hello2morrow.com/products/sonargraph/architect9 Accessed on 18.11.2020
[40] http://wala.sourceforge.net/wiki/index.php/Main_Page Accessed on 18.11.2020
[41] http://kieker-monitoring.net/ Accessed on 18.11.2020
[42] https://github.com/ExplorViz/docs/wiki#quick-start-guides Accessed on 18.11.2020
[43] https://www.elastic.co/guide/en/apm/get-started/current/overview.html Accessed on 18.11.2020
[44] https://fluxicon.com/disco/ Accessed on 18.11.2020
[45] https://github.com/dbeaver/dbeaver/wiki Accessed on 18.11.2020

Biographies

Deepali Bajaj has over 14 years of teaching experience as Assistant Professor in Department of Computer Science, Shaheed Rajguru College of Applied Sciences for Women (University of Delhi). She is currently doing her research in the area of Cloud and Distributed Computing. Her key research areas are Microservices and Function as a service (FaaS) of serverless technology. She has authored several national and international research publications.

Urmil Bharti has over 14 years of teaching experience as Assistant Professor in Department of Computer Science, Shaheed Rajguru College of Applied Sciences for women (University of Delhi). Earlier she has more than 10 years of industry experience. Her last designation was Senior Quality Analyst. She is currently doing her research in the area of Cloud and Distributed Computing. Her key research area is open source serverless frameworks. She has authored several national and international research publications.

Anita Goel is an Associate Professor in Department of Computer Science, Dyal Singh College, University of Delhi, India. She has received her Ph.D. in Computer Science and Masters in Computer Applications from Jamia Millia Islamia and Department of Computer Science (University of Delhi), respectively. She has a work experience of more than 30 years. She is a visiting faculty to Delhi Technological University and NIIT University. From 2009–10, she was Fellow in Computer Science, at Institute of Life Long Learning (ILLL) in University of Delhi. She has served as member of program committee of International conferences like IEEE BigData Congress 2015 and ICWI 2015. She has guided several students for their doctoral studies and has travelled internationally to present research papers. She has authored books in Computer Science and has several national and international research publications.

S. C. Gupta is B.Tech (EE) from IIT Delhi and has worked at Computer Group at Tata Institute of Fundamental Research and NCSDCT (now C-DAC Mumbai), Till recently, he worked as Deputy Director General, Scientist-G and Head of Training at National Informatics Centre, New Delhi and was responsible for keeping its 3000 scientists/ engineers up to date in various technologies. He has extensive experience in design and development of large Complex Software Systems. Currently he is a Visiting Faculty at Dept of Computer Science and Engineering, IIT Delhi. His research interests includes Software Engineering, Data Bases and Cloud Computing. He has been teaching Cloud Computing at IIT Delhi, which includes emerging disruptive technologies like SDN and SDS. He has guided many M.Tech & Ph.D. Research students in these technologies and has many publications in Software Engineering and Cloud Technology in National and International Conferences and Journals.