

ULRR

Decomposition of monolith applications into microservices architectures: a systematic review

| | |
|---------------|---|
| Item Type | Article |
| Authors | Abgaz, Yalemisew;McCarren, Andrew;Elger, Peter;Solan, David;Lapuz, Neil;Bivol, Marin;Jackson, Glenn;Yilmaz, Murat;Buckley, Jim;Clarke, Paul |
| Citation | IEEE Transactions on Software Engineering, 2023, 49 (8), pp. 4213-4242 |
| Publisher | Institute of Electrical and Electronics Engineers |
| Download date | 2026-06-09 00:17:52 |
| Item License | https://creativecommons.org/licenses/by-nc-sa/4.0/ |
| Link to Item | https://doi.org/10.34961/researchrepository-ul.25334518 |

Decomposition of Monolith Applications Into Microservices Architectures: A Systematic Review

Yalemisew Abgaz , Andrew McCarren , Peter Elger, David Solan , Neil Lapuz , Marin Bivol,
Glenn Jackson , Murat Yilmaz , Jim Buckley , and Paul Clarke 

(Survey Paper)

Abstract—Microservices architecture has gained significant traction, in part owing to its potential to deliver scalable, robust, agile, and failure-resilient software products. Consequently, many companies that use large and complex software systems are actively looking for automated solutions to decompose their monolith applications into microservices. This paper rigorously examines 35 research papers selected from well-known databases using a Systematic Literature Review (SLR) protocol and snowballing method, extracting data to answer the research questions, and presents the following four contributions. First, the Monolith to Microservices Decomposition Framework (M2MDF) which identifies the major phases and key elements of decomposition. Second, a detailed analysis of existing decomposition approaches, tools and methods. Third, we identify the metrics and datasets used to evaluate and validate monolith to microservice decomposition processes. Fourth, we propose areas for future research. Overall, the findings suggest that monolith decomposition into microservices remains at an early stage and there is an absence of methods for combining static, dynamic, and evolutionary data. Insufficient tool support is also in evidence. Furthermore, standardised metrics, datasets, and baselines have yet to be established. These findings can assist practitioners seeking to understand the various dimensions of monolith decomposition and the community’s current capabilities

in that endeavour. The findings are also of value to researchers looking to identify areas to further extend research in the monolith decomposition space.

Index Terms—Monolith application decomposition, monolith to microservices migration, microservices architecture, microservices identification, static analysis, dynamic analysis.

I. INTRODUCTION

WITH the passage of time, successful software systems grow large and become complex, due to the addition of a plethora of functionalities resulting in highly coupled but less cohesive components [1], [2]. In these large and complex systems, monolith architectures [3] embody the centralisation of functionality in large individual components, giving rise to inherent limitations in terms of scalability, maintenance and deployment performance [1], [4], [5], [6], [7], [8]. In contrast, microservice architectures are distributed, favouring the decomposition of systems into various relatively small and independent components that may be invoked as required [9], [10], delivering benefits in areas such as increased scalability and improved deployment frequency [2], [4], [11], [12].

Prior to the introduction of microservice architectures, monolith architectures were commonly adopted. However, as software systems continued to grow in size and with demand growing for ever-faster release cycles, the need for partitioning systems into separately compilable services came to the fore. Also, with the advent of cloud-based infrastructure innovations such as Software-as-a-Service [13] and Function-as-a-Service [14], the relative benefits of monolith architectures have been reduced [15]. As a result, there is growing interest in microservice architectures [16], [17].

For companies with existing monolith-based systems, a particular challenge is the decomposition of these systems into coherent microservice-based implementations. This decomposition is sometimes focused on supporting migration engineers in the identification of microservice candidates by analysing the application’s domain [18], [19], [20]. Other techniques involve analysing the source code [21], [22], [23], [24], execution traces [25], and version related information [26], [27].

While a great deal of important work has been conducted in the decomposition space to date, the existing published material tends to focus on addressing a specific scenario, domain, or

Manuscript received 15 February 2023; revised 5 May 2023; accepted 11 June 2023. Date of publication 23 June 2023; date of current version 15 August 2023. This work was supported in part by the Department of Enterprise, Trade and Employment, Ireland (<https://enterprise.gov.ie/en/>) under the Disruptive Technologies Innovation Fund under Grant DTIF DT20180116, and in part by SFI, Science Foundation Ireland (<https://www.sfi.ie/>) under Grant SFI 13/RC/2094_P2 to Lero - the Science Foundation Ireland Research Centre for Software. Recommended for acceptance by F. Ferrucci. (Corresponding author: Yalemisew Abgaz.)

Yalemisew Abgaz, Neil Lapuz, Glenn Jackson, and Paul Clarke are with the School of Computing, SFI Research Centre for Software, Dublin City University and Lero, D09 N920 Dublin, Ireland (e-mail: yalemisewm.abgaz@dcu.ie; Neil.lapuz2@mail.dcu.ie; gjackson@live.ie; paul.m.clarke@dcu.ie).

Andrew McCarren is with Dublin City University and Insight, the SFI Research Centre for Data Analytics, School of Computing, Dublin City University, D09 N920 Dublin, Ireland (e-mail: andrew.mccarren@dcu.ie).

Peter Elger and Marin Bivol are with fourTheorem Limited, The Rubicon, Cork Institute of Technology, T12 Y275 Cork, Ireland (e-mail: peter.elger@fourtheorem.com; marin.bivol@fourtheorem.com).

David Solan is with FINEOS Corporation Limited, D03 FT97 Dublin, Ireland (e-mail: david.solan@fineos.com).

Murat Yilmaz is with the Faculty of Engineering, Department of Computer Engineering, Gazi University, 06560 Ankara, Turkey (e-mail: my@gazi.edu.tr).

Jim Buckley is with the Lero/Department of CSIS, University of Limerick, V94 T9PX Limerick, Ireland (e-mail: Jim.Buckley@lero.ie).

This article has supplementary downloadable material available at <https://doi.org/10.1109/TSE.2023.3287297>, provided by the authors.

Digital Object Identifier 10.1109/TSE.2023.3287297

programming language. We present a consolidation of the innovations to date, and we introduce a framework as a means to organise and classify the key methods and approaches that may be adopted when undertaking decomposition projects. To do so, this research adopts a Systematic Literature Review (SLR) [28], [29], [30] and snowballing to address the following research objective: *to systematically identify and organise existing research on the decomposition of monolith applications into microservices*. To this end, this research initially identifies 5022 search results from major computer science literature databases and applies four rigorous refinement steps to identify 33 research papers. After applying a snowballing method, this research finally examines 35 research articles.

Various existing perspectives are incorporated into this research, including, at an overview level, contributions seeking to define migration road maps [11], [31], works reporting on migration problems and challenges [2], [32], [33], [34], [35], and works outlining differing migration processes [36], [37]. Examples of more specifically targeted work include works directed at the containerisation of existing monoliths [38], at different migration patterns [4], [39], and at differing migration approaches [40], [41], [42], [43].

Central to our findings is the observation that monolith decomposition into microservices is a complicated and expensive task. Many different approaches have been proposed, each of which may be used in isolation or combined for greater effect, and as demanded by the context. This suggests that automated or semi-automated tool support is advantageous and therefore, various existing tooling is also identified and analysed in this review. More importantly, it suggests the need for an organising framework for the area and consequently, the Monolith To Microservices Decomposition Framework (M2MDF) introduced in this research provides a comprehensive map for researchers and practitioners to navigate this complicated landscape.

This research concludes that although monolith decomposition has gained more attention in recent years, it remains at a preliminary stage for the following reasons: (i) the lack of integrated, comprehensive data collection and analysis methods with regard to crucial aspects of the monolith application, (ii) the lack of comparison between various decomposition methods and algorithms, (iii) the lack of validated and widely accepted metrics and benchmarks to measure the quality of resulting microservice candidates, and (iv) the shortage of integrated tools to support the various stages of microservices extraction pipelines.

The remaining sections of this paper are organised as follows. The background and related work section (Section II) introduces monolith to microservices decomposition, its challenges, and related literature. Section III presents the Systematic Literature Review (SLR) methodology employed along with a description of the analyses performed. In Section IV, the M2MDF is presented along with the detailed results of the SLR. Section V identifies existing gaps in the field, with Section VI discussing acknowledged threats to validity. Section VII presents a summary of the work in the context of specific implications for practitioners and researchers, and includes a post-review reflection that evaluates the utility of the proposed M2MDF by examining literature published following the primary review

timeframe. Section VIII presents concluding remarks and proposed future research directions.

II. BACKGROUND AND RELATED WORK

A. Monolith versus Microservices Architectures

An application based on a monolith architecture is recognised as one which combines all (or many of) its modules into large components that are self-contained and independent from other applications [1]. Monolith architecture brings the benefit of localising significant portions of application functionality in a single manageable space but, as it becomes overly large and complex, it can inhibit maintenance and deployment flexibility [44]. Whenever a module in a monolith changes, an entire application may need to be reintegrated, rebuilt, retested, and restarted upon deployment [1]. Moreover, the scalability of monolith applications may introduce significant challenge as application usage grows due to size and load distribution [45].

Microservices architectures are presented as an alternative to monolith architectures. They are organised around business functions that perform a single task and maintain their data in a decentralised manner. They are built on the principles of single responsibility, high cohesion, low coupling, scalability, deployability, and low perturbation (meaning, minimal disturbance to other microservices that comprise the software system) [46], [47], [48], based on components that can be evolved and deployed independently [12], [49].

B. Monolith to Microservices Decomposition

Many companies have chosen to continue with monolith implementations, as they have large existing monolith investments and limited knowledge of the complex process required to decompose their monoliths into microservices [50].

But, even for the the large proportion of organisations embracing microservices, manual decomposition of the monolith application is a challenging task [2] and may require existing application experts to devote a considerable volume of time to the tedious task of profiling and fully understanding the intricacies of the monolith code. As a complementary or alternative approach, automated code analysis tools have been used to identify service boundaries within a monolith. But, as we shall outline in Section IV, research dealing with automated system analysis and microservices identification is still somewhat in its infancy (although growing steadily).

C. Existing Literature Reviews on the Decomposition of Monolith Applications to Microservices

Very recently, the amount of secondary research on the decomposition of monolith applications to microservices has been growing. The studies investigate a wide range of research questions related to the rationale of the migration [11], the problems and challenges of the migration process [32], microservices analysis [40], [41], and strategies for supporting microservices extraction. Other secondary studies investigate the broader aspects of microservices architecture including the use of microservices architecture in DevOps [51], tools and techniques to support

microservices development [52], microservices granularity [53], and cloud migration processes [35].

Abdellatif et al. [40] conducted a review of 41 studies (2004–2019) to identify the inputs, processes, outputs and usability of service identification approaches used for the modernisation of legacy software systems. The aim of this work was to assist practitioners in the selection of a suitable approach for identifying services. The authors presented a multi-layer taxonomy of existing approaches used in the broader context of service identification and web services whereas, we focus on the approaches used to decompose monolith applications, particularly into microservice architectures. Another distinction is that Abdellatif's study focuses on service identification without a particular focus on microservices, resulting in only three overlapping studies.

Velepucha and Flores [32] conducted an SLR to identify the problems and challenges associated with microservices migrations. To answer their research question, the authors reviewed 37 studies and identified major problems concerning suitable tool selection, team reorganisation, the complete or partial migration decision, microservices identification, and multiple database consistency. The identified problems cover technical, functional, and behavioural aspects of the migration process. Although this study provides practitioners with crucial information as to what problems and challenges they should expect in the migration process, it does not provide a detailed overview of the availability and usability of decomposition approaches and tools that are in use.

The question of why and how monolith legacy systems are migrated to microservices is investigated by Wolfart et al. [11]. The authors identified 11 migration driving forces: optimised scalability, independent and automated deployment, easier maintenance and evolution, independence of teams, loosely coupled services, cohesive services, technology flexibility, infrastructure facilities, agility enabling, easier reuse, and reduced time. The authors furthermore categorised the modernisation process into eight activities organised into four phases: initiation, planning, execution, and monitoring. Bushong et al. [41] also investigated the practice of analysing microservices architecture focusing on analysis and debugging methods used for microservice systems. The most relevant aspect of this research is the investigation of the practice of migration of monolith applications into microservices, but with less emphasis on the decomposition techniques.

A rapid review has been conducted on the migration from a monolith architecture to microservices by Ponce et al. [33] with the aim of gathering, organising, and analysing migration techniques. The review, which comprised of 20 research articles, investigated available migration techniques, the types of systems the techniques are applied to, the validation methods used, and the challenges faced. Three migration techniques were identified: Model-Driven (MD), Static Analysis (SA), and Dynamic Analysis (DA), with 70% of the applications being web-based and more than 90% of the applications being Java-based. As part of the review, the paper also discovered that case studies are the most widely used validation method, followed by experiments and examples. This review is informative and related to the focus of our research, however, the paper acknowledges that while

rapid reviewing provides a lightweight methodology to analyse the migration process [54], it can result in reduced coverage.

Complementary to the above rapid review [33], a systematic review was conducted to identify and classify existing refactoring approaches in the context of monolith application decomposition, by comparing 10 existing studies [44]. The paper classifies the studies using static code analysis aided (SCA), meta-data aided (MDA), workload-data aided (WDA), and dynamic microservices composition (DMC) approaches. The authors of the review evaluated the selected methods using parameters including granularity, input and output types, result evaluation, tool support, and validation. The review outlines existing approaches using a decision guide but does not cover the microservices decomposition methods. In addition, the review has a broad scope since it includes greenfield microservices development as well as monolith application decomposition. A re-examination of the findings of this review is crucial because of its scale and because a considerable number of new studies on the topic have been published recently: significant growing interest has been observed over the past three years with an influx of research studies examining the use of novel tools and techniques to assist with the migration problem.

A systematic mapping study of microservices construction was conducted in [55], where 103 primary studies were examined to identify, classify, and evaluate the state-of-the-art microservices architecture solutions. Although related to our research, this particular mapping study does not focus directly on decomposition projects. Kazanavičius and Mažeika [56] also conducted a literature review on the migration of legacy software to microservices architecture to understand the techniques employed and the challenges faced. The paper studied the benefits and drawbacks of certain earlier studies and presented some interesting comparisons between refactoring and rebuilding decisions. However, it is not directed at decomposition specifically, is limited to only six earlier studies and the SLR methodology used to select the studies is not discussed in detail.

An overview of the lessons learned and the associated difficulties/challenges of the migration process is investigated in [42]. The review focuses on papers discussing migration difficulties. However, only five papers emerge from the SLR, with a further seven papers based on the suggestion of the authors. Although this study reports the challenges, it does not cover aspects of the migration process such as data collection, analysis, decomposition, and evaluation.

Additional studies covered related aspects of the migration process. An earlier study investigated the importance of variability (the adaptability of a system for a particular context) for supporting the extraction of microservices from monolith legacy systems in industry [37]. Cojocaru et al. [57] further studied the attributes used to assess the quality of microservices that are derived from monolith applications, proposing minimal indicators for use in evaluating the quality of microservices and Service-Oriented Architecture (SOA).

As a summary, we extracted the research questions and the broader topics of the existing review papers (in reverse chronological order) in Table I. This is helpful, as it reifies the focus of these related earlier works. It also helps to focus our research. Extending earlier works, this paper conducts a detailed review that

TABLE I
RESEARCH QUESTIONS OF EXISTING LITERATURE REVIEWS

| Paper | Research Question(s) | Primary Topic | |
|-------|---|-----------------|-------------------------|
| [32] | What problems and challenges are there for the migration process of monolithic applications to microservices? | Migration: | Problems & challenges |
| [40] | (i) What are the inputs used by Service Identification Approaches (SIA)?, (ii) What are the processes followed by SIA?, (iii) What are the outputs of SIAs?, and (iv) What is the usability of SIAs? | Migration: | Microservice Extraction |
| [11] | Why and how are monolithic legacy systems migrated to microservices? | Migration: | Microservice Extraction |
| [41] | (i) What methods and techniques are used in microservice analysis?, (ii) What are the problems or opportunities that are addressed using microservice analysis techniques?, (iii) Does microservice analysis overlap with other areas of software analysis, or are new methods or paradigms needed?, and (iv) What potential future research directions are open in the area of microservice analysis? | Migration: | Microservice Extraction |
| [53] | (i) What are the activities undertaken to adopt microservices?, (ii) What are the modelling approaches used to define the granularity of a microservice?, (iii) Which quality attributes are considered when reasoning about microservice granularity and how are they captured?, and (iv) How is reasoning about microservice granularity described? | Microservices | Architecture |
| [33] | (i) What are the migration techniques proposed in the literature?, (ii) In what types of systems have the proposed techniques been applied?, (iii) What type of validation do the authors of the techniques use?, and (iv) Are there challenges associated with migration from monolith to microservices? | Migration: | Microservice Extraction |
| [44] | (i) What are existing architectural refactoring approaches in the context of decomposing a monolithic application architecture into Microservices? and (ii) How can they be classified with regards to the techniques and strategies used? | Migration: | Microservice Extraction |
| [42] | (i) Which strategies have been reported in the literature to support the migration of legacy software systems to microservices-based architecture? and (ii) Which lessons learned have been reported in the literature regarding challenges and advantages perceived as a consequence of the aforementioned migration? | Migration: | Microservice Extraction |
| [51] | (i) What is the frequency and type of published research on MSA in DevOps?, (ii) What are the existing research themes on MSA in DevOps and how can they be classified and mapped?, (iii) What problems have been reported when implementing MSA in DevOps?, (iv) What solutions have been employed to address the problems?, (v) What challenges have been reported when implementing MSA in DevOps?, (vi) What methods are used to describe MSA in DevOps?, (vii) What MSA design patterns are used in DevOps?, (viii) What quality attributes are affected when employing MSA in DevOps?, (ix) What tools are available to support MSA in DevOps?, and (x) What are the application domains that employ MSA in DevOps? | Microservices | Architecture |
| [55] | (i) What are the publication trends of research studies about architecting with microservices?, (ii) What is the focus of research on architecting with microservices, and (iii) What is the potential for industrial adoption of existing research on architecting with microservices? | Microservices | Architecture |
| [52] | (i) What type of research is conducted on microservices?, (ii) What are the main practical motivations behind microservices related research?, and (iii) What are the emerging standards and de facto tools on microservices solutions? | Microservices | Architecture |
| [58] | (i) What are the main practical motivations behind using microservices?, (ii) What are the existing methods, techniques and tool support to enable microservice architecture development and operation?, and (iii) What are the existing research issues and What should be the future research agenda? | Microservices | Architecture |
| [35] | (i) What are the existing approaches proposing a migration model for moving legacy applications to cloud environments in the literature?, (ii) What is the current state of these approaches w.r.t. evaluation?, (iii) What generic criteria, as typically expected for a software development methodology, are supported by these approaches?, and (iv) What cloud-specific criteria are supported by these approaches? | Cloud Migration | |

includes the different phases of the decomposition of monolith applications into microservices. It extends the findings of the existing reviews and draws a comprehensive and interrelated picture of the various decomposition techniques, benchmarks, and metrics, identifying gaps and outlining future directions. Although several SLRs are presented on the migration of monolith applications in general and the decomposition to microservices in particular, there is no single SLR that fully address our research objective (and consequently our research questions).

III. METHODOLOGY

This paper follows the three-phase literature survey process (planning, reviewing & reporting) proposed in [28], [29], [30]. The planning phase includes identifying the need for a review and the development of a review protocol. It is described in Section III-A The review phase includes selection of the primary studies, assessment of the studies, data extraction, and data synthesis. It is detailed in Sections III-B and III-C. Finally, the reporting phase focuses on documenting the review, which includes document observation, and the reporting of results. It is described in Sections IV and V.

A. Planning the Survey

The planning phase examined the research motivation, ultimately leading to the development of research questions.

1) *Identifying the Need for an SLR*: Of itself, the absence of comprehensive and up-to-date secondary research examining monolith-to-microservices decomposition highlights the need for a comprehensive SLR. Although there are rapid and partial reviews of the area, their focus is limited to specific aspects of the migration and they lack an in-depth analysis and organisation of the methods and algorithms. The majority tend not to address benchmarking or report on the benchmarking employed. The searches used in this study have not discovered any comprehensive secondary research that addresses all these aspects and, consequently the research questions in Section III-A2 were derived. We believe that answering these questions will significantly contribute towards consolidating and supplementing existing research with rich analysis.

Furthermore, evidence from earlier research studies indicates a demand for research of this type, that the topic is of interest to academic and industrial practitioners, and that the body of related literature is growing [55]. Our own initial searches

confirmed this to be the case and established that despite growth in the volume of publications year-on-year, there was no systematic, comprehensive, and robust evaluation of the state of the art focusing on a holistic view of monolith to microservices decomposition in recent years. It is this position that fundamentally motivates the SLR presented in this paper.

2) *Specifying the Research Questions*: When conducting an SLR, it is crucial to identify relevant research questions with the capacity to deliver unambiguous answers [28]. We identified four such research questions in this area:

RQ1 What are the primary phases of monolith-to-microservices decomposition and the major constituent elements of those phases?

RQ2 What are the existing approaches, tools and methods observed in the decomposition of monolith applications into microservices?

RQ3 What are the metrics, datasets, and benchmarks used for evaluating and validating monolith decomposition into microservices?

RQ4 What research gaps can be identified in the current literature?

3) *Defining and Evaluating the Review Protocol*: This work has been conducted in the context of the Future Software Systems Architecture (FSSA) project based at Dublin City University and Lero, the Science Foundation Ireland Research Centre for Software. Formal industrial collaborators include the FINEOS Corporation and fourTheorem Limited. The review process was led by the first author who prepared the SLR protocol by selecting the topics and the search strings. The protocol, as presented below, was internally evaluated by a team of seven researchers who are members of the FSSA project working in the area of monolith code migration as well as two expert practitioners in the microservices industry. The protocol was applied iteratively and, at each iteration, the scope, inclusion/exclusion criteria, and the search terms were revised as appropriate to address the research questions.

B. Selection of Primary Studies

Guided by the research questions, initial terms representing the research topic were extracted. We identified three main topics and built the search terms around these topics: Monolith, Microservices, and Decomposition. To formulate the search keywords under each topic, we further considered the previous literature reviews in the area. Using synonyms and related terms, each of the topics was expanded to include additional search terms as follows: (i) monolith, existing, and legacy, (ii) microservice and micro-service, and (iii) decomposition, migration, identification, extraction, refactoring, modularisation, transformation, transition, and conversion. The keywords under the three main topics were combined using the Boolean 'OR' and 'AND' operators along with a wildcard(*) representation of the keywords to ensure a higher recall. The topics and search strings were reviewed by the third and last authors.

The final search string is represented as: *(monolith* OR exist* OR legacy) AND (microservice* OR micro-service*) AND (decompos* OR migrat* OR identif* OR extract* OR refactor* OR modular* OR transform* OR transit* OR conver*)*. Major

platforms such as IEEE Xplore and ACM Digital Library only support a fixed number of Boolean operators and wildcards, effectively forcing splitting the search string into three queries: (i) *(monolith* OR existing OR legacy) AND (microservice* OR micro-service*) AND (decompos* OR migrat* OR identif*)*, (ii) *(monolith* OR existing OR legacy) AND (microservice* OR micro-service*) AND (extract* OR refactor* OR modular*)*, and (iii) *(monolith* OR existing OR legacy) AND (microservice* OR micro-service*) AND (transform* OR transit* OR conver*)*. After removing duplicate studies, the results from the three queries were combined.

1) *Initial Search*: We searched on the established platforms for the publication of robust peer-reviewed computer software engineering research, including IEEE Xplore, ACM Digital Library, Science Direct, SpringerLink, Wiley Online, and Scopus. Where possible, we conducted an advanced search on all metadata available on these platforms. Google Scholar has been used to promote the retrieval of recent papers that may not yet be indexed on the established publication platforms. The search was conducted on all platforms by the first author in collaboration with the fifth and sixth authors.

We also adopted *Publish or Perish*¹ to harvest and organise the metadata, which is also used as an additional sanity check to ensure the discovery of relevant papers. The last search on the platforms was conducted on the 28th of October, 2021. We restricted the search to peer-reviewed scientific publications found in journals, conferences, and workshops between 2015 and 2021 inclusive, as microservices literature only started to gain significant momentum from 2015 on [52], [59]. A total of 5022 studies were initially retrieved.

2) *Refinement*: The screening of the studies was conducted by applying general criteria for the exclusion of studies on the search results. To refine the studies, we applied filtering based on: (i) the title, (ii) the title, abstract and conclusion, and (iii) full-text analysis, by applying the inclusion and exclusion criteria. Studies were screened by applying the following exclusion criteria (EC): (EC1) Duplicate Study, (EC2) Books and Patents, (EC3) Non-peer-reviewed Study, (EC4) Secondary Study, (EC5) Study written in languages other than English, and (EC6) Study published before 2015. Note that there is an additional step at the end of the process to protect against oversight of significant pre-2015 published material (details of which are provided at the end of this subsection). Further studies that did not satisfy the inclusion criteria were removed. The three inclusion criteria (IC) are: (IC1) the primary objective of the study should be the decomposition of monolith applications into microservices, (IC2) the study should include structured and preferably automatic or semi-automatic decomposition approaches, and (IC3) the study should sufficiently describe the decomposition method, code, algorithm, and its evaluation (i.e., abstracts and extended abstracts are not included).

Refinement Step 1: By merging the results obtained from the search platforms, we automatically removed duplicate studies, incomplete data, books, websites, and reports which resulted in 968 studies. *Refinement Step 2*: Refinement step 2 was conducted by inspecting the title of the studies. The main focus in this

¹<https://harzing.com/resources/publish-or-perish/>

step is to further refine studies that do not satisfy the inclusion and exclusion criteria. This step resulted in 126 studies. In circumstances where it was not possible to decide based only on the title, the studies are promoted to the next refinement step. *Refinement Step 3*: The studies that passed refinement step 2 are further refined by carefully inspecting the abstract and the conclusion sections. After the application of the criteria, 92 studies passed to the full-text review. Refinement step 2 and Refinement step 3 were conducted by the first author as they involve applying objective exclusion criteria using the collected metadata and because this was an initial filtering only, to be refined by further group-wise filtering later on.

Refinement Step 4: Next, a full-text review of the studies was conducted against the inclusion criteria, resulting in 33 selected papers. Refinement step 4 which is based on the full-text analysis, was conducted by the wider team strictly applying the inclusion and exclusion criteria by organising a series of weekly literature review presentations with the third, fifth, sixth, and seventh author. Specifically, the first and the fifth authors jointly participated in the full-text screening by focusing particularly on the methodology, experiment and evaluation sections. Their deliberations were presented at a series of literature review meetings conducted jointly with the first, third, sixth and seventh authors to decide the final selection, based on the full consensus reached after these review meetings. These review presentations were venues for critically reflecting on the reviewed studies, filtering studies, identifying gaps, assessing the validity of proposed methods fostering theoretical debates and even triggering replication of selected proposed methods. Additional high-level review meetings were conducted to further scrutinise the M2MDF framework and the associated analysis results, which have been conducted by the first, second, third, fourth, and last authors. The details of the studies following each refinement step are included in the replication package [87].

Snowballing: One forward and backward snowballing iteration [88] was then conducted by the first author, on the 33 papers. Using backward snowballing, we extracted 941 references from the reference section of the studies. Using forward snowballing, we extracted 594 studies citing the 33 selected studies. After extracting the references and citations from the 33 studies, we conducted two major refinement steps each containing additional minor refinement steps discussed as follows. *Refinement Step 5*: We combined the forward and backward snowballing results and filtered out duplicate and incomplete records, along with books, patents, non-peer reviewed studies and publications before 2015 resulting in 212 studies. *Refinement Step 6*: Again, we compared the 212 studies with the 968 studies obtained from the literature search to further refine the studies. Further inclusion and exclusion criteria are applied on the full-text of the studies. As a result, an additional two studies from Springer-Link and the Association for the Advancement of Artificial Intelligence (AAAI) were included in the study. The details of the snowballing process and its results are also included in the replication package [87].

Quality Assessment Criteria. Based on the quality assessment of primary studies proposed in [89], [90], the following quality assessment questions that correspond to the inclusion criteria are adopted to determine the quality of the studies concerning

TABLE II
SUMMARY OF THE SEARCH RESULTS

| Platforms | All Search Results | Selected Studies |
|---------------------|--------------------|------------------|
| SpringerLink | 2247 | 11 |
| IEEE Xplore | 267 | 12 |
| ACM Digital Library | 1191 | 5 |
| Science Direct | 367 | 3 |
| Wiley Online | 461 | 1 |
| Scopus | 489 | 1 |
| Snowballing | | 2 |
| Total | 5022 | 35 |

the objective, method, and coverage of the studies, respectively. Q1) Does the study's primary objective explicitly focus on the decomposition of monolith applications into microservices? Q2) Does the study include structured and preferably automatic or semi-automatic decomposition approaches? Q3) Does the study sufficiently describe the decomposition method, code, algorithm, and evaluation? These questions are implicitly used in the refinement stages. For each candidate study during a particular refinement stage, each question is answered (in the order of its appearance) using a numeric value 0 or 1, where the value 0 indicates that the study does not answer the specific question, and the value 1 indicates that the study answers the question. Studies that answer all three questions are included in the review.

Munir et al. [91] defines quality assessment for SLRs in terms of 'rigor and relevance'. Note that two of our quality assessment guidelines address relevance. But, in this case, 'rigor', referring to the quality of the empirical work performed in the primary studies, is not appropriate as it leads to an assessment of the quality of empirical results that are then synthesized with the 'whole' to address the RQ. Instead our third criteria refers to the primary studies explicitly addressing the component phases, approaches, tools, methods, and (evaluation) metrics, datasets and benchmarks in monolithic application decomposition. This is in line with the stated research questions in this work, but does give a Systematic-Mapping-study feel to this SLR.

An additional, critical review of the overall methodology and research findings has been conducted by the eighth and ninth authors, who are FSSA project Advisory Board Members and have therefore been generally advising on FSSA technical implementations. This final critical review process was iterative in nature and required a number of review cycles and feedback.

Ultimately, 35 relevant studies (see Table II) are considered in this research. The search and snowballing process is summarised in Fig. 1. Note that from this section onward, we refer to the selected studies using their chronological study number as P1, P2, ..., P35 to explicitly highlight and distinguish included studies from other references (refer to Table III).

As all the works included in this study are from leading academic dissemination fora (publishers/conferences/journals), where they have already been peer-reviewed for quality by expert reviewers, we did not see the need to re-iterate with ranking based on quality review for these papers, beyond the quality assessment criteria already performed. It is nevertheless important to emphasise that different sources, even among those subject to peer review, will not present with the same quality level. Indeed,

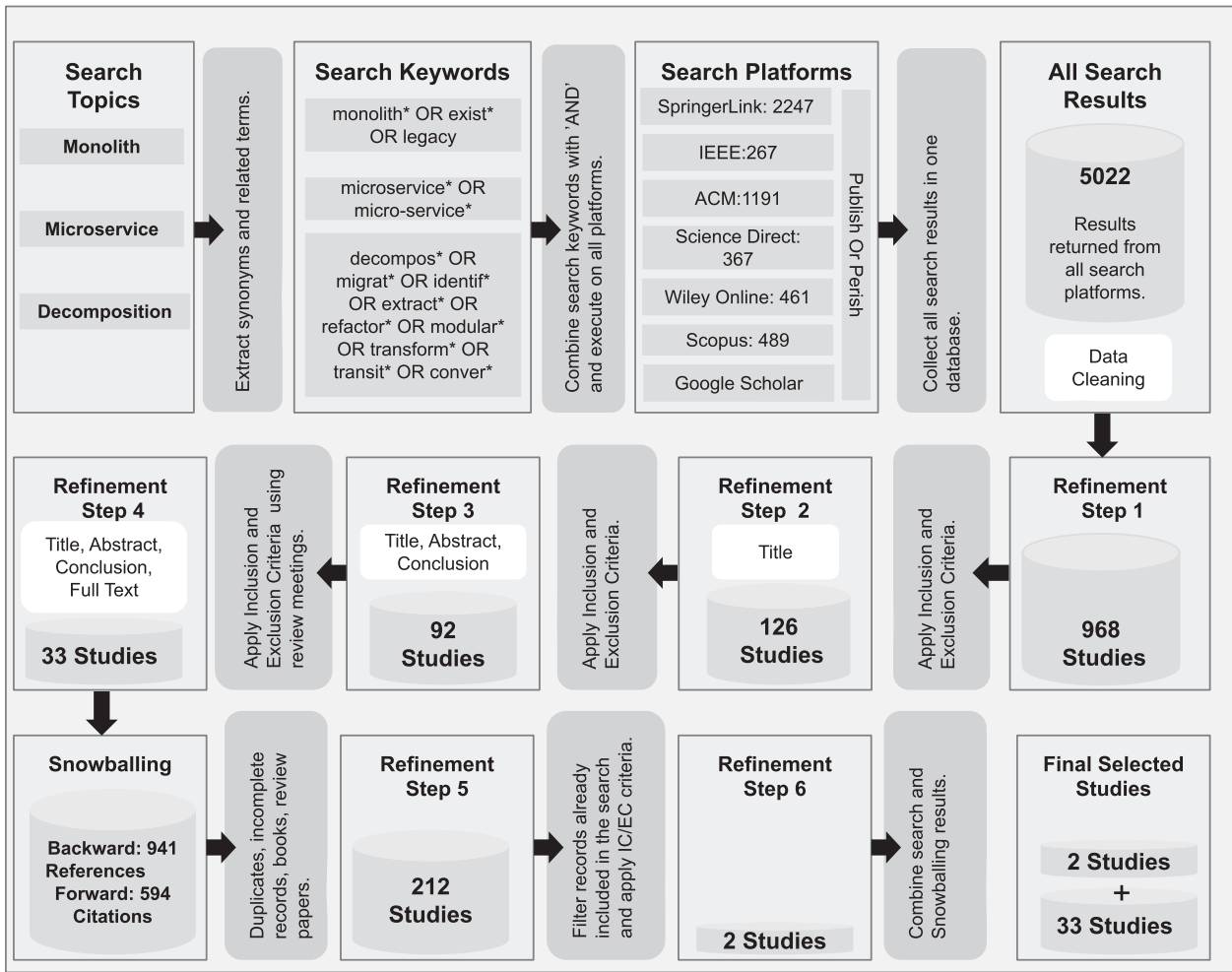


Fig. 1. SLR search and snowballing steps.

it has been observed that there exists no standard definition of quality for software engineering research studies [92].

As a confirmative step, we examined works in the 2011–2014 time frame as a means to reduce the risk that any significant earlier work was not incorporated into this systematic review. The original search string was explored using Google Scholar. A total of 19 additional papers were discovered in this step. Nine studies do not relate to monolith to microservices decomposition at all and were excluded. Five studies are thesis reports, where two reports are written in languages other than English. The five studies are excluded because of EC3. The other three results are book publications excluded due to EC2. Only two studies discuss monolith and microservices in the software engineering context: the first covers microservices development with a brief mention of monolith partitioning [93] and did not satisfy IC2. The second study presents the experiences and lessons learned in incremental migration and refactoring [94]. A revision of this second paper is published in 2016 and it is therefore covered in the 2015–2021 timeline.

C. Analysis of the Data

The analysis of data and the steps (Refer Fig. 2) that are employed to extract the data are discussed in the following sections.

1) *RQ1: Grounded Theory Components:* The M2MDF analysis was primarily based on memoing, coding, constant comparison and theoretical sorting on the selected literature, proposed in [95], [96] as part of a grounded theory (GT). Given the diverse, informal, and diffuse nature of the initial memos and codes, it would be impractical to list those memos and codes in the replication package, and ultimately of limited utility to the reader due to their overwhelming number. In addition, readers interested in replicating the approach in a truly GT fashion should not be guided/constrained by our codes. Hence, instead we present a snippet of a selected paper [P24] with the associated memos and codes, to give the reader a feel for our approach. We wish to highlight that given the nature of the study, it would not have been appropriate to apply a full grounded theory process: GT activities such as theoretical sampling and theoretical saturation, for example, would have little meaning when applied to a pre-defined dataset (the 35 papers).

Obvious and Inviolable Element Identification: It should be noted that certain migration elements were obvious and inviolable (even if they had not been assembled in an end-to-end framework up to this point). For example, it is clear that monolith data collection is required. Analysis of that data, and identification of the resultant microservices are also fundamental migration steps.

TABLE III
SELECTED STUDIES

| ID | Title | Year | Source | Type |
|----------|--|------|----------|------|
| P1 [60] | A Microservice Decomposition Method Through Using Distributed Representation of Source Code | 2021 | WOS | J |
| P2 [61] | A Multi-Criteria Strategy for Redesigning Legacy Features as Microservices: An Industrial Case Study | 2021 | IEEE | C |
| P3 [18] | A Multi-model Based Microservices Identification Approach | 2021 | SD | J |
| P4 [62] | Graph Neural Network to Dilute Outliers for Refactoring Monolith Application | 2021 | ACM | C |
| P5 [63] | Identification of Microservices from Monolithic Applications through Topic Modelling | 2021 | ACM | C |
| P6 [64] | Microservice Remodularisation of Monolithic Enterprise Systems for Embedding in Industrial IoT Networks | 2021 | Springer | C |
| P7 [65] | Migrating Production Monolithic Systems to Microservices Using Aspect Oriented Programming | 2021 | WO | J |
| P8 [66] | Mono2Micro: A Practical and Effective Tool for Decomposing Monolithic Java Applications to Microservices | 2021 | ACM | C |
| P9 [67] | Monolith to Microservice Candidates using Business Functionality Inference | 2021 | IEEE | C |
| P10 [68] | A Decomposition and Metric-Based Evaluation Framework for Microservices | 2020 | Springer | C |
| P11 [69] | A Model-Driven Approach Towards Automatic Migration to Microservices | 2020 | Springer | W |
| P12 [70] | Automated Microservice Identification in Legacy Systems with Functional and Non-Functional Metrics | 2020 | IEEE | C |
| P13 [71] | Determining Microservice Boundaries: A Case Study Using Static and Dynamic Software Analysis | 2020 | Springer | C |
| P14 [72] | Extracting Microservices's Candidates from Monolithic Applications: Interface Analysis and Evaluation Metrics Approach | 2020 | IEEE | C |
| P15 [73] | From Monolithic Architecture Style to Microservice one Based on a Semi-Automatic Approach | 2020 | IEEE | C |
| P16 [74] | Microservice Decomposition via Static and Dynamic Analysis of the Monolith | 2020 | IEEE | C |
| P17 [75] | Partial Migration for Re-architecting a Cloud Native Monolithic Application into Microservices and FaaS | 2020 | Springer | C |
| P18 [76] | Remodularization Analysis for Microservice Discovery Using Syntactic and Semantic Clustering | 2020 | Springer | C |
| P19 [77] | System Decomposition to Optimize Functionality Distribution in Microservices with Rule Based Approach | 2020 | IEEE | C |
| P20 [78] | Transforming Monolithic Systems to Microservices - An Analysis Toolkit for Legacy Code Evaluation | 2020 | IEEE | C |
| P21 [19] | A Dataflow-driven Approach to Identifying Microservices from Monolithic Applications | 2019 | Springer | C |
| P22 [79] | From a Monolith to a Microservices Architecture: An Approach Based on Transactional Contexts | 2019 | Springer | C |
| P23 [80] | Migration of Software Components to Microservices: Matching and Synthesis | 2019 | ACM | C |
| P24 [25] | Service Candidate Identification from Monolithic Systems based on Execution Traces | 2019 | IEEE | J |
| P25 [81] | Tool Support for the Migration to Microservice Architecture: An Industrial Case Study | 2019 | Springer | C |
| P26 [21] | Towards Automated Microservices Extraction Using Multi-objective Evolutionary Search | 2019 | Springer | C |
| P27 [82] | Unsupervised Learning Approach for Web Application Auto-decomposition into Microservices | 2019 | SD | J |
| P28 [26] | An Automatic Extraction Approach: Transition to Microservices Architecture from Monolithic Application | 2018 | ACM | C |
| P29 [83] | Extracting Candidates of Microservices from Monolithic Application Code | 2018 | IEEE | C |
| P30 [27] | Function-Splitting Heuristics for Discovery of Microservices in Enterprise Systems | 2018 | Springer | C |
| P31 [84] | Migrating Web Applications from Monolithic Structure to Microservices Architecture | 2018 | ACM | C |
| P32 [85] | Extraction of Microservices from Monolithic Software Architectures | 2017 | IEEE | C |
| P33 [20] | From Monolith to Microservices: A Dataflow-Driven Approach | 2017 | IEEE | C |
| P34 [86] | Microservices Identification Through Interface Analysis | 2017 | Springer | C |
| P35 [24] | Service Cutter: A Systematic Approach to Service Decomposition | 2016 | Springer | C |

J = Journal, C = Conference, W = Workshop, WO = Wiley Online, SD = Science Direct

Beneath these (and other) migration phases, further sub-classification was required. In some cases, these sub-classifications were already explicitly provided in previous studies. For example, two of the four categories for data collection were previously identified in [97], while a further data collection category was identified in [P28]. A further example of the often-explicit presentation of sub-categories can be seen in [33], which explicitly identifies both Static Analysis (SA) and Dynamic Analysis (DA). [33] also identifies Model-Driven (MD) analysis which is also presented in our results, but where it is ultimately classified as Domain Analysis. In [33], we also note that Examples, Experiments and Case Studies are reported as the major categories for evaluation. These are noted elsewhere in the literature and directly feed into the derivation of these sub-categories.

Element Aggregation: Thus, the research is concerned with knitting existing, explicitly-identified and established categories into an end-to-end decomposition framework. Hence the elements of GT were combined with a light form of quantitative content analysis [98] in a mixed method. In studies where the information is not explicitly presented, we synthesised the appropriate phases and sub-categories based on GT, employing an analysts' lens of phases and approaches-within-phases, in the organisational spirit of [99]'s approach to GT. In studies

where phases and sub-categories were explicitly identified, a light form of content analysis was performed where those phases and sub-categories were noted and quantified.

Iterative Refinement: At the end of each full-text review, this information was analysed, compared and assigned categories. A new category was assigned in cases where no existing category was deemed suitable. The framework was revised incrementally to accommodate the new categories and to refine existing categories.

2) *RQ2-RQ4:* To answer RQ2, RQ3, and RQ4 a data-extraction template was generated based, in part, on the emerging M2MDF, but also on the specific RQs. After application of this template to individual sets of papers, review meetings presented the results of the data extraction process to the wider group for discussion and review. While the data analysis activity was undertaken by the first author, the analysis outputs were reviewed by all authors. Full-text versions of all the studies were shared among all authors enabling them to assess the accuracy of the process.

Using the data collected from the data extraction stage, the data synthesis stage focused on comparing, organising and re-organising the extracted data in a format that answers the research questions. It is important to note that this process was also revisited by the group, every time a paper was reviewed.

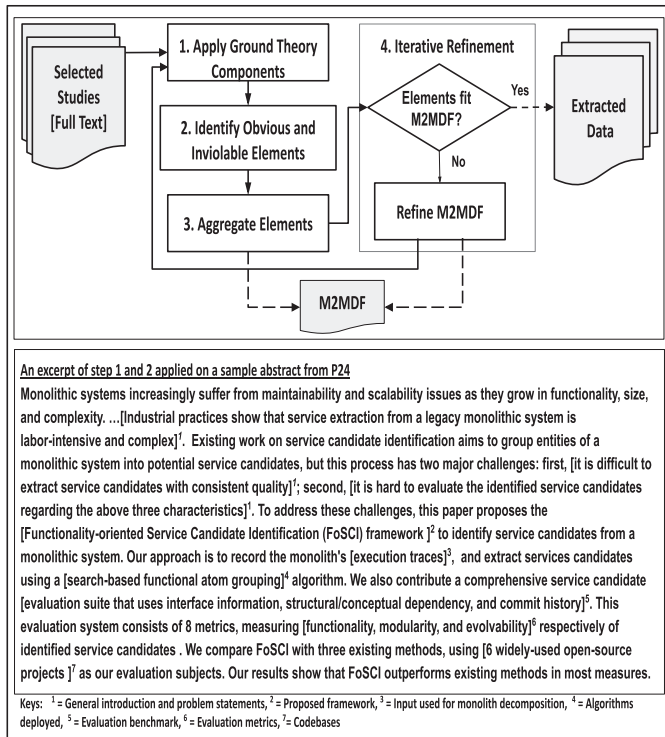


Fig. 2. Data analysis steps. The data analysis step begins with reviewing the full text of each selected paper step-by-step by applying GT components, identifying and aggregating elements. The process involves several iterations of reviews and presentation meetings resulting in a continuous refinement of the M2MDF and extraction of the data required to answer the research questions. A simplified excerpt of the memoing, coding and aggregation process is included at the bottom of the diagram.

After several revisions conducted during the literature review presentation meetings, we reached the final analysis result.

IV. LITERATURE REVIEW

Initially, Section IV-A presents an overview of the literature reviewed in this work. Thereafter, Section IV-B discusses the organisation of existing monolith-to-microservices decomposition research, as per RQ1. Then Section IV-C addresses RQ2 by focusing on the collection and analysis of monolith data, and on the resulting microservices identification and optimisation. Section IV-D reports on the evaluation of approaches and thus addresses RQ3, with Section IV-E briefly reporting on deployment of identified microservices to complete the description of the organisational framework.

A. Overview of the Selected Literature

In this subsection, the profile of the research included in this review is presented in terms of date of publication and publication venue.

1) *Temporal Distribution of the Studies*: On the evidence of our analyses, research in the decomposition of monoliths to microservices is gaining significant traction in recent years. Consistent with [47], since the time the term “microservice” was discussed in 2011, the number of studies has shown a steady increase from just a single paper in 2016, to three papers in 2017,

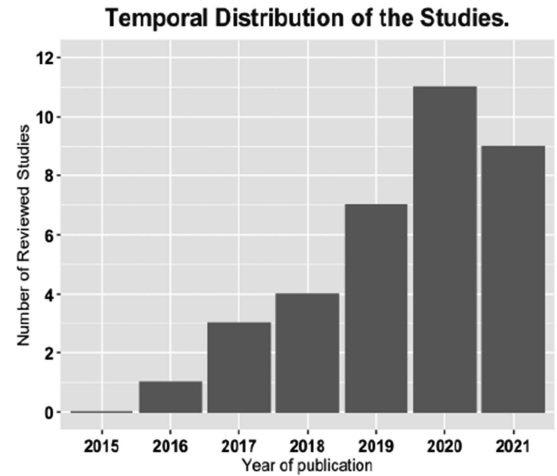


Fig. 3. Temporal distribution of the studies.

four in 2018, seven in 2019, 11 in 2020 and nine to October in 2021 (refer to Fig. 3).

Semi-automatic migration was first identified in 2016 [24] and since then, both semi-automatic and automatic decomposition approaches have been suggested. The early works propose solutions to decompose monolith applications to microservices using static, dynamic, and evolutionary approaches. Certain recent papers introduce genetic algorithms [21], [25] and Neural Network methods [62]. These recent contributions examine the use of feature evolution to extract microservices.

2) *Publication Venue*: The studies were distributed across several conferences. The Asia-Pacific Software Engineering Conference (ASPEC) and the European Conference on Service-Oriented and Cloud Computing (ESOCC) contributed three studies each. The Service-Oriented System Engineering conference (SOSE), IEEE International Conference on Web Services (ICWS), the European Conference on Software Architecture (ECSA), International Conference on Advanced Information Systems Engineering (CAiSE), International Conference on Service-Oriented Computing (ICSOC), and the International Conference on Software Architecture (ICSA) have contributed two studies each (refer to Table III). The rest of the studies were distributed across major software engineering conferences. The five journal papers are published in *IEEE Transactions on Software Engineering*, *the Journal of Systems Architecture*, *the Journal of Systems and Software*, *Software Practice and Experience*, and *Scalable Computing: Practice and Experience*.

Of the 35 papers included in this review, 12 were published by IEEE, 12 by Springer, five by ACM, three by Science Direct, and one each in Web of Science, Wiley Inter Science, and the Association for the Advancement of Artificial Intelligence (AAAI). This analysis demonstrates that the publication of microservices migration research is fragmented across conferences ranging from generic software engineering to more niche web services events. This is perhaps not surprising as the core architectural migration challenge is fundamentally a software engineering one, but one that may be implemented in the context of taking greater advantage of web/cloud-based computing innovations.

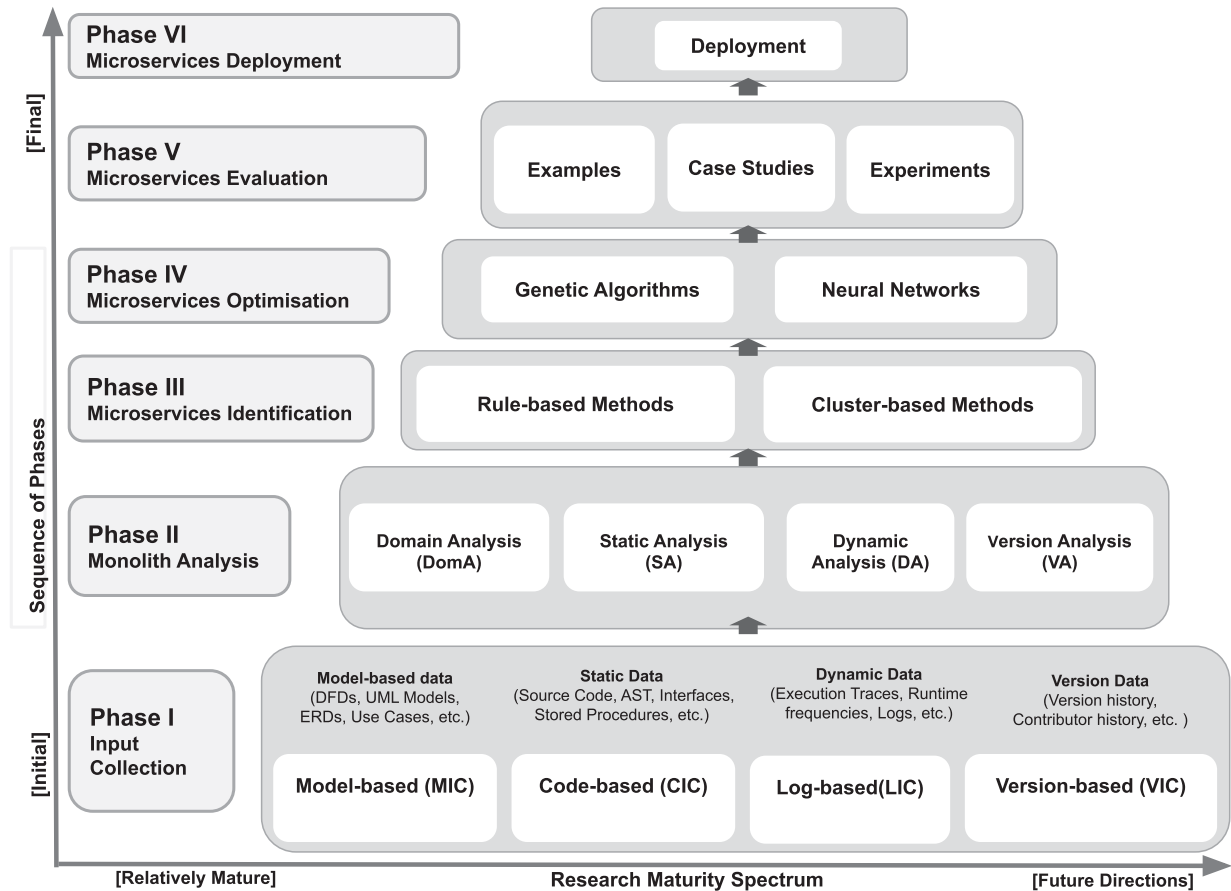


Fig. 4. Monolith to microservices decomposition framework (M2MDF) The maturity spectrum (x -axis) plots newer research areas further to the right, while the sequence of phases (y -axis) positions earlier phases closer to the bottom. For example, Model-based (MIC) data has been used for data input for some time and is utilised at the start of the monolith decomposition process (Phase I). In contrast, Version Analysis (VA) is a relatively recent research focus for decomposition studies, and it is part of Phase II, Monolith Analysis. This framework emerged from the literature following our approach discussed under Section III-C Analysis of Data particularly focusing on RQ1.

B. A Framework for Classification and Comparison of Monolith Application Decomposition

Existing decomposition frameworks focus on specific aspects such as monolith structure analysis [100], microservices validation [101], and microservices assessment [100], [102], but no single study captures all phases and techniques available for robust decomposition. In order to address RQ1, this research has systematically produced the Monolith to Microservices Decomposition Framework (M2MDF).²

Given the methodology employed, it can be argued that the proposed framework significantly reflects the phases and approaches proposed across the incorporated studies. It also serves as a taxonomy for the classification of the proposed approaches. The conceptual phases of the framework are as follows:

- Phase I–Input Collection.
- Phase II–Monolith Analysis.
- Phase III–Microservices Identification.

- Phase IV–Microservices Optimisation.
- Phase V–Microservices Evaluation.
- Phase VI–Microservices Deployment.

On the left side of Fig. 4, the major phases of a monolith to microservices decomposition are presented. Although indicated as a sequential process, some studies entirely skip certain steps. For example, Phase IV Microservices Optimisation is not observed in all works. By explicitly identifying these phases, the M2MDF framework, facilitates a scoped analysis of the individual works by logically partitioning the major activities associated with the end-to-end migration task. Along with the framework, we present a mapping of the studies when applicable (subsequently, in Fig. 5).

Fig. 4 further depicts the relative maturity of individual techniques, where maturity relates to the relative newness of the technique in the context of monolith to microservice decomposition. Maturity in this context is viewed in light of software process maturity where a matured process is repeatable, defined, managed, and optimised [103]. Capturing this maturity perspective is not easily given to quantitative representation, especially as the adoption of certain techniques will inevitably overlap. Therefore, the maturity dimension presented in Fig. 4 should be viewed as indicative and not as entirely discrete.

²The M2MDF and the detailed analysis of the literature essentially co-evolved in this research. Therefore, the M2MDF could be presented either prior to or following the detailed literature findings. The authors have elected to present the M2MDF prior to the detailed literature review findings as it will aid readers in visualising the broader landscape prior to progressing to the details.

However, in more-general terms, we can say that techniques closer to the left-hand side, such as domain analysis and static analysis have been under consideration for a longer time frame. Whereas techniques further to the right, such as the incorporation of version control data, are relatively newer.

1) *Phase I–Input Collection*: The decomposition efforts commence by acquiring data that describes the essential characteristics of the monolith application, for example, its domain model, codebases, log files, or code versions. Practically, many of these characteristics of the monolith, particularly domain models, are identified by prior reviewing of design documents, architecture diagrams, or by interviewing the custodians of the systems. However, input collection from monolith codebases, log files and versions are often supported by sophisticated tools and methods.

One or more of these inputs are used in practice, however, no single work has been identified that combines all these inputs. This observation is likely related to the large effort required to implement any one of these collection techniques and the significant differences of knowledge and skillsets required to successfully apply them. Theoretically, all the inputs could be used in combination by collecting data from domain-driven artefacts, the source/executable codebase, log files, and the different revisions of the monolith collected from version control systems such as Git. The combined use of these various data sources could be of significant importance, given that greater volumes of pertinent data hold the potential for improved understanding of the wider behaviour of a monolith ecosystem and could contribute to improved decisions concerning monolith decomposition.

2) *Phase II–Monolith Analysis*: The monolith analysis phase focuses on filtering and transforming the collected data into a representation that is suitable for the subsequent phases. This phase may adopt multiple stages of analysis including domain analysis and static analysis to extract the structural relationships between the artefacts of the monolith, while dynamic analysis and version analysis focus on enriching the relationship with frequencies and associations observed on the execution and the evolution of the codebase over time respectively.

3) *Phase III–Microservices Identification*: The microservices identification phase uses heuristics to guide the microservices identification process by partitioning the monolith application into suitable microservice candidates [104]. Clustering algorithms are widely used to extract microservices by representing the monolith data as a graph/matrix and treating the problem as a clustering problem that tries to identify artefacts that can be grouped as microservices without any prior awareness of the number or size of the resulting microservices.

4) *Phase IV–Microservices Optimisation*: Some emerging studies adopted a two-phased approach to microservice identification that first generates a large pool of microservice candidates in order to subsequently select the optimal microservice partitions. Attempts to optimise microservice configurations in this way have employed genetic algorithms in addition to cluster-based microservices identification methods. Genetic algorithms assist in the identification of optimal combinations of packages, classes or methods within the monolith application using objective functions based on constructs such as cohesion,

coupling, and semantic similarity. Evolutionary algorithms also employ heuristics to guide the optimal selection of microservices. This microservices optimisation phase is often valuable when transforming large parts of an existing monolith application into microservices, where the efficiency of microservices interaction and execution in the target system is conceivably of significant concern. Where just a single microservice is to be extracted for a specific purpose, this optimisation phase may not be implemented as no significant microservices orchestration may be required in the target implementation.

5) *Phase V–Microservices Evaluation*: Evaluation of the success of monolith decomposition into microservices has received limited attention in the literature to date. Most often the assessment of the resulting microservices is informally reported by experts evaluating the proposed candidate microservices. In recent years, more structured evaluation approaches have been proposed including examples, case studies, and experiments. Example-based evaluation uses illustrative examples to show how the identification process is applied. In case studies, a target monolith is used to evaluate the migration process in detail by looking at relevant cases, whereas in experiments, selected codebases are migrated to microservices and experimentally evaluated, using a range of quality metrics such as coupling, cohesion, modularity, and evolvability.

6) *Phase VI–Microservices Deployment*: The deployment phase focuses on the implementation of the proposed microservices candidate(s) to evaluate if the proposed technique is successful or the extracted microservices candidate(s) exhibit(s) the desired properties. Many of the existing works have not reported how the proposed candidate services are built from the monolith code and delivered as a microservice. While the previous phases can propose effective microservices, it still requires a huge effort by the programmers to compose, build, and repack the candidate microservice, in order to provide a complete microservice offering.

C. Existing Approaches, Tools and Methods

This section explores RQ2 by examining the approaches, tools and methods reported in decomposition studies. This includes the data gathered, the analyses employed, and the microservice identification and optimisation methods used in the literature.

1) *Phase I. Input Collection*: The collection phase involves the acquisition of monolith application data that can later be used to identify candidate microservices from within the monolith. The related works are first categorised based on the data collection methods, and thereafter on the type of data collected.

a) *Data Collection Methods*: The data collection methods used in the selected studies are extracted into four categories: Model-based Input Collection (MIC) [97], Code-based Input Collection (CIC) [97], Version-based Input Collection (VIC) (also known as evolutionary coupling [P28]), and Log-based Input Collection (LIC).

MIC uses domain-driven design artefacts (e.g., data flow diagrams, entity relationship diagrams, and process models) to extract the features of the monolith application. CIC focuses on the collection of the structure and behaviour of the monolith

TABLE IV
DATA COLLECTION METHODS USED IN THE STUDIES

| Methods | % | Studies |
|--------------------------------------|-------|--|
| Code-based Input Collection (CIC) | 68.6% | [P1, P2, P4-10, P13-15, P18-20, P22-26, P29-31, P34] |
| Model-based Input Collection (MIC) | 11.5% | [P3, P21, P33, P35] |
| Log-based Input Collection (LIC) | 5.7% | [P17, P27] |
| MIC+CIC | 5.7% | [P11, P16] |
| CIC+VIC | 5.7% | [P28, P32] |
| CIC+LIC | 2.8% | [P12] |
| Version-based Input Collection (VIC) | 0.0% | Not yet used independently. |

using the source code or executable code of the application. VIC uses simultaneous or consecutive versions of the source code or domain-driven artefacts to extract features that are useful to characterise the monolith application (essentially because versioning data can be indicative of significant feature level contributions over time). Finally, LIC uses log files that are generated during the run time of the application.

One example of log files are performance logs, which contain information indicating method-level CPU and memory consumption (and perhaps additional data also) [70]. Alternatively, web access logs contain information such as client IP, requested resource URI, timestamp, HTTP status code, and document size [75]. In some cases, web access logs may, however, include just a subset of this data, for example the URI, document size and response time [82].

Additionally, there are studies that combine one or more of the methods, particularly a combination of CIC either with MIC (CIC+MIC) or with VIC (CIC+VIC). From the analysis, it is clear that CIC is the dominant approach. This research did not identify any study where VIC has been used independently. A summary of the distribution of the studies based on the type of collected data is presented in Table IV.

b) Type of Collected Data: Two types of data are prominent in the literature: Static Data (SD) and Dynamic Data (DD). SD focuses on the source code of the application reflecting the structure, whereas DD deals with the data generated during the execution of the monolith application. Data may also be obtained using MIC and VIC. Model-based Data (MD) focuses on domain-driven design artefacts, and Version-based Data (VD) focuses on the change history of the source code. While SD is the most widely used type of data (42.9%), it is also used in combination with DD, VD, and MD (refer to Table V).

A frequent combination is the use of SD and DD (SD+DD) to increase the understanding of the monolith application and raise the efficiency of the microservices identification process. In [P2, P6, P10, P13], static data is used to represent the system with a call graph where the nodes represent components and the edges represent dependencies between them. Then, operational data is collected at runtime to identify dependencies and compute the weight of the edges. In [P16], source code packages are mapped to bounded contexts using static analysis, while dynamic data is employed to refine these contexts by computing their associated runtime frequency. These runtime frequencies can then be used

TABLE V
TYPE OF ANALYSIS AND TYPE OF COLLECTED DATA

| Analysis | Data | % | Studies |
|------------------------|----------|-------|---|
| Static Analysis (SA) | SD | 42.9% | [P1, P4, P5, P9, P11, P14, P15, P18, P20, P22, P23, P25, P26, P29, P34] |
| Dynamic Analysis (DA) | DD | 20.0% | [P7, P8, P12, P17, P19, P24, P27] |
| SA+DA | SD+DD | 17.1% | [P2, P6, P10, P13, P30, P31] |
| Domain Analysis (DomA) | MD | 11.5% | [P3, P21, P33, P35] |
| SA+VA | SD+VD | 5.7% | [P28, P32] |
| SA+DA+DomA | SD+DD+MD | 2.8% | [P16] |

in a postmortem analysis to further discover new candidate microservices.

In a broadly similar fashion to [P13], [P31] uses static code analysis to represent a call graph of the monolith application, subsequently overlaying dynamic information to indicate the frequency of edge execution. This tendency to combine different types of data is confirmed in earlier research [33], but in our analyses, we note an increase in combining not only static and dynamic data but also static and version-based data [P28, P32]. Some correspondence between the data collection methods and the type of data collected is observable. For example, although all the methods that use MIC collected MD data, CIC methods are used to collect either SD or DD. Furthermore, DD collection is not restricted to CIC, but it is also collected using LIC as in [P17, P27] where log files are used.

2) Phase II. Monolith Analysis: Monolith analysis strategies are essentially comprised of three distinct perspectives. The first perspective concerns the type of analysis employed, the second perspective is the desired unit of analysis (for example, class-level or package-level), and the third perspective, depending on the type of data and unit of analysis, concerns the various forms of data representation that may be employed. This section outlines the findings with respect to all three perspectives.

a) Type of Analysis: The studies included in this SLR suggest that four major analysis types may be used in isolation or in combination: Domain Analysis (DomA), Static Analysis (SA), Dynamic Analysis (DA), and Version Analysis (VA). These analysis types correspond to the type of the collected data. Although log analysis appears in some of the studies [P12, P17, P27], the analysis employed is similar to dynamic analysis. Thus we treat it as a subset of the dynamic analysis.

DomA focuses on the analysis of various models of the monolith application that have been employed during the requirement analysis and modelling stage of software development [105], [106], [107], [108] (or those extracted using reverse engineering methods). DomA includes data flow diagrams, activity diagrams, use case descriptions, entity relationship diagrams, and other UML artefacts that describe the process and intent of the program. Specific examples of DomA include the business process model [P3]; use case specification, business logic analysis, and data analysis [P21]; dataflow diagrams [P33]; and nanoentities [P35]. DomA is essentially concerned with identifying domains of interest within the existing monolith. These domains of interest are key themes or logical functional elements that can

be observed in the system. DomA is sometimes referred to as *model-driven* or *domain-driven*.

SA is sometimes referred to as static code analysis and is performed by studying the (non-executing) source code of an application [109], [110]. Typically, supporting software-based tooling is employed. Fifteen of the studies relied entirely on SA, with an additional nine studies using SA in combination with DA, VA, or DomA. This indicates that SA is a prevalent and important choice for microservices identification. SA is fundamentally aligned with static data (SD).

The essential difference between SA and DomA concerns the fact that DomA analyses the monolith using models employed to structure/build the source code rather than directly analysing the actual source code or associated executing programs. In areas outside of monolith decomposition [111], DomA has been used to refer to a SA technique that involves analysis of the flow of data in a program source file.³

DA is sometimes referred to as dynamic code analysis and involves the examination of the properties of a program while it is executing [112]. DA aids the understanding of the runtime behaviour of an application. DA of monolith applications can be *online* (where the program is examined as it executes) using code instrumentation features provided by IDEs (such as AspectJ Runtime, AspectJ Tools, and AspectJ Weaver) [113]. DA may also be conducted *offline*, where the runtime data is collected and stored for later analysis (for example, in log files).

DA collects rich information about executing programs, including the thread id, calling class, called class, calling methods, called methods, method parameters, timestamp, and database queries. Where DA is performed on non-object-oriented programs, collected data may refer to *functions* in place of *methods* and *classes*. Although DA is getting more attention recently, the published literature suggests that it remains less utilised. Only 20% of the studies used DA exclusively, with a further 20% using it in combination with SA or DomA. This could be due to the difficulty of collecting the dynamic data during the execution of the monolith and/or the overhead it introduces to the running monolith [113].

VA examines monolith versions to learn how the monolith was extended or changed over time. Although no study applied VA exclusively, VA has been applied in combination with SA. For instance, SA can produce a call graph that is supplemented with evolutionary coupling data (the code that was co-committed) [P28]. An alternative to this approach involves injecting VA into early-stage SA while the monolith's graph is still under construction [P32]. This alternative approach shares some conceptual underpinning with [P13] in that, artefacts of the monolith are analysed using source code repository information to identify items that appear to co-evolve or evolve according to a similar timeline. Using this approach, increased co-evolution can raise the edge weighting between the affected items.

[P16] combines SA, DA and DomA. Process documentation and interviews, along with context diagrams, use cases, domain context and context maps are first used to identify context boundaries. Static analysis and dynamic analysis are then applied in order to further refine the bounded contexts. Although

TABLE VI
UNIT OF ANALYSIS (GRANULARITY)

| Level | % | Studies |
|-------------------|-------|---|
| Class | 54.3% | [P1, P4, P5, P8-13, P15, P18, P20, P22, P24, P26, P28, P29, P30, P32] |
| Method | 14.2% | [P2, P6, P14, P31, P34] |
| Package | 11.5% | [P7, P16, P19, P25] |
| Software Artefact | 11.5% | [P3, P21, P33, P35] |
| URI | 5.7% | [P17, P27] |
| SW Component | 2.8% | [P23] |

this approach combines three of the four identified analyses, the effort required to conduct such detailed domain analysis may render it a less attractive solution (this may be particularly relevant for larger monolith applications).

b) Unit of Analysis: Existing research on microservices decomposition uses a wide spectrum of granularity when extracting microservices. *Unit of analysis* in this review refers to the smallest unit of the system a study uses when seeking to identify microservices. Units can be as small as individual methods or functions, ranging up to classes and packages. An additional unit of analysis could involve, in the case of domain analysis, software artefact-based units. These could, for example, include use cases and data stores. Table VI presents an overview of the unit of analysis adopted by the studies identified in this review, with the following paragraphs providing a summary of the different approaches.

Method-level analysis explores the use of methods as the base unit of analysis, thereafter extracting microservices by recombining highly-related methods [P2, P6, P14, P31, P34]. In this approach, one method from a given class could be merged with methods from other classes to make a new microservice. However, method-level analysis, due to its fine granularity, requires relatively large hardware computing resource support. The implementation of method-level microservices may also require additional validation and testing, particularly when methods are extracted from different classes and merged into a new service.

This validation process includes, initially, ensuring the behaviour of the method when it is run outside of its class and, subsequently, ensuring how the composed methods function when they are used in combination with methods from other classes. Existing research has not yet addressed these challenges, however, method-level analysis can potentially produce highly-optimised microservices as it can cross class boundaries, thus identifying chains of related methods. Applying method-level analysis, it is possible to extract individual methods as microservices, which may be particularly well-aligned with Function-as-a-Service (FaaS) based cloud computing [14]. Certain utility methods, for example a method that provides a payment capability across a monolith, could also be identified using method-level granularity.

Class-level analysis elevates the base unit of granularity up to the level of a class. Given that classes perform a vital role in established monolith programming languages, for example, Java and C++, and that the computational analysis cost at a class-level is likely to be considerably less than at a method-level, it is perhaps unsurprising to discover that class-level analysis

³<https://glossary.istqb.org/en/term/data-flow-analysis>

granularity is widely adopted in the literature [P1, P4, P5, P8-13, P15, P18, P20, P22, P24, P26, P28, P29, P30, P32].

Using class-level analysis, classes within a monolith are clustered based on several criteria including coupling and cohesion [P10, P21, P26], topic modelling [P1, P5], and business object modelling [P9, P30], leading to the identification of candidate microservices. They may be represented as nodes in a call graph, with edges in this case representing various object-oriented constructs, including object references, inheritance relationships, and evolutionary couplings. Later, this information is used to cluster closely related classes into candidate microservices.

Package-level analysis, as implemented in [P7, P16, P19, P25], adopts a higher level granularity than class-level or method-level analysis. Source code packages (SCPs), which may be comprised of large numbers of classes and associated methods, are used as the base unit of analysis. The general approach with package-level analysis involves identifying packages that can be grouped together as microservices. Package-level analysis does not seek to split packages when projecting candidate microservices. This may be advantageous where packages are highly decoupled and exhibit high internal cohesion. However, if packages are very large or externally dependent for end-to-end service completion, the microservices resulting from this approach could be prohibitively large in size. For well-managed codebases where larger services are preferred, package-level analysis could offer a fast-track to miniservice or macroservice [114] identification. In such a scenario, product-level decomposition risks may be reduced as there is less fragmentation of the existing monolith.

Software artefact-based analysis [P3, P21, P33, P35] involves the use of artefacts for candidate microservices identification. Although the term software artefact refers to various by-products of a software development process [115], [116], in the decomposition context, it mainly refers to documents such as data flow diagrams, use cases, process models, and ER diagrams [P21, P33]. Source code and executable files are not included in this type of analysis. This is the least clear of all the proposed analysis approaches, perhaps because the mapping between actual source code classes and meta-level artefacts is not always obvious. Indeed, meta-level artefacts may not be produced in all settings, or they may become inconsistent with the codebase over time.

Analysis based on Uniform Resource Identifiers (URIs) [P17, P27] occurs in the context of web-based monoliths. URIs represent different resources available to a system. With URI-based analysis, a monolith is decomposed based on its association with, and the frequency of its association with, URIs at runtime. Finally, one paper [P23] has not explicitly reported the unit of analysis except indicating that it used software components.

c) Input Data Representation: The input data is represented using a variety of data structures. For example, several studies (51.4%) [P2, P3, P4, P5, P9, P11, P13, P19, P22, P25, P26, P28, P29, P30, P31, P32, P33, P35] have used graphs for input data representation. Graph nodes are used to represent granular elements such as methods, classes, packages, and processes, while edges represent the call relationship between the nodes. Both directed edges (that indicate the flow of information or the direction of the call) and undirected edges are employed to represent rich data. The edges may carry further information

including static and dynamic call frequencies or weights, process flows, and similarity and distance metrics. Graphs can also be used in combination with topic modelling that uses *bag of words* [P25].

Execution traces in tuple form have been used to represent input data (14.3%) [P8, P10, P17, P24, P27]. This involves tracking various items potentially relevant to the decomposition task, including the thread Id, calling/called class names, calling/called method names, parameters and timestamps. Furthermore, indicators for entry and exit methods can be retained for the tuples. Although there is no clear distinction between data representation using graphs and matrices, there are papers that explicitly use matrix or vector representations (14.3%) [P1, P6, P12, P18, P20], set (5.7%) [P21, P34], and word embedding [P14] as sets. Extended Backus-Naur Form (EBNF) is used in [P23] to formally represent software components. The remainder of the studies do not explicitly report their data representation methods (8.6%) [P7, P15, P16].

3) Phase III. Microservices Identification: This section presents the microservice identification tools reported in the literature. Thereafter, the microservice identification methods are presented. These generally fall into two categories: rule-based approaches and cluster-based approaches.

a) Microservices Identification Tools: Some preliminary work has been carried out to build tools for microservices decomposition. One such tool is *Service Cutter* [P35] which has been used not just for microservices identification but also for initial attempts to benchmark effectiveness in [P13, P21, P33, P34]. Various other tools have been developed explicitly towards a microservice extraction agenda using static and dynamic data. *Kieker* [P12, P24] extracts dynamic data from Java applications [117], *Dbeaver* [P16] is used to extract database tables accessed by a monolith at runtime, *ExploreViz* [P16] is a tool for 3-D visualisation of candidate microservices based on static and dynamic data, and *OpenAPI* (formerly *Swagger*) is a language-agnostic interface that is used to define and extract web service interfaces.

Tools such as *Arcan* [P25], *DISCO* [P10], *Microservice Miner* [P11, P13], *MSExtractor* [P26], *Decomposer* [P34] and *Scipy Python Library* [P22] are also used to assist in the extraction of microservices. Each of these tools supports specific extraction algorithms which are identified in the respective papers. These data collection tools are typically built to support specific programming languages (usually Java) and are therefore not immediately applicable to codebases in other languages. Additionally, various in-house algorithms have been developed as part of the studies [P9, P18, P24, P30] and have been utilised in the related literature. This demonstrates that software-based tools are an important dimension when decomposing monolith applications into microservices.

b) Rule-Based Microservice Identification Methods: Rule-based identification methods involve varying degrees of direct human involvement and engagement in the process by providing some guidelines or rules about how to partition the monolith source code. Although it covers only 11.5% of the studies, rule-based methods [P11, P19, P21, P33] are observed in model-driven approaches, with the rules being defined based upon prior knowledge of the application domain and the

experience of the decomposition engineers, such as in [P19, P21, P33]. Rules make extensive use of dataflows, use cases, and data stores as input, which could produce high-quality results. In [P33], the authors formulate four rules to apply on purified data flow diagrams. These include, for example, rules related to the number of inputs, and operations included in a single business process.

Although capable of producing microservices with fine-grained service boundaries [P21], rule-based approaches require significant manual human effort owing to their high dependency on expert knowledge. This limitation may render rule-based approaches less attractive for the decomposition of larger and more complex codebases. Rule-based methods have also been used in combination with SA and DA, for example in [P11, P16], where abstract syntax trees are analysed, followed by manual inspection of the resulting candidate microservices.

c) Cluster-Based Microservice Identification Methods: We use the term cluster-based microservice identification methods to refer to various unsupervised machine learning algorithms. These algorithms are widely observed in the literature, and they tend to favour different forms of statistical clustering. Many clustering techniques have been proposed, including hierarchical, K-means, affinity propagation, Girvan-Newman, fast community, SArF, Minimal Spanning Tree (MST), Epidemic Label Propagation, collaborative clustering, and the Hungarian method as in [118]. While variants of hierarchical clustering algorithms are used to generate dendrograms that identify the cutting points for the desired number of microservices, some require prior selection of the cluster size, but others determine the cluster size automatically.

Heuristics-based clustering, as used in [P30], defines two rules related to subtype and common subgraph (a step which enables the grouping of call graphs). The CO-GCN3 (Clustering and Outlier-Aware Graph Convolution Network) approach has also been applied to cluster classes [P4]. This demonstrates that neural network (NN) based approaches are considered to have some value in microservice identification. Indeed, it is claimed that NN-based approaches can present improved results when compared with the widely used clustering variants [62]. Other contributions use genetic algorithms [P2, P12] to partition the classes into cohesive but loosely coupled clusters. This involves rearranging the groupings over a number of generations and mutations. Topic modelling, using Latent Dirichlet Allocation (LDA) and Seeded LDA (SLDA) algorithms, have also been applied in dependency graph analysis and topic detection for microservices identification [P25].

An overview of the reported unsupervised identification algorithms is presented in Table VII, which demonstrates that a wide variety of algorithms have been applied to identify microservices from monolith applications. However, it should be noted that the use of clustering algorithms for the identification of microservices from large-scale monolith applications has not yet been explored in-depth in the literature. This may be due to the lack of resources and access to large-scale (millions of lines of code) enterprise systems. Thus, further research is warranted to determine the efficacy of clustering based analysis for larger systems, especially when method-level granularity is selected.

TABLE VII
UNSUPERVISED MICROSERVICES IDENTIFICATION ALGORITHMS

| Algorithms | Studies | # |
|--|--------------------------|---|
| Hierarchical Clustering | [P8, P15, P20, P22, P24] | 5 |
| Kmeans Clustering | [P6, P17, P18, P27, P31] | 5 |
| Affinity Propagation Clustering (Unspecified) | [P1, P14] | 3 |
| Genetic Algorithm | [P13, P30] | 2 |
| Fast Community Clustering | [P2, P12] | 2 |
| Girvan-Newman and Epidemic Label Propagation (ELP) | [P28] | 1 |
| Louvain Algorithm | [P35] | 1 |
| Graph Matching for Ontologies (GMO) | [P5] | 1 |
| Dependency Graph Analysis and Topic Modelling (LDA and SLDA) | [P23] | 1 |
| Network based Community Detection Algorithm | [P25] | 1 |
| SArF [24] | [P9] | 1 |
| Minimum Spanning Tree (MST) | [P29] | 1 |
| Hungarian Algorithm | [P32] | 1 |
| CO-GCN3 | [P34] | 1 |
| | [P4] | 1 |

4) Phase IV. Microservices Optimisation: Although microservices optimisation can be integrated into the identification phase, recent studies have treated the optimisation method as a distinct phase [119]. Unsupervised clustering renders multiple microservices candidates which can be used as input to an optimising algorithm, for example, a genetic algorithm. Furthermore, fitness functions have been proposed as way to evaluate proposed microservices.

a) Optimisation Algorithms: Distinct from the clustering algorithms applied to identify candidate microservices, genetic algorithms have been utilised to refine optimal combinations of software units (for example, classes) in pursuit of higher optimisation in the target microservices system [P12, P24]. Optimisation algorithms are less popular in the studies: 29/35 (82%) studies have not used them. However, they are gaining popularity in recent studies (in the past 3 years) where three studies [P12, P24, P26] used NSGA II (Non-dominated Sorting Genetic Algorithm II) [120], one study [P4] used NSGA III, and another study [P23] used a combination of NSGA II and SPEA II (a Strength Pareto Evolutionary Algorithm) [121]. The Study that used CO-GCN3 [P4] has employed the ADAM optimiser [122].

Genetic algorithms have shown improvement in identifying microservices when used in combination with clustering algorithms [P24, P26]. However, a recent (and still ongoing) study suggests that the use of NSGA-III in multi-objective optimisation could contribute to efficient microservices candidate identification without going through the clustering step [119].

b) Connectivity Fitness Functions: Structural and Conceptual Intra-Connectivity and Structural and Conceptual Inter-Connectivity fitness functions are used to assess the optimisation process in [P24]. Structural Connectivity is calculated based on the number of edges between classes in the so-called functional atom groups, whereas Conceptual Connectivity is calculated based on shared terms in the class identifiers. Fitness has also been evaluated on the basis of CPU and memory consumption

where candidate microservices are extracted at a class-level granularity using NSGA-II [P12].

D. Phase V: Microservices Evaluation

In this section, we investigate the metrics, datasets, and benchmarks that are used for evaluating and validating the extraction process and its results. For the purpose of this work, we classify evaluation efforts into two categories for discussion: the metrics employed for evaluation and the benchmarking systems used as the basis for the evaluation of datasets.

Evaluation metrics allow us to estimate candidate microservice qualities, by characterizing and quantifying their relevant features. These metrics enable architects to understand, compare, and improve candidate microservices and decomposition methods. In contrast benchmarks and datasets refer to the systems and system artefacts chosen as the subject matter for the decomposition.

The majority of the studies (88%) have reported the evaluation of their resulting microservices. 71% of the studies have used some evaluation metrics, whereas 17% reported the use of manual evaluation. All identified papers from 2021 [P1-P9] have used metrics to evaluate their results. But this was not the prevailing case in earlier research which incorporated higher levels of manual evaluation [P29, P33, P34, P35]. Only four studies did not report the evaluation or do not evaluate the resulting microservice candidates at all.

1) *Evaluation Metrics*: Analysis of the literature suggests that evaluation metrics are comprised of coupling metrics, cohesion metrics, non-functional metrics, and cluster size metrics.

a) *Coupling Metrics*: The most prevalent and stable metrics with long-lasting literature support in the software development domain are coupling and cohesion [57], [123], [124]. While coupling measures the degree of dependence between different microservices [57], cohesion measures the single responsibility of a microservice. Apart from the generic coupling measure used in [P2, P10, P15, P20, P32], many variants of coupling metrics are used in the reviewed studies. Afferent coupling and efferent coupling [P3, P21], structural coupling [P18, P30], instability [P3, P21], and Interface Number (IFN) [P4, P5, P8, P24] are used across the studies. Whereas some of these coupling variants are well understood in the software engineering domain, the parameters used to compute the metrics vary, and it is this observable variation that makes the comparison of different studies difficult, even when seemingly identical coupling metrics are assessed.

b) *Cohesion Metrics*: Typical cohesion metrics reported in the literature, include Lack of Cohesion (LOC) [P2, P9, P14, P15, P18, P30] [125], Cohesion at Message level (CHM) [P1, P5, P24, P26], Cohesion at Domain level (CHD) [P1, P5, P24, P26], and relational cohesion [P3, P21] as in [126]. Other related metrics in this category are modularity [P2, P9] and structural modularity (SMQ) [P4, P5, P8, P24] which are used to measure the strength of the resulting microservices. Conceptual similarity using the number of shared terms among services and structural similarity using the number of calls between selected artefacts are used in computing SMQ in [P24]. These metrics are used to measure the strength of the dependency between artefacts that make up the microservice, thus indicating the

cohesion of a microservice. Methods that employ graph-based representations tend to also use coupling and cohesion metrics along with the number of imports, number of calls between artefacts, and structural and semantic similarities. This association demonstrates that effective automated microservices extractions are largely dependent on variants of cohesion and coupling criteria.

c) *Non-Functional Metrics*: Other non-functional metrics are also utilised to evaluate microservices. Execution time [P7, P12, P18, P27, P30], CPU usage [P7, P12, P18], memory usage [P7, P12, P18, P27, P30], and network overhead [P2, P18] are the widely used performance measures. Efficiency is also employed as a metric. It is measured by comparing the execution time of a monolith versus a microservice [P6, P30]. By comparing memory and disk usage over time [89] [P6, P30], scalability can be evaluated. There is also evidence of availability measurement. This is achieved by comparing the total up-time and down-time when performing tasks (per 100 requests) [P6, P30].

d) *Cluster Size Metrics*: Various clustering size metrics have also been employed. These include the number of classes per cluster [P10, P22] and the number of generated microservices (number of clusters) [P9, P28]. Non-Extreme Distribution (NED) has been used to evaluate the even distribution of the classes in the clusters [P4, P8, P9]. These measures are usually based on the count of entities of interest to architects and provide useful input when comparing the results of different configurations of the same microservices decomposition approach. Manual evaluation, mainly using questionnaires and interviews, is also used frequently [P2, P11, P13, P15, P29, P33, P34, P35], particularly in the older studies [P33, P34, P35] (as already noted).

Coupling and cohesion metrics in combination have become the most widely used metrics in recent publications [P2, P3, P4, P5, P15, P18, P21, P24, and P30]. Performance related metrics are also gaining popularity [P6, P7, P12, P18, P23, P27, P30], particularly in evaluating the performance of the candidate microservices in cloud environments. Although there is a rising interest in using these metrics, there are subtle differences in their calculation particularly when they are used along with data collected from static, dynamic, and version-based datasets. A list of the metrics that are used by two or more studies is provided in Table VIII. The asterisk (*) in the "Studies" rows indicates that the study has used additional metrics that are not used in other studies.

2) *Evaluation Benchmarks*: Various codebases are available for the purpose of evaluation, and different validation methods have been utilised in different studies. It is also the case that differences in programming languages presents as a distinct feature of evaluation efforts.

a) *Evaluation Codebases*: For consistent evaluation of proposed decomposition methods, the availability of datasets and benchmarking data is crucial but this is potentially the least explored area [127], [128]. The selection of monolith applications for experimental purposes is driven by the availability of monolith source code (often Open Source Software), the equivalent microservices, and access to the monolith programming environment. Due to the absence of published and agreed-upon

TABLE VIII
MICROSERVICES CANDIDATE IDENTIFICATION AND EVALUATION METRICS

| Studies | Coupling (Generic) | Afferent Coupling | Efferent Coupling | Structural Coupling | Instability | IFN | LOC | CHM | CHD | Relational Cohesion | Modularity | SMQ | Functionality | Execution Time | CPU Usage | Memory Usage | Network Overhead | Efficiency | Scalability | Availability | Cluster Size (Classes) | No. of Microservices | NED | Manual Comparison | None |
|---------|--------------------|-------------------|-------------------|---------------------|-------------|-----|-----|-----|-----|---------------------|------------|-----|---------------|----------------|-----------|--------------|------------------|------------|-------------|--------------|------------------------|----------------------|-----|-------------------|------|
| P1 | | | | | | | | X | X | | | | | | | | | | | | | | | | |
| P2* | X | | | | | | X | | | | | | | | | | X | | | | | | | | X |
| P3 | | X | X | | X | | | | | X | | | | | | | | | | | | | | | |
| P4 | | | | | | X | | | | | X | X | | | | | | | | | | | | X | |
| P5* | | | | | | X | | X | X | | | X | | | | | | | | | | | | | |
| P6 | | | | | | | | | | | | | | | | | | X | X | X | | | | | |
| P7 | | | | | | | | | | | | | | X | X | X | | | | | | | | | |
| P8* | | | | | | X | | | | | | X | | | | | | | | | | | | X | |
| P9* | | | | | | | X | | | | X | | | | | | | | | | | | X | X | |
| P10* | X | | | | | | | | | | | | | | | | | | | | X | | | | |
| P11 | | | | | | | | | | | | | | | | | | | | | | | | | X |
| P12 | | | | | | | | | | | | | | X | X | X | | | | | | | | | X |
| P13 | | | | | | | | | | | | | | | | | | | | | | | | | X |
| P14* | | | | | | | X | | | | | | | | | | | | | | | | | | |
| P15* | X | | | | | | X | | | | | | | | | | | | | | | | | | X |
| P16 | | | | | | | | | | | | | | | | | | | | | | | | | X |
| P17 | | | | | | | | | | | | | | | | | | | | | | | | | X |
| P18 | | | | X | | | X | | | | | | | X | X | X | X | | | | | | | | |
| P19 | | | | | | | | | | | | | | | | | | | | | | | | | X |
| P20* | X | | | | | | | | | | | | X | | | | | | | | | | | | |
| P21 | | X | X | | X | | | | | X | | | | | | | | | | | | | | | |
| P22* | | | | | | | | | | | | | | | | | | | | | X | | | | |
| P23 | | | | | | | | | | | | | | | | | | | | | | | | | X |
| P24* | | | | | | X | | X | X | | | X | X | | | | | | | | | | | | |
| P25* | | | | | | | | | | | | | | | | | | | | | | | | | |
| P26* | | | | | | | | X | X | | | | | | | | | | | | | | | | |
| P27 | | | | | | | | | | | | | | X | | X | | | | | | | | | |
| P28* | | | | | | | | | | | | | | | | | | | | | | | X | | |
| P29 | | | | | | | | | | | | | | | | | | | | | | | | | X |
| P30 | | | | X | | | X | | | | | | | X | | X | | X | X | X | | | | | |
| P31* | | | | | | | | | | | | | | | | | | | | | | | | | |
| P32* | X | | | | | | | | | | | | | | | | | | | | | | | | |
| P33 | | | | | | | | | | | | | | | | | | | | | | | | | X |
| P34 | | | | | | | | | | | | | | | | | | | | | | | | | X |
| P35 | | | | | | | | | | | | | | | | | | | | | | | | | X |

evaluation benchmarks, individual studies have generally applied bespoke evaluations. However, in a positive development, there is a trend towards increasing the availability of artefacts and data from experimental decomposition for external analysis.

Recently, certain monolith applications along with their associated microservices implementations have emerged. These include *JPetStore* [P5, P8, P12, P24, P26, P31], *DayTrader* [P4, P8, P9, P31], *Acme Air* [P4, P8, P9, P27], *Petclinic* [P7, P9, P29, P32], and *Cargo Tracking System* [P21, P34, P35]. The other studies comprising this review use unique codebases to demonstrate and validate their proposed methods. *JPetStore* would appear to be emerging as a popular Java monolith for decomposition experimentation, perhaps because it is relatively lightweight and makes various versions available. Other opportunities for general evaluation also exist in the form of *ChurchCRM*, *SugarCRM* and the *Fujitsu Purchasing* COBOL-based application.

The number and purpose of applications that are used in each study varies significantly. One study has used 200 open-source codebases to conduct its experiment [P5], whereas another study

has used 21 different codebases [P32]. A further two studies use seven codebases [P8, P24]. But overall, only 10% of the studies have used more than four applications as part of their evaluation. The remainder of studies have used four or less, as follows: four [P4, P9, P31], three [P15, P34], two [P7, P12, P14, P18, P22, P26, P28, P29, P30], one [P2, P3, P11, P16, P17, P19, P21, P25, P27, P33, P35], and zero [P23]. *Petclinic* and *Fujitsu Purchasing* are used for manual evaluation in [P29].

b) *Evaluation Methods*: Previous reviews [33], [126] classify validation methods into one or more of the following: case studies, experiments, and examples. Experiment-based validation [P1, P3, P4, P5, P6, P8, P9, P12, P14, P15, P20, P23, P24, P26, P27, P28, P30, P31, P32, P33, P34] is utilised by 60% of the studies and is the dominant method. One of these works [P8] uses a survey along with its experiment. Case study [P2, P7, P17, P19, P21, P25, P29, P35] is the next most dominant method where 22.8% of the papers applied the approach to evaluate their methods. A combination of experiment and case study [P18, P22] contributes 5.71%, and example-based methods [P11, P13] are also adopted by 5.71% of the studies.

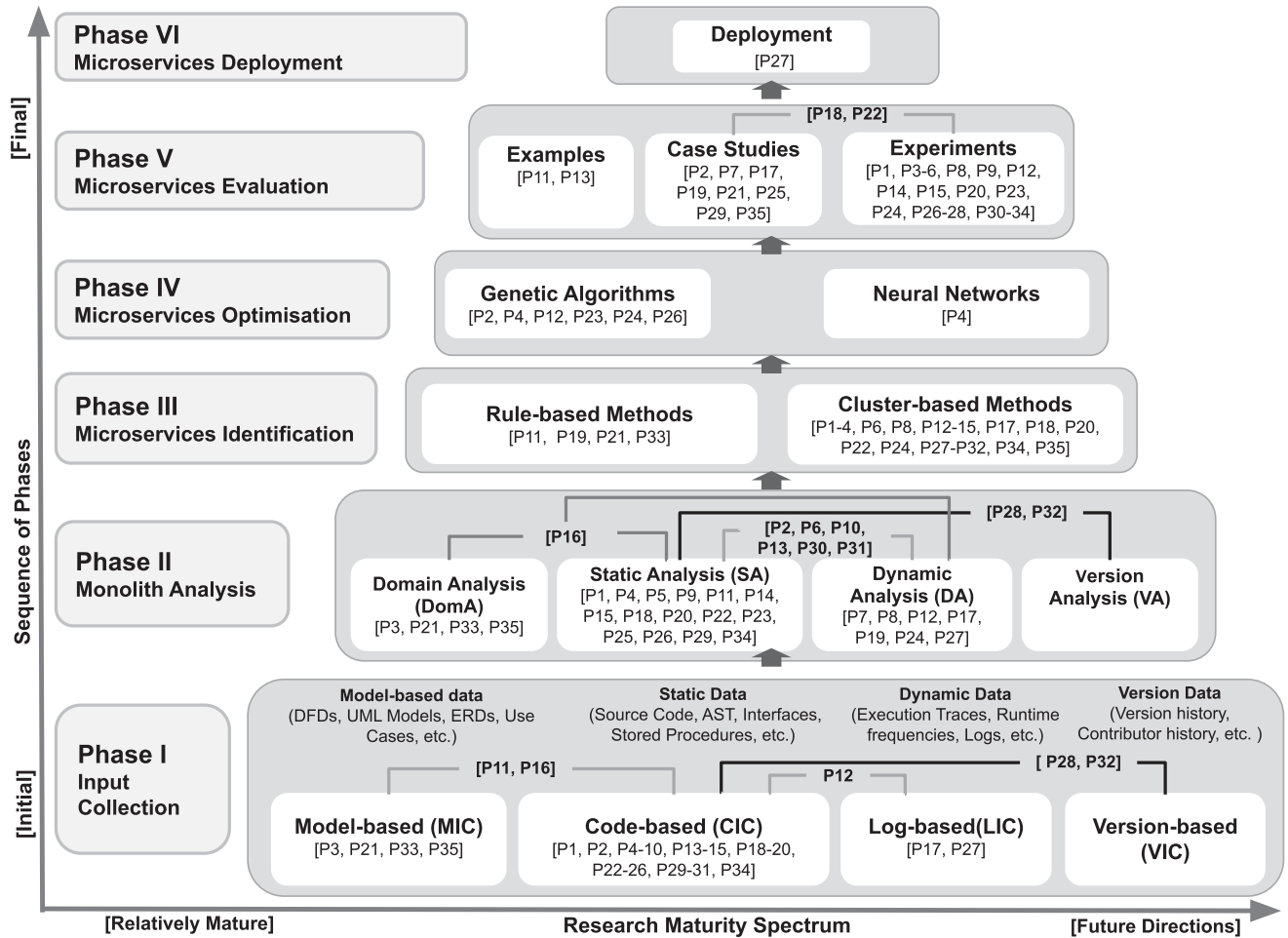


Fig. 5. Monolith to microservices decomposition framework (M2MDF) with mapping to individual studies.

Whilst at an earlier point, case studies were reported to be the primary validation techniques [126], it is clear that experiment-based validation is both of growing interest and now the prevailing method. This could be attributed to the progress made in the last few years towards the availability of widely applicable metrics and benchmarks, enabling comparison of different decomposition efforts.

c) *Evaluated Programming Languages*: A final difference across studies relates to programming languages. Here we see that the significant majority (68.6%) examine Java-based applications [P2, P3, P4, P5, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P20, P21, P22, P24, P25, P26, P28, P31, P34, P35]. PHP [P18, P30] accounts for 5.6%. Java and COBOL in combination [P29], Java, Python and Ruby in combination [P32], ASP.Net [P19], and Python [P17] each account for 2.8%. With 74.2% of studies using one or more Java-based applications, there is clearly a strong tendency for decomposition studies to incorporate Java codebases. This may be because many of the existing decomposition tools (i.e., for dynamic and static data collection and analysis) are designed for use with Java [113]. Whilst this is somewhat appropriate given the prevalence of Java-based systems, it is also an example of the “researching under the lamppost” problem [129]. As a result, legacy technologies with possibly the greatest need for translation into microservices are studied less.

E. Phase VI: Microservices Deployment

Deployment focuses on the implementation of the extracted microservices and studying whether the identification process achieves its objectives. We found just a single study [P27] that addresses this deployment phase but it does not discuss the deployment process in detail. In the examined studies, the lack of technical coverage on the deployment of the extracted microservices leaves a major gap regarding the suitability for production environments. This could be addressed, in future research, with industrial case studies where monolith applications are transformed into microservices and their quality evaluated in a production environment. Finally, Fig. 5 presents the mapping of the studies relative to the M2MDF diagram.

V. GAPS AND FUTURE RESEARCH

This section answers RQ4 by presenting the research and practice gaps in the area of monolith decomposition into microservices. We categorise these gaps according to the phases of the M2MDF.

A. Phase I: Collection Gaps

Existing research exploits MIC, CIC, LIC, VIC, or a combination of these data collection methods. These collection methods

TABLE IX
SUMMARY OF RESEARCH GAPS AND FUTURE RESEARCH DIRECTIONS

| PhaseNo | Research Gap | Future Research |
|----------------|---|---|
| Collection | FR1 Lack of efficient and comprehensive data collection methods and tools. | Efficient automated tool support for extraction and visualisation of domain data as well as static, dynamic, evolutionary, and model data is required. This may include data based on expert practice as well as some code instrumentation tools that enable the extraction of execution traces, interception of URIs and database calls from codebases written in different programming languages. |
| | FR2 Methods for an efficient combination of static, dynamic, and evolutionary data are missing. | Development of methods enabling the efficient combination of various data collection methods such as static, dynamic, evolutionary, and domain-driven data. This includes identifying suitable data representation methods. |
| Analysis | FR3 Dynamic analysis does not ensure full code coverage of the monolith application and does not show the full behaviour of the application. | Research on software testing methods that promote maximum code coverage, revisiting existing dynamic data collection methods and tools for improved coverage is essential. Dynamic code coverage from the perspective of utility classes and external libraries is another area of investigation. Coverage analysis should recognise several external packages and utility classes that are not the core components of the target monolith application. |
| | FR4 Rigorous criteria for the selection of the unit of analysis are not yet available. | Research on the selection of the appropriate granularity level of analysis (i.e., method level, fragment level, business object level, class level, and package level) should investigate which unit of analysis is appropriate to the monolith application under investigation. |
| | FR5 Business process awareness is largely unaddressed. | Research to determine the existing business processes enabled in monoliths would be of value in microservices transformations. This could also facilitate business process improvement in migrated systems. |
| Identification | FR6 Supervised machine learning methods are not yet exploited for identifying and extracting microservices candidates. | Research on the applicability of supervised machine learning methods may contribute towards training algorithms to learn from a repository of decomposed codebases. Since the availability of microservice versions of monolith applications is growing, it may be time to focus on supervised algorithms. |
| | FR7 Little research on the metrics and objective functions for optimisation. | Research focusing on: (i) the selection of metrics to serve multi-objective optimisation, (ii) the selection of optimisation algorithms, and (iii) comparison of the performance of the algorithms. |
| Optimisation | FR8 Lack of research on the applicability of neural network-based approaches and their optimisation functions. | As neural network, particularly deep learning, approaches are advancing in the clustering and classification problem spaces, it is worth considering how these techniques can be used in monolith application decomposition. |
| | FR9 The existing datasets are not representative of the size and complexity of industrial codebases. | Building a large-scale, representative dataset that can serve as a gold standard for microservices extraction experiments is another challenging but beneficial research area. Such a dataset should include appropriate test cases for data generation. Incorporation of non-Java-based systems would address the lack of attention on other programming languages. |
| Evaluation | FR10 Absence of standardised evaluation metrics and a standard open dataset. | Research towards building standard metrics and evaluation criteria. |
| | FR11 Absence of comparative studies on the performance of candidate microservices identification algorithms. | Comparative studies of clustering algorithms for candidate microservices identification in different decomposition settings will contribute towards saving the time and effort of architects when screening and implementing decomposition options. |
| | FR12 Limited use of semantic similarity metrics for the purpose of assessing candidate services. | Exploring various semantic similarity metrics, comparative study of existing source code similarity metrics and assessing the impact of such measures on the quality of the candidate microservices warrants empirical investigation. |
| Deployment | FR13 The inclusion of feedback from the deployment of candidate microservices into the decomposition process has not yet received adequate attention. | Research on methods for evaluating the performance of candidate microservices during/after the deployment of the microservices will complete the feedback loop. |
| | FR14 Insufficient tool support for the end-to-end decomposition process. | The development of an IDE that includes different collection, analysis, and identification methods. Improved visualisation for proposed microservices architectures. Incorporation of evaluation methods. |

extract SD, DD, MD, or VD to represent the structure and behaviour of the monolith application. The data provides crucial information regarding monolith implementation and organisation, however, this research suggests the following gaps.

i) MIC approaches require significant manual input and experts with a detailed understanding of the monolith. For large enterprise applications, this may prove prohibitively slow or expensive. There would appear therefore to be significant opportunity for automation of domain-driven artefact collection. This process could benefit from studying the decision-making process of experts to understand the tasks they conduct and the artefacts they focus on when they try to identify microservices. It is important to note that such artefacts may not accurately (or even approximately) reflect the actual system.

ii) Although some studies address the issue to an extent [P13, P31], data collection methods that efficiently combine static, dynamic, and evolutionary data are not yet fully employed.

This is further related to the absence of a common formal representation of the input data and its core content. Up to this point, call graphs extracted from static data have been somewhat enriched with dynamic data using the frequency of the artefacts involved in the call graph. Some papers compute the static and dynamic data separately, while others combine the two into one measure (usually in the form of a weighting). Combining collection techniques can improve the quality of the microservices identification process since additional perspectives bestow increased monolith knowledge (as identified in other domains [130]).

B. Phase II: Analysis Gaps

This research has identified the following gaps in relation to analysis phase.

i) When employed in isolation, dynamic data collection cannot guarantee full code coverage or that all operational use

case scenarios have been catered for, with the result that the sole use of dynamic analysis carries increased risk of unintended operational issues in target microservices systems. To address this problem, one study [P24] proposes the use of comprehensive functional tests that ensure full functional coverage of the monolith code. However, this has not been systematically tried and tested. Furthermore, the design and implementation of tests conveying complete functional coverage can be very expensive [131] and error-prone. Furthermore, it is foreseeable that unimagined operational use case scenarios might be accidentally overlooked. Techniques to remedy this problem should be examined in future research.

ii) Criteria for determining the appropriate unit of analysis are not presently available. Researchers and practitioners would benefit from guidance in this area. For example, method level analysis may be preferred over package level analysis if the resulting microservices are to be relatively small in size but the trade-off in complexity and monolith size need to be considered.

iii) The analysis approaches identified in this work do not examine the efficiency of existing business processes. Rather the research focus is on a technical evaluation of the monolith to identify candidate microservices. A monolith-based system that is in use for many years may not be optimised in respect of the business processes that the monolith supports. For some firms, the process of pivoting to a distributed architecture from a monolith system may require additional business-level concerns. One of these additional concerns may be the redesign of business processes delivered in the monolith system. This however is outside the scope of the current research.

C. Phase III: Identification Gaps

Existing approaches tend to focus on unsupervised methods [P1, P2, P3, P4, P6, P12, P14, P15, P17, P18, P22, P24, P25, P27, P28, P29, P30, P31, P32, P34, P35] to extract microservice candidates, followed by rule-based methods [P11, P16, P19, P21, P33] that employ domain-driven analysis. However, opportunities exist for increased use of supervised methods as well as extended examination of current unsupervised algorithms.

We found no evidence of the application of supervised machine learning approaches and reinforcement learning methods. The absence of sufficient training data to support supervised learning methods is a major constraining factor. A further inhibitor to supervised learning techniques stems from the absence of a generalised conceptual representation of the decomposition model and process. Input data and expected output data formats and structures would need to be developed in order to support future supervised learning techniques.

However, it should not be inferred that these techniques have no potential value in this space. For example, supervised techniques might increase the opportunity for human involvement in the decomposition loop. This human agent could be an established expert for the monolith under examination, and as such, could raise the quality of decisions reached. Owing to their potential as complementary or independent techniques, supervised methods and reinforcement learning techniques have been identified as future research directions.

D. Phase IV: Optimisation Gaps

The optimisation phase has to-date received only modest attention. NSGA-II [P12, P23, P24, P26] and NSGA-III [P4] [119] have both been employed. Defining optimisation functions that represent the desired characteristics of the final output requires further research, along with supporting metrics. A neural network-based approach has also been proposed [P4], perhaps indicating that deep learning algorithms hold some promise in terms of microservice candidate optimisation. Research on the use of such approaches in the decomposition process, either in combination with clustering or in isolation, can be considered to be in its very early stage. Nevertheless, advanced evolutionary approaches may become more prevalent in decomposition projects as the associated monoliths become larger and more complex.

E. Phase V: Evaluation Gaps

Having identified candidate microservices, evaluating their suitability is a vital task. Various supporting metrics have been proposed, including coupling and cohesion measures. A variety of evaluation methods have also been utilised, including experimental validation, case studies, and manual evaluations. However, current evaluations of candidate microservices generated during monolith decomposition exhibit a number of gaps.

i) There is an absence of datasets. Studies have mostly used small or medium scale (typically less than 200,000 lines of code) open-source applications, mainly developed using Java. Although Java is among the top programming languages used to build enterprise applications, C/C++ and Python are also widely used in this space [132]. COBOL, which accounts for billions of code lines in large monolith systems [133] has not received the attention it warrants. Furthermore, although recent papers show diverse test cases in their experiment [P5, P8, P24, P32], the limited overlap of test cases across the studies undermines robust evaluation. *JPetstore* is used in six studies, with *DayTrader*, *Acme Air* and *PetClinic* each used in four studies.

ii) There remains a requirement to determine appropriate decomposition evaluation metrics. Coupling, cohesion, and many other metrics are used frequently for the identification and evaluation of microservices [134]. But there is considerable variation in specific metric definitions. Standardised evaluation metrics together with benchmarking datasets are essential for candidate microservice evaluation. The availability of such resources would also enable supervised machine learning approaches by providing essential training data.

iii) Several unsupervised learning algorithms have been proposed, including variants of hierarchical clustering [P8, P15, P22, P24], kmeans [P6, P18, P27, P31], and Girvan-newman [P13, P35]. However, a comparative study of these algorithms when applied to monolith to microservices decomposition is not available. This frustrates the work of researchers and practitioners, especially in terms of algorithm selection. This gap needs to be addressed by making experimental data and the source code openly available (as in P24) to encourage comparison.

iv) The use of semantic similarity (sometimes referred to as *conceptual similarity*) is increasing in recent years [P6, P18, P24,

P32, P34]. Thus, there appears to be opportunity for further use of these techniques as an enabler of program understanding, for example by integrating semantic similarity measures such as Wu & Palmer [135], Lin [136], and Resink [137].

F. Phase VI: Deployment Gaps

To date, just a single study considers the deployment of extracted microservices [p27]. This is undesirable because the ultimate test of microservice fitness will only arise once deployed. Through examination of deployments, errors and inefficiencies in microservice extraction methods can be identified.

Though not strictly a deployment concern, there is also an absence of end-to-end tooling to automate (or partly automate) the identification, extraction and deployment of microservices. Such tooling might support the selection of collection methods with different granularity levels (method, fragment, class, package). It might also support the selection of various other key considerations, including the clustering approach and the optimisation methods. A further capability of an end-to-end tool might permit the visualisation of resulting microservices candidates, along with quality metrics to compare the candidate microservices. Initial tooling implementations such as *Service Cutter* [24] and *Kieker* demonstrate the potential in this space, but much work is yet required.

VI. THREATS TO VALIDITY

A. External Validity

The primary threat to external validity is related to the selection and inclusion of primary studies, and to the search time frame which covers the period 2015–2021. As a result, selected studies may not be fully representative of the state of the art in the decomposition of monolith applications to microservices. Furthermore, and although the authors involved in the activity were engaged full time on the associated research programme and therefore well versed in the topic, the selection of studies for inclusion was based solely on consensus. Had the selection process incorporated inter-rating to record and formally process the individual author evaluations, the process as a whole would be both more robust and more transparent. To mitigate the potential impact of threats surrounding study selection, search keywords were carefully developed, evolved, and the major computer software databases were included in the scope of the searches.

Three search strings were used and the results were systematically combined. Alternative search platforms such as (Publish or Perish,⁴ google scholar and semantic scholar) were used to identify potentially omitted articles. In addition, the SLR methodology was faithfully applied and later stages subject to group review. Likewise the analysis was subject to group review and snowballing was used to expand the paper collection. In combination, these methodological steps significantly reduce the possibility that papers were missed or that some significant earlier contributions were overlooked. As a subsequent mitigation following completion of the SLR, we performed a targeted

literature search in the period 2011–2014. No additional works of relevance were identified in this subsequent search.

Supplementary materials, available online, are likely to exist, especially in the non-academic sphere and future work could look to integrate any industry-led innovations in this important research area. Industrial applications have been proposed to address monolith decomposition into microservices, for example IBM's *Mono2Micro* tool⁵ and Amazon's *AWS Microservice Extractor for .NET*.⁶ However, the technical implementation details for these industrial applications are not publicly available.

Patents and grey literature are also not within the scope of this work and if examined, they might contain additional relevant material. Although the inclusion and exclusion criteria were strictly applied, their application in Refinement Steps 1 and 2 was conducted solely by the first author, as was the snowballing element of Refinement Step 6. Had additional authors been engaged in these steps, it would have reduced the effect of author selection bias.

In terms of the quality of the selected works, only studies that underwent rigorous peer-review were included, where leading academic publishers were the central focus of the search. These included IEEE, ACM, Springer, and Science Direct. And the review protocol should largely assuage the affect of subjective paper selection decisions, as studies relevant to the topic were selected by the consensus of the majority. Furthermore, identified works were included in the research where they fell within the search timeline, providing a focus on more up-to-date and state-of-the-art work. To the knowledge of the researchers, all works of central relevance to the research objective and within the search window have been included.

Finally, and although appropriate techniques from Grounded Theory were employed to systematically identify the major phases and their application sequence in a monolith to microservices decomposition, it is inevitably the case that universal agreement on phase boundaries, naming and sequencing, will not be possible given the broad nature of the problem and the fact that individual firms and research efforts might design alternative and specific decomposition pipelines. Nevertheless, this work is the first published study that the authors are aware of that has attempted to systematically identify and classify the general problem space as presently reported in the prominent peer review literature to date.

B. Internal Validity

It is important to highlight that technology solutions - such as is the focus of this work - may ultimately only present a partial solution to a difficult challenge. At various points in our work, we have noted the important role of human experts as part of the decomposition process. We do not envisage an immediate future devoid of this important human contribution. Rather, as codebases to be migrated inevitably present as larger and more complex, the role of supporting technology may grow to support humans charged with this important strategic activity.

⁵<https://www.ibm.com/cloud/blog/announcements/ibm-mono2micro>

⁶<https://aws.amazon.com/about-aws/whats-new/2021/11/aws-microservice-extractor-net/>

⁴<https://harzing.com/resources/publish-or-perish>

The transition from a monolith architecture to a microservices architecture can enable certain key strategic objectives, including increased frequency in the delivery of new features with less impact on an operational system (sometimes referred to as *low perturbation*). The realisation of such objectives is constrained by the cost of decomposition projects and the quality of the resulting microservices-based system. And while microservices architectures deliver certain desirable benefits for certain market offerings, they are not the only approach that may be adopted: more generalised and larger granularity service-oriented offerings, such as so-called *modular* monoliths may be a more pragmatic alternative in certain settings. However, even for modular monolith transitions, the technologies and phases identified in this work are likely to be largely relevant.

This research has created and applied the M2MDF as a means to compare existing techniques adopted in up-to-now largely disparate migration studies. The framework itself is built based on steps derived by inspecting all the implicit and explicit decomposition steps used in the selected papers, using aspects of grounded theory. Although the framework has been systematically derived, it is possible that other research efforts might have reached different classification decisions.

VII. DISCUSSION

This systematic review has produced various important contributions for practitioners. While microservices architectures have been in use for some time, the task of decomposing a monolith application into microservices can vary significantly from context to context. Furthermore, with the passage of time, improved tooling has enabled the part-automation of aspects of the process. No previous work has been identified that consolidates all these concerns into a single work and mapped the various existing contributions into a decomposition framework. This is one of the substantial contributions of this research, and it may be adopted by both practitioners and researchers to effectively explore the decomposition challenge in the context of the end-to-end process and the existing literature.

One of the greatest challenges of any monolith to microservices decomposition effort can be identified in the fact that there is uncertainty regarding the many aspects of the task. For practitioners more accustomed to designing, building and maintaining monolith-based systems, learning is required to understand the nature of microservices architectures. However, this is just the start of the decomposition activity, and a great deal of technical innovation may be required to effectively execute a decomposition project. The material presented in Section IV-C provides clear and categorised approaches to the various elements of the decomposition. In Table IV, the different methods for monolith data collection are presented, with Table V presenting the different types of analysis that may be employed. In both sections, there is a rich variety of techniques available and no single study has employed all of them. Indeed, economics may prevail in practice, and practitioners may need to choose which techniques they can deploy in line with budgets and timelines. The key point here is that all the known techniques are presented in the M2MDF, and therefore, practitioners can

quickly evaluate their options without the need for an extensive literature review.

A further challenge for practitioners arises from the decision regarding the unit of analysis to be employed. In the context of a Java system, this could be at a package level, or it could be more granular, including down to the individual method level. This is an important consideration for practitioners as the unit of analysis will affect the size of the projected microservices. Packages may initially present as intuitively appealing fragmentation points (assuming low coupling across package boundaries), but in practice this might result in relatively large microservices that may not be suited to all manners of hardware deployment. For example, large microservices may not be suited to a Function-as-a-Service paradigm [14]. Table VI presents a summary of the identified units of analysis and maps them to individual studies.

Many different techniques can be employed when attempting to identify microservices in data collected from monolith systems, and a variety of tools exist to support this activity. Section IV-C3 addresses these dimensions, identifying the tools reported in the literature and systematically elaborating the various different algorithms that can be utilised (refer to Table VII). The provision of this information in one single source consolidates the field to date and will assist practitioners in quickly navigating the available tools and algorithms to support monolith decomposition into microservices.

Evaluating the effectiveness of a proposed decomposition is perhaps the most elusive problem in the migration agenda. As presented in Table VIII, there are a various evaluation metrics in use across many different studies. While these metrics will be of use to practitioners, the broader challenge of devising a means to examine decomposition effectiveness is a task perhaps best suited to the research community. The materials presented in Section IV-D summarise the accumulated work to date which is very much fragmented. But it is nevertheless a starting point for future research seeking to work towards standardised techniques for evaluating the effectiveness of proposed microservices.

Some promising work on benchmarking has been conducted and it is presented in Section IV-C3. However, this work has some significant limitations, especially in the context of the programming languages examined. Furthermore, the absence of clear guidance on what constitutes an effective monolith decomposition contributes to challenges in this space. It is in these areas that future research can make important contributions. Without a clearer understanding of these concerns, it will not be possible to have robust and consistent examination of proposed decomposition techniques and their effectiveness in comparison to existing reported decompositions. Although this is a particularly challenging aspect of the problem domain, in the fullness of time more elements of the decomposition task may become automated and therefore, it will be essential to have consistent methods of evaluation (because the role of manual human input may be reduced).

Beyond evaluation and benchmarking, to date only modest attention (a single study) has examined the microservice deployment dimension. Future studies, especially in practice, should actively examine the deployment phase. Microservices

should be both deployable and effective once deployed, otherwise the substantial effort to identify microservices might be undermined.

While this research demonstrates that static analysis is the most commonly used analysis technique (refer to Section IV-C1), this does not necessarily imply that it is the most effective approach. Rather, it may be the case that static analysis is more accessible. Various tools exist to support static code analysis, it is a long-established field. Furthermore, there is no requirement to install a monolith system and generate traffic through it in order to obtain static data. In contrast, to obtain dynamic data, a monolith system must be executing and processing requests. This may require the isolation of test bed infrastructure and the execution of test cases. Although there is a greater set up time and cost associated with the collection of dynamic data, the fact that the dynamic data (e.g., stack trace) identifies the actual internal runtime behaviour of a monolith is potentially very valuable. Whereas static analysis can present the various theoretical considerations (e.g., all possible paths and dependencies), a robust set of test cases can identify the actual considerations (e.g., the actual paths navigated at runtime under expected traffic). It is perhaps the case therefore that the concurrent use of static and dynamic data may facilitate greater pragmatism and effectiveness in monolith decomposition.

A key consideration for any decomposition project is the scope and type of data storage employed in an existing monolith. Where a monolith employs a large, centralised data store (for example, a relational database), the decomposition into microservices can be particularly complicated. If constructs such as database stored procedures are utilised, then some of the monolith program logic may in effect be implemented inside the database. This raises various issues, including the impact on static code analysis which may not automatically examine internal database logic. Related research has partially addressed this challenge, for example through the use of *Dbeaver* in [P16] (refer to Section IV-C3) but various other studies largely ignore the data layer which might present as a fundamental limitation. This research suggests that an examination of the data layer should be one of the first considerations for any decomposition project.

On a point of terminology, this research has identified that there is some terminological inconsistency across the research domain. Most notably, this arises in the case of the core focus: monolith decomposition into microservices. It is not uncommon for related research to refer to this as a *migration*, for example in the case of [4], [31], [32], as opposed to a *decomposition*. Clearly, either term could be utilised, but perhaps the term *decomposition* is more appropriate given that any firm seeking to rearchitect a monolith into microservices will also have various other migration-oriented tasks. For example, the firm may require organisational adjustments, and to do so business processes may need to be adapted in order to pivot to a more continuous form of software engineering [138]. In this work, therefore, we prefer the term *decomposition* over *migration* and it is for this reason that the framework produced in this study has adopted the title: Monolith to Microservices Decomposition Framework.

A. Post-Review Reflection

To evaluate if the literature review was representative of the up-to-date material in the field, a further post-review reflection was undertaken. In this step, we applied the SLR protocol and conducted the literature search for the time period of October 2021 to April 2023. A total of 2712 additional studies were discovered, and following application of the original refinement steps, 21 new studies were identified. The temporal distribution shows a growth of publications with one additional study in 2021 and 18 studies in 2022. Two studies were identified in the period January 2023 to April 2023. All additional studies in the 2021 calendar year [134] and the 2023 year to April [139], [140] were included. Since the volume of literature in 2022 is large, eight studies were randomly selected for consideration [141], [142], [143], [144], [145], [146], [147], [148]. The first author extracted the information from the studies to be used in evaluating the proposed M2MDF and possibly to also highlight new findings.

The majority of the studies [139], [144], [145], [147], [148] used CIC for data collection and collected SD to perform static analysis. Three studies [141], [142], [143] collected SD+DD and performed SA+DA on the data, and one study [140] collected DD and performed DA. One study [134] used CIC+VIC for data collection and performed SA+VA, and another study [146] used CIC+MIC to collect the data and applied SA+DomA to extract microservices. These findings are consistent with the data-collection and data analysis findings reported on in Section IV-C.

While classes remain the most dominant unit of analysis [134], [139], [140], [141], [143], [144], [145], [148], other units of analysis are also employed, such as methods [142] and the combination of software artefacts and methods [146]. A new development here is that one study [147] presented its unit of analysis purely based on APIs that are extracted from the codebases using the *OpenAPI (Swagger)* tool. Concerning input representation, a graph is the most common input data representation method [139], [140], [142], [143], [144], [145], [147], [148] followed by matrix or vector [134], [141], and in one case the representation was unspecified [146]. In many of these studies, the graph representations are later converted into a matrix to serve as an input for clustering algorithms. Regarding microservices identification tools, *Kicker* [142], *Java Call Graph* [134], *Service Cutter* [146] and *MoDISCO/DISCO* [145], [146] are used. Several hierarchical clustering algorithms are used with new variants of the existing hierarchical algorithms. Neural network-based hierarchical clustering [146] and graph deep clustering [140] are new additions. One study [140] used Loss Function for the optimisation of their graph deep clustering algorithm.

In terms of the use of metrics, variations of cohesion, coupling and non-functional metrics are widely used. An increase in the number of studies using more than one metric and comparing previous microservices extraction methods over legacy codebases as benchmarks are widely observed [140], [141], [142], [143], [144], [145], [148]. Precision, recall, and accuracy are further reported in recent studies, that could be considered as a piece of evidence towards a growing interest in supervised

machine learning approaches [139], [145], [146], [147]. The majority of the studies [140], [141], [142], [143], [144], [145], [146], [148] use one or more of the evaluation codebases such as *JPetStore*, *DayTrader*, *Acme Air*, *Petclinic* and *Cargo Tracking System*. However, new evaluation codebases, such as *Plants* [140], [141], [143], [148] and *MyITS* [139], are also observed. Java is the dominant evaluated programming language that is used throughout these selected studies [134], [140], [141], [142], [143], [144], [145], [146], [147], [148]. One study [139] developed an extraction tool for codebases that are written in the PHP programming language.

These results from this post-review reflection indicate that the proposed M2MDF framework captures the major methods for input collection, monolith analysis, microservices identification, and evaluation metrics.

VIII. CONCLUSION

This research has identified 35 studies using a SLR protocol and snowballing method, and systematically reviewed studies examining the decomposition of monolithic applications into microservices. Various existing literature addresses aspects of the decomposition task, but up to this point, there is no single published study that organises and addresses the end-to-end decomposition task. We find that monolith decomposition into microservices is a complicated, large, varied and relatively new task. However, with the demand for more frequent software delivery and innovations in cloud computing that support distributed software systems, it is the view of the authors that monolith decomposition into microservices is a domain of importance for software engineering. The evidence in the literature suggests growing interest in this domain.

This research observed an absence of a description of the end-to-end decomposition process. We therefore applied elements of Grounded Theory to systematically develop the M2MDF as a means to outline the key tasks that comprise monolith decomposition. This enabled the research to address RQ1 (*What are the primary phases of monolith-to-microservices decomposition and the major constituent elements of those phases?*). The primary phases of the M2MDF are: I. Input Collection, II. Monolith Analysis, III. Microservices Identification, IV. Microservices Optimisation, V. Microservices Evaluation, and VI. Microservices Deployment.

Within *Input Collection*, we identify four major methods for collection: Model-based (concerned with Specification/Design artefacts), Code-based (concerned with the source code), Log-based (concerned with logs produced by a system), and Version-based (concerned with changes to systems over time). Using the data obtained from these collection methods, we find that there are four major *Monolith Analysis* methods: Domain Analysis (concerned with discovering domains/themes in a system), Static Analysis (analysis conducted without executing a system), Dynamic Analysis (analysis of systems at run time), and Version Analysis (analysis of changes to systems over time).

The *Microservices Identification* phase employs two major methods: Rule-based methods (human input, defining partitioning rules) and Clustering methods (applying unsupervised machine learning algorithms). In the *Microservices*

Optimisation phase, candidate microservices are optimised using two identified techniques: Genetic Algorithms (to refine optimal combinations of software units within microservices) and Neural Networks (as a means to cluster classes into microservices). Through systematic examination of M2MDF phases I-IV, this research has answered RQ2 (*What are the existing approaches, tools and methods observed in the decomposition of monolith applications into microservices?*)

In *Microservice Evaluation*, we find three distinct evaluation approaches: Case Studies (in-depth evaluation of individual decomposition), Examples (manual evaluation of individual decomposition), and Experiments (experimenting with different decomposition approaches). By systematically investigating this M2MDF phase, this research has answered RQ3 (*What are the metrics, datasets, and benchmarks used for evaluating and validating monolith decomposition into microservices?*). Finally, the *Microservices Deployment* phase is concerned with observing extracted microservices in production systems. Just a single study reports attempting microservices deployment at this time.

Given the relative newness of this area and the potential for growing interest in this space, we identify key areas for future research. A summary of the gaps identified is presented in Table IX, which answers RQ4 (*What research gaps can be identified in the current literature?*). The review suggests that major gaps exist in several areas. Much of the reported work to date is focused on just a small number of programming languages and is heavily biased towards Java. Many large enterprise monolith systems have been built using technologies other than Java (e.g., COBOL and C/C++) and we suggest that this is an area that requires more attention.

Gaps also arise from the inconsistent use of metrics at the present time. Coupling and cohesion are regularly utilised, but they are not consistently measured across the studies. As a result, it is not possible to reliably compare the effectiveness of methods proposed in different studies. Future work could seek to establish a standard set of metrics for use in monolith analysis and microservices identification. And it is not just in the analysis and identification phases that metrics are required. There is a clear absence of consistent evaluation of resulting microservices, for which the publication of datasets is warranted. A dataset should comprise various elements: including the monolith source code, the extracted microservices, and should identify the metrics utilised at the various stages. In addition to the analysis and evaluation metrics identified above, non-functional metrics should also be employed. For example, metrics to report on performance and hardware utilisation.

It is the view of the authors that the material produced by this research can have an important impact on a complicated pursuit in contemporary software engineering. The relentless drive towards better, faster, cheaper software has entered a disruptive and challenging new phase. Because they are large in size, legacy systems based on monolith architectures are slow to deploy and, when making small changes, large components might need to be rebuilt, retested and redeployed. Monoliths may be too large to be deployed to certain serverless infrastructure, for example Function-as-a-Service [14]. As a result, the potential economic benefits of services such as AWS

Lambda, Google Cloud Functions and Microsoft Azure Functions (where hardware is available on demand and is charged in a pay-per-use model) are not realisable.

A word of caution is warranted. Although monolith architectures may inhibit faster and cheaper software objectives, they have proven resilient and useful over a lengthy time period. The structure and organisation permitted in monolith architectures can be amenable to developer understanding of systems. Furthermore, the development of monolith systems can be considered to be substantially refined at this point; many large operational systems are based on the monolith architecture and they are effective in delivering their functionality. While the march of technological innovation continues and economic arguments advocate emerging constructs such as serverless computing, monolith to microservice decompositions are large and expensive, and they are not risk free. It is also the case that to be successful, the broader migration activity may require adaptation to various other processes. For example, the hardware provisioning model may change and other work practices may need to be reengineered. Microservices may not be warranted in some contexts.

The work presented in this research can assist researchers and practitioners tasked with monolith decomposition into microservices. It can help firms understand decomposition tasks more completely, thereby assisting decisions surrounding proposed decompositions. Where decompositions are sanctioned, the M2MDF identifies the major options available to practitioners. For researchers, the present literature on monolith decomposition into microservices is fragmented. This research consolidates the available literature and organises the decomposition landscape. In addition to this important contribution, various significant gaps are identified. The research community can address these gaps in areas such as extending programming language support, introducing robust and standardised metrics, and in producing datasets for the consistent evaluation of migrations.

REFERENCES

- [1] N. Dragoni et al., "Microservices: Yesterday, today, and tomorrow," in *Present and Ulterior Software Engineering*, M. Mazzara and B. Meyer, Eds., Cham, Switzerland: Springer, 2017, pp. 195–216. [Online]. Available: https://doi.org/10.1007/978-3-319-67425-4_12
- [2] M. Kalske, N. Mäkitalo, and T. Mikkonen, "Challenges when moving from monolith to microservice architecture," in *Proc. Int. Conf. Web Eng.*, I. Garrigós and M. Wimmer, Eds., Cham, Switzerland: Springer, 2018, pp. 32–47. [Online]. Available: https://doi.org/10.1007/978-3-319-74433-9_3
- [3] N. C. Mendonça, C. Box, C. Manolache, and L. Ryan, "The monolith strikes back: Why Istio migrated from microservices to a monolithic architecture," *IEEE Softw.*, vol. 38, no. 5, pp. 17–22, Sep./Oct. 2021. [Online]. Available: <https://doi.org/10.1109/MS.2021.3080335>
- [4] A. Balalaie, A. Heydarnoori, P. Jamshidi, D. A. Tamburri, and T. Lynn, "Microservices migration patterns," *Softw.: Pract. Experience*, vol. 48, no. 11, pp. 2019–2042, 2018. [Online]. Available: <https://doi.org/10.1002/spe.2608>
- [5] D. Taibi, V. Lenarduzzi, and C. Pahl, "Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation," *IEEE Cloud Comput.*, vol. 4, no. 5, pp. 22–32, Sep./Oct. 2017. [Online]. Available: <https://doi.org/10.1109/MCC.2017.4250931>
- [6] M. Ahmadvand and A. Ibrahim, "Requirements reconciliation for scalable and secure microservice (de)composition," in *Proc. IEEE 24th Int. Requirements Eng. Conf. Workshops*, Los Alamitos, CA, USA, 2016, pp. 68–73. [Online]. Available: <https://doi.ieeecomputersociety.org/10.1109/REW.2016.026>
- [7] F. Auer, V. Lenarduzzi, M. Felderer, and D. Taibi, "From monolithic systems to microservices: An assessment framework," *Inf. Softw. Technol.*, vol. 137, 2021, Art. no. 106600. [Online]. Available: <https://doi.org/10.1016/j.infsof.2021.106600>
- [8] P. Clarke, R. V. O'Connor, and B. Leavy, "A complexity theory viewpoint on the software development process and situational context," in *Proc. Int. Conf. Softw. Syst. Process*, New York, NY, USA, 2016, pp. 86–90. [Online]. Available: <https://doi.org/10.1145/2904354.2904369>
- [9] O. Zimmermann, "Microservices tenets," *Comput. Sci. Res. Develop.*, vol. 32, no. 3, pp. 301–310, Jul. 2017. [Online]. Available: <https://doi.org/10.1007/s00450-016-0337-0>
- [10] S. Newman, *Building Microservices*, 1st ed. Sebastopol, CA, USA: O'Reilly Media, Inc., 2015.
- [11] D. Wolfart et al., "Modernizing legacy systems with microservices: A roadmap," in *Proc. Int. Conf. Eval. Assessment Softw. Eng.*, New York, NY, USA, 2021, pp. 149–159. [Online]. Available: <https://doi.org/10.1145/3463274.3463334>
- [12] T. Smith, "New research shows 63% of enterprises are adopting microservices architectures," Sep. 2018. Accessed: Feb. 01, 2021. [Online]. Available: <https://dzone.com/articles/new-research-shows-63-percent-of-enterprises-are-a>
- [13] S. W. Schütz, T. Kude, and K. M. Popp, "The impact of software-as-a-service on software ecosystems," in *Proc. Int. Conf. Softw. Bus.*, G. Herzworm and T. Margaria, Eds., Berlin, Germany: Springer, 2013, pp. 130–140.
- [14] J. Grogan et al., "A multivocal literature review of function-as-a-service (FaaS) infrastructures and implications for software developers," in *Systems, Software and Services Process Improvement*, M. Yilmaz, J. Niemann, P. Clarke, and R. Messnarz, Eds., Cham, Switzerland: Springer, 2020, pp. 58–75. [Online]. Available: https://doi.org/10.1007/978-3-030-56441-4_5
- [15] M. Villamizar et al., "Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS Lambda architectures," *Serv. Oriented Comput. Appl.*, vol. 11, no. 2, pp. 233–247, Jun. 2017. [Online]. Available: <https://doi.org/10.1007/s11761-017-0208-y>
- [16] M. Loukides and S. Swoyer, "Microservices adoption in 2020," Jul. 2020. Accessed: Sep. 10, 2021. [Online]. Available: <https://www.oreilly.com/radar/microservices-adoption-in-2020/>
- [17] IBM Corporation, "Microservices in the enterprise, 2021: Real benefits, worth the challenges," Mar. 2021. Accessed: Sep. 10, 2021. [Online]. Available: <https://www.ibm.com/downloads/cas/OQG4AJAM>
- [18] M. Daoud, A. El Mezouari, N. Faci, D. Benslimane, Z. Maamar, and A. El Fazziki, "A multi-model based microservices identification approach," *J. Syst. Architecture*, vol. 118, 2021, Art. no. 102200. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1383762121001442>
- [19] S. Li et al., "A dataflow-driven approach to identifying microservices from monolithic applications," *J. Syst. Softw.*, vol. 157, 2019, Art. no. 110380. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0164121219301475>
- [20] R. Chen, S. Li, and Z. Li, "From monolith to microservices: A dataflow-driven approach," in *Proc. IEEE 24th Asia-Pacific Softw. Eng. Conf.*, Nanjing, 2017, pp. 466–475. [Online]. Available: <https://doi.org/10.1109/APSEC.2017.53>
- [21] I. Saidani, A. Ouni, M. W. Mkaouer, and A. Saied, "Towards automated microservices extraction using multi-objective evolutionary search," in *Proc. 17th Int. Conf. Serv.-Oriented Comput.*, S. Yangui, I. Bouassida Rodriguez, K. Drira, and Z. Tari, Eds., Cham, Switzerland: Springer, 2019, pp. 58–63. [Online]. Available: https://doi.org/10.1007/978-3-030-33702-5_5
- [22] A. Furda, C. Fidge, O. Zimmermann, W. Kelly, and A. Barros, "Migrating enterprise legacy source code to microservices: On multitenancy, statefulness, and data consistency," *IEEE Softw.*, vol. 35, no. 3, pp. 63–72, May/Jun. 2018. [Online]. Available: <https://doi.org/10.1109/MS.2017.440134612>
- [23] A. Selmadji, A.-D. Seriai, H. L. Bouziane, C. Dony, and R. O. Mahamane, "Re-architecting OO software into microservices," in *Proc. Eur. Conf. Serv.-Oriented Cloud Comput.*, K. Kritikos, P. Plebani, and F. de Paoli Eds., Cham, Switzerland: Springer, 2018, pp. 65–73. [Online]. Available: https://doi.org/10.1007/978-3-319-99819-0_5
- [24] M. Gysel, L. Kölbener, W. Giersche, and O. Zimmermann, "Service cutter: A systematic approach to service decomposition," in *Proc. Eur. Conf. Serv.-Oriented Cloud Comput.*, M. Aiello, E. B. Johnsen, S. Dustdar, and I. Georgievski, Eds., Cham, Switzerland: Springer, 2016, pp. 185–200. [Online]. Available: https://doi.org/10.1007/978-3-319-44482-6_12

- [25] W. Jin, T. Liu, Y. Cai, R. Kazman, R. Mo, and Q. Zheng, "Service candidate identification from monolithic systems based on execution traces," *IEEE Trans. Softw. Eng.*, vol. 47, no. 5, pp. 987–1007, May 2021. [Online]. Available: <https://doi.org/10.1109/TSE.2019.2910531>
- [26] S. Eski and F. Buzluca, "An automatic extraction approach: Transition to microservices architecture from monolithic application," in *Proc. 19th Int. Conf. Agile Softw. Develop.: Companion*, New York, NY, USA, 2018, Art. no. 25. [Online]. Available: <https://doi.org/10.1145/3234152.3234195>
- [27] A. A. C. De Alwis, A. Barros, A. Polyvyanyy, and C. Fidge, "Function-splitting heuristics for discovery of microservices in enterprise systems," in *Service-Oriented Computing*, C. Pahl, M. Vukovic, J. Yin, and Q. Yu, Eds., Cham, Switzerland: Springer, 2018, pp. 37–53. [Online]. Available: https://doi.org/10.1007/978-3-030-03596-9_3
- [28] B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering – A systematic literature review," *Inf. Softw. Technol.*, vol. 51, no. 1, pp. 7–15, 2009. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0950584908001390>
- [29] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil, "Lessons from applying the systematic literature review process within the software engineering domain," *J. Syst. Softw.*, vol. 80, no. 4, pp. 571–583, 2007. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S016412120600197X>
- [30] B. Kitchenham, "Procedures for performing systematic reviews," Keele Univ., Keele, U.K., vol. 33, pp. 1–26, 2004. [Online]. Available: <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=29890a936639862f45cb9a987dd599dce9759bf5>
- [31] H. S. da Silva, G. Carneiro, and M. Monteiro, "Towards a roadmap for the migration of legacy software systems to a microservice based architecture," in *Proc. 9th Int. Conf. Cloud Comput. Serv. Sci.*, SciTePress, 2019, pp. 37–47. [Online]. Available: <https://doi.org/10.5220/0007618400370047>
- [32] V. Velepucha and P. Flores, "Monoliths to microservices - migration problems and challenges: A SMS," in *Proc. IEEE 2nd Int. Conf. Inf. Syst. Softw. Technol.*, 2021, pp. 135–142. [Online]. Available: <https://doi.org/10.1109/ICI2ST51859.2021.00027>
- [33] F. Ponce, G. Márquez, and H. Astudillo, "Migrating from monolithic architecture to microservices: A rapid review," in *Proc. IEEE 38th Int. Conf. Chilean Comput. Sci. Soc.*, Concepción, Chile, 2019, pp. 1–7. [Online]. Available: <https://doi.org/10.1109/SCCC49216.2019.8966423>
- [34] J. Ghofrani and D. Lübke, "Challenges of microservices architecture: A survey on the state of the practice," in *Proc. 10th Central Eur. Workshop Serv. Comp.*, N. Herzberg, C. Hochreiner, O. Kopp, and J. Lenhard, Eds., Dresden, Germany: CEUR-WS.org, 2018, pp. 1–8. [Online]. Available: <http://ceur-ws.org/Vol-2072/paper1.pdf>
- [35] M. F. Gholami, F. Daneshgar, G. Low, and G. Beydoun, "Cloud migration process—A survey, evaluation framework, and open challenges," *J. Syst. Softw.*, vol. 120, pp. 31–69, 2016. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0164121216300966>
- [36] R. Lichtenthaler, M. Prechtel, C. Schwille, T. Schwartz, P. Cezanne, and G. Wirtz, "Requirements for a model-driven cloud-native migration of monolithic web-based applications," *SICS Softw.-Intensive Cyber-Phys. Syst.*, vol. 35, pp. 89–100, Aug. 2020. [Online]. Available: <https://doi.org/10.1007/s00450-019-00414-9>
- [37] L. Carvalho, A. Garcia, W. K. G. Assunção, R. Bonifácio, L. P. Tizzei, and T. E. Colanzi, "Extraction of configurable and reusable microservices from legacy systems: An exploratory study," in *Proc. 23rd Int. Syst. Softw. Product Line Conf.*, New York, NY, USA, 2019, pp. 26–31. [Online]. Available: <https://doi.org/10.1145/3336294.3336319>
- [38] P. Mahanta and S. Chouta, "Translating a legacy stack to microservices using a modernization facade with performance optimization for container deployments," in *Proc. OTM Confederated Int. Conf. "On Move to Meaningful Internet Syst."*, C. Debruyne, H. Panetto, W. Guédria, P. Bollen, I. Ciuciu, G. Karabatis, and R. Meersman, Eds., Cham, Switzerland: Springer, 2020, pp. 143–154. [Online]. Available: https://doi.org/10.1007/978-3-030-40907-4_14
- [39] P. Jamshidi, C. Pahl, and N. C. Mendonça, "Pattern-based multi-cloud architecture migration," *Softw.: Pract. Experience*, vol. 47, no. 9, pp. 1159–1184, 2017. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2442>
- [40] M. Abdellatif et al., "A taxonomy of service identification approaches for legacy software systems modernization," *J. Syst. Softw.*, vol. 173, 2021, Art. no. 110868. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0164121220302582>
- [41] V. Bushong et al., "On microservice analysis and architecture evolution: A systematic mapping study," *Appl. Sci.*, vol. 11, no. 17, 2021, Art. no. 7856. [Online]. Available: <https://www.mdpi.com/2076-3417/11/17/7856>
- [42] H. C. D. S. Filho and G. D. F. Carneiro, "Strategies reported in the literature to migrate to microservices based architecture," in *Proc. 16th Int. Conf. Inf. Technol.-New Gener.*, S. Latifi, Ed., Cham, Switzerland: Springer, 2019, pp. 575–580. [Online]. Available: https://doi.org/10.1007/978-3-030-14070-0_81
- [43] M. Garriga, "Towards a taxonomy of microservices architectures," in *Software Engineering and Formal Methods*, A. Cerone and M. Roveri, Eds., Cham, Switzerland: Springer, 2018, pp. 203–218. [Online]. Available: https://doi.org/10.1007/978-3-319-74781-1_15
- [44] J. Fritzsche, J. Bogner, A. Zimmermann, and S. Wagner, "From monolith to microservices: A classification of refactoring approaches," in *Proc. Workshop Softw. Eng. Aspects Continuous Develop. New Paradigms Softw. Prod. Deployment*, J.-M. Bruel, M. Mazzara, and B. Meyer, Eds., Cham, Switzerland: Springer, 2019, pp. 128–141. [Online]. Available: https://doi.org/10.1007/978-3-030-06019-0_10
- [45] Nortal, "Are monolithic software applications doomed for extinction?," 2017. Accessed: Jan. 10, 2023. [Online]. Available: <https://nortal.com/blog/are-monolithic-software-applications-doomed-for-extinction>
- [46] F. Despoudis, "Understanding solid principles: Single responsibility," 2017. Accessed: Oct. 10, 2020. [Online]. Available: <https://codeburst.io/understanding-solid-principles-single-responsibility-b7c7ec0bf80>
- [47] M. Fowler and J. Lewis, "Microservices," 2014. Accessed: Oct. 10, 2020. [Online]. Available: <http://martinfowler.com/articles/microservices.html>
- [48] R. C. Martin, *Agile Software Development: Principles, Patterns, and Practices*. Englewood Cliffs, NJ, USA: Prentice Hall, 2002.
- [49] P. Di Francesco, P. Lago, and I. Malavolta, "Migrating towards microservice architectures: An industrial survey," in *Proc. IEEE Int. Conf. Softw. Architecture*, Seattle, WA, USA, 2018, pp. 29–2909. [Online]. Available: <https://doi.org/10.1109/ICSA.2018.00012>
- [50] C. Tozzi, "6 reasons not to adopt microservices," Jan. 2018. Accessed: Feb. 17, 2021. [Online]. Available: <https://containerjournal.com/features/microservices-use-not-use-question/>
- [51] M. Waseem, P. Liang, and M. Shahin, "A systematic mapping study on microservices architecture in devops," *J. Syst. Softw.*, vol. 170, 2020, Art. no. 110798. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0164121220302053>
- [52] H. Vural, M. Koyuncu, and S. Guney, "A systematic literature review on microservices," in *Proc. Int. Conf. Comput. Sci. Appl.*, O. Gervasi, B. Murgante, S. Misra, G. Borruso, C. M. Torre, A. M. A. Rocha, D. Taniar, B. O. Apduhan, E. Stankova, and A. Cuzzocrea, Eds., Cham, Switzerland: Springer, 2017, pp. 203–217. [Online]. Available: https://doi.org/10.1007/978-3-319-62407-5_14
- [53] S. Hassan, R. Bahsoon, and R. Kazman, "Microservice transition and its granularity problem: A systematic mapping study," *Softw.: Pract. Experience*, vol. 50, no. 9, pp. 1651–1681, 2020. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2869>
- [54] B. Cartaxo, G. Pinto, and S. Soares, "The role of rapid reviews in supporting decision-making in software engineering practice," in *Proc. 22nd Int. Conf. Eval. Assessment Softw. Eng.*, New York, NY, USA, 2018, pp. 24–34. [Online]. Available: <https://doi.org/10.1145/3210459.3210462>
- [55] P. Di Francesco, P. Lago, and I. Malavolta, "Architecting with microservices: A systematic mapping study," *J. Syst. Softw.*, vol. 150, pp. 77–97, 2019. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0164121219300019>
- [56] J. Kazanavičius and D. Mažeika, "Migrating legacy software to microservices architecture," in *Proc. IEEE Open Conf. Elect. Electron. Inf. Sci.*, Vilnius, Lithuania, 2019, pp. 1–5. [Online]. Available: <https://doi.org/10.1109/eStream.2019.8732170>
- [57] M. Cojocar, A. Uta, and A. Oprescu, "Attributes assessing the quality of microservices automatically decomposed from monolithic applications," in *Proc. IEEE 18th Int. Symp. Parallel Distrib. Comput.*, Amsterdam, The Netherlands, 2019, pp. 84–93. [Online]. Available: <https://doi.org/10.1109/ISPDC.2019.00021>
- [58] C. Pahl and P. Jamshidi, "Microservices: A systematic mapping study," in *Proc. 6th Int. Conf. Cloud Comput. Serv. Sci.*, 2016, pp. 137–146. [Online]. Available: <https://doi.org/10.5220/0005785501370146>
- [59] C. Pautasso, O. Zimmermann, M. Amundsen, J. Lewis, and N. Josuttis, "Microservices in practice, part 1: Reality check and service design," *IEEE Softw.*, vol. 34, no. 1, pp. 91–98, Jan./Feb. 2017. [Online]. Available: <https://doi.org/10.1109/MS.2017.24>
- [60] O. Al-Debagy and P. Martinek, "A microservice decomposition method through using distributed representation of source code," *Scalable Comput.*, vol. 22, pp. 39–52, Feb. 2021. [Online]. Available: <https://doi.org/10.12694/scpe.v22i1.1836>

- [61] W. K. G. Assunção et al., "A multi-criteria strategy for redesigning legacy features as microservices: An industrial case study," in *Proc. IEEE Int. Conf. Softw. Anal. Evol. Reengineering*, 2021, pp. 377–387. [Online]. Available: <https://doi.org/10.1109/SANER50967.2021.00042>
- [62] U. Desai, S. Bandyopadhyay, and S. G. Tamilselvam, "Graph neural network to dilute outliers for refactoring monolith application," in *Proc. 35th AAAI Conf. Artif. Intell. 33rd Conf. Innov. Appl. Artif. Intell. 11th Symp. Educ. Adv. Artif. Intell.*, 2021, pp. 72–80.
- [63] M. Brito, J. Cunha, and J. A. Saraiva, "Identification of microservices from monolithic applications through topic modelling," in *Proc. 36th Annu. ACM Symp. Appl. Comput.*, New York, NY, USA, 2021, pp. 1409–1418. [Online]. Available: <https://doi.org/10.1145/3412841.3442016>
- [64] A. A. C. De Alwis, A. Barros, C. Fidge, and A. Polyvyanyy, "Microservice modularisation of monolithic enterprise systems for embedding in industrial IoT networks," in *Proc. Int. Conf. Adv. Inf. Syst. Eng.*, M. La Rosa, S. Sadiq, and E. Teniente, Eds., Cham, Switzerland: Springer, 2021, pp. 432–448. [Online]. Available: https://doi.org/10.1007/978-3-030-79382-1_26
- [65] A. F. A. A. Freire, A. F. Sampaio, L. H. L. Carvalho, O. Medeiros, and N. C. Mendonça, "Migrating production monolithic systems to microservices using aspect oriented programming," *Softw.: Pract. Experience*, vol. 51, no. 6, pp. 1280–1307, 2021. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2956>
- [66] A. K. Kalia, J. Xiao, R. Krishna, S. Sinha, M. Vukovic, and D. Banerjee, "Mono2Micro: A practical and effective tool for decomposing monolithic Java applications to microservices," in *Proc. 29th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, New York, NY, USA, 2021, pp. 1214–1224. [Online]. Available: <https://doi.org/10.1145/3468264.3473915>
- [67] S. Agarwal et al., "Monolith to microservice candidates using business functionality inference," in *Proc. IEEE Int. Conf. Web Serv.*, 2021, pp. 758–763. [Online]. Available: <https://doi.org/10.1109/ICWS53863.2021.00104>
- [68] D. Taibi and K. Systä, "A decomposition and metric-based evaluation framework for microservices," in *Proc. Int. Conf. Cloud Comput. Serv. Sci.*, D. Ferguson, V. Méndez Muñoz, C. Pahl, and M. Helfert, Eds., Cham, Switzerland: Springer, 2020, pp. 133–149. [Online]. Available: <https://doi.org/10.48550/arXiv.1908.08513>
- [69] A. Bucchiarone, K. Soysal, and C. Guidi, "A model-driven approach towards automatic migration to microservices," in *Proc. Int. Workshop Softw. Eng. Aspects Continuous Develop. New Paradigms Softw. Prod. Deployment*, J.-M. Bruel, M. Mazzara, and B. Meyer, Eds., Cham, Switzerland: Springer, 2020, pp. 15–36. [Online]. Available: https://doi.org/10.1007/978-3-030-39306-9_2
- [70] Y. Zhang, B. Liu, L. Dai, K. Chen, and X. Cao, "Automated microservice identification in legacy systems with functional and non-functional metrics," in *Proc. IEEE Int. Conf. Softw. Architecture*, Salvador, Brazil, 2020, pp. 135–145. [Online]. Available: <https://doi.org/10.1109/ICSA47634.2020.00021>
- [71] T. Matias, F. F. Correia, J. Fritzsche, J. Bogner, H. S. Ferreira, and A. Restivo, "Determining microservice boundaries: A case study using static and dynamic software analysis," in *Proc. Eur. Conf. Softw. Architecture*, A. Jansen, I. Malavolta, H. Muccini, I. Ozkaya, and O. Zimmermann, Eds., Cham, Switzerland: Springer, 2020, pp. 315–332.
- [72] O. Al-Debagy and P. Martinek, "Extracting microservices' candidates from monolithic applications: Interface analysis and evaluation metrics approach," in *Proc. IEEE 15th Int. Conf. Syst. Syst. Eng.*, Budapest, Hungary, 2020, pp. 289–294. [Online]. Available: <https://doi.org/10.1109/SoSE50414.2020.9130466>
- [73] A. Selmadji, A. Seriai, H. L. Bouziane, R. Oumarou Mahamane, P. Zaragoza, and C. Dony, "From monolithic architecture style to microservice one based on a semi-automatic approach," in *Proc. IEEE Int. Conf. Softw. Architecture*, Salvador, Brazil, 2020, pp. 157–168. [Online]. Available: <https://doi.org/10.1109/ICSA47634.2020.00023>
- [74] A. Krause, C. Zirkelbach, W. Hasselbring, S. Lenga, and D. Kröger, "Microservice decomposition via static and dynamic analysis of the monolith," in *Proc. IEEE Int. Conf. Softw. Architecture Companion*, Salvador, Brazil, 2020, pp. 9–16. [Online]. Available: <https://doi.org/10.1109/ICSA-C50368.2020.00011>
- [75] D. Bajaj, U. Bharti, A. Goel, and S. C. Gupta, "Partial migration for re-architecting a cloud native monolithic application into microservices and FaaS," in *Proc. Int. Conf. Inf. Commun. Comput. Technol.*, C. Badica, P. Liatsis, L. Kharb, and D. Chahal, Eds., Singapore: Springer, 2020, pp. 111–124. [Online]. Available: https://doi.org/10.1007/978-981-15-9671-1_9
- [76] A. A. C. De Alwis, A. Barros, C. Fidge, and A. Polyvyanyy, "Remodularization analysis for microservice discovery using syntactic and semantic clustering," in *Proc. Int. Conf. Adv. Inf. Syst. Eng.*, S. Dustdar, E. Yu, C. Salinesi, D. Rieu, and V. Pant, Eds., Cham, Switzerland: Springer, 2020, pp. 3–19. [Online]. Available: https://doi.org/10.1007/978-3-030-49435-3_1
- [77] F. D. Eyitemi and S. Reiff-Marganiec, "System decomposition to optimize functionality distribution in microservices with rule based approach," in *Proc. IEEE Int. Conf. Serv. Oriented Syst. Eng.*, Oxford, U.K., 2020, pp. 65–71. [Online]. Available: <https://doi.org/10.1109/SOSE49046.2020.00015>
- [78] C. Bandara and I. Perera, "Transforming monolithic systems to microservices - An analysis toolkit for legacy code evaluation," in *Proc. IEEE 20th Int. Conf. Adv. ICT Emerg. Regions*, 2020, pp. 95–100. [Online]. Available: <https://doi.org/10.1109/ICTer51097.2020.9325443>
- [79] L. Nunes, N. Santos, and A. Rito Silva, "From a monolith to a microservices architecture: An approach based on transactional contexts," in *Proc. Eur. Conf. Softw. Architecture*, T. Bures, L. Duchien, and P. Inverardi, Eds., Cham, Switzerland: Springer, 2019, pp. 37–52. [Online]. Available: https://doi.org/10.1007/978-3-030-29983-5_3
- [80] A. Christoforou, L. Odysseos, and A. Andreou, "Migration of software components to microservices: Matching and synthesis," in *Proc. 14th Int. Conf. Eval. Novel Approaches Softw. Eng.*, 2019, pp. 134–146. [Online]. Available: <https://doi.org/10.5220/0007732101340146>
- [81] I. Pigazzini, F. A. Fontana, and A. Maggioni, "Tool support for the migration to microservice architecture: An industrial case study," in *Proc. Eur. Conf. Softw. Architecture*, T. Bures, L. Duchien, and P. Inverardi, Eds., Cham, Switzerland: Springer, 2019, pp. 247–263. [Online]. Available: https://doi.org/10.1007/978-3-030-29983-5_17
- [82] M. Abdullah, W. Iqbal, and A. Erradi, "Unsupervised learning approach for web application auto-decomposition into microservices," *J. Syst. Softw.*, vol. 151, pp. 243–257, 2019. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0164121219300408>
- [83] M. Kamimura, K. Yano, T. Hatano, and A. Matsuo, "Extracting candidates of microservices from monolithic application code," in *Proc. IEEE 25th Asia-Pacific Softw. Eng. Conf.*, Nara, Japan, 2018, pp. 571–580. [Online]. Available: <https://doi.org/10.1109/APSEC.2018.00072>
- [84] Z. Ren et al., "Migrating web applications from monolithic structure to microservices architecture," in *Proc. 10th Asia-Pacific Symp. Internetware*, New York, NY, USA, 2018, Art. no. 7. [Online]. Available: <https://doi.org/10.1145/3275219.3275230>
- [85] G. Mazlami, J. Cito, and P. Leitner, "Extraction of microservices from monolithic software architectures," in *Proc. IEEE Int. Conf. Web Serv.*, Honolulu, HI, USA, 2017, pp. 524–531. [Online]. Available: <https://doi.org/10.1109/ICWS.2017.61>
- [86] L. Baresi, M. Garriga, and A. De Renzis, "Microservices identification through interface analysis," in *Proc. Eur. Conf. Serv.-Oriented Cloud Comput.*, F. De Paoli, S. Schulte, and E. Broch Johnsen, Eds., Cham, Switzerland: Springer, 2017, pp. 19–33. [Online]. Available: https://doi.org/10.1007/978-3-319-67262-5_2
- [87] Y. Abgaz et al., "Replication package for Decomposition of monolithic applications into microservices architectures: A systematic review," Feb. 2023. [Online]. Available: <https://doi.org/10.5281/zenodo.7899996>
- [88] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proc. 18th Int. Conf. Eval. Assessment Softw. Eng.*, New York, NY, USA, 2014, Art. no. 38. [Online]. Available: <https://doi.org/10.1145/2601248.2601268>
- [89] S. Li et al., "Understanding and addressing quality attributes of microservices architecture: A systematic literature review," *Inf. Softw. Technol.*, vol. 131, 2021, Art. no. 106449. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0950584920301993>
- [90] T. Dybå and T. Dingsøy, "Empirical studies of agile software development: A systematic review," *Inf. Softw. Technol.*, vol. 50, no. 9, pp. 833–859, 2008. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0950584908000256>
- [91] H. Munir, M. Moayyed, and K. Petersen, "Considering rigor and relevance when evaluating test driven development: A systematic review," *Inf. Softw. Technol.*, vol. 56, no. 4, pp. 375–394, 2014. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0950584914000135>
- [92] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. Berlin, Germany: Springer, 2012.
- [93] D. Namiot and M. Sneps-Sneppé, "On micro-services architecture," *Int. J. Open Inf. Technol.*, vol. 2, pp. 24–27, 2014.
- [94] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices architecture enables DevOps: Migration to a cloud-native architecture," *IEEE Softw.*, vol. 33, no. 3, pp. 42–52, May/Jun. 2016.

- [95] K.-J. Stol, P. Ralph, and B. Fitzgerald, "Grounded theory in software engineering research: A critical review and guidelines," in *Proc. 38th Int. Conf. Softw. Eng.*, New York, NY, USA, 2016, pp. 120–131. [Online]. Available: <https://doi.org/10.1145/2884781.2884833>
- [96] S. Jantunen and D. C. Gause, "Using a grounded theory approach for exploring software product management challenges," *J. Syst. Softw.*, vol. 95, pp. 32–51, 2014. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0164121214000776>
- [97] J. Kazanavičius and D. Mazeika, "Analysis of legacy monolithic software decomposition into microservices," in *Proc. Doctoral Consortium/ForumDB&IS*, Tallinn, Estonia, 2020, pp. 25–32. [Online]. Available: <https://ceur-ws.org/Vol-2620/paper4.pdf>
- [98] T. Harwood and T. Garry, "An overview of content analysis," *Marketing Rev.*, vol. 3, pp. 479–498, 2003.
- [99] A. Strauss and J. Corbin, *Basics of Qualitative Research*. Newbury Park, CA, USA: Sage, 1990.
- [100] D. Taibi and K. Systä, "From monolithic systems to microservices: A decomposition framework based on process mining," in *Proc. 9th Int. Conf. Cloud Comput. Serv. Sci.*, D. Ferguson, V. Munoz, M. Helfert, and C. Pahl, Eds., Heraklion, Crete, Greece: Scitepress, 2019, pp. 153–164. [Online]. Available: <https://doi.org/10.5220/0007755901530164>
- [101] M. Cojocar, A. Uta, and A. Oprescu, "MicroValid: A validation framework for automatically decomposed microservices," in *Proc. IEEE Int. Conf. Cloud Comput. Technol. Sci.*, Sydney, Australia, 2019, pp. 78–86. [Online]. Available: <https://doi.org/10.1109/CloudCom.2019.00023>
- [102] F. Auer, M. Felderer, and V. Lenarduzzi, "Towards defining a microservice migration framework," in *Proc. 19th Int. Conf. Agile Softw. Develop.: Companion*, New York, NY, USA, 2018, Art. no. 27. [Online]. Available: <https://doi.org/10.1145/3234152.3234197>
- [103] M. C. Paulk, B. Curtis, M. B. Chrissis, and C. V. Weber, "Capability maturity model, version 1.1," *IEEE Softw.*, vol. 10, no. 4, pp. 18–27, Jul. 1993.
- [104] B. Arasteh, A. Fatolahzadeh, and F. Kiani, "Savalan: Multi objective and homogeneous method for software modules clustering," *J. Softw. Evol. Process*, vol. 34, no. 1, Jan. 2022, Art. no. e2408. [Online]. Available: <https://doi.org/10.1002/smr.2408>
- [105] H. J. Rosenblatt, *Systems Analysis and Design*. Boston, MA, USA: Cengage Learning, 2013. [Online]. Available: https://books.google.ie/books?id=h_PUASRnDDYC
- [106] M. Chinosi and A. Trombetta, "BPMN," *Comput. Standards Interfaces*, vol. 34, no. 1, pp. 124–134, Jan. 2012. [Online]. Available: <https://doi.org/10.1016/j.csi.2011.06.002>
- [107] B. P. Model, "Notation (BPMN) version 2.0," OMG Specification, Object Management Group, 2011, pp. 22–31. [Online]. Available: <http://www.omg.org/spec/BPMN/2.0>
- [108] A. Dennis, *Systems Analysis and Design*, 5th ed. Hoboken, NJ, USA: Wiley, 2012.
- [109] N. Ayewah, W. Pugh, D. Hovemeyer, J. D. Morgenthaler, and J. Penix, "Using static analysis to find bugs," *IEEE Softw.*, vol. 25, no. 5, pp. 22–29, Sep./Oct. 2008. [Online]. Available: <https://doi.org/10.1109/MS.2008.130>
- [110] A. Møller and M. I. Schwartzbach, "Static program analysis," Dept. Comput. Sci., Aarhus Univ., Oct. 2018. Accessed: Sep. 10, 2021. [Online]. Available: <http://cs.au.dk/amoeller/spa/>
- [111] F. Nielson, H. R. Nielson, and C. Hankin, *Data Flow Analysis*. Berlin, Germany: Springer, 1999, pp. 35–139. [Online]. Available: https://doi.org/10.1007/978-3-662-03811-6_2
- [112] T. Ball, "The concept of dynamic analysis," *SIGSOFT Softw. Eng. Notes*, vol. 24, no. 6, pp. 216–234, Oct. 1999. [Online]. Available: <https://doi.org/10.1145/318774.318944>
- [113] N. Lapuz, P. Clarke, and Y. Abgaz, "Digital transformation and the role of dynamic tooling in extracting microservices from existing software systems," in *Systems, Software and Services Process Improvement*, M. Yilmaz, P. Clarke, R. Messnarz, and M. Reiner, Eds., Cham, Switzerland: Springer, 2021, pp. 301–315. [Online]. Available: https://doi.org/10.1007/978-3-030-85521-5_20
- [114] N. Gupta, "Microservice, miniservice, and macroservice," Aug. 2020. Accessed: Mar. 14, 2021. [Online]. Available: <https://dzone.com/articles/micro-service-mini-service-and-macro-service>
- [115] D. M. Fernández et al., "Artefacts in software engineering: A fundamental positioning," *Softw. Syst. Model.*, vol. 18, no. 5, pp. 2777–2786, 2019. [Online]. Available: <https://doi.org/10.1007/s10270-019-00714-3>
- [116] M. Silva and T. Oliveira, "Towards detailed software artifact specification with SPEMArti," in *Proc. Int. Conf. Softw. Syst. Process*, New York, NY, USA, 2011, pp. 213–217. [Online]. Available: <https://doi.org/10.1145/1987875.1987912>
- [117] A. van Hoorn, J. Waller, and W. Hasselbring, "Kieker: A framework for application performance monitoring and dynamic software analysis," in *Proc. 3rd ACM/SPEC Int. Conf. Perform. Eng.*, New York, NY, USA, 2012, pp. 247–248. [Online]. Available: <https://doi.org/10.1145/2188286.2188326>
- [118] H. W. Kuhn and B. Yaw, "The Hungarian method for the assignment problem," *Nav. Res. Logistics Quart.*, vol. 2, no. 1/2, pp. 83–97, 1955.
- [119] L. Carvalho et al., "Search-based many-criteria identification of microservices from legacy systems," in *Proc. Genet. Evol. Computation Conf. Companion*, New York, NY, USA, 2020, pp. 305–306. [Online]. Available: <https://doi.org/10.1145/3377929.3390030>
- [120] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002. [Online]. Available: <https://doi.org/10.1109/4235.996017>
- [121] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength pareto evolutionary algorithm," TIK-Rep., 2001. [Online]. Available: <http://hdl.handle.net/20.500.11850/145755>
- [122] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.
- [123] E. Yourdon and L. L. Constantine, *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*, 1st ed. Englewood Cliffs, NJ, USA: Prentice-Hall, 1979.
- [124] M. Hitz and B. Montazeri, "Measuring coupling and cohesion in object-oriented systems," in *Proc. Int. Symp. Appl. Corporate Comput.*, 1995, pp. 25–27.
- [125] D. Athanopoulos, A. V. Zarras, G. Miskos, V. Issarny, and P. Vassiliadis, "Cohesion-driven decomposition of service interfaces without access to source code," *IEEE Trans. Serv. Comput.*, vol. 8, no. 4, pp. 550–562, Jul./Aug. 2015. [Online]. Available: <https://doi.org/10.1109/TSC.2014.2310195>
- [126] J. Fritsch, J. Bogner, S. Wagner, and A. Zimmermann, "Microservices migration in industry: Intentions, strategies, and challenges," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol.*, Cleveland, OH, USA, 2019, pp. 481–490. [Online]. Available: <https://doi.org/10.1109/ICSME.2019.00081>
- [127] A. Brogi, A. Canciani, D. Neri, L. Rinaldi, and J. Soldani, "Towards a reference dataset of microservice-based applications," in *Proc. Int. Conf. Softw. Eng. Formal Methods*, A. Cerone and M. Roveri, Eds., Cham, Switzerland: Springer, 2018, pp. 219–229. [Online]. Available: https://doi.org/10.1007/978-3-319-74781-1_16
- [128] M. I. Rahman, S. Panichella, and D. Taibi, "A curated dataset of microservices-based systems," *Joint Proc. Inforte Summer Sch. Softw. Maintenance Evol.*, vol. 2520, pp. 1–9, 2019. [Online]. Available: <https://ceur-ws.org/Vol-2520/paper1a.pdf>
- [129] C. Wohlin, "Software engineering research under the lamppost," in *Proc. 8th Int. Joint Conf. Softw. Technol.*, J. Cordeiro, D. A. Marca, and M. van Sinderen, Eds., Reykjavík, Iceland: SciTePress, 2013, pp. IS–11.
- [130] A. Razzaq, A. Wasala, C. Exton, and J. Buckley, "The state of empirical evaluation in static feature location," *ACM Trans. Softw. Eng. Methodol.*, vol. 28, no. 1, Dec. 2018, Art. no. 2. [Online]. Available: <https://doi.org/10.1145/3280988>
- [131] M. Bozkurt, M. Harman, and Y. Hassoun, "Testing and verification in service-oriented architecture: A survey," *Softw. Testing Verification Rel.*, vol. 23, no. 4, pp. 261–313, 2013. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/stvr.1470>
- [132] T. S. C. Tiobe, "Tiobe index for January 2021," Jan. 2021. Accessed: Feb. 02, 2021. [Online]. Available: <https://www.tiobe.com/tiobe-index/>
- [133] D. Cassel, "COBOL is everywhere. Who will maintain it?" 2017. Accessed: Feb. 24, 2022. [Online]. Available: <https://thenewstack.io/cobol-everywhere-will-maintain/>
- [134] A. Santos and H. Paula, "Microservice decomposition and evaluation using dependency graph and silhouette coefficient," in *Proc. 15th Braz. Symp. Softw. Compon. Architectures Reuse*, New York, NY, USA, 2021, pp. 51–60. [Online]. Available: <https://doi.org/10.1145/3483899.3483908>
- [135] Z. Wu and M. Palmer, "Verb semantics and lexical selection," in *Proc. 32nd Annu. Meeting Assoc. Comput. Linguistics*, Las Cruces, NM, USA, 1994, pp. 133–138. [Online]. Available: <https://www.aclweb.org/anthology/P94-1019>
- [136] D. Lin, "An information-theoretic definition of similarity," in *Proc. 15th Int. Conf. Mach. Learn.*, San Francisco, CA, USA: Morgan Kaufmann, 1998, pp. 296–304.
- [137] P. Resnik, "Using information content to evaluate semantic similarity in a taxonomy," in *Proc. 14th Int. Joint Conf. Artif. Intell.*, San Francisco, CA, USA: Morgan Kaufmann, 1995, pp. 448–453.

- [138] R. V. O'Connor, P. Elger, and P. M. Clarke, "Continuous software engineering—A microservices architecture perspective," *J. Softw.: Evol. Process*, vol. 29, no. 11, 2017, Art. no. e1866.
- [139] S. Rochimah and B. Nuralamsyah, "Decomposing monolithic to microservices: Keyword extraction and BFS combination method to cluster monolithic's classes," *Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi)*, vol. 7, no. 2, pp. 263–270, Mar. 2023. [Online]. Available: <http://jurnal.iaii.or.id/index.php/RESTI/article/view/4866>
- [140] L. Qian, J. Li, X. He, R. Gu, J. Shao, and Y. Lu, "Microservice extraction using graph deep clustering based on dual view fusion," *Inf. Softw. Technol.*, vol. 158, 2023, Art. no. 107171. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0950584923000253>
- [141] K. Sellami, M. A. Saied, A. Ouni, and R. Abdalkareem, "Combining static and dynamic analysis to decompose monolithic application into microservices," in *Proc. Int. Conf. Serv.-Oriented Comput.*, J. Troya, B. Medjahed, M. Piattini, L. Yao, P. Fernández, and A. Ruiz-Cortés, Eds., Cham, Switzerland: Springer, 2022, pp. 203–218.
- [142] L. Cao and C. Zhang, "Implementation of domain-oriented microservices decomposition based on node-attributed network," in *Proc. 11th Int. Conf. Softw. Comput. Appl.*, New York, NY, USA, 2022, pp. 136–142. [Online]. Available: <https://doi.org/10.1145/3524304.3524325>
- [143] V. Nitin, S. Asthana, B. Ray, and R. Krishna, "CARGO: AI-guided dependency analysis for migrating monolithic applications to microservices architecture," in *Proc. IEEE/ACM 37th Int. Conf. Automated Softw. Eng.*, New York, NY, USA, 2023, Art. no. 20. [Online]. Available: <https://doi.org/10.1145/3551349.3556960>
- [144] K. Sellami, M. A. Saied, and A. Ouni, "A hierarchical DBSCAN method for extracting microservices from monolithic applications," in *Proc. Int. Conf. Eval. Assessment Softw. Eng.*, New York, NY, USA, 2022, pp. 201–210. [Online]. Available: <https://doi.org/10.1145/3530019.3530040>
- [145] I. Trabelsi et al., "From legacy to microservices: A type-based approach for microservices identification using machine learning and semantic analysis," *J. Softw.: Evol. Process*, 2022, Art. no. e2503. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.2503>
- [146] M. Dehghani, S. Kolahdouz-Rahimi, M. Tisi, and D. Tamzalit, "Facilitating the migration to the microservice architecture via model-driven reverse engineering and reinforcement learning," *Softw. Syst. Model.*, vol. 21, no. 3, pp. 1115–1133, Jun. 2022. [Online]. Available: <https://doi.org/10.1007/s10270-022-00977-3>
- [147] X. Sun, S. Boranbaev, S. Han, H. Wang, and D. Yu, "Expert system for automatic microservices identification using API similarity graph," *Expert Syst.*, 2022, Art. no. e13158. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1111/exsy.13158>
- [148] A. Mathai, S. Bandyopadhyay, U. Desai, and S. Tamilselvam, "Monolith to microservices: Representing application software through heterogeneous graph neural network," in *Proc. 31st Int. Joint Conf. Artif. Intell.*, L. D. Raedt, Ed., 2022, pp. 3905–3911. [Online]. Available: <https://doi.org/10.24963/ijcai.2022/542>



Yalemisew Abgaz received the BSc and MSc degrees from Addis Ababa University, and the PhD degree from Dublin City University, in 2013. He is an assistant professor with the School of Computing, Dublin City University. He has been a senior research fellow with the Future Software Systems Architecture (FSSA) Project, Lero Research Centre, a principal investigator on ChIA Project, research fellow and postdoctoral researcher with the ADAPT Centre, Dublin City University affiliated with the Austrian Centre for Digital Humanities, Austrian Academy

of Sciences, and a postdoctoral researcher with Maynooth University. He is mainly interested in topics at the intersection between software engineering, data analytics, Semantic Web technologies, knowledge representation, and natural language processing including ontology development, ontology evolution, semantic search, semantic publishing, information retrieval, and computational creativity.



Andrew McCarren received the BSc and PhD degrees from Dublin City University. He is an associate professor with the School of Computing, Faculty of Engineering and Computing, Dublin City University. He is also a funded SFI investigator with the Insight Centre of Data Analytics and his primary research interests include the application of data analytics and data mining techniques in Fintech, agriculture, food, health and human performance.



Peter Elger received the degrees in physics and computer science. He is a co-founder and CEO of fourTheorem, an AWS Advanced consulting partner specialising in next generation cloud architecture, and high performance Serverless. He started his career with the Joint-Jet Undertaking in the U.K. building acquisition, control and data analytics systems for nuclear fusion research. He has held technical leadership roles across a broad base of the industry in both the research and commercial sectors including, software disaster recovery, telecommunications, and social media. Prior to founding fourTheorem, he was co-founder and CTO of two companies; Stitcher Ads, a social advertising platform and nearForm, a Node.js consultancy specialising in digital transformation initiatives. An author and contributor to several books and academic papers, his most recent book 'AI as a Service' is available from Manning Publications.



David Solan is head of product engineering with FINEOS. FINEOS is a global market leader in core insurance technology for disability, life, accident, and health with more than 50 Insurance Carriers and Government Accident Compensation organisations. He has more than 25 years of industry experience in all aspects of software delivery from requirements, architecture, SDLC governance, and post-production support. He has led the transition from traditional development practices to Agile at scale and is responsible for delivering a high-quality end-to-end SAAS-based platform for some of the largest insurance carriers in the world.



Neil Lapuz received the bachelor's degree in computer applications and software engineering from Dublin City University, in 2019, and the master's degree. He is a software engineer with Oviva, Zurich. With his interest in monolithic to microservice migration, he joined the Future Software Systems Architecture (FSSA) project with the aim of contributing to the project, and conducting his research.



Marin Bivol received the BSc degree from the School of Computing, Dublin City University, in 2019. He is a senior software developer and a professional AWS cloud architect with fourTheorem. He has worked as a software developer with the Future Software Systems Architecture (FSSA) Project and continues to contribute to it. His research interests include monolith to microservices automated transformation, IoT, container, and database edge technologies.



Glenn Jackson received the BA degree from the School of Computer Science, Trinity College Dublin, in 2015. He is a software developer working for Vectra AI, a company specialised in AI-driven threat detection and response. He has worked on the Future Software Systems Architecture (FSSA) Project that was aligned with Lero, the Science Foundation Ireland Research Centre for Software, School of Computing, Dublin City University.



Jim Buckley received the PhD degree in computer science from the University of Limerick, in 2002. He is a professor with the Computer Science and Information Systems Department, University of Limerick, Ireland and is a co-principle investigator with Lero, the Irish Research Centre for Software. He was awarded the (Lero) Director's prize for Research Excellence in 2020 and his main research interests focus on supporting software developers who are tasked with maintaining and evolving software systems. Thus, specific areas of interests include feature location, software comprehension, and architectural analysis of such systems.



Murat Yilmaz received the master's degree in software engineering from the University of Minnesota, with a particular focus on game theory in software engineering, and the PhD degree from Dublin City University. He is an associate professor with the Computer Engineering Department, Gazi University, where he also serves as the deputy director of the Informatics Institute. His professional journey spans more than thirteen years in the software development industry, complemented by a decade of academic experience, equipping him with a profound knowledge

base and diverse expertise. He is actively engaged in numerous projects and has an extensive portfolio of academic works published in internationally recognized conferences and journals, with a primary focus on the software process, software management, empirical software engineering, algorithmic game theory, virtual reality, serious games, and gamification.



Paul Clarke is an associate professor with Dublin City University (DCU) and is a member of Lero, the Science Foundation Ireland Research Centre for Software. His research interests include software practices, software architecture, and artificial intelligence. He is principal investigator on the Future Software Systems Architectures Project, an initiative which brings together industrial practitioners and academia to collaborate on machine learning techniques for architectural remodelling. He is presently DCU BSc in computer science programme chair, and National Head of Delegation for Ireland to ISO/IEC Joint Technical Committee 1, Sub Committee 7, Systems and Software Engineering. He is currently serving as Steering Committee chair for the International Conference on Software and Systems Processes (ICSSP) and as an editor for the European System, Software and Service Process Improvement (euroSPI) Conference.