International Journal of Service Science, Management, Engineering, and Technology
Volume 13 • Issue 1

# A Comparative Study of Meta-Data-Based Microservice Extraction Tools

Kerem Akkaya, Fibabanka, Turkey
Tolga Ovatman, Istanbul Technical University, Turkey*
https://orcid.org/0000-0001-5918-3145

## ABSTRACT

Microservices are state-of-the-art architectural patterns for developing highly resilient and scalable applications to be hosted on the web. One of the tedious and challenging aspects on applying microservice pattern is the process of transforming legacy "monolithic" software to microservices. This process, named microservice extraction, is mostly done manually because of its complicated nature and the need for domain expertise. In this study, meta-data-driven tools that aim to perform automatic extraction of microservices are focused. Initially, the studies that perform automatic microservice extraction are reviewed to identify public meta-data-based tools. Afterwards, the identified tools are investigated by their relative success in extracting microservices. Reviewed tools suffer over-decomposition during extraction, and most of them fail to provide a mature basis when considered to exhibit practical value to be used in real-world cases.

## KEYWORDS

Automated Software Engineering Tools, Microservice Architecture, Microservice Extraction

## INTRODUCTION

Microservices are the state of art architectural style for building highly scalable and reusable software, typically used in web-based systems. Although there exist many different definitions of the concept, a possible brief definition for a microservice can be: "a composable and reusable, loosely coupled functionality, encapsulated in a fine-grained manner, that may depend on other internal microservices when served".

Different techniques can be applied to ensure this property and provide a better throughput during replica runs. Microservices are gradually replacing "monolithic services" as its predecessor. Although it is hard to provide a brief description of a monolithic service, a possible definition might be: "a coarsely encapsulated end-to-end functionality that can be served independent of other internal services".

Service oriented computing and web services have been a major architectural pattern in web-based software development in 2010s (Sbaï and Guerfel, 2016; Kim and Lennon, 2017). There have been

DOI: 10.4018/IJSSMET.298677 *Corresponding Author

many studies performed on various aspects of service-oriented computing and web services such as context-awareness of services and security of service composition (Guermah et al, 2021; Rouached, 2021). Recently with the emergence of cloud-based systems, service-oriented computing research has begun to transform as well (Guerfel et al., 2017; Kumar and Bhardwaj, 2020). In this advancing field, microservices are the state-of-the-art architectural patterns that seizes the development strategy of modern cloud-based web applications.

There are some factors behind the software industry's incline towards microservices. Most of these factors stem from the incapabilities of legacy monolithic services and recent developments in software technology that enable fine-grained governance of software services. Some of these factors can be listed as follows:

- Obstacles and costs in reusing and integration of monolithic services.
- Virtualization and cloud-based technologies that enable rapid deployment and automatic scaling of service instances.
- Recent developments in monitoring and tracking deployed software components.
- Expanding tool support for software development life-cycle automation.

This gradual shift towards microservices brings about its own set of challenges. An example to such challenges is the migration from legacy software to microservices. The major effort, for this challenge, is on decomposing a living monolithic software at hand to identify an equivalent set of microservices whilst keeping the functionality of the legacy software. This decomposition and identification effort is called "microservice extraction (or decomposition)" in the literature. Most of the time, such an extraction requires a great amount of architectural and domain expertise. There exists recent effort and even tool support in software literature that try to automate this task using different software development artifacts.

This study aims to evaluate meta-data based microservice extraction tools over a set of case studies to identify the current capabilities and shortcomings of the tool support in the literature by applying a comparative analysis (Gupta and Garg, 2015). In the study, publicly available, microservice extraction tools present in the literature are used to extract microservices from six different projects where two of these projects are real world banking software. Present capabilities of the tools are evaluated from a user level perspective and the results are compared both in a cross-tool manner and by direct comparison with manual microservice extraction.

This study adds to the software engineering body of knowledge with the following contributions:

- A focused literature review is presented on microservice extraction tools. Since there exists a few studies that aim microservice extraction, present literature review studies review on a more holistic level. The focus is on the studies that provide a publicly available tool support. (Section: "Microservice extraction approaches with tool support")
- Tool support is evaluated in terms of user level functionality (Section: "Meta-data based microservice extraction tools by user experience")
- Meta-data based microservice extraction tools in the literature are cross-compared. (Sections: "Cross comparison of microservice extraction tools" and "Microservice extraction tools in the wild")
- Meta-data based tools are evaluated over a set of case studies and results are compared with microservices extracted by domain experts. (Sections: "Cross comparison of microservice extraction tools" and "Microservice extraction tools in the wild")
- An overall evaluation of the meta-data based microservice extraction tools is provided and areas of improvement are proposed. (Section: "Overall evaluation of meta-data based microservice extraction tools and areas for improvement")

## MICROSERVICE EXTRACTION APPROACHES WITH TOOL SUPPORT

### Microservice Extraction Process

Microservice extraction can be described as an architectural or high-level refactoring activity that involves identifying and encapsulating software component clusters as microservices. Extracted services can be developed, hosted and maintained independently to some extent but they can also work in collaboration to provide the user level functionality of the legacy software. This description is somewhat vague because there is no formal definition that defines the number and clustering characteristics of the microservices to be extracted. Because of this vagueness, microservice extraction is mostly done by domain experts and it can be seen as one of the "more artistic" aspects of software development discipline.

Extraction process generally involves application of both decomposition and clustering to identify microservices in legacy software. Practicing microservice extraction initially requires fine-grained decomposition of current services or software components to identify clustering possibilities of highly coupled software components. According to the artifacts or medium being used to apply microservice extraction, the method and decomposable units used in the decomposition may change. For instance, a meta-data based approach can use class definitions and composition relation in decomposition while a run-time analysis approach might use objects and run-time method calls or workload data.

After the decomposition, clustering can be applied by using various different criteria that aims to define meaningful coupling relations among used components. Automated clustering approaches generally use graph clustering in this purpose where manual clustering approaches mostly rely on functional and non-functional requirements. There exist approaches that involve such requirements in clustering process that we will review in the next section.

Even though automated microservice extraction can be conceptually overviewed as above, in practice, there are many different characteristics that can be used to develop automated microservice extraction. An overview of such approaches and tools is provided in the next section.

### Microservice Extraction Approaches and Tools

In this study two recent surveys have been adopted to base the presented literature review. The focus is on the approaches with tool support in the study by Fritzsch et al. (2018) and additionally recent studies with tool support between 2018 and 2021 are also reviewed. The characteristics identified by Carvalho et al. (2019) and Kirby et al. (2021) in microservice extraction are also used when needed.

There exists a number of review studies on the area in the recent years (Kazanavičius and Mažeika, 2019; Ponce et al., 2019; Schröer et al., 2020; Stojkov and Stojanov, 2021). However, all of these studies focus on methodological comparison instead of performing a comparative study on tool support and practical aspects of the provided tool. We aim to fill this gap in the current literature by providing an extensive comparative study on meta-data based tools and identify possible research directions for the researchers and practitioners studying in the field.

In their study Jonas Fritzsch et al. reviewed and compared ten refactoring approaches that aim to extract microservices from legacy systems. They initially classified the approaches based on the decomposition of the legacy software and use various other criteria such as atomic unit granularity, outputs, validation of approach and tool support. In their comparison of ten different approaches they have identified the approaches with and without the public tool support in Table 1. Fields in the table are explained below:

- Study: Citation to the related paper.
- Public Tool Support: The name of the tool is presented with a public URL as a reference.
- Tool Input: A subset of Fritzsch et al.'s study is used in this respect:

- Static analysis: Input obtained by running a static analysis on software code and/or design artifacts like Unified Modeling Language (UML) diagrams or more conventional diagrams like data flow diagrams (DFDs).
- Meta-data: Input obtained by analyzing information indirectly related with software code such as non-functional requirements, repository information, etc.
- Workload data: Stands for input obtained by analyzing run-time data.

Approaches in Table 1 that has been proposed between 2016 and 2018 are grouped together regarding the public tool support. Four of the approaches provide a public tool support where three of these four approaches use meta-data based input (version control history data, non-functional requirements, etc.) as the medium to perform the extraction.

In this study, this table is further extended by considering the studies that has been published after 2018 (also one study from 2017 as an exception). We use the key terms "microservice extraction", "microservice decomposition", "microservice refactoring" to conduct a literature search in scientific publishing and indexing websites, namely IEEE, ACM, Springer, Elsevier, Wiley, IGI and Google Scholar. An extra level of exploration is also conducted by reviewing the studies that cites studies listed in Table 1. As a result, the extension in Table 2 is constructed.

Considering Table 1 and Table 2 together it can be seen that there exist only six publicly available tools that present an implementation of the proposed approach. The tool support for study (ms Dashboard, 2017) (Mayer and Weinreich, 2018) is accessible by registration but it contains a dashboard application rather than the extraction tool itself. Those tools are mainly implemented between 2016 and 2018. There is a trend towards utilizing more artifacts to perform better in extraction process. Service Cutter, Decomposer and Microservice-extractor tools use only meta-data and MicADO tool use only workflow data while relatively closer dated MicroART and SubType uses all three types of input in the classification.

Another interesting aspect about the studies listed above is, a follow up paper on the presented extraction approach or its implementation is rarely the case for the studies. Many of the papers that contains "Towards..." keyword in their title contains studies that has not yet been complemented by additional research. This situation may be due to the narrow four-year time interval of the conducted review, since most of the authors continue publishing about microservices or distributed systems. Still it is possible to comment that most of the microservice extraction tools hold an impression/decline on being discontinued after initial implementation. Another indication of this situation is the latest

**Table 1. Tool support crosscut of Fritzsch et al.'s study (Fritzsch et al., 2018)**

| Study | Public Tool Support | Tool Input |
|---|---|---|
| (Ahmadvand and Ibrahim, 2016) | - | Meta-data |
| (Escobar et al., 2016) | - | Static analysis |
| (Levcovitz et al., 2016) | - | Static analysis |
| (Procaccianti et al., 2016) | - | Meta-data and workload data |
| (Hassan et al., 2017) | - | Workload data |
| (Mustafa et al., 2018) | - | Workload data |
| (Gysel et al., 2016) | Service Cutter (Gysel, 2015) | Meta-data |
| (Baresi et al., 2017) | Decomposer (Garriga, 2016) | Meta-data |
| (Klock et l., 2017) | MicADO (Klock, 2017) | Workload data |
| (Mazlami et al., 2017) | Microservice-extractor (Mazlami, 2016) | Meta-data |

**Table 2. Extension to extraction studies' tool support**

| Study-Year | Public Tool Support | Tool Input |
|---|---|---|
| (Chen et al., 2017) | - | Static analysis and meta-data |
| (Amiri, 2018) | - | Meta-data |
| (Eski and Buzluca, 2018) | - | Static analysis and meta-data |
| (Jin et al., 2018) | - | Workload data |
| (Mayer and Weinreich, 2018) | ms Dashboard (2017) | Static analysis, meta-data and workload data |
| (Munezero et al., 2018) | - | Meta-data |
| (Tyszberowicz et al., 2018) | - | Static analysis and meta-data |
| (Granchelli et al., 2017) | MicroART (Granchelli, 2017) | Static analysis, meta-data and workload data |
| (De Alwis et al., 2018) | SubType (De Alwis, 2018) | Static analysis, meta-data and workload data |
| (Nunes et al., 2019) | Mono2Micro (Mono2Micro, 2019) | Static analysis, architect interaction |
| (Pigazzini et al., 2019) | Arcan (Arcan, 2019) | Static analysis |
| (Saidani et al., 2019) | - | Static analysis |
| (Taibi and Systä, 2019) | - | Workload data |
| (Park et al., 2020) | - | Static analysis |
| (Al-Debagy and Martinek, 2021) | - | Static analysis |
| (Daoud et al., 2021) | - | Meta-data |
| (Zaragoza et al., 2021) | - | Static analysis |
| (Junfeng Zhao and Ke Zhao, 2021) | - | Static analysis |

commit date on the presented tools' github repositories mostly changes between 2-4 years before the preparation date of this study. An exception to this situation is the tool Mono2Micro (Mono2Micro, 2019) which is still being actively maintained.

## Public Microservice Extraction Tools and Their Characteristics

In this subsection, a more detailed overview of the meta-data based extraction tools is presented together with the related scientific studies that back them up. Table 3 and Table 4 presents an overall comparison between the microservice extraction tools in the literature. This comparison is presented to obtain a broader idea about the main idea behind each tool and the evaluation of these tools. A more detailed comparison on user experience and capabilities of the meta-data based tools will be presented in the latter section. When examined in detail, the tools presented in the literature trend towards using more diverse input during their analysis such as static code analysis and run-time workload analysis while the former tools mostly rely on models and application programming interface (API) specification.

Another important trend can be seen in evaluation of the tools where the latter studies rely on evaluation by real world case studies where former tools perform evaluation on custom coded software. A common trend among the techniques being used is graph or string clustering, which seems to be natural because of the characteristics of the microservice extraction process. An alternative approach is presented in MicroART tool where many different pieces of information is mapped to a meta-model, specific to microservice modeling domain. However, the details of the mapping procedure are not provided in the related papers' content (Granchelli et al., 2017).

**Table 3. Extraction tools' overall comparison - Inputs and methods**

| Tool Name | Input | | | Used Methods |
|---|---|---|---|---|
| | *Static Input* | *Meta-data Input* | *Workload Input* | |
| **Service Cutter** | - | Use cases, E-R models, etc. | - | Graph clustering and software metrics |
| **Decomposer** | - | OpenAPI specs. | - | String similarity |
| **Microservice-extractor** | - | Repository history | - | Graph clustering |
| **MicADO** | - | - | Access logs, performance metrics | Clustering by genetic algorithm |
| **MicroART** | Code | Repository history, documentation | Communication logs, container info. | Domain specific model mapping |
| **SubType** | Code | Business object models | Execution call graph | Graph clustering and object relations |
| **Mono2Micro** | Code | - | - | Dependency graph clustering and transactional context by architect interactions |
| **Arcan** | Code | - | - | Graph clustering on three different views of the system |

**Table 4. Extraction tools' overall comparison - tool evaluation method**

| Tool Name | Evaluation | |
|---|---|---|
| | **Number of cases** | **Comparison** |
| **Service Cutter** | 2 custom coded software, many artificially generated input | number of correctly extracted services compared to monolithic version |
| **Decomposer** | 1 custom coded software, 2 example projects from eventuate.io, 14 API specs from APIs.Guru | number of correctly extracted services compared to monolithic version |
| **Microservice-extractor** | 21 open source projects | custom metrics compared to monolithic version |
| **MicADO** | 1 real-world ERP software | non-functional metrics compared to monolithic version |
| **MicroART** | 1 custom coded software | architecture comparison with monolithic version |
| **SubType** | 2 real-world CRM software | non-functional metrics compared to monolithic version |
| **Mono2Micro** | 2 real-world software | custom metrics compared to monolithic version |
| **Arcan** | 1 real-world software | manual evaluation via developer interaction |

A final common pattern on the extraction tools can be seen on their quantitative evaluation of the tools. All of the papers perform evaluation by comparing with legacy monolithic software. There is no apparent comparison between different extraction approaches in any of the studies. Another vague area, specifically mentioned in some of the papers is the set of metrics to use when evaluating the extraction process. Naturally, some studies compare the number and qualifications of the extracted services where others use non-functional and performance metrics targeting properties such as scalability, fault tolerance, end-to-end performance, number of requests, etc. Some specific studies come up with their own set of metrics where even development team effort is considered.

Basic characteristics and functions of those tools in Table 3 and Table 4 can be briefly summarized as follows:

- Service cutter is aimed to assist software architects when they make service design decisions by considering software quality attributes. This tool assembles 16 coupling criteria grouped in 4 classes such as cohesiveness and compatibility (Gysel et al., 2016).
- Decomposer is a decomposition framework that uses identification process consists of matching the terms used in the OpenAPI specifications supplied as input against a reference vocabulary to suggest possible decompositions. This approach calculates semantic similarities by pre-computed database of collocations and similar words by their frequencies (Baresi et al., 2017).
- Microservice-extractor presents three coupling approaches and embed those in a graph-based clustering algorithm by using classes and its dependencies among them. The tool firstly gets source code of the legacy software from public source control repository and then transforms it into graph representation by employing coupling strategies. A coupling strategy is used to mine the information in the legacy software and create an undirected, edge-weighted graph representation. Logical coupling, semantic coupling and contributor coupling are the main strategies proposed as contributions (Mazlami et al., 2017).
- MicADO is an automated approach for optimizing the performance and scalability of a microservice architecture by modifying the placement of features in microservices based on the workload of a microservice system. Software operation data is collected as workload for the system to be transformed. This approach suggests a clustering of these features in microservices by employing genetic algorithms, optimized for the given workload (Klock et al., 2017).
- MicroART is a microservice architecture recovery system that performs by obtaining source code residing in a code repository and a reference of the container engine managing the microservice architecture. This tool's main focus is to extract the deployment architecture of a microservice-based system (Granchelli et al., 2017).
- Subtype is a microservice recommendation system that handles source code and its database meta-data as an input. In the next step, this tool analyzes business objects over how they apply CRUD operations and their business object relationships by using graph clustering algorithms. As a result; it generates recommendations as an output based on two different heuristics (De Alwis et al., 2018).
- Mono2Micro uses static analysis to extract call graph from java code and visualize how call graph contains call interdependency clusters by the help of dendrograms. Later, by interacting with a software architect, this dendrogram is clustered in a custom way over its visualization. Clustering process relies on model-view-controller based nature of the project and try to identify so called "transactional contexts" where business transactions are identified within the code to be based on building microservices. Presented tool is still being actively developed and maintained which is a distinguishing aspect among other tools analyzed in this study Mono2Micro (Mono2Micro, 2019).
- Arcan is a previously developed tool targeted to detect architectural smells. This tool has been extended to detect features and domains by running a dependency graph analysis over different perspectives of the system. These perspectives are call clusters which are detected by connected component analysis; critical features which are identified as common classes in call paths and

logical characteristic of components which is performed by mapping classes to one of model-view-controller layers. This tool is not open source and available upon request by mailing to researchers which gives little open information on its development status (Arcan, 2019).

Our comparisons in this study will continue by covering the tools that rely solely on meta-data based analysis: Service cutter, Microservice-extractor and Decomposer.

This subgroup is preferred to be analyzed because of the possibility to set up and use these software more rapidly compared to others. To be more specific, tools that rely on workload collection as an input requires a relatively long inspection period to collect and format necessary data which makes the application of these tools harder for the user to grasp and adapt. On the other hand, meta-data based tools can be applied relatively easier and more rapidly on legacy software. We believe that the key idea behind using such a tool is to gain rapid insight about possible microservices that can be extracted from a legacy software to reduce the manual microservice extraction time. Tools that rely on workload data damages the necessary rapidness of the process which might make them less usable in practice. Because of this insight we have decided to experience a rapid application process of the meta-data based tools and compare them in this context.

## META-DATA BASED MICROSERVICE EXTRACTION TOOLS BY USER EXPERIENCE

In this section, the user experience offered by the meta-data based extraction tools is presented. Presented analysis involves three stages. At the first stage we compare the overall characteristics of the selected tools such as input format, visualization support, etc. At the second stage we comment about the setup and booting sequence of each tool and relatively compare the initial effort needed by each tool. At the final stage we compare the usability of each tool over the offered extraction processes. It is important to note that comparisons in this section, especially in stages two and three are relatively subjective, tools are compared based on authors' own experience, but presented comments are about overall adversities of the processes, as much as possible, to provide meaningful insight.

Table 5 compares the selected tools over various criteria. An initial distinctive characteristic about the tools is the visualization support of the tools. Decomposer tool relies on command line usage and uses text-based input-output, making the tool less interactive in terms of visualization when compared to other two. A more significant distinction is about the input artifacts that the tools use. All of the tools rely on analyzing inter-dependencies between software components but Microservice-extractor uses lower level dependency definitions as well as development team activity information from the code repository as a distinctive characteristic. On the other hand, Decomposer utilizes semantic similarity between the entity names to couple the components together. Service cutter relies on system specifications and 16 different coupling criteria that can be extracted from system specifications.

Significant differences between the tools can be summarized as follows: Service cutter relies on various criteria on the system specification stage where Microservice-extractor considers development stage information as well. Decomposer uses a comparably smaller set of input and relies only on API specifications but this tool uses semantic similarity based on natural language processing techniques on top of using relational coupling like the other two tools.

In the second stage initial setup process and booting difficulties of the selected tools are compared. This property is important in terms of a good user experience because a tedious initial effort may build a barrier on the application of the tool. An initial concern on setting up the tool environment is the availability of publicly available code to tweak and customize the setup process. All of the selected tools host their code in publicly available git repositories. All of the selected tools can be installed on an isolated environment (such as a virtual machine) with no preliminary knowledge, by following the order of installation described in its documentation.

**Table 5. Meta-data based extraction tools' characteristics**

| Characteristic | Service Cutter | Microservice-extractor | Decomposer |
|---|---|---|---|
| **Input artifact** | Non-functional requirements, entity and interface definitions | Class dependencies, team activity and change history | Service interfaces |
| **Input format** | JSON | git URL/API | JSON (OpenAPI specifications) |
| **Process Tuning** | Yes | Yes | No |
| **Code availability** | Open source | Open source | Open source |
| **Application type** | Web | Web | Command line |
| **Output visualization** | Yes | Yes | No |
| **Code generation** | No | No | No |
| **Main technique** | Girvan-Newman clustering (2004) and ELP clustering (Raghavan et al., 2007) | MST Clustering (Mazlami et al., 2017) | DISCO word similarity (Kolb, 2009) |
| **Integration** | No | No | No |
| **Output format** | image | image, JSON | test |
| **Used criteria** | 16 different coupling criteria | 3 coupling strategies | semantic similarity of entity names |

Service cutter has two main components. The first one is the editor application designed as a Graphical User Interface (GUI) that allows using the back-end engine. This application was developed using node.js framework. The second part is the application named as engine, which is used as the back-end used by the editor, and the spring boot application where calculations are made. Node.js and dependent libraries must be installed in the environment to run the editor.

The installation process of Service cutter is easy however some extra components must be installed to run the software. This project is organized as a pom maven project so compilation and booting process does not take a significant amount of time. The most obvious drawback is to understand the tool parameters required to perform microservice extraction. Inputs require a comprehensive knowledge of the entire system.

The easiest booting process that was faced is provided by the Decomposer. Decomposer is not served from a web server, it has only a single main method that classify service method interfaces according to dictionary based on DISCO framework (Kolb, 2009).

Microservice-extractor is designed in the form of two separate executable components as a front-end developed using Angular frameworks and a back-end that uses spring boot framework. It is a little bit more difficult to boot this application compared to the Service cutter and Decomposer.

Microservice-extractor is run as a web-project so its installation process requires some extra additional steps. Especially the front-end is organized in two separate projects. These two separate projects should be run in two separate processes, which introduces additional effort in booting the tool. However, all of the instructions to run project properly are explained in detail in its public repository.

For the last phase, we comment on the usability for each tool over five criteria listed, defined and examined in Table 6. The criteria that were used to investigate the usability is partly based on ISO/IEC 25010 standard (ISO/IEC25010, 2011) and partly on a similar evaluation performed by Taibi et al. (2019) in their study. Operability criteria in ISO standard is examined under two parts, autonomicity and tunability.

In terms of first-time usage experience and documentation support within and outside the software, Service cutter presents the most comprehensible and detailed support. Microservice-extractor

**Table 6. Subjective evaluation of microservice extraction tools' usability**

| Criteria | Definition | Evaluation |
|---|---|---|
| Learnability/ Understandability | Perceived easiness for the users to achieve specified goals of learning to use the tool in the specified context of use | Service Cutter > Microservice-extractor = Decomposer |
| Adaptability | Perceived easiness for the users to prepare required input for the software | Decomposer [3] Microservice-extractor > Service Cutter |
| Autonomicity/ Self efficacy | Perceived easiness to perform the basic functionality of the software | Microservice-extractor > Decomposer > Service Cutter |
| Tunability | Perceived easiness to tweak the properties that make it easy to operate and control the software | Service Cutter > Microservice-extractor [3] Decomposer |
| Interface aesthetics/ Ease of use | Perceived easiness to apply the tool with a pleasing and satisfying interaction using the provided user interface | Service Cutter > Microservice-extractor > Decomposer |

and Service cutter have detailed prospective to execute actions on their platforms. Github pages, documentations and instructions are sufficient to run extraction on these platforms. Decomposer is a relatively simpler but it has some shortcomings when it comes to providing guidance through software use. Even though it is very easy to use Decomposer, it requires great effort to rationalize the output provided by it and tune the tool accordingly.

Service cutter is also used and examined by many different academic studies and thesis. This much of investigation and documentation support makes it more preferable in terms of understandability compared to other tools under investigation. While both Service cutter and Microservice-extractor provides user friendly web interfaces, Service cutter is more understandable compared to Microservice-extractor by the documentation in its user interface and user orientation functionality provided by the tool.

Decomposer and Microservice-extractor both work on software artifacts that are produced during the development phase of the software while Service cutter needs additional effort and transformations on the software artifacts to be able to run efficiently. Decomposer and Microservice-extractor both may have downsides and additional minimal effort during the preparation process. For instance, even though unlikely for the web services, it might not be always possible to come up with OpenAPI compatible API specifications. Likewise, Microservice-extractor uses git-based repository histories which may not be present for all cases. Service cutter, on the other hand, requires its own format to be prepared as input which contains information about system interfaces, their relations and nano-entities.

Microservice-extractor tool provides a more automated platform to decompose classes into clusters compared to the rest. Decomposer also works in a fully automatic way but the incompatibilities between the terms in the source code and the semantic vocabulary might require extra effort to apply the tool correctly. Service cutter provides a similar level of applicability once a wide range of parameters are tuned properly and the required input is defined properly and in detail, which requires manual effort. While the branching state machine has both write and read-only-paths throughout its execution cycle, a read-only-path can be failed by another machine with a write-path, in order to prevent the read event to return the wrong result.

Service cutter provides a wide range of parameters that lets the user tune the way the decomposition is performed ranging from the number of microservices to be extracted to the method to be used while clustering entities in the software. Microservice-extractor and Decomposer comparably provides less parameters which is better for ease of use and autonomicity but worse for tuning the way extraction approach is applied on the legacy software. It is hard to find a significant difference between the

Decomposer and Microservice-extractor, each of them has its own advantages but our preference was over Microservice-extractor because of the ease of parameter tuning over a comprehensible GUI.

Primarily, it is crucial to point out that Decomposer does not provide a GUI while others do. As a command line program, Decomposer have its merits to be run more simply than the rest but it also lacks necessary level of interaction when it comes to parameter tuning, output visualization, etc. When it comes to compare the software with GUI, Service cutter provides a slightly better impression because of the ability to interactively tune the way extraction is performed by providing side-by-side comparison between the input parameters and output clusters (see Figure 1). Microservice-Extractor provides an easy-to-use, simple jGraph based interface to provide its output but it is relatively untidy compared to Service cutter (see Figure 2).

## CROSS COMPARISON OF MICROSERVICE EXTRACTION TOOLS

This section provides a cross-comparison between the selected tools over common case studies that has been used in the microservice extraction studies. As Table 7 shows there is a little consensus on the type of the case studies that might be used to evaluate the microservice extraction process. Many of the tools use custom coded software or artificial input, where some of them use proprietary software and two of them use publicly available input that can be used in cross comparison. We have selected the following software that has been used frequently by the microservice extraction studies surveyed in this paper.

- Pet Clinic Application (Nicoll, 2016): Pet clinic has been used for evaluation by Microservice-Extractor. Petclinic is an example application that has also been used by Spring boot community as an on-boarding example project. These projects main functionality is to reserve veterinarians

**Figure 1. Portion of a sample extraction output by Service Cutter**

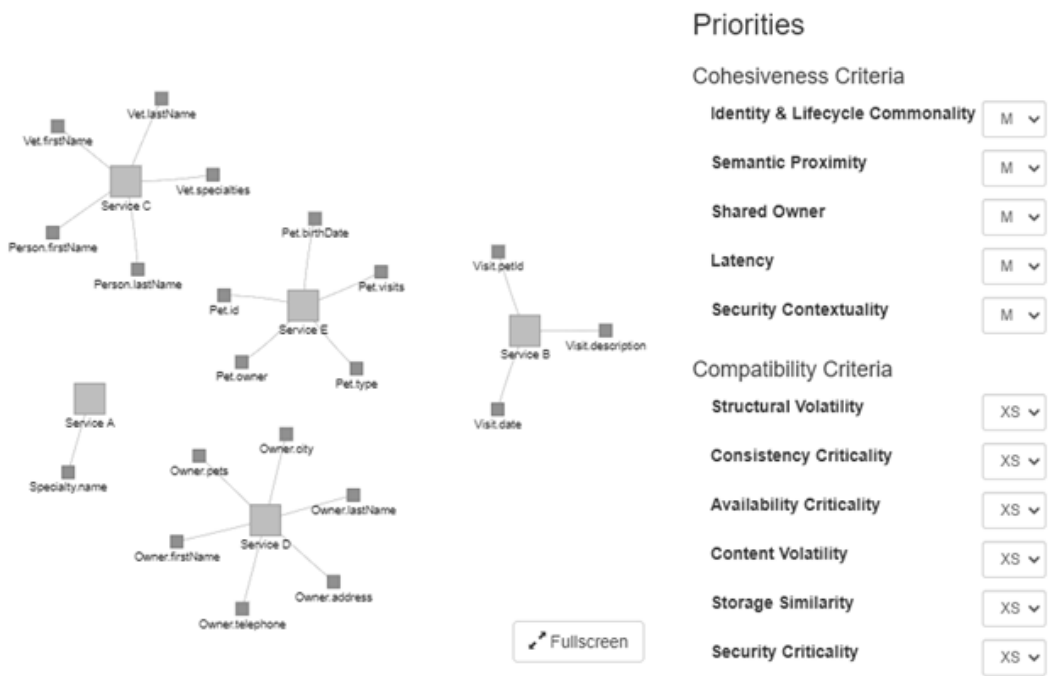
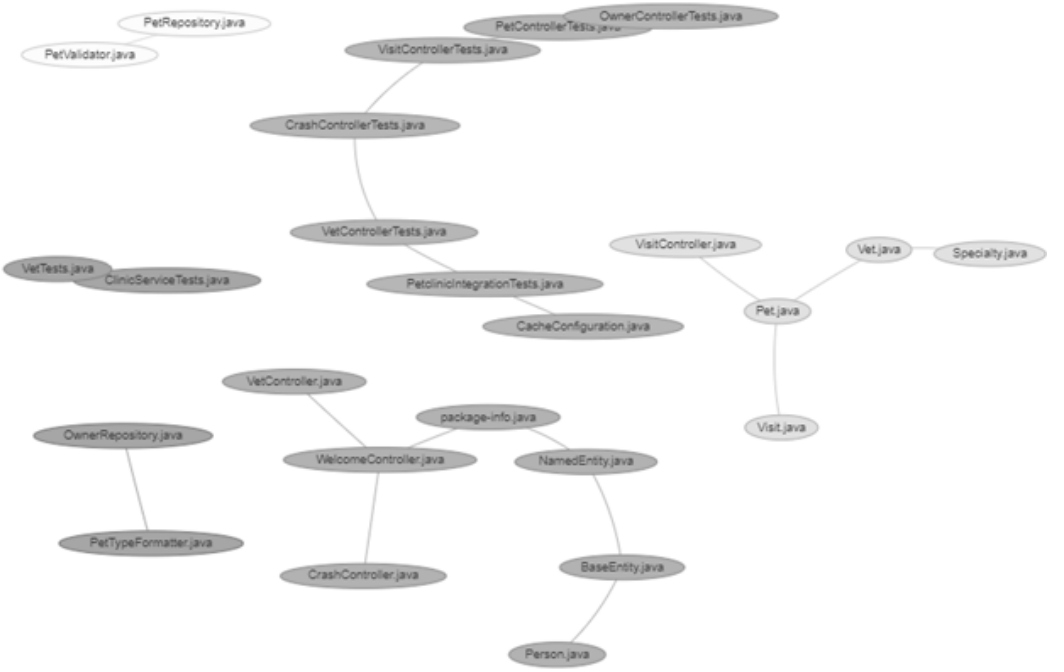


**Figure 2. Portion of a sample extraction output by Microservice-extractor**




and manage animal visits. It is possible to organize veterinarian types and animal visits and manage owner information for pets.

- Cargo Application (Hammarback, 2013): Cargo application has been used both on Service cutter and Decomposer. Cargo application is a web application that manages cargo routes, handling, itinerary and administration processes,
- Trading System Application (Souza, 2018): We chose a project that has not been used for evaluation in any of the discussed papers. Trading system is a web application that manages users trading orders, positions and recommendations. Its main functions involve new management, account management and order/position management.
- Acme Air Application (Spyker, 2015): This is another project that has not been used for evaluation in any of the discussed papers. Acme air similar to other projects is web application to manage airlines companies' operations. Its principal functions include flight operations, user management, flight inquiry and reporting.

Selected three tools have been used over the software briefed above to compare the extracted services and the software components that has been included in the services. At the meantime, an independent software developer has manually analyzed these software and extracted the microservices to be used as a ground truth for evaluating the tools' performance.

During tools' execution, a substantial amount of parameter tweaking is required to obtain comparably the most meaningful set of microservices. In the present comparisons, the best results provided by each tool is intuitively compared after the mentioned parameter tweaking process.

From an overall perspective the number of microservices extracted by each microservice tool is presented in Table 7. Most notably, almost all of the tools presented in the table suffer from over-decomposition of microservices. One of the reasons behind this situation is the inclusion of many highly coupled utility classes into automatic decomposition process. Even though utility class clusters

**Table 7. Number of services extracted from public software by tools under investigation**

| | Service Cutter | Microservice-extractor | Decomposer | Manual decomposition |
|---|---|---|---|---|
| **Pet Clinic App.** | 5 | 5 | 6 | 4 |
| **Acme Air App.** | 8 | 15 | 24 | 5 |
| **Cargo App.** | 7 | 25 | 9 | 5 |
| **Trading App.** | 10 | 16 | 21 | 6 |

have been excluded from the extraction results there still exists some of the clusters that contains both utility classes and domain classes. However, even though such results are ignored there are still over-decomposed microservices extracted by the tools that has been used in this study.

Detailed comparison of automatic service extraction and manual extraction for Pet Clinic software is presented in Table 8. Before moving on to further discussion an explanation on how to interpret the results in the following tables shall be presented. Software specific comparison tables' first column always lists the manually extracted services, represented by a suitable name for the microservice. Afterwards, a tool specific id is manually assigned to each microservice extracted by the tool. This id is a simple number that starts from 1 and ends by the number of microservices extracted by the tools. As mentioned before, some tools extracted microservices that are solely composed of utility classes such as DateTime or Test classes. Such microservices are not included in the presented tables except the ones that also contain domain classes.

As the second third and fourth columns of our result tables the extracted microservices are clustered into manually extracted services for each tool under investigation. Some automatically extracted microservices might belong to more than one manually extracted microservice, which indicates that the components inside the microservice should be distributed among many other services. Additional rows are added that are separated by a line from manually extracted microservices. These microservices can be identified as possible additional microservices that can be treated as a separate microservice that were handled in a different way during manual decomposition. These microservices are found to be too small or treated as a component of another microservice during the manual decomposition. Instead of interpreting manual decomposition as favorable, those additional microservices extracted by tools reveals alternative ways to perform service decomposition which is a positive aspect for such tools.

Finally, if the tool did not identify any services that can be affiliated with a service that has been identified in manual decomposition the term "None" is used in the corresponding cell.

**Table 8. Services extracted from pet clinic application software**

| Manually extracted services | Service Cutter | Microservice-extractor | Decomposer |
|---|---|---|---|
| Owner | 4 | 4 | 1-3 |
| Pet | 2 | 1,2,5 | 4 |
| Visit | 3 | 1,2 | 5 |
| Vet | 1,5 | 1-3 | 6 |
| | | Controllers (3) | FindOwner (2) |
| | | | ListOwner (3) |

For the pet clinic software in Table 8, manual decomposition identified four services as follows:

- Pet microservice to provide services about pet operations.
- Vet microservice to provide services about veterinary operations.
- Owner microservice to provide services about pet owner operations.
- Visit microservice to provide services about veterinary visit operations.

On the other hand, automatic decomposition tools extracted five or more services for Pet clinic.

- Service cutter successfully identified pet, vet, visit and owner services. In addition to these, it also identified a specialty microservice which is used to provide operations on veterinary specialty.
- Microservice-extractor tool also identified five services but in a different way. This tool identified two small services called "Owner repository" and "Pet repository" where CRUD operations of these two entities are clustered. It also identified three more services, containing a mixed interaction between veterinary, pet and visit classes.
- Decomposer identified six microservices. It successfully identified the four microservices of the manual decomposition, but it also extracted an additional owner information retrieval microservice and owner finding microservice.

For the acme air software in Table 9, manual decomposition identified five services as follows:

- Flight operations microservice to provide services regarding flights such as assigning flight times.
- Flight query microservice to provide services that enables to query present flights.
- User operations microservice to provide services regarding users such as editing user information.
- Book flight microservice to provide services booking flights and operations on booking information.
- Reporting microservice to provide services on reporting system operations.

On the other hand, automatic decomposition tools extracted many more services for acme air.

- Service cutter over-decompose flight operations and booking operations, which is reasonable since these two services were the largest microservices in manual decomposition. Service cutter also extracted a microservice, related with trip operations which handle information specific to flight trip such as actual trip time, actual arrival and departure times, etc.
- Microservice-extractor over-decomposed but identified the same services as in manual decomposition.

**Table 9. Services extracted from acme air application software**

| Manually extracted services | Service Cutter | Microservice-extractor | Decomposer |
|---|---|---|---|
| Flight operations | 1,4,5,7 | 8,11,13,15 | 1,7,14,19 |
| Flight query | 1 | 8,11,15 | 7,8,14,18 |
| User operations | 4 | 2,10,12 | 6,9,11,16-18,21-24 |
| Book flight | 3,4,6,8 | 4,9,15 | 2-4,6,10,12,13,18,20,22 |
| Reporting | 2 | 3,7,14 | 1,8,15,24 |
| | Trip (1) | | Session (5,11,18) |

- Decomposer both over-decomposed and extracted an additional microservice that is related to user sessions during operations. This extra microservice is questionable since it is more related with system operations rather than the domain itself.

For the cargo application software in Table 10, manual decomposition identified five services as follows:

- Cargo admin microservice to provide services administrator operations on cargo services.
- Cargo listing microservice is used to filter and list current cargo information.
- Itinerary microservice provides functionality to fill in itinerary for a cargo delivery.
- Delivery microservice to provide information on a cargo on the actual distribution process.
- Handling microservice provides monitoring and managing interface in order to take customer cargo from their addresses.

On the other hand, automatic decomposition tools extracted more services for cargo application.

- Service cutter slightly over-decomposed but identified the same services as in manual decomposition.
- Microservice-extractor over-decomposed but identified the same services as in manual decomposition.
- Decomposer slightly over-decomposed but identified the same services as in manual decomposition.

For the trading application software in Table 11, manual decomposition identified six services as follows:

- Order microservice to manage trading orders.
- News microservice to provide services regarding trading news published by the system.
- Recommendations microservice performs recommendations based on previous user activity and by experts.
- Account microservice handles user account operations.
- Portfolio microservice to handle user trading entities.
- Position microservice handles the long-term trading orders.

On the other hand, automatic decomposition tools extracted many more services for trading application.

**Table 10. Services extracted from cargo application software**

| Manually extracted services | Service Cutter | Microservice-extractor | Decomposer |
|---|---|---|---|
| Cargo admin | 1,5 | 7,11,12,20 | 1-5 |
| Cargo listing | 1,5 | 7 | 3 |
| Itinerary | 4 | 3,5,10,21,23,25 | 4,5 |
| Delivery | 2,3,6 | 2,3,5,10,11,14,15,21,23,25 | 6-8 |
| Handling | 7 | 4,8 | 9 |

**Table 11. Services extracted from trading application software**

| Manually extracted services | Service Cutter | Microservice-extractor | Decomposer |
|---|---|---|---|
| Order | 2,4,7,9 | 1,3,4,7,10,15,16 | 7,10-12,14-16 |
| News | 6,10 | None | 9 |
| Recommendation | 5,8 | 5,6,8 | 6,17 |
| Account | 5 | 6 | 1,5 |
| Portfolio | 3,7 | 4,9 | 4,6,19,21 |
| Position | 1,3,7,9 | 12-14,16 | 2-4,13,15,16,18,19 |

- Service cutter clustered services that regard orders and positions as well but in a different way, by clustering components in those microservices differently.
- Microservice-extractor over-decomposed but also fail to identify a microservice that can be affiliated with news microservice. The tool scattered the components in the news microservice in a great extent. News microservice lists latest news related to economics.
- Decomposer over-decomposed but its output meets with our expectations.

As an overall evaluation and comparison among the extraction tools, an observable pattern is present the way each tool performs the extraction. Comparably, Service cutter is able to avoid the over-decomposition that can be counted as a common flaw for the extractor tools under investigation. Microservice-extractor is also successful in matching the manually decomposed services like Service cutter but it also over-decomposes by a great extent. In three out of four cases, Decomposer identified an additional microservice which has not been separated by manual decomposition which can provide a different perspective during microservice extraction process.

## MICROSERVICE EXTRACTION TOOLS IN THE WILD

After applying microservice extraction tools over code repositories that has been subjects to previous studies two in-house software from a real-world bank's software vendor will be applied. Just like the applications in the previous section, first microservices extracted by a manual process is obtained from a group of developers and the outcome is compared with the extracted microservices by the tools under investigation. Additionally, this time manual decomposition is performed by developers with domain expertise on the subject, since the applications under microservice extraction is still being used and developed actively as in-house applications in a commercial bank in Turkey.

Two different applications have been used, namely "Notification Hub" and "Product Organization", in the presented analyses. Before explaining those applications in detail, the number of microservices discovered by each tool and by manual decomposition is presented in Table 12. It can be seen that

**Table 12. Number of services extracted from in-house developed software**

| | Service cutter | Microservice-extractor | Decomposer | Manual decomposition |
|---|---|---|---|---|
| **Notification hub** | 7 | 9 | 7 | 5 |
| **Product organization** | 8 | 19 | 17 | 7 |

similar results have been obtained for the real-world software where over-decomposition is seen as a common flaw for all the tools under investigation.

The first application we apply the microservice extraction tools was called Notification Hub. Notification hub is the notification management system for the bank's mobile users. Its main function is to send mails or push notifications and short messages to customers. Also, some other systems use this tool to send notifications. This tool is organized as a spring boot project and runs on a web server in a containerized environment. We have chosen this project because of the simplicity; therefore, its code organization and metrics are more simple than other monolithic software in company. Moreover, it has a structure more prone to microservice decomposition cycle.

For the notification hub software in Table 13, manual decomposition identified five services as follows:

- User notification microservice provides services related to common operations in different kinds of user notification operations.
- Notification template microservice governs the template preparation and application services.
- Push notifications microservice provides services to identify and publish push notifications.
- Device microservice provides services to send notification to different kind of medium called channels such as mobile application, short message, email, etc.
- User token microservice provides services to manage user tokens used for authentication purposes.

On the other hand, automatic decomposition tools extracted more services for notification hub.

- Service cutter almost made a one-to-one mapping with manual extraction. There is also an additional microservice extracted by the tool which contains components that govern the user operations and some utility classes. This additional microservice can be seen as the single drawback of the tool.
- Microservice-extractor over-decomposed most of the services. However, interestingly, it also merged device and user token services into a single service. It also identified an additional service as "Validation Service" which governs the validation operations applied during the software services. Validation service may be seen as being composed of utility classes but it also is a favorable recommendation to use a microservice for validation operations.
- If the over-decomposition is ignored, Decomposer extracted the same services as in manual decomposition.

Product Organization is the fundamental sub-system of the main monolithic system named core banking which governs all the fundamental banking operations offered by the bank. This

**Table 13. Services extracted from notification hub application software**

| Manually extracted services | Service Cutter | Microservice-extractor | Decomposer |
|---|---|---|---|
| User notification | 6,7 | 3,4,6,9 | 1,3,4,6,7, |
| Notification template | 6 | 5,9 | 2 |
| Push notification | 2 | 2-4,8,9 | 2,5,7 |
| Device | 3 | 1 | 6,7 |
| User token | 1,3 | 1 | 5,6 |
| | Application (4,5,7) | Validation (7) | |

sub-module is consisted of management of organizations, menus, companies, users and their authorizations in Core Banking. This project has been chosen because of its high complexity and its generic implementation details.

The main drawback of the core banking project is the heavily use of reflection module provided by Java framework. This kind of structure is particularly challenging when it comes to automatic analysis because of the difficulties in resolving class and method dependencies. Too much use of the reflection causes more effort in extracting classes, methods and their relationships, dependencies and during the analysis of software quality metrics in a proper way.

For the Product Organization software in Table 14, manual decomposition identified seven services as follows:

- User operations microservice clusters operations on the real persons using the system.
- Organization operations microservice clusters operations on the organizations using the system.
- Authorization operations microservice operates on users, companies and customers authorizations.
- Role menu operations microservice provides us to manage menu authorizations and visibilities for each actor in the system.
- Role menu listing microservice designed to list menu and its components according to roles.
- User menu listing microservice is used to manage users and their menu operations.
- Organization listing microservice provides simple services to list organizations and corresponding users in the system.

On the other hand, automatic decomposition tools extracted more services for product organization.

- Service cutter extracted the same services as in manual. The tool tried to pack together classes from user operations microservice and authorization microservice which is reasonable since these services are quite coupled. Similar behavior was observed for "operation" and "listing" services as well. This is also reasonable since listing services can be also treated as operations microservices.
- Microservice-extractor also extracted all services by over-decomposition. This tool tends to pack common classes from whole the listing operations in smaller microservices.
- For this case, Decomposer produced similar output with Microservice-extractor, over-decomposed and scattered among listing services and operations microservices consecutively.

As a positive aspect, all the microservice extraction tools produced similar results as in cases for public code repositories, they provided more microservices than manual decomposition; when

**Table 14. Services extracted from product organization application software**

| Manually extracted services | Service Cutter | Microservice-extractor | Decomposer |
|---|---|---|---|
| User operations | 1,4,7 | 4,10,16 | 9,16,17 |
| Organization operations | 1,2 | 4,5,7,9,15 | 2-4,7 |
| Authorization operations | 3,4,7 | 3,6,7,12 | 1,15 |
| Role menu operations | 3,5 | 1,4,7,10 | 13,16 |
| Role menu listing | 5 | 8,14,18,19 | 6,7,12-14 |
| User menu listing | 4-6 | 2,14,16,19 | 6,11-14 |
| Organization listing | 2,5,8 | 11,13,14,17 | 5-8,10,13-14 |

investigated deeper it can be seen that extraction tools provided even smaller microservices that provide, most of the time, the same microservices as in manual decomposition.

## OVERALL EVALUATION OF META-DATA BASED MICROSERVICE EXTRACTION TOOLS AND AREAS FOR IMPROVEMENT

During the previous sections experimental results on the microservice extraction tools under investigation is presented and insight about interpretation of these results is provided. In this section an overall evaluation for meta-data based microservice extraction tools is provided based on the comments in the previous sections and additional comments for specific improvement ideas.

Our evaluations are performed under two categories. First group of comments would be on the overall usability for the meta-data based microservice extraction tools. In the second part, extraction performance of the tools is evaluated and possible points of improvements are provided.

### Evaluating Usability of Microservice Extraction Tools

Most of the comments on the usability of the tools provides a trade-off. It is possible to begin by the input that the tools are accepting in order to perform their evaluations. All in all, the leaner that the tool accepts as input the easier it becomes to use the tool. Decomposer and Microservice-extractor accept inputs that can be obtained and extracted more easily from static artifacts of the development process where Service cutter requires comparably more effort to obtain the inputs to be presented to the tool. This situation results in a deeper analysis of the software at hand but on the other hand as the amount of effort to use the tool increases it becomes less preferable against a manual service extraction in comparison.

Considering the comparison with manual efforts we believe it is very important that the microservice extraction tool to be used accepts a widely accepted standardized format of input, or provide necessary tools to transform the software development artifacts into such formats to be more usable. A general area of improvement for tools that utilize static analysis or meta-data should pay a great deal of attention to automatize the way the input is obtained.

Another area of automation about the tools under investigation is the amount of parameter tweaking that is needed in order to run the tool. Again, the trade-off in here is usability against adaptive performance where the more tweaking user is able to perform the more customizable tool becomes. Service cutter provides the broadest amount of tweaking available compared to other tools where user may choose different types of clustering algorithms as well as the number of services that she expects from the tool. The key point for usability in providing a broad set of parameters lies in how explainable each parameter (or set of parameters) is presented to the user. For instance, it is very useful to have different kind of clustering methods to be used during extraction process but the proper application areas of these clustering methods should not be left to user intuition. The tool should provide simple explanations under what circumstances it is better to use each clustering method to provide insight to the user.

Providing a graphical user interface (GUI) is also another important property that the service extraction tools need to provide. Most of the time, the software under investigation contains many different classes and components and for the user to obtain insight about how the extraction process is performed providing a picture of component clustering makes a lot of different. Moreover, provided GUI should also neatly allow user to apply and experience results of different parameter tweaking results which we believe is provided by Service cutter in a better way. A functionality that is viable by a GUI is the amount of interaction with the user, where the user may interfere with clustering results or provide a custom cluster based on her own domain expertise. This property is not directly offered yet by the tools under investigation and seems like a promising area of research for such kind of tools.

A common advantage of all tools is their source code being publicly available; user may develop additional features or refactor available features to adapt the tool for specific areas of applications.

On the other hand, all the tools need substantial amount of effort to set up and boot when used as on premise.

As an overall evaluation, a microservice extraction tool that adopts the best features of the three tools under investigation might be superior in terms of usability in this area. Microservice-extractor uses the leanest and most easily obtainable inputs where Service cutter provides better parameter tweaking and graphical user interaction and Decomposer involves automatic semantic reasoning on top of static analysis. Additionally, custom/editable service clustering by the user interaction and an adaptive learning engine that considers custom clusters in future decision would increase the usability of the tools by a great extent.

## Evaluating Performance of Microservice Extraction Tools

It shall be appropriate to begin with favorable properties on the performance of the meta-data based microservice extraction tools. All the tools extracted services that can be mapped to manually extracted microservices by a great extent. Because of the fine-grained microservices extracted by the tools, most of the time a group of extracted microservices can be mapped to a manually extracted microservice. Another important trend can be seen in evaluation of the tools where the latter studies rely on evaluation by real world case studies where former tools perform evaluation on custom coded software. A common trend among the techniques being used is graph or string clustering, which seems to be natural because of the characteristics of the microservice extraction process. An alternative approach is presented in MicroART tool where many different pieces of information is mapped to a meta-model, specific to microservice modeling domain. However, the details of the mapping procedure are not provided in the related papers' content (Granchelli et al., 2017).

Another favorable property of the tools under investigation is for some of the cases they have also offered an authentic microservice which was not extracted by manual decomposition but provide insight on performing a different clustering among software classes. Providing different perspective by such kind of identifications is one of the good reasons behind using an automated tool.

The obvious drawback in meta-data based microservice extraction seems to be over-decomposition of services during extraction. For all of the cases, all of the investigated tools provide finer grained microservices that the manual extraction process. Service cutter, by allowing user to input the expected number of microservices, performs the best in this respect but even this tool provided more microservices than manual extraction. The possible reason behind this situation may be the lack of semantic information behind the clustering process. For instance, Microservice-extractor uses information from developer activities to bridge this gap, while Decomposer uses a semantic dictionary and service Cutter uses mostly the user input. However, still they were not able to identify necessary amount of (possibly semantic) relations between the identified clusters.

A possible way to deal with this drawback is to use additional information about the software that is being analyzed. Such information may stem either from the user or from other artifacts and run-time information. Literature review in the previous sections suggest that microservice extraction has also been expanded to that direction as well. On the other hand, obtaining more information may increase required effort and decrease usability of the software. Meta-data based extraction tools has a potential in providing rapid support during microservice extraction process to user and using already present software artifacts without a need of transformation to a possible required format greatly increases potential usage of the tool. For instance, Microservice-extractor can be directly applied on code repositories without any necessity to perform input tweaking which serves as a potential advantage. Hence, focusing on automatic extraction of more meaningful information from meta-data input is a very valuable direction for microservice extraction studies.

A final remark about the extraction performance of the tools is the obvious need of manual interference during the run-time of the tools. Even though manual interference is an important disadvantage during input preparation stage of the tools, manual interference

during run-time is more acceptable and even improve the overall performance of the tools. Service cutter provides the most user interaction and it can be seen that it provides better results compared to other tools which provides a valuable insight on increasing the tools' performance. Microservice extraction is closer to an exploratory study rather than a black box process where user interaction and learning from those interactions is the most valuable input that can be provided to an extraction tool.

## CONCLUSION AND LIMITATIONS OF THE STUDY

In this study three publicly available microservice extraction tool are compared that work by only using meta-data information about software such as design models, developer activity, interface definitions, static analysis, etc. Presented analyses show that all the available tools suffer from over-decomposition of the services where a finer grained service extraction is performed with respected to manual decomposition performed by experienced developers and domain experts. Common usability disadvantages of the tools are also identified and insight about the future approaches is provided to perform better microservice extraction.

Almost all of the recent studies focus on methodological comparison instead of performing a comparative study on tool support and practical aspects of the provided tool. Our study fills this gap in the current literature by providing an extensive comparative study on meta-data based tools and identify possible future directions for the researchers and practitioners studying in the field.

As for the limitations, primarily, the focus of this study is on a narrow context where three different tools that work only by using meta-data (and/or software models) and performing static analysis on software are experienced. Microservices are still a developing area of software development and automatic extraction is a niche area which results in a small number of available tools. Microservice extraction is currently in stage of lower technological levels where provided tools are mostly academic studies that are not yet mature enough to be used in industry. This study's aim was to provide insight on the valuable method of meta-data based extraction and in order to provide repeatable experiments the tools that are publicly available have been used.

Another limitation of this study is the number of case studies that have been applied for the tools under investigation. Because of the nature of software development, developer insight might become more important than only using quantitative measures. Manual comparison is avoided as much as possible during the comparison of the tools which resulted in relatively a few case studies. However, most of the microservice extraction literature use small number of case studies during evaluation and perform deeper analysis on the case studies since it is more valuable to show that the extraction tool performs closer results to manual decomposition.

Last but not least, because of the mentioned qualitative nature of the process, subjective comments have been provided on the tools that are being compared. During the evaluation process isolated groups of developers are employed consecutively to use the tools for extraction and perform manual decomposition to avoid any kind of bias that may stem from either of the extraction methods. Nevertheless, comments on microservice extraction might differ from developer to developer resulting in different kinds of manual decomposition. In order to lesser these differences a relatively small number of projects have been used for cross comparison and domain experts' idea have been obtained for the in-house software decomposition comparison.

The key insight was on developing tools that need less effort on the inception phase (such as input preparation) and utilizing smarter user interaction during the service extraction time. This study can be extended to involve tools that use workload input and analyze the marginal benefit of using workload information on microservice extraction. It is also possible to extend the research further by addressing the commercial microservice extraction tools such as AWS' microservice extractor .NET.

## ACKNOWLEDGMENT

This research was supported by İTÜ Nova projects jointly under projects numbered 2020000784 and 2021001003 in collaboration with Fibabanka A.Ş.

## FUNDING AGENCY

The publisher has waived the Open Access Processing fee for this article.

## REFERENCES

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

Al-Debagy, O., & Martinek, P. (2021). Dependencies-based microservices decomposition method. *International Journal of Computers and Applications*, 1–8. doi:10.1080/1206212X.2021.1915444

Amiri, M. J. (2018, July). Object-aware identification of microservices. In *2018 IEEE International Conference on Services Computing (SCC)* (pp. 253-256). IEEE. doi:10.1109/SCC.2018.00042

Baresi, L., Garriga, M., & De Renzis, A. (2017). Microservices identification through interface analysis. In *European Conference on Service-Oriented and Cloud Computing* (pp. 19-33). Springer. doi:10.1007/978-3-319-67262-5_2

Carvalho, L., Garcia, A., Assunção, W. K., Bonifácio, R., Tizzei, L. P., & Colanzi, T. E. (2019). Extraction of configurable and reusable microservices from legacy systems: An exploratory study. In *Proceedings of the 23rd International Systems and Software Product Line Conference*-Volume A (pp. 26-31). doi:10.1145/3336294.3336319

Carvalho, L., Garcia, A., Assunção, W. K., de Mello, R., & de Lima, M. J. (2019). Analysis of the criteria adopted in industry to extract microservices. In *2019 IEEE/ACM Joint 7th International Workshop on Conducting Empirical Studies in Industry (CESI) and 6th International Workshop on Software Engineering Research and Industrial Practice (SER&IP)* (pp. 22-29). IEEE. doi:10.1109/CESSER-IP.2019.00012

Chen, R., Li, S., & Li, Z. (2017). From monolith to microservices: A dataflow-driven approach. In *2017 24th Asia-Pacific Software Engineering Conference (APSEC)* (pp. 466-475). IEEE. doi:10.1109/APSEC.2017.53

Daoud, M., El Mezouari, A., Faci, N., Benslimane, D., Maamar, Z., & El Fazziki, A. (2021). A multi-model based microservices identiftcation approach. *Journal of Systems Architecture*, *118*, 102200. doi:10.1016/j.sysarc.2021.102200

De AlwisA. A. C. (2018). *SubType Github Page*. Retrieved from https://github.com/AnuruddhaDeAlwis/Subtype

De Alwis, A. A. C., Barros, A., Polyvyanyy, A., & Fidge, C. (2018). Function-splitting heuristics for discovery of microservices in enterprise systems. In *International Conference on Service-Oriented Computing* (pp. 37-53). Springer. doi:10.1007/978-3-030-03596-9_3

Escobar, D., Cárdenas, D., Amarillo, R., Castro, E., Garcés, K., Parra, C., & Casallas, R. (2016). Towards the understanding and evolution of monolithic applications as microservices. In *2016 XLII Latin American Computing Conference (CLEI)* (pp. 1-11). IEEE. doi:10.1109/CLEI.2016.7833410

Eski, S., & Buzluca, F. (2018). An automatic extraction approach: Transition to microservices architecture from monolithic application. In *Proceedings of the 19th International Conference on Agile Software Development: Companion* (pp. 1-6). doi:10.1145/3234152.3234195

Fritzsch, J., Bogner, J., Zimmermann, A., & Wagner, S. (2018). From monolith to microservices: A classification of refactoring approaches. In *International Workshop on Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment* (pp. 128-141). Springer.

Garriga, M. (2016). *Decomposer Github Page*. Retrieved from https://github.com/mgarriga/decomposer

Granchelli, G. (2017). *MicroART Github Page*. Retrieved from https://github.com/microart/microART-Tool

Granchelli, G., Cardarelli, M., Di Francesco, P., Malavolta, I., Iovino, L., & Di Salle, A. (2017). Towards recovering the software architecture of microservice-based systems. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)* (pp. 46-53). IEEE. doi:10.1109/ICSAW.2017.48

Guerfel, R., Sbaï, Z., & Ayed, R. B. (2017). On the Use of Similarity or Query Languages in Cloud Discovery Based on Ontology. *International Journal of Service Science, Management, Engineering, and Technology*, *8*(3), 60–78. doi:10.4018/IJSSMET.2017070104

Guermah, H., Guermah, B., Fissaa, T., Hafiddi, H., & Nassar, M. (2021). Dealing With Context Awareness for Service-Oriented Systems: An Ontology-Based Approach. *International Journal of Service Science, Management, Engineering, and Technology*, *12*(4), 110–131. doi:10.4018/IJSSMET.2021070107

Gupta, M., & Garg, A. (2015). A Comparative Analysis of Content Delivery Network and Other Techniques for Web Content Delivery. *International Journal of Service Science, Management, Engineering, and Technology*, *6*(4), 43–58. doi:10.4018/IJSSMET.2015100104

Gysel, M. (2015). *Service Cutter Github Page*. Retrieved from https://github.com/ServiceCutter/ServiceCutter

Gysel, M., Kölbener, L., Giersche, W., & Zimmermann, O. (2016). Service cutter: A systematic approach to service decomposition. In *European Conference on Service-Oriented and Cloud Computing* (pp. 185-200). Springer. doi:10.1007/978-3-319-44482-6_12

Hammarback, J. (2013). *DDD Sample Cargo Application*. Retrieved from https://sourceforge.net/projects/dddsample

Hassan, S., Ali, N., & Bahsoon, R. (2017). Microservice ambients: An architectural meta-modelling approach for microservice granularity. In *2017 IEEE International Conference on Software Architecture (ICSA)* (pp. 1-10). IEEE. doi:10.1109/ICSA.2017.32

ISO/IEC25010. (2011). *Systems and Software Engineering: Systems and Software Quality Requirements and Evaluation (SQuaRE): System and Software Quality Models*. International Standards Organization(34:2910).

Jin, W., Liu, T., Zheng, Q., Cui, D., & Cai, Y. (2018). Functionality-oriented microservice extraction based on execution trace clustering. In *2018 IEEE International Conference on Web Services (ICWS)* (pp. 211-218). IEEE. doi:10.1109/ICWS.2018.00034

Kazanavičius, J., & Mažeika, D. (2019). Migrating legacy software to microservices architecture. In 2019 Open Conference of Electrical, Electronic and Information Sciences (eStream) (pp. 1-5). IEEE. doi:10.1109/eStream.2019.8732170

Kim, J. H., & Lennon, S. J. (2017). Descriptive content analysis on e-service research. *International Journal of Service Science, Management, Engineering, and Technology*, *8*(1), 18–31.

Kirby, L. J., Boerstra, E., Anderson, Z. J., & Rubin, J. (2021). Weighing the Evidence: On Relationship Types in Microservice Extraction. In *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)* (pp. 358-368). IEEE.

Klock, S. (2017). *MicADO Github Page*. Retrieved from https://github.com/AMUSEResearch/MicADO

Klock, S., Van Der Werf, J. M. E., Guelen, J. P., & Jansen, S. (2017). Workload-based clustering of coherent feature sets in microservice architectures. In *2017 IEEE International Conference on Software Architecture (ICSA)* (pp. 11-20). IEEE. doi:10.1109/ICSA.2017.38

Kolb, P. (2009). Experiments on the difference between semantic similarity and relatedness. In *Proceedings of the 17th Nordic Conference of Computational Linguistics (NODALIDA 2009)* (pp. 81-88). Academic Press.

Kumar, V., & Bhardwaj, A. (2020). Deploying Cloud-Based Healthcare Services: A Holistic Approach. *International Journal of Service Science, Management, Engineering, and Technology*, *11*(4), 87–100. doi:10.4018/IJSSMET.2020100106

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

Mayer, B., & Weinreich, R. (2018). An approach to extract the architecture of microservice-based software systems. In 2018 IEEE symposium on service-oriented system engineering (SOSE) (pp. 21-30). IEEE. doi:10.1109/SOSE.2018.00012

Mazlami, G. (2016). *Microservice-extractor Github Page*. Retrieved from https://github.com/gmazlami/microserviceExtraction-backend

Mazlami, G., Cito, J., & Leitner, P. (2017). Extraction of microservices from monolithic software architectures. In *2017 IEEE International Conference on Web Services (ICWS)* (pp. 524-531). IEEE. doi:10.1109/ICWS.2017.61

Mono2Micro. (2019). *Mono2Micro Github Page*. Retrieved from https://essere.disco.unimib.it/wiki/arcan

Mono2Micro. (2019). *Mono2Micro Github Page*. Retrieved from https://github.com/socialsoftware/mono2micro

ms Dashboard. (2017). *Microservice visualization service*. Retrieved from https://www.se.jku.at/microservice-dashboard/

Munezero, I. J., Mukasa, D. T., Kanagwa, B., & Balikuddembe, J. (2018). Partitioning microservices: A domain engineering approach. In *2018 IEEE/ACM Symposium on Software Engineering in Africa (SEiA)* (pp. 43-49). IEEE.

Mustafa, O., Gómez, J. M., Hamed, M., & Pargmann, H. (2018). GranMicro: A black-box based approach for optimizing microservices based applications. In *From Science to Society* (pp. 283–294). Springer.

Newman, M. E., & Girvan, M. (2004). Finding and evaluating community structure in networks. *Physical Review. E*, *69*(2), 026113.

Nicoll, S. (2016). *Spring PetClinic Sample Application*. Retrieved from https://github.com/spring-projects/spring-petclinic

Nunes, L., Santos, N., & Silva, A. R. (2019). From a monolith to a microservices architecture: An approach based on transactional contexts. In *European Conference on Software Architecture* (pp. 37-52). Springer.

Park, J., Kim, D., & Yeom, K. (2020). An Approach for Reconstructing Applications to Develop Container-Based Microservices. *Mobile Information Systems, 2020*.

Pigazzini, I., Fontana, F. A., & Maggioni, A. (2019,). Tool support for the migration to microservice architecture: An industrial case study. In *European Conference on Software Architecture* (pp. 247-263). Springer.

Ponce, F., Márquez, G., & Astudillo, H. (2019). Migrating from monolithic architecture to microservices: A Rapid Review. In *2019 38th International Conference of the Chilean Computer Science Society (SCCC)* (pp. 1-7). IEEE.

Procaccianti, G., van der Sanden, B., Li, G., Luo, Z., Hertzberger, B., Kopershoek, M., & Oprescu, A. M. (2016). *Towards a microservices architecture for cloud*. VU University Amsterdam.

Raghavan, U. N., Albert, R., & Kumara, S. (2007). Near linear time algorithm to detect community structures in large-scale networks. *Physical Review. E*, *76*(3), 036106.

Rouached, M. (2021). Web Service Composition Security: A Three-Dimensions Overview. *International Journal of Service Science, Management, Engineering, and Technology*, *12*(3), 154–174.

Saidani, I., Ouni, A., Mkaouer, M. W., & Saied, A. (2019). Towards automated microservices extraction using muti-objective evolutionary search. In *International Conference on Service-Oriented Computing* (pp. 58-63). Springer.

Sbaï, Z., & Guerfel, R. (2016). CTL model checking of web services composition based on open workflow nets modeling. *International Journal of Service Science, Management, Engineering, and Technology*, *7*(1), 27–42.

Schröer, C., Kruse, F., & Gómez, J. M. (2020, September). A Qualitative Literature Review on Microservices Identification Approaches. In *Symposium and Summer School on Service-Oriented Computing* (pp. 151-168). Springer.

Souza, A. (2018). *Sample Trading System Github Page*. Retrieved from https://github.com/apssouza22/trading-system

Spyker, A. (2015). *Sample AcmeAir Github Page*. Retrieved from https://github.com/acmeair

Stojkov, A., & Stojanov, Z. (2021). Review of methods for migrating software systems to microservices architecture. *Journal of Engineering Management and Competitiveness*, *11*(2), 152–162.

Taibi, D., & Systä, K. (2019). From monolithic systems to microservices: A decomposition framework based on process mining. *8th International Conference on Cloud Computing and Services Science, CLOSER*.

Tyszberowicz, S., Heinrich, R., Liu, B., & Liu, Z. (2018). Identifying microservices using functional decomposition. In *International Symposium on Dependable Software Engineering: Theories, Tools, and Applications* (pp. 50-65). Springer.

Zaragoza, P., Seriai, A. D., Seriai, A., Shatnawi, A., Bouziane, H., & Derras, M. (2021). Refactoring monolithic object-oriented source code to materialize microservice-oriented architecture. *16th International Conference on Software Technologies (ICSOFT)*, *117*, 126.

Zhao, J., & Zhao, K. (2021, August). Applying Microservice Refactoring to Object-Oriented Legacy System. In *2021 8th International Conference on Dependable Systems and Their Applications (DSA)* (pp. 467-473). IEEE.

*Kerem Akkaya is currently studying for a Game and Interaction Technologies MSc degree at Istanbul Technical University. He received his undergraduate degree from Computer Engineering at Ege University. His research interests mainly focus on microservices and service-oriented software engineering.*

*Tolga Ovatman received the B.Sc. degree in computer engineering from Hacettepe University, Turkey, in 1999, and the M.Sc. and Ph.D. degrees in computer engineering from Istanbul Technical University (ITU), Turkey, in 2005 and 2011, respectively. He is an Associate Professor with the Computer Engineering Department, ITU. His research interests include distributed systems, cloud computing, model checking, and object-oriented design.*