Faculdade de Engenharia da Universidade do Porto

Defining Metrics for the Identification of Microservices in Code Repositories

Domingos Francisco Panta Junior

Mestrado em Engenharia de Software

Supervisor: Jácome Cunha

Second Supervisor: Lázaro Costa

July 19, 2023

Defining Metrics for the Identification of Microservices in Code Repositories

Domingos Francisco Panta Junior

Mestrado em Engenharia de Software

July 19, 2023

Resumo

Microsserviços tornaram-se o estilo de arquitetura mais utilizado entre todas as estratégias de desenvolvimento de software disponíveis. No entanto, as pesquisas sobre esse tema estão no início, o que dificulta a localização de aplicações de microsserviços em escala para análise. Portanto, há uma grande necessidade de novas investigações, bem como ferramentas para apoiar novos desenvolvimentos no campo de microsserviços.

O primeiro objetivo deste trabalho é coletar características de microsserviços encontradas na literatura e traduzi-las em características mensuráveis no código. Com isso, fornecemos um conjunto abrangente de características, bem como métricas para identificá-las no código.

Um segundo objetivo é usar essas métricas para identificar a base do código seguindo um estilo de arquitetura de microsserviço. Essa solução é disponibilizada por meio de uma ferramenta que permite aos usuários encontrar microsserviços em escalas e filtrá-los de acordo com suas necessidades. Isso pode ser usado para encontrar exemplos de microsserviços em uma linguagem de programação específica ou para criar corpora para estudos de pesquisa.

Nossa avaliação mostra que nosso algoritmo pode identificar microsserviços com uma precisão de 85%.

Palavras-chave: Microserviços, Corpus de software, Métricas

Abstract

Microservices have become the most used architectural style among all available software development strategies. However, it is difficult to find microservice applications at scale for analysis. Therefore, there is a great need for new investigations as well as tools to support new developments in the field of microservices.

The first goal of this work is to collect microservices characteristics found in the literature and translate them into measurable features in the code. With this, we provide a comprehensive set of characteristics as well as metrics to identify them in the code.

A second goal is to design an algorithm to use such metrics to identify code basis following a microservice architectural style. This solution is made available through a tool that allows users to find microservices at scales and filter them according to their needs. This can be used to find examples of microservices in a specific programming language or to create corpora for research studies.

Our evaluation shows our algorithm can identify microservices with a precision of 85%.

Keywords: Microservices, Software Corpus, Metrics

Acknowledgements

I would like to thank my family for always being there for me, even from afar, throughout my academic journey. My wife, Georgia Ricalde, for the love, support, and motivation, without which I would not finish this work. FEUP and all MESW faculty for the opportunity to join the program and the knowledge shared from begging until the end. Professor Jacome Cunha, for asking the right questions to drive this research forward and for all the guidance in making this idea a reality.

Domingos Francisco Panta Junior

“ Learn from yesterday,
live for today,
hope for tomorrow.
The important thing is not to stop questioning.”

Albert Einstein

Contents

List of Figures

List of Tables

Abbreviations and Symbols

MSA	Microservice Application
URL	Uniform Resource Locator
API	Application Programming Interface
UML	Unified Modeling Language
POC	Proof Of Concept
GUI	Graphical User Interface

Chapter 1

Introduction

This chapter starts by presenting and explaining the context in which the topic is inserted. It is then followed by the definition of the problem aimed to be solved as well as the proposed solution. Additionally, the structure of this document is described.

1.1 Context

One of the most important phases of any scientific investigation is the validation of the results obtained. Software corpus paves the way for analysis replication [5], which helps collect metrics for research validation. In investigations regarding mining code repositories, there are very few works trying to create software corpora. This is particularly more challenging in the microservices research field because the different services might be spread over many repositories, and the identification of such services might depend on the documentation about them found in those repositories. Therefore, even though there are some examples of tools created for either building a software corpus, such as the work by Giuseppe et al. [21], or facilitating static analysis on it, like the tool created by Caracciolo et al. [5], there is not a significant amount of research about mining microservices.

Moreover, examining and connecting data with a view to discover valuable actions in software projects is the main goal of the Mining Software Repositories (MSR) area of expertise [15]. One of the motivations behind this is the recurring nature of software projects. When a new project is started, managers and developers rely on previously acquired experience to intuitively estimate tasks that have been done in the past. In modern projects, there are plenty of tools available to support the development process by organizing activities and requirements and producing code. Thus, the project development is actually recorded in software repositories so that data can be analyzed and used as a historical reference to support decision-making [15]. The data available in software repositories are so plural that a whole movement towards this field of research has emerged because of it [23]. Taking a publicly available project hosted in a code repository such as GitHub as an example, just by browsing in it, it is possible to see the directories created for the project, often there is a README.md file by convention, which sometimes is the only source of documentation to describe the purpose of the project, its functionalities, and usage. Additionally, files and their respective extensions indicate the programming languages of choice. And the content of the source code files reveals business rules as requirements implemented. Another important factor is that there is data about the users who contribute to the repository as well as their interactions.

Finally, research on microservice applications (MSA)[1] is a continuously evolving field with ongoing research and advancements. Therefore, there are already various definitions for MSA. What most authors agree on is the goal of microservices, which is to have highly cohesive services defined around business capabilities, that can be deployed independently (self-contained) and use a lightweight communication protocol such as RESTFul[2] or gRPC[3] [25]. MSA migrations and new implementations have been heavily boosted by the advent of cloud computing. While cloud computing provides precise control over resources used by deployed applications, MSAs have the required software quality characteristics to take full advantage of that given flexibility. Hence, the microservice architecture has become the most obvious choice for large applications going forward. Therefore, there is still much to be learned about this recent architectural style, and in order to research or implement MSAs, practitioners often need them at scale with a view to perform experiments, visualize how the actual code of an MSA adheres to the characteristics of this architecture, etc. Thus, researchers and consultants take great benefit from having many and varied MSAs available for their work.

1.2 Problem Definition

There is a need for a corpus of MSAs for several reasons. While a researcher might want a dataset set of MSAs implemented in a specific programming language, another might need to find out libraries and frameworks used by MSAs. Another example would be a software engineer looking at how MSAs have their lightweight protocol quality requirement implemented or what has been implemented in messaging broker technologies. The list of possible usages for identified applications matching specified criteria is vast. For this reason, the main problem we wish to solve is how to find MSAs at scale.

1.3 Objectives

The main goals of this research are to:

- Define a collection of characteristics that allow us to identify MSAs.

- Devise a way to identify such characteristics in the source code.

[1] We use the term microservice to refer to a service of a microservice application, which is an application that follows the microservice architectural style. In this work, we use the term monolith to refer to other architectures.

[2] `https://restfulapi.net/`

[3] `https://grpc.io/`

- Create a method to automatically identify microservices on code repositories.
- Validate the identification method by running it against code repositories.
- Implement a tool to allow users to find and filter (e.g., by programming language) large amounts of MSAs.

1.4 Contributions

This research project makes the following contributions:

- **Systematic Literature Review** - State-of-the-art review in which works about microservices related to this topic were grouped and analyzed with the intention of building a set of heuristics to help distinguish microservices from other architecture styles.
- **Microservice Identification Algorithm** - The heuristics found in the literature review were observed in the code of a curated list of MSAs and further utilized to generate metrics about MSA in an automated fashion.
- **Microservice Mining Tool** - Implementation of the GitHub Microservice Mining Tool that incorporates the algorithm and provides a graphical user interface (GUI) to mine microservices from code repositories.
- **Algorithm Evaluation** - Usage of the created tool and analysis of the algorithm's produced results to determine its efficiency.

1.5 Document Structure

In addition to this chapter, where the context surrounding the research topic was presented along with the problem aimed to be solved, the goals of the work, and the intended contributions to the research community, there are seven more chapters. In Chapter 2, we present the state-of-the-art research relevant to the topic in a systematic manner. In Section 5.1, we present the proposed solution designed to address the problem at hand. In Chapter 3, we present the metrics gathered from code repositories. In Chapter 4, we present the algorithm designed to identify MSAs. In Chapter 5, we present the prototype tool implemented. In Chapter 6, we present the evaluation of the results obtained. In Chapter 7, we draw conclusions and refer to future works.

Chapter 2

Systematic Literature Review

In this chapter, we describe our light-weight systematic literature review aiming to detect literature about MSA corpus, identification, and mining from code repositories. The goal of this review is to verify the existence of an MSA corpus, understand how MSAs are distinguished from other applications, and also how one can design a process for mining MSAs.

2.1 Research Method

The research method for this review followed the three main phases of the review process as specified by Keele et al. [19]: *Planning the Review*, *Conducting the Review*, and *Reporting the Review*.

2.1.1 Planning the Review

The planning phase was started by identifying and choosing relevant sources for software engineering research. After some experimentation and skimming on some results, the digital libraries which presented the most relevant articles for the topic were:

- ACM Digital library
- IEEExplore
- ScienceDirect
- Wiley

With the exception of the Wiley digital library, the other selected libraries are among the seven relevant sources for software engineering listed by Brereton et al. [3] which increased the hopes for good results in this empirical process.

Following, a search was performed on the selected libraries aiming to identify existing reviews on the topic, and so the search query for this experiment was: *(Mining AND (microservices OR "micro-services") and GitHub)*. Even though the query did produce results for some primary studies, it did not find any reviews on the topic which ensured the need for this review.

Research Questions Being the research questions the main portion of a review [19], we took special care defining them. We started by revisiting the goals defined at Section 1.3, and analyzing what needed to be known to accomplish them. Therefore, considering that we want to discover ways of mining microservice applications from source code by empirically building heuristics of microservices identification so that we can make them easily available for practitioners by means of a tool, we asked the following questions:

- *RQ1*: What are the corpus of microservices currently available for researchers and/or practitioners?
- *RQ2*: What aspects of microservices applications can be used for their classification as microservices applications?
- *RQ3*: Can we define an algorithm to recognize MSAs' code repositories based on their characteristics?

To answer these questions, we started with the definition of a search query that would contemplate all areas of interest of the researched topic, namely microservices, mining software repositories, and mining code repositories. During the first phase of experimentation on the ACM Digital library, those terms searched on the full text of all publications available produced over 13,000 results. After skimming over some titles and abstracts, it was clear that many of those publications were not related to the research topic. Thus, it was necessary to filter the results either by narrowing down search terms or the section of the publications in which the terms were being searched. As a result, another experiment was performed matching the terms against the abstract of the publications which produced 879 results. This rather larger number of papers still had many publications off-topic. Therefore, it was clear that we should not only restrict the search to the title and the abstract of the papers but also refine the search query. The result of this refinement produced a total of 238 publications across all four chosen digital libraries, which had a much more close relation to the topic. The final refined query used was the following:
((microservice OR "micro-service") AND (identification OR corpus OR mining OR repository OR GitHub))

2.1.2 Conducting the Review

During the conduction of the review, the defined search was executed in each one of the digital libraries selected and all research works resultant of this search were properly identified, extracted, selected by using inclusion and exclusion criteria, synthesized, and then reported.

Identification of research As detailed in Table 2.1, we identified and extracted from the digital libraries a total of 238 publications. Given the considerable number of works, they were imported into Mendeley, a reference management software chosen to support these tasks.

Selection of primary studies

Table 2.1: Research Identification Summary

Digital Library	Publications
ACM	109
IEEE	93
ScienceDirect	22
Wiley	14
Total	238

The selection of relevant literature related to the theme was based upon the analysis of titles and abstracts of the identified research to verify if they met the inclusion or exclusion criteria presented below.

Inclusion Criteria

- Primary study on the topic.
- Publications about microservices:

  - corpus, in which any set of MSAs are presented and can be reused.

  - design structure, in which the architecture of MSAs is standardized as a model to be followed for new developments.

  - identification from existing systems, where MSA's characteristics that distinguish them from other applications are highlighted.

  - commonly used technologies (libraries and frameworks)

  - mining, where any process for mining MSAs from code repositories is explained.

Exclusion Criteria

- Publication about systems built on microservices architecture.
- Publications about microservices:

  - pitfalls resolution, in which problems related to MSA's development or maintenance are presented.

  - identification from green field development, where the definition of each MSA candidate is determined from business requirements or processes.

Additionally to this criteria, only publications in the English language were considered; any date of publication up until December of 2022 was valid, as well as any authors, article setting, research design, or sampling method.

Data Extraction & Synthesis

The data extraction stage of the process is designed to collect all information required to answer the research questions, while the synthesis summarizes the results of included primary studies, which are selected based on the inclusion and exclusion criteria. As shown by Table 2.2, the global number of publications considered related to the topic was 44, and 194 of the papers met the exclusion criteria. This meant only 18.5% of information had a direct bearing on the search topic. Appendix A lists the included publications. Additionally, the complete list of all publications is available at `https://anonymous.4open.science/r/icsme23-A500/data/sample/articles.xlsx`.

Table 2.2: Included & Excluded Publications By Digital Library

Digital Library	Publications	Included #	Excluded #
ACM	109	18	91
IEEE	93	16	77
ScienceDirect	22	5	17
Wiley	14	5	9
Total	238	44	194

2.2 Results

In this section, the final stage of a systematic review, the contents of the selected articles will be discussed. The main focus in this stage is to group articles highlighting their main topic in regards to how they help to answer the research questions.

2.2.1 RQ1: *What are the corpus of microservices currently available for researchers and/or practitioners?*

In this systematic review, the only work identified that could be considered to have examples of microservices as a small dataset used to validate microservice implementations was the work by Podolskiy et al. [22], where a set of 137 microservices are used to revel MSAs architecture patterns. Therefore, it is safe to say that there is no software corpus of microservice applications available for researchers that is scalable, easily accessible, or dynamically updated based on a set of metrics, which is the ultimate goal of this research.

2.2.2 RQ2: *What aspects of microservices applications can be used for their classification as microservices applications?*

Considering the successful validation of **RQ1**, the next most important task is to design a solution to properly identify microservices so that *mining* them becomes possible. This microservice identification process was divided into two parts, the first one was to inspect the selected literature to find MSAs' most common characteristics. The second part was to manually analyze a set of MSAs source code to determine what types of metrics could be possible to extract to be used to search for other MSAs, perhaps and ideally in an automated fashion. This process will be detailed in this section.

Microservices Characteristics

While there are several definitions of microservices, many authors agree that they should be small in size [22, 9, 16, 20] [27], that is, implement limited loosely coupled and highly cohesive functionality. Also, they should use light-weight protocol for extra-service communication [28] [22] [14] [12], such as RESTFul and gRPC, and message brokers for interservice communication [28], such as Kafka and RabbitMQ[1]. Additionally, they should be independently deployable [22] [2] [6] [20], and for this, containers technology is used 95% of the time for new MSA implementations [10]. On top of that, some authors argue that each microservice should have its own database [7] [30] [26], others go further and state that database sharing is in fact considered architectural smells [28]. The list below summarizes MSAs characteristics found in this literature view:

- Small in size (found in 31 out of 44 papers)
- Highly cohesive (found in 29 out of 44 papers)
- Light-weight protocol (found in 24 out of 44 papers)
- Independently deployable (found in 37 out of 44 papers)
- Database Ownership (found in 11 out of 44 papers)
- Message broker (found in 12 out of 44 papers)

The details of which characteristics were found in which paper can be found at `https://anonymous.4open.science/r/icsme23-A500/data/sample/articles.xlsx`, in the tab "included-characteristics". This list of characteristics is the starting point for a heuristic process to form a set of (dynamic) metrics to mine MSAs, described in the next section.

From Microservices Characteristics to Search Metrics

Once the most common set of quality attributes of MSAs was known, it was necessary to verify which ones could be found in the source code so that these heuristics could become metrics. Therefore, a small set of MSAs code repositories were manually analyzed as a proof of concept (POC). The goal of this manual analysis was to verify that some characteristics can be detected in code while others cannot. This small set of MSAs' repositories was extracted from the bigger set of 137 microservices in the work of Podolskiy et al. [22], and the extracted data is presented in Table 2.3. The repositories for this sample can be found in the following URLs (numbers correspond to the numbers in column "Repo"):

1. `https://github.com/zalando/zally`
2. `https://github.com/icecrime/vossibility-collector`
3. `https://github.com/sczyh30/vertx-blueprint-microservice`
4. `https://github.com/uc-cdis/cdis-data-client`

[1] `http://kth.diva-portal.org/smash/get/diva2:813137/FULLTEXT01.pdf`

5. https://github.com/uber/cadence

6. https://github.com/pingcap/tidb

7. https://github.com/influxdata/chronograf

As a result, it is possible to observe that the characteristics (in the columns) can be quantifiable, and so this value may be used as a threshold to search MSAs at scale.

- **Small in size**: can be measured by the contents of the columns:
  1. Size: which represents the size of the repositories in kilobytes.
  2. Files: the number of files in the "Main Language" of the repository.
  3. Contents: the number of files summed with the number of folders
- **Highly cohesive**: this characteristic can be measured from the source code. For instance, Jin et al. [18] has proposed a way to do it. However, to calculate it, it is necessary to analyze the source code, which implies having implementations for each language an MSA may be written on. This is unfeasible and would limit our approach. Thus, we do not consider this metric.
- **Light-weight protocol**: can be measured by the column Rest, which represents the presence RESTFul related technology, such as HTTP external call in the code.
- **Independently deployable**: the presence of a "Dockerfile" can suggest that the application is standalone.
- **Database ownership**: column "DB" indicates that a database connection configuration was found, so the application most likely owns a database.
- **Message broker**: column "Msg" indicates that a message broker configuration was found or code that shows the existence of a "Producer" or "Consumer" process, which suggests usage of messaging technology.

In addition, based on our own experience and on the analysis of MSA repositories, we also consider the following metrics:

- ***Documentation***: column "Ms" represents the presence of the words "microservices" in the READ.me file or anywhere in the code.
- ***Heavy-weight protocol***: column "Soap" represents the use of a Simple Object Access Protocol (SOAP) in the repository, which is an anti-pattern for MSAs.
- ***Logging***: represents the presence of technologies related to logging services, which is a good practice in MSAs.

Table 2.3: Manually Collected MSAs Metrics

Repo	Size	Files	Contents	Ms	DB	Dockerfile	Rest	Msg	Soap	Logging
1	12236	224	628	No	Yes	Yes	Yes	No	No	No
2	12761	381	601	No	Yes	Yes	Yes	Yes	No	Yes
3	1877	6	474	Yes	Yes	Yes	Yes	No	No	Yes
4	7818	34	54	No	Yes	Yes	Yes	No	No	No
5	58871	381	1916	Yes	Yes	Yes	Yes	Yes	No	No
6	398633	2292	4621	No	Yes	Yes	Yes	Yes	No	No
7	98156	356	1926	No	Yes	Yes	Yes	Yes	No	No

To sum up, the set of MSA's characteristics extracted from the selected articles, which were yielded in the search string of this light-weight systematic review, formed a strong pavement to build a method for microservice identification from source code. This experiment obtained enough evidence that this manually executed process can certainly be implemented in a more elegant and automated solution to broaden the universe of examples it may rely upon to produce yet a more vast, dynamic, and accurate set of metrics for microservice identification.

2.2.3 RQ3: *Can we define an algorithm to recognize MSAs' code repositories based on their characteristics?*

To address **RQ3**, we need to understand two aspects of software repositories. The first one is what are the existing types of software repositories, which are shown in the listing below by [15], and the second is comprehending the MSR process.

- **Source control repositories:** "They track all the changes to the source code along with meta-data about each change, e.g., the name of the developer who performed the change, the time the change was performed and a short message describing the change."
- **Bug repositories:** "These repositories track the resolution history of bug reports or feature requests that are reported by users and developers of large software projects."
- **Archived communications:** "These repositories track discussions about the various aspects of a software project throughout its lifetime."
- **Deployment logs:** "These repositories record information about the execution of a single deployment of a software application or different deployments of the sample application."
- **Code repositories:** "These repositories archive the source code for a large number of projects."

MSR process

One of the biggest contributions of *the MSR Cookbook* [17], was to summarize all recommendations of MSR research and define a well-structured four steps process organized in four different themes to define an agreeable standard in this research field. This process and themes are shown in Fig. 2.1, and their description highlights the most relevant parts of this topic:

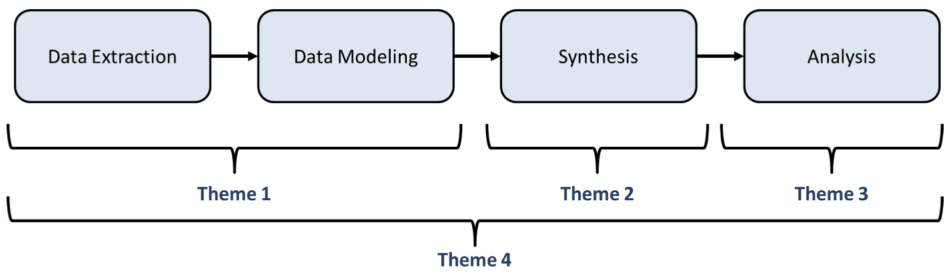


Figure 2.1: Steps of a typical MSR process [15]

- **Theme 1: Data Acquisition and Preparation** is the starting point of the process. It concentrates on gathering raw data from repositories and then pre-processing it. In this case, most of the data is the source code itself, the most important of all software development artifacts. In addition, there are plain text data where the software documentation is present. An important point to mention is the dependency on the Source Control Management (SCM) system, which in this case is limited to *git*.
- **Theme 2: Synthesis** entails the utilization of machine learning algorithms such as clustering, classification, or prediction of the data. Often, during this phase, straightforward analysis presents better results than complex ones.
- **Theme 3: Analysis** - cares about the results, and more importantly, their proper interpretation. This is the most important phase as it dictates the conclusions drawn at the end of the studies. For this step, manually verifying the generated results is crucial.
- **Theme 4: Sharing and replication** are about making the study reproducible. Sharing the process, data, and tools used helps external validation and perhaps the evolution of the study. This is significant because most studies of MSR apply certain techniques to certain data sources, and this alone is not enough to provide proof that the results can be replicated in other data sources.

To summarize, despite the existence of many types of software repositories, this research focuses only on *code repositories*. Additionally, the MSR process describes the phases for mining all types of software repositories, which addresses **RQ3**. However, the results of the search strings used in this systematic review did not find any work where such methods are explicitly applied for *mining* MSAs. Consequently, this leaves room for further research and experimentation to find out the results of the utilization of existing MSR processes and associated methods in the unexplored field which is this current research.

Chapter 3

Metrics

The definition and usage of suitable metrics are at the very core of this research work. This is the reason why, when defining them, we considered both the literature and observations made from code repositories sample projects in an iterative heuristic process. In this chapter/section, we will dive deeper into the details of the data collected from each of one of those 10 metrics mentioned in Section 5.1. We extracted instances of each metric in an automated fashion to form an expanded dataset as the one presented at Table 2.3. In order to do that in a systematic manner, we used the MSR process described at Section 2.2.3 as a reference to manipulate the dataset with the goal of having data ready to be input into our classification algorithm.

Bearing in mind that the ultimate goal of this research is to correctly identify microservices, or better yet, distinguish microservices from monolith systems, we decided that at least two datasets would be required. The first one should be a curated list of microservices. This was identified in the related work of Podolski et al. [22], which contains 137 samples. For the second one, we needed a list of monolith systems, which we found in the work of Brito et al. [4]. Next, we pre-processed the list of code repositories by removing any invalid (e.g. not pointing to a GitHub repository) or duplicated URLs. As a result, we gathered two equally sized lists of code repositories: Appendix B with 101 microservices, and Appendix C with the same count of monoliths.

Afterward, we used GitHub Rest API[1] to search inside each code repository from our dataset to verify if they met the criteria used to identify the metrics which we were trying to find. In that event, 190 repositories were successfully searched from the list of 202, 96 of which were monoliths and 94 microservices. The remaining 12 were excluded due to a lack of data in response to the API calls. The summary of this search is presented next with plots generated in RapidMiner[2].

3.1 Binary Metrics

We considered 7 out of 10 metrics to be binary because a code repository either has a certain metric present or not. So, for this type of metric, we searched the code repositories looking for

[1]`https://docs.github.com/en/rest`
[2]`https://rapidminer.com/`

any mention of certain keywords, which were different for each metric. To illustrate the results of these searches in the code, we will present the keywords used and a bar chart for each metric, where MSAs are represented on the left and monoliths on the right.

Documentation Metric: For this metric, we searched the code repositories looking for any mention of the keywords “microservice” or “micro-service”. As we can see in Fig. 3.1, for MSAs these keywords were found in 31% of the repositories (for 29 out of 94) while for monoliths it was less than 1% (3 out of 96). Therefore, the MSA Mention metric is more likely to be found in MSAs than it is in monoliths.

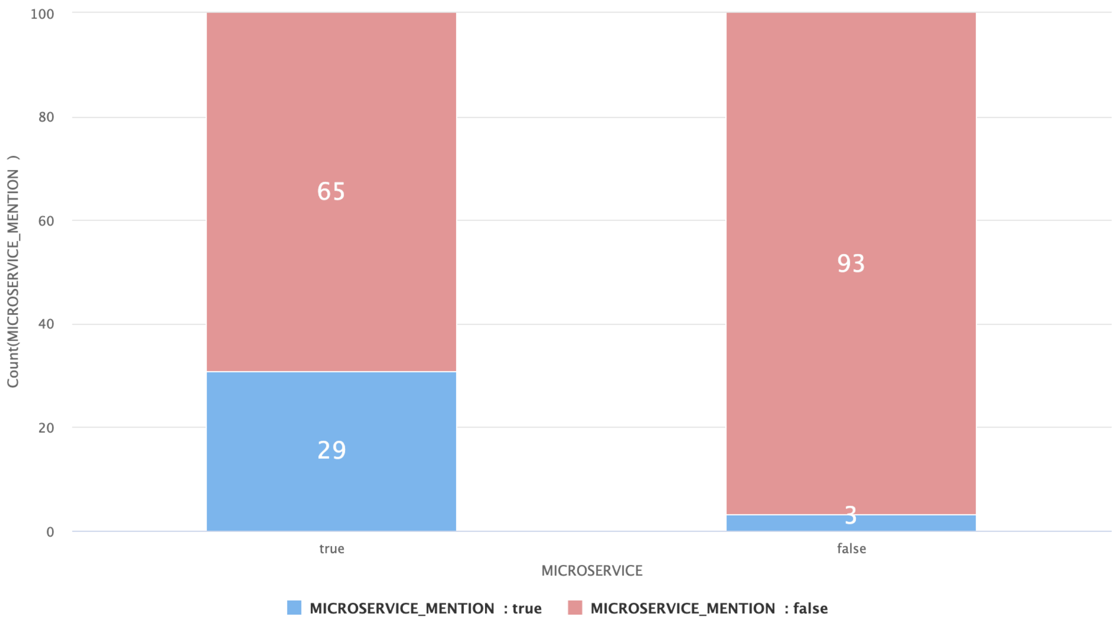


Figure 3.1: Documentation Metric

Database Ownership Metric: For this metric, we searched the code repositories looking for any mention of the keywords “database”, “oracle”, “MySQL”, “SQL Server”, “SQLite”, “Postgres”, “Cassandra” or “MongoDB” in addition to files having the “.sql” or “.bd” extensions. As we can see in Fig. 3.2, for MSAs these keywords were found in 97% of the repositories (for 91 out of 94) and for monoliths, it was 98% (94 out of 96). Therefore, the Database Ownership metric was highly present in both types of repositories and it is equally likely to be found in both.

Independently Deployable Metric: For this metric, we searched the code repositories looking for any mention of the keywords “docker” or “docker-compose” as well as files written in the languages Dockerfile or Makefile. As we can see in Fig. 3.3, for MSAs these keywords were found in 99% of the repositories (for 93 out of 94) while for monoliths it was only in 29% (28 out

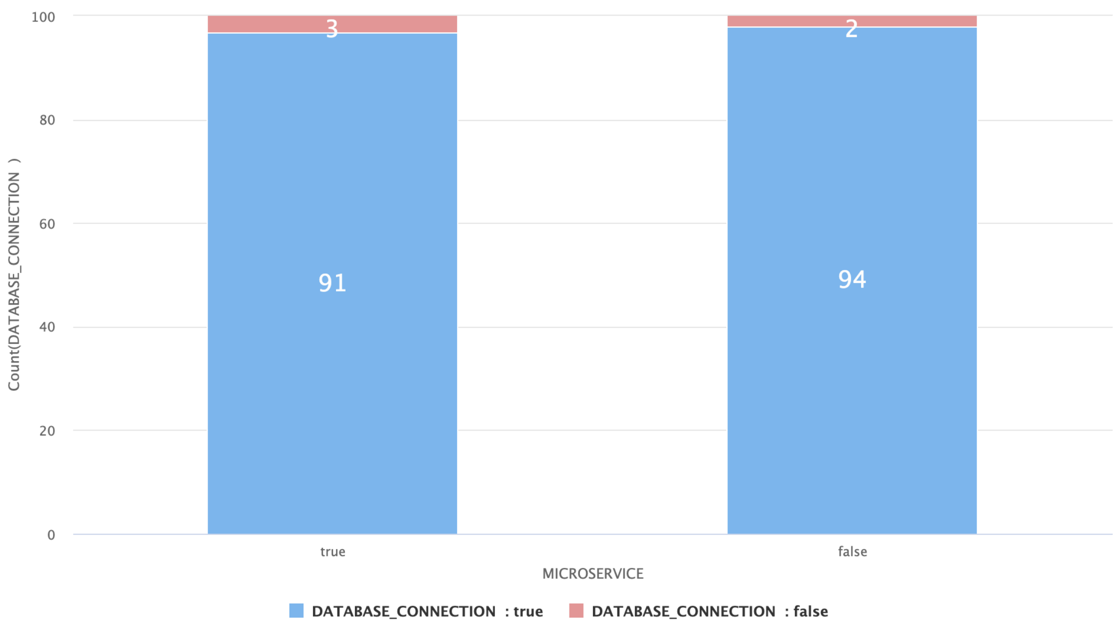


Figure 3.2: Database Ownership Metric

of 96). Therefore, the Independently Deployable metric is much more likely to be found in MSAs than it is in monoliths.

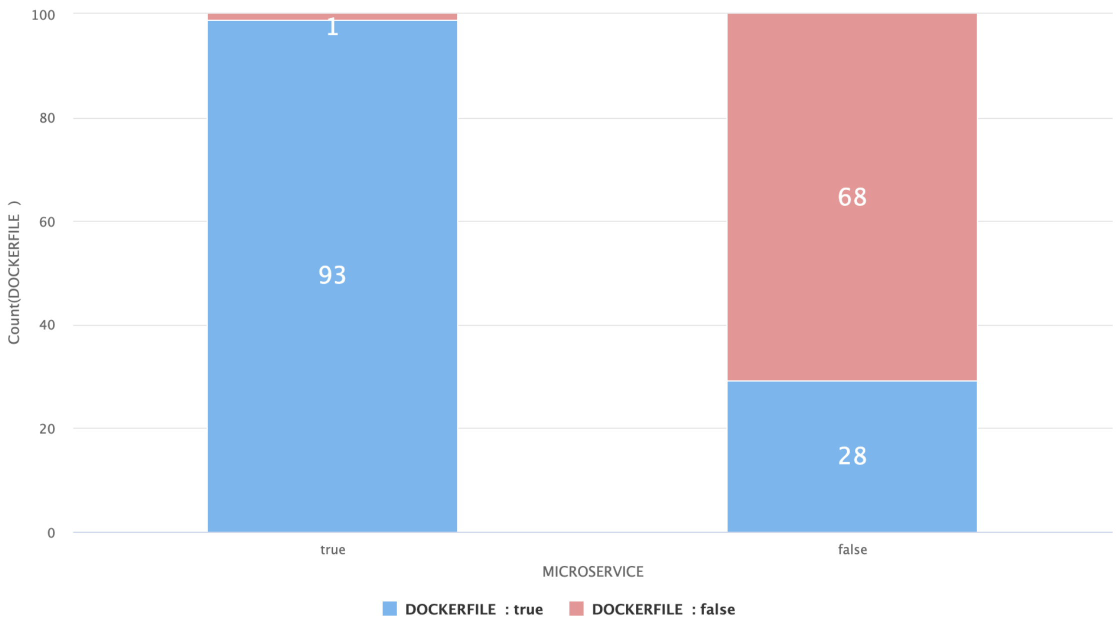


Figure 3.3: Independently Deployable Metric

Light-weight Protocol: For this metric, we searched the code repositories looking for any mention of the keywords “http” or “https”. As we can see in Fig. 3.4, for MSAs these keywords were found in 99% of the repositories (for 93 out of 94) while for monoliths it was found in 100% of the cases(96 out of 96). Therefore, the Light-weight protocol Metric was highly present in both types of repositories, and it is equally likely to be found in both.

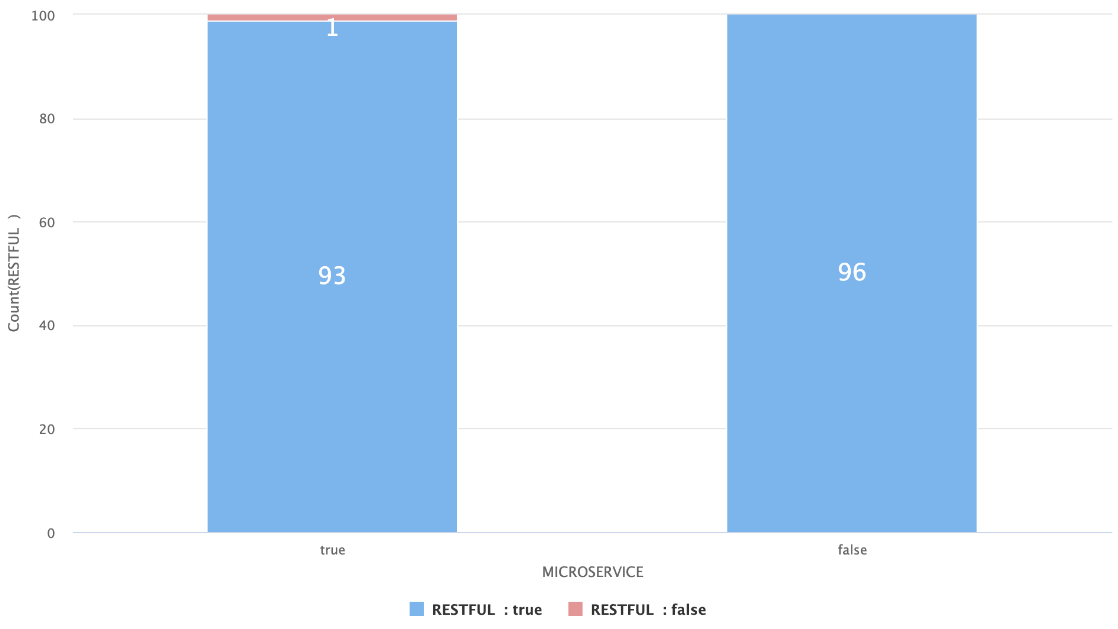


Figure 3.4: Light-weight Protocol Metric

Message Broker: For this metric, we searched the code repositories looking for any mention of the keywords “Kafka”, “RabbitMQ”, “producer”, “consumer”, or “amqp”. As we can see in Fig. 3.5, for MSAs these keywords were found in 71% of the repositories (for 67 out of 94) while for monoliths it was found in 33% of the cases(32 out of 96). Therefore, the message broker metric was much more likely to be found in MSAs than it is in monoliths.

Heavyweight Protocol Metric: For this metric, we searched the code repositories looking for any mention of the keyword “soap”. Especially for this metric, we were more interested in the absence of the SOAP protocol since it is considered a heavy-weight protocol. As we can see in Fig. 3.6, for MSAs these keywords were not found in 94% of the repositories (for 88 out of 94) while for monoliths it was not found in 84% of the cases(81 out of 96). Therefore, the Heavyweight protocol metric indicates that the SOAP protocol is being less and less used.

Logging Metric: For this metric, we searched the code repositories looking for any mention of the keywords “logstash”, “Datadog”, “Syslog-ng”, “Rsyslog”, “rsyslog”, “Logagent”, “Graylog”, or “Fluentd”. As we can see in Fig. 3.7, for MSAs these keywords were found in 38% of the

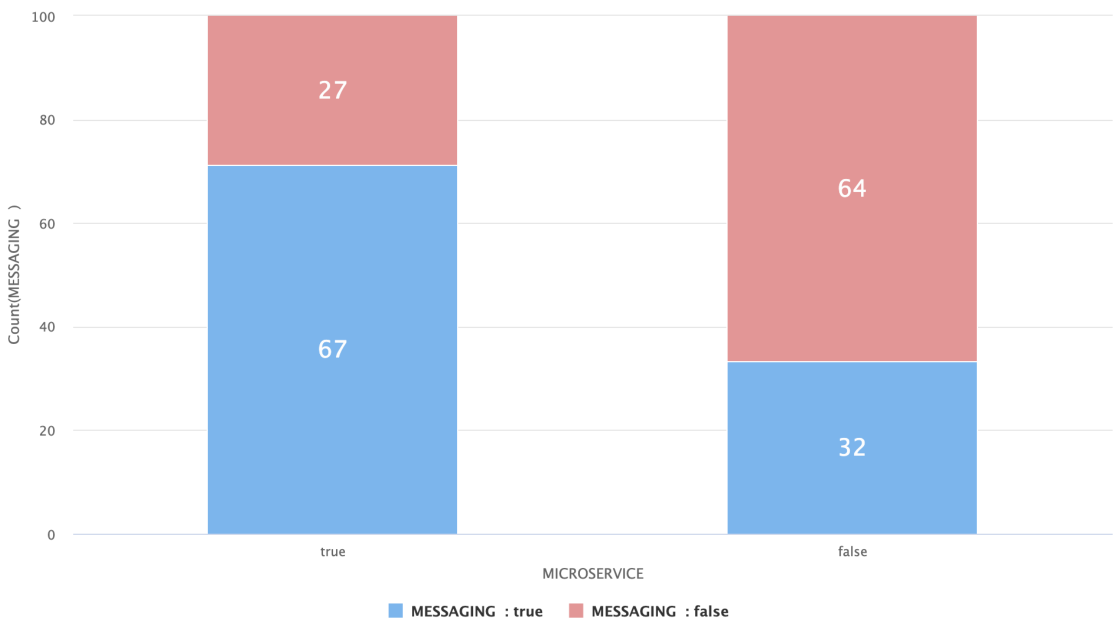


Figure 3.5: Message Broker Metric

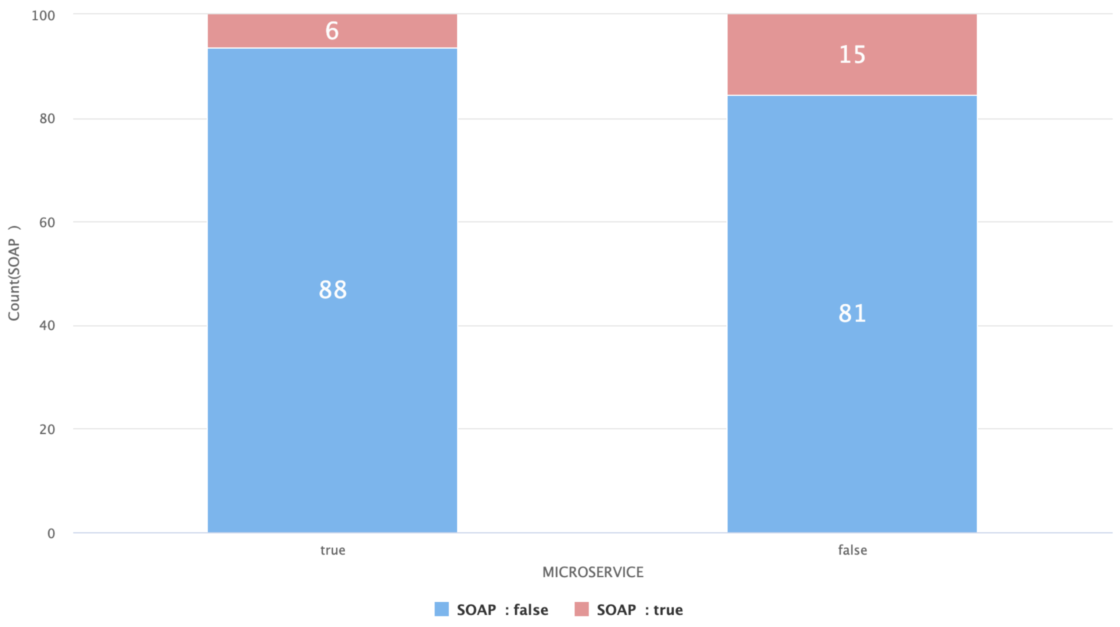


Figure 3.6: Heavyweight Protocol Metric

repositories (for 36 out of 94) while for monoliths it was found in 5% of the cases (5 out of 96). Therefore, the Logging Metric is more likely to be found in MSAs than it is in monoliths.

In summary, we observed that four out of seven metrics had greater chances of being found in

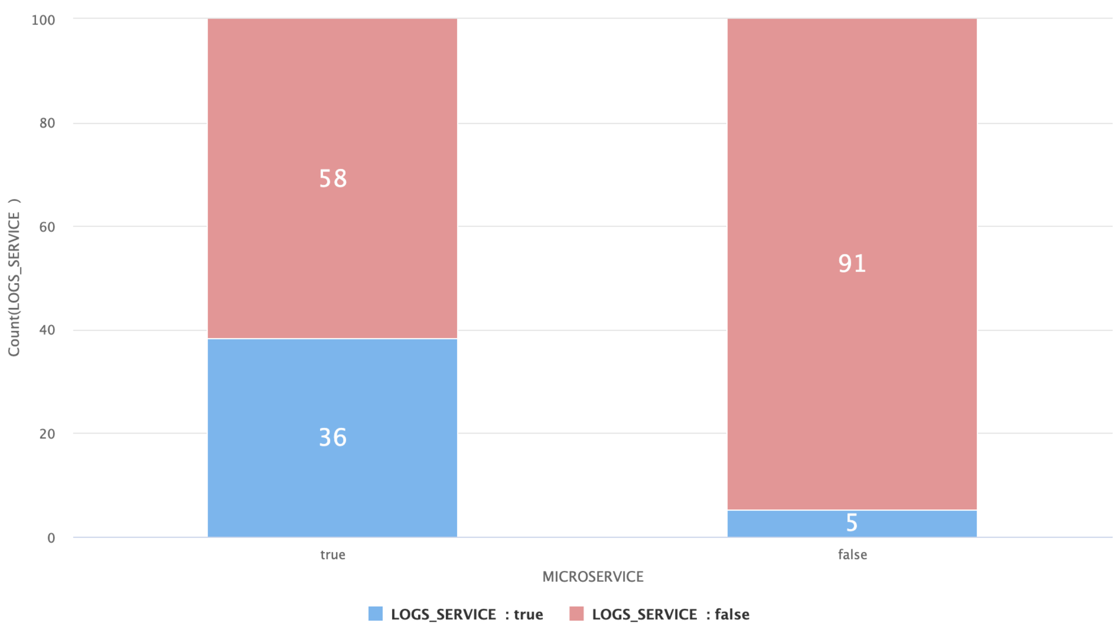


Figure 3.7: Logging Metric

MSAs, which supports the study started at Section 2.2.2. Additionally, the three remaining metrics had similar chances of being found in both types of repositories.

3.2 Continuous Metrics

Unlike the binary metrics, when looking at the data pulled from GitHub for the continuous metrics, namely, Size(kb), Files, Files+Folders (see Section 2.2.2), we could not see a clear pattern that we could use to distinguish MSAs from monoliths in response the question of how small they were. Therefore, we adapted the method introduced by [1], which presents the following process to derive thresholds from benchmark data:

1. metrics extraction: metrics are extracted from a benchmark of software systems.
2. weight ratio calculation: for each entity, compute the weight percentage within its system, i.e., we divide the entity weight by the sum of all weights of the same system. For each system, the sum of all entities WeightRatio must be 100%.
3. entity aggregation: aggregate the weights of all entities per metric value, which is equivalent to computing a weighted histogram (the sum of all bins must be 100%).
4. system aggregation: we normalize the weights for the the number of systems and then aggregate the weight for all systems.
5. weight ratio aggregation: order the metric values in an ascending way and take the maximal metric value that represents 1%, 2%, ..., 100% of the weight.

6. thresholds derivation: thresholds are derived by choosing the percentage of the overall code we want to represent. Considering four categories:
    - low risk (between 0 and 70%)
    - moderate risk (70 and 80%)
    - high risk (80 and 90%)
    - very-high risk (> 90%)

Since each value of the continuous metric was already the representation of each system (e.i. if a system had a size of 450kb, then this value was representative of the whole system globally), the steps for weight ratio calculation and entity aggregation become obsolete. The remaining measures were implemented and applied to each of the three metrics considering a low-risk threshold for the threshold derivation step.

Additionally, we noted that some repositories of microservices had only one microservice while others had a set of them. Consequently, we needed to differentiate between the two types of MSA repositories to derive a more helpful threshold for each variety. Thus, we distinguish applications that contain a single microservice, applications composed of several microservices (which we refer to as MSAs sets), and applications that follow other architectures. This led us to consider the databases metric from a different perspective and also to consider the number of programming languages used:

- ***Databases***: from the literature, it is clear that each MSA should have its database [9, 8, 30]. This means that if a repository has more than one database, it may contain an MSA set. Thus, we count the number of databases configured as services in each repository by reading their docker-compose file[3]. We gathered every database keyword present in our benchmark data to use as a curated list of classifiable data storage services: sqlserver, kafka, datahub, hana, MySQL, postgres, Cassandra, spark, mongo, db, neo4j, clickhouse, rabbitmq, redis, dynomite, bd, subscriber, redpanda, SQL, database, bigtable, datanode, vertical, namenode, spark, hadoop, pingcap, grafana.

- ***Programming languages***: if the number of programming languages is high, it may mean that the repository contains a set of MSAs. Thus, we use the following curated list of programming languages gathered from our benchmark data to count the number of programming languages present in each repository: JavaScript, Java, Go, PHP, Python, TypeScript, Groovy, Ruby, R, Lua, Scala, COBOL, Erlang, Kotlin, Swift, C, C#, C++, Perl, ASP.NET, Objective-C, Visual Basic .NET, Assembly, Blade, CoffeeScript, Cython, Rust, Dart, MATLAB, Awk, Emacs Lisp, Objective-C++, Vala, D, XSLT, q, ActionScript, Elixir, Euphoria, Visual Basic and ASP.

[3] `https://docs.docker.com/compose/features-uses/`

Finally, we checked if repositories had their count of database services configured and programming languages greater than one to classify them as a set of MSA and then generated a threshold for each one of the three types of repositories.

Next, we will show line charts for the metrics size, files, and files+ folder renamed "All Contents." The X-axis shows the aggregated weight of the metric with a vertical line marking the low-risk threshold derivation of 70%, while the Y-axis shows the metrics' values.

Size Fig. 3.8, Fig. 3.9, and Fig. 3.10 show the distribution of sizes accumulated from 0 up to 100% to MSAs, MSAs sets and monoliths, respectively. Accordingly, the values selected as thresholds were 442014, 492578, and 379774 since they are the max values before the aggregated weight overcame 70% of the total distribution. So they help classify each respective type of system with a low risk of incorrect type.

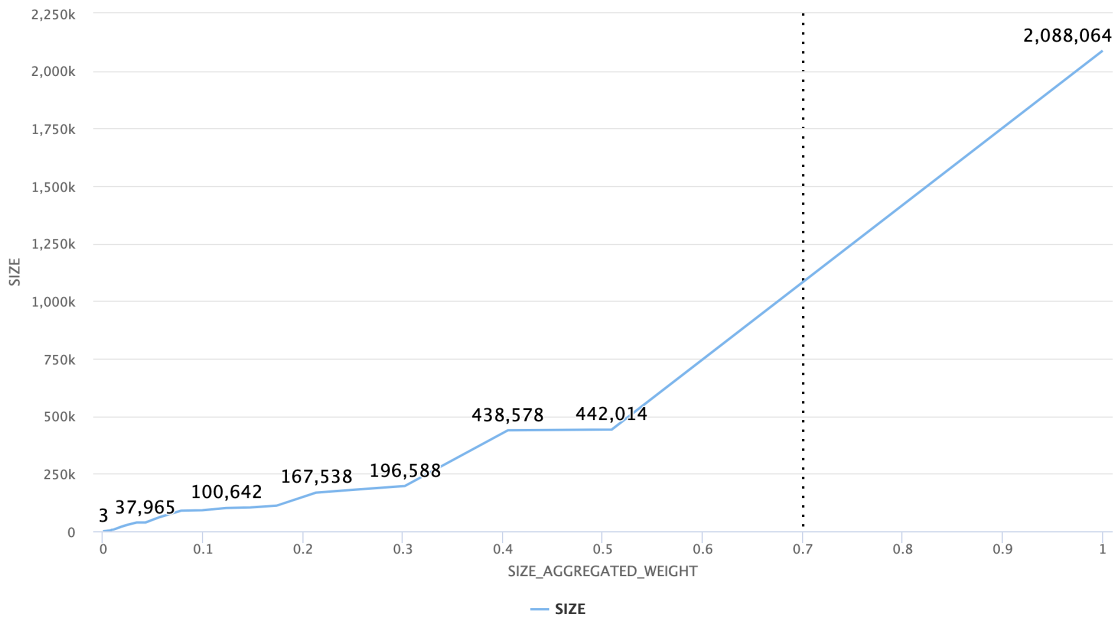


Figure 3.8: MSA Size Metric

Files Fig. 3.11, Fig. 3.12, and Fig. 3.13 show the distribution of file count for each repository accumulated from 0 up to 100% to MSAs, MSAs sets, and monoliths, respectively. Accordingly, the values selected as thresholds were 4925, 8790, and 3156 since they are the max values before the aggregated weight overcame 70% of the total distribution. So they help classify each respective type of system with a low risk of incorrect variety.

All Contents Fig. 3.14, Fig. 3.15, and Fig. 3.16 show the distribution of allContents(files+folders) accumulated from 0 up to 100% to MSAs, MSAs sets, and monoliths respectively. Accordingly, the values selected as thresholds were 7159, 13661, and 3491 since they are the max values before

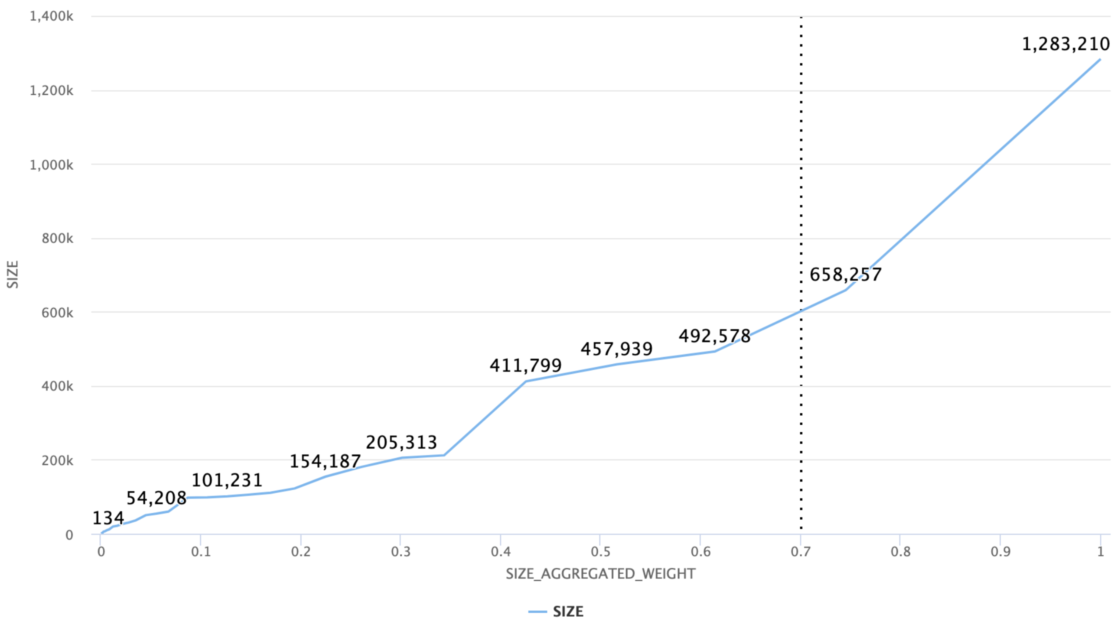


Figure 3.9: MSA set Size Metric

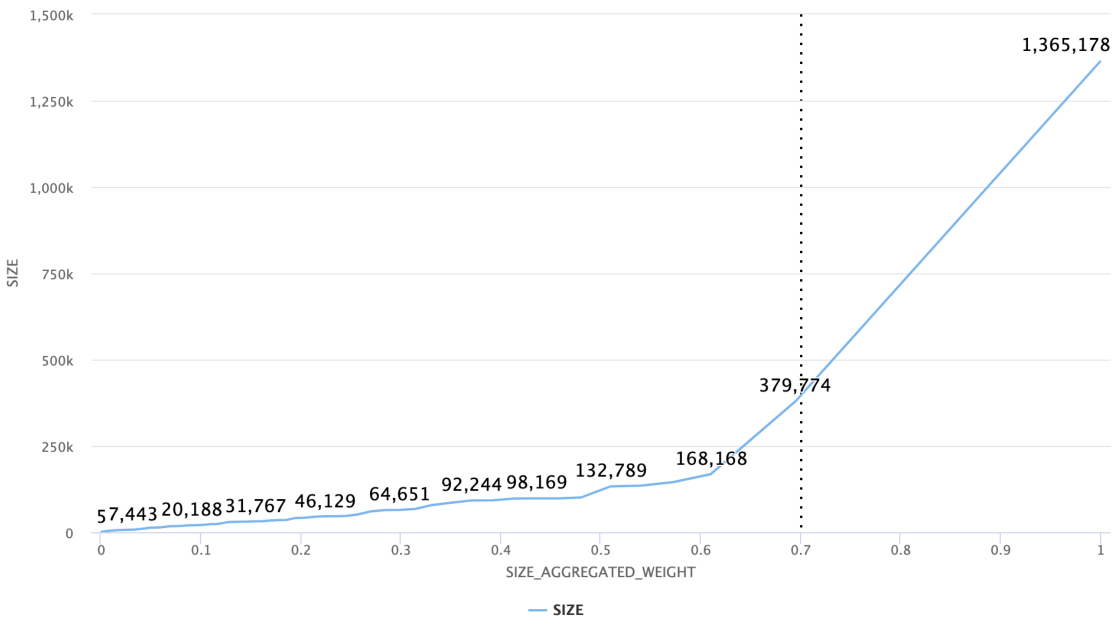


Figure 3.10: Monoliths Size Metric

the aggregated weight overcame 70% of the total distribution. So they help classify each respective type of system with a low risk of incorrect variety.

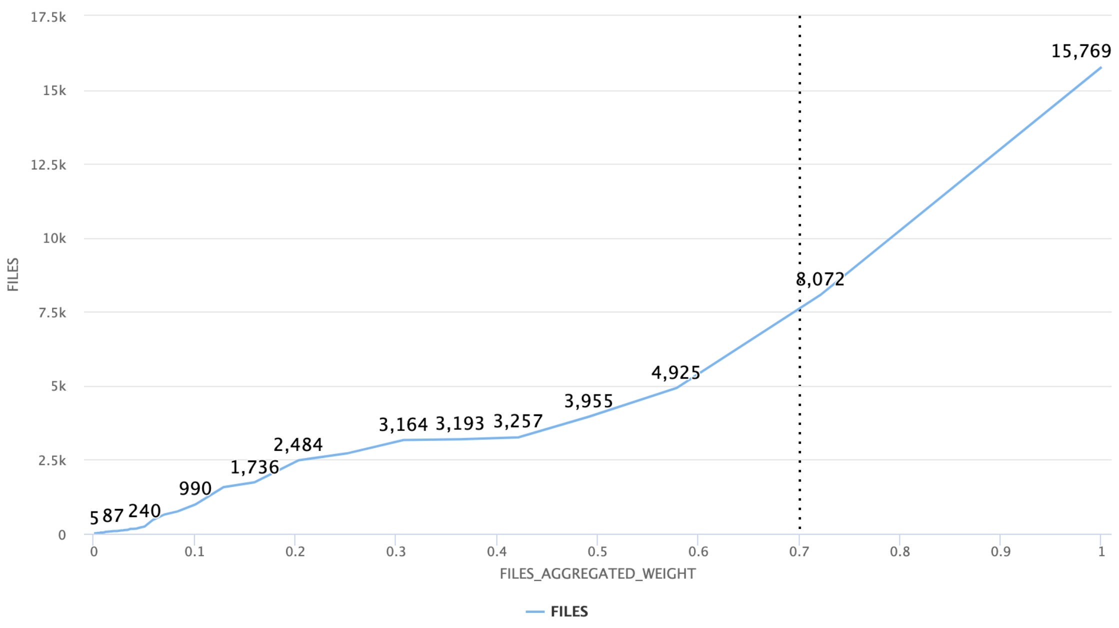


Figure 3.11: MSA Files Metric

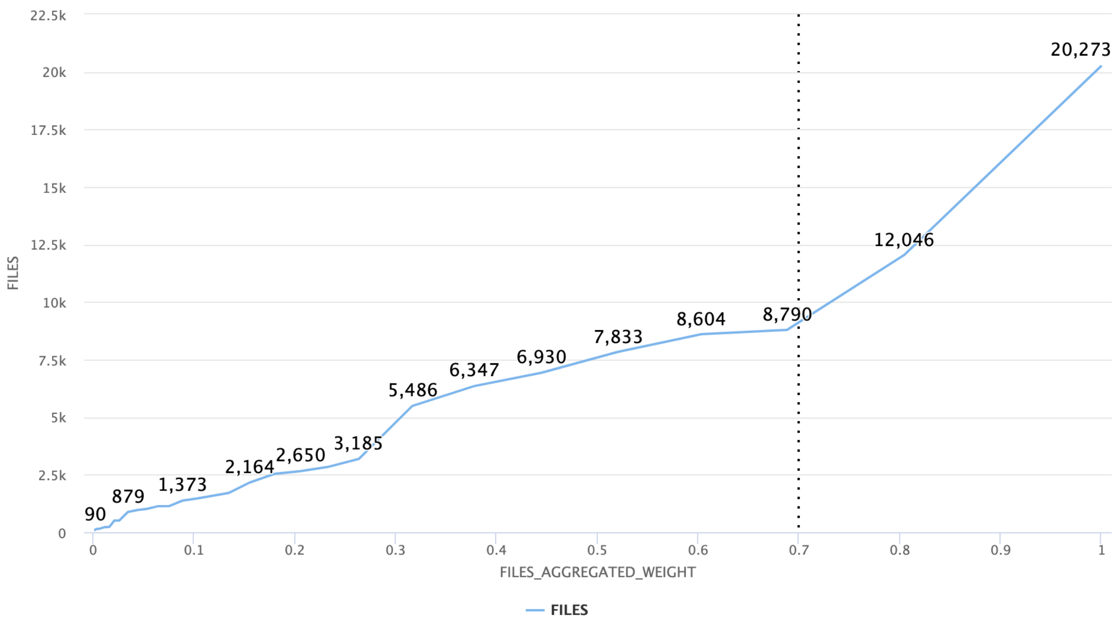


Figure 3.12: MSA set Files Metric

3.3 Conclusion

In summary, we collected data from seven binary and three continuous metrics. Searching the code repositories to detect its presence for the binary type metric was sufficient. In contrast, for the continuous metric, a method for threshold derivation was necessary to determine the upper

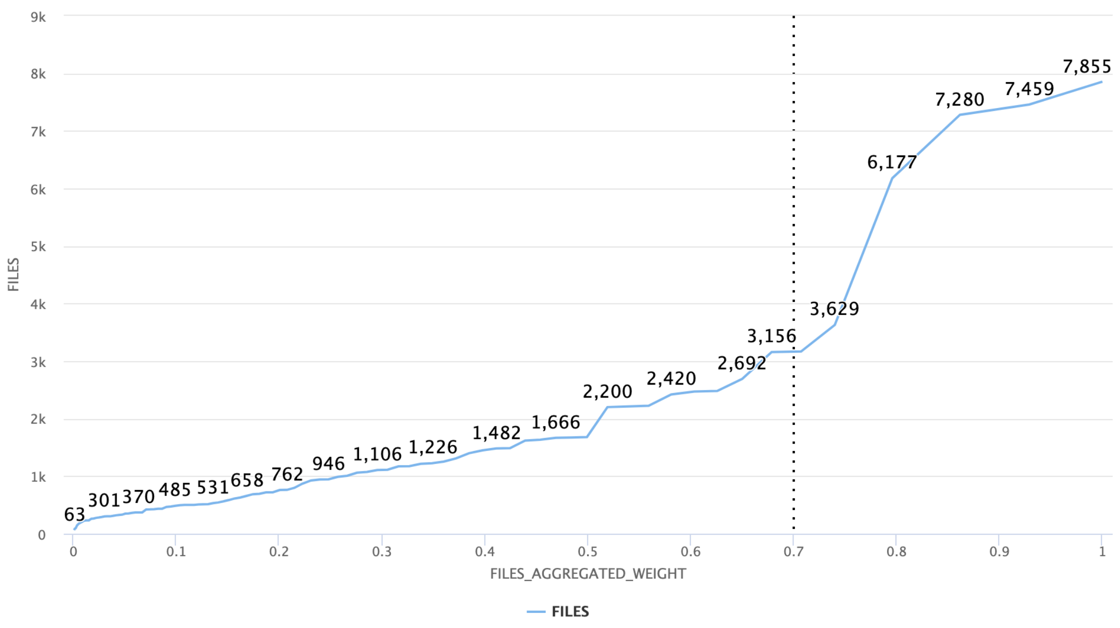


Figure 3.13: Monoliths Files Metric

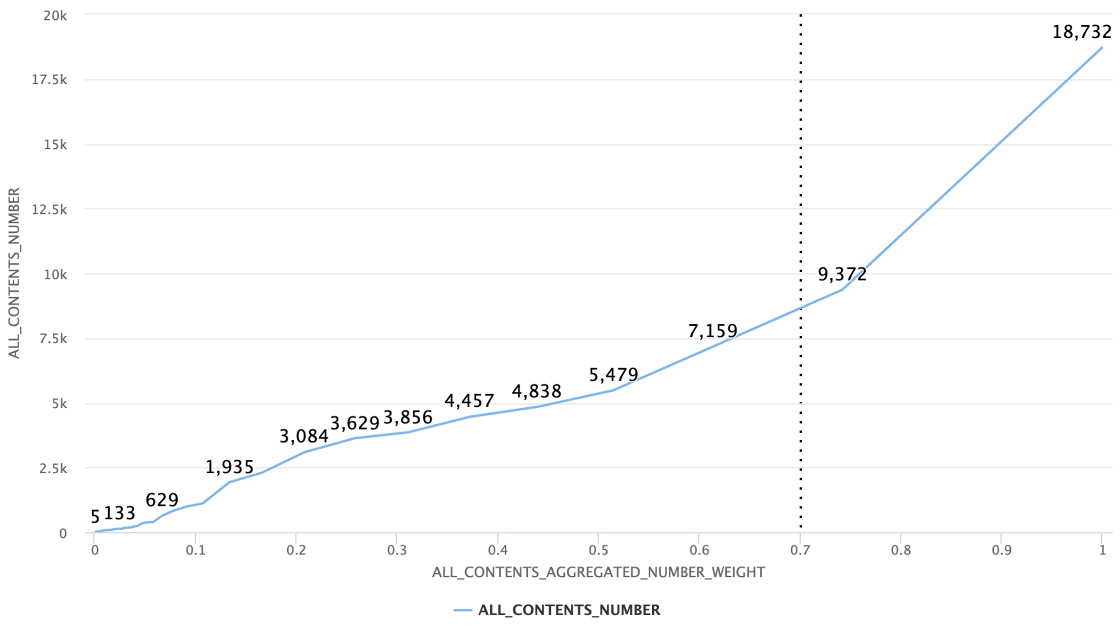


Figure 3.14: MSA AllContents Metric

range limit for each metric. Hence, at this stage, we had enough data about the metrics to use as input to a classification algorithm to identify MSAs.

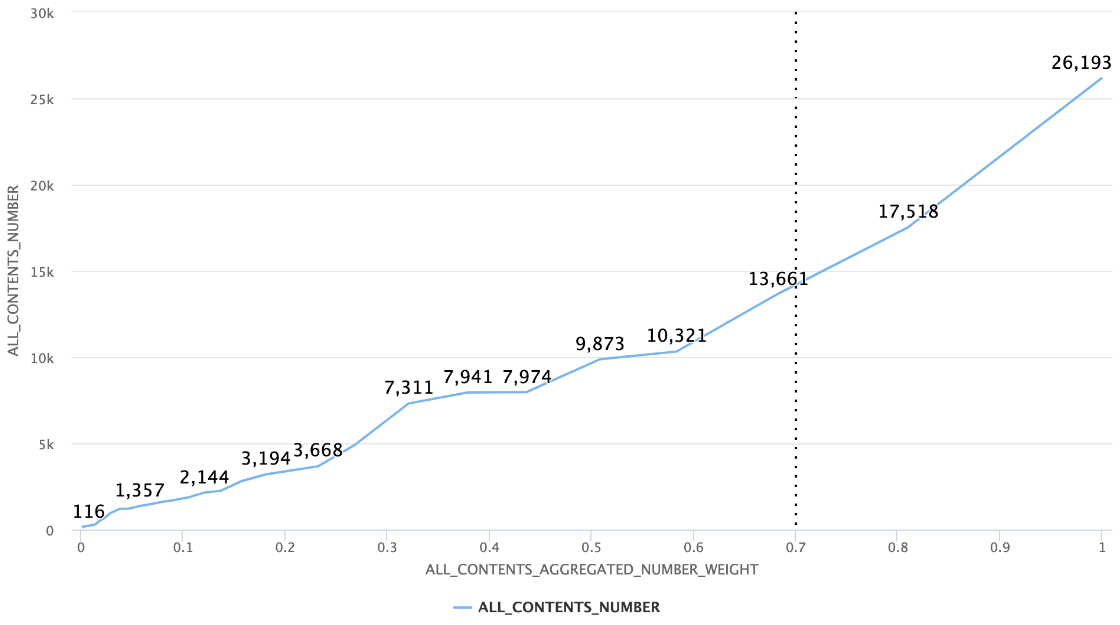


Figure 3.15: MSA set AllContents Metric

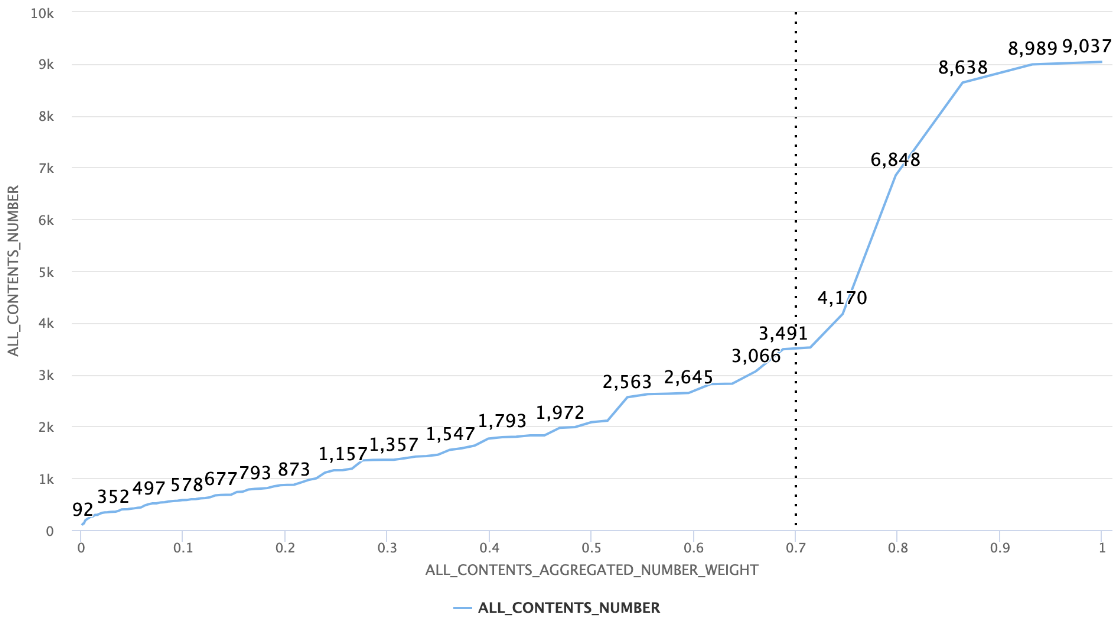


Figure 3.16: Monoliths AllContents Metric

Chapter 4

Classification Algorithm

Once all metrics for the list of 190 code repositories were successfully processed, we ought to decide on a method in which we could leverage the collected metrics for the identification of MSAs as well as MSAs sets. There were two possible paths we could follow: we could create a model with a machine learning algorithm, or we could design our algorithm. Considering that the process should be as transparent as possible so that users could know exactly the reasons for repositories classification, we opted to design our own algorithm. At a broad level, the solution involves analyzing a code repository by examining 10 metrics, computing a score that reflects the repository's performance in those metrics, and employing this score to determine its classification. Essentially, the solution entails a comprehensive evaluation process that involves measuring the repository's implementation of some MSAs' characteristics and then using this information to assign it to an appropriate category. Next, we present the pseudo-code of the developed solution broken-down to the function level. In addition, at the end of this chapter, it is possible to find a sequence diagram of the same solution providing an illustrative view of the design.

4.1 Calculate Scores and Set Classification

Algorithm 1: This is the starting point of the whole process for the classification. In Line 2, the list of repositories is retrieved from the database, and an iteration on this list is started in Line 3. Hence, for each repository, we want to recover the metrics fetched and stored from GitHub, calculate a score for it, and based on this score and two indicators (*hasMsSetInd* in Line 7 and *hasMonoInd*[1] in Line 10), define a classification for it. While the MSA set indicator (*hasMsSetInd*) had been idealized during the analysis of the continuous metrics in Chapter 3, the other indicator (*hasMonoInd*) was created during the development, and it intended to flag the system having front-end related languages, typical HTML and CSS, and one or two programming languages since, from our empirical analysis, other systems usually have up to two programming languages. The score calculation is executed in the method call to Algorithm 2 in Line 11. Next, a call to the method Algorithm 5

[1] In the code we use *MONOLITH* to refer to other systems not following a microservice architecture.

to get the classification for the repository is made in Line 12. Finally, the result is set in Line 13 and Line 14 and saved in Line 15.

Algorithm 1 Calculate Scores and Set Classification

```
1: function CALCULATESCORESANDSETCLASSIFICATION
2:     $codeRepoMetricsList \leftarrow repository.findAll()$
3:     for each $metrics$ in $codeRepoMetricsList$ do
4:         $codeRepo \leftarrow metrics.getCodeRepo()$
5:         $pgls \leftarrow metrics.getProgrammingLanguages()$
6:         $dbs \leftarrow metrics.getDatabaseServices()$
7:         $hasMsSetInd \leftarrow pgls > 1$ and $dbs > 1$
8:         $langs \leftarrow metrics.getLanguages()$
9:         $ftLang \leftarrow$ HASFRONTENDLANGUAGES$(langs)$
10:        $hasMonoInd \leftarrow dbs \leq 2$ and $ftLang$
11:        $score \leftarrow$ CALCULATESCORE$(metrics, hasMsSetInd)$
12:        $class \leftarrow$ GETREPOCLASS$(score, hasMsSetInd,$
               $hasMonoInd)$
13:        $codeRepo.setClassification(classification)$
14:        $codeRepo.setScore(score)$
15:        $codeRepoService.save(codeRepo)$
16:    end for
17: end function
```

4.1.1 Calculate Score

Algorithm 2: Even though some metrics were found in both MSAs and monoliths, such as Lightweight protocol and Database ownership as presented in Chapter 3, we considered all metrics in the calculation of the score because we noticed that the higher the number of metrics found, the more likely it would mean that it was an MSA, and removing those metrics found in both types of repositories would not change that, but it would reduce the clarity in the reasoning for the classification repositories as MSAs. Therefore, the score calculation was based on one principle: each metric may score 0 or 1 point. The score started at 0 in Line 2. It accumulates the points from Line 6 to Line 8 by calling the method Algorithm 3 for continuous metrics, and Algorithm 4 for binary metrics from Line 9 to Line 15. The total value is returned in Line 16.

Get Numerical Score (Algorithm 3) For continuous metrics, if the value for the metric was smaller than the threshold value defined for each metric (size, files, or all contents), it scores 1 (Line 3), otherwise 0 (Line 5).

Get Boolean Score (Algorithm 4) For binary metrics, if the value for the metric was true (the metric was detected in the code repository during the search with GitHub API), it would score 1 (Line 3), otherwise 0 (Line 5).

Algorithm 2 Calculate Score

```
 1: function CALCULATESCORE(m, hasMsSetInd)
 2:     $score \leftarrow 0$
 3:     $sT \leftarrow threshold.getByMetric(SIZE, hasMsSetInd)$
 4:     $fT \leftarrow threshold.getByMetric(FILES, hasMsSetInd)$
 5:     $cT \leftarrow threshold.getByMetric(ALL\_CONTENTS,$
          $hasMsSetInd)$
 6:     $score \leftarrow score + \text{GETNSCORE}(m.getSize(), sT)$
 7:     $score \leftarrow score + \text{GETNSCORE}(m.getFiles(), fT)$
 8:     $score \leftarrow score + \text{GETNSCORE}(m.getAllContents(), cT)$
 9:     $score \leftarrow score +$
          $\text{GETBSCORE}(m.isDockerfile(), DOCKERFILE))$
10:     $score \leftarrow score +$
          $\text{GETBSCORE}(m.isLogsService(), LOG\_SERVICE))$
11:     $score \leftarrow score +$
          $\text{GETBSCORE}(m.isDbConnection(), DATABASE))$
12:     $score \leftarrow score +$
          $\text{GETBSCORE}(m.isMessaging(), MESSAGING))$
13:     $score \leftarrow score +$
          $\text{GETBSCORE}(m.isRestful(), REST))$
14:     $score \leftarrow score +$
          $\text{GETBSCORE}(m.isMsMention(), MS\_MENTION))$
15:     $score \leftarrow score + \text{GETBSCORE}(!m.isSoap(), SOAP))$
16:     return $score$
17: end function
```

Algorithm 3 Get Numerical Score

```
1: function GETNSCORE($metricValue, thresholdValue, metric$)
2:     if $metricValue \leq thresholdValue$ then
3:         return 1
4:     end if
5:     return 0
6: end function
```

Algorithm 4 Get Boolean Score

```
1: function GETBSCORE(metricValue, metric):
2:     if metricValue then:
3:         return 1
4:     end if
5:     return 0
6: end function
```

4.1.2 Get Code Repository Classification

Algorithm 5: Here is where the classification is set not only based on the score but also considering the indicators defined in Algorithm 1. In Line 2, the target minimum classification score for MSA is set to 7. This value was chosen after an iterative empirical process classifying MSAs while

keeping as few false positives (others classified as MSA) as possible. In Line 3, the first evaluation of the score is made, and if its value is not greater than 7, the project is immediately considered not to be an MSA (Line 18). Otherwise, if its indicator to be a microservice set is true – Line 4 and Line 5 – it is classified as an MSA set. Otherwise, on the one hand, if it has not an indicator that it is another type of system (Line 7), it is classified as an MSA (Line 15); on the other hand, if it has an indicator that it is another type of system (Line 7), it loses 1 point (Line 8), but if the score is still greater than 7 (Line 9 and Line 10), then it is classified as an MSA set, if the score is lower, it is classified as not being an MSA (Line 12).

Algorithm 5 Get Code Repository Classification

```
 1: function GETREPOCLASS($score, hasMsSetInd, hasMonoInd$)
 2:     $MS\_CLASSIFICATION\_SCORE \leftarrow 7.0$
 3:     if $score > MS\_CLASSIFICATION\_SCORE$ then
 4:         if $hasMsSetInd$ then
 5:             return MICROSERVICE_SET
 6:         end if
 7:         if $hasMonoInd$ then
 8:             $score \leftarrow score - 1$
 9:             if $score > MS\_CLASSIFICATION\_SCORE$ then
10:                 return MICROSERVICE_SET
11:             else
12:                 return MONOLITH
13:             end if
14:         else
15:             return MICROSERVICE
16:         end if
17:     else
18:         return MONOLITH
19:     end if
20: end function
```

Fig. 4.1 shows the sequence diagram for the classification algorithm highlighting the call to Algorithm 5 and with some additional details omitted on the pseudo-code for simplicity.

In conclusion, the designed algorithm can utilize the 10 different metrics in its scoring system to accumulate up to 10 points. Code repositories with scores over 7 are classified as MSAs or MSAs set. Otherwise, they are generally considered other architectures. While the 7 binary metrics are static since they rely upon the premise that those metrics are either found (in code repositories) or not, the 3 continuous metrics are dynamic and so they can change over time whenever the input dataset from which the thresholds are derived changes. Thus, on the one hand, a porting of the algorithm (70%) will keep it stabilized and prevent it from changing its classification drastically. On the other hand, the other portion of the algorithm (30%) will learn over time, and its classification will be influenced by its input data.

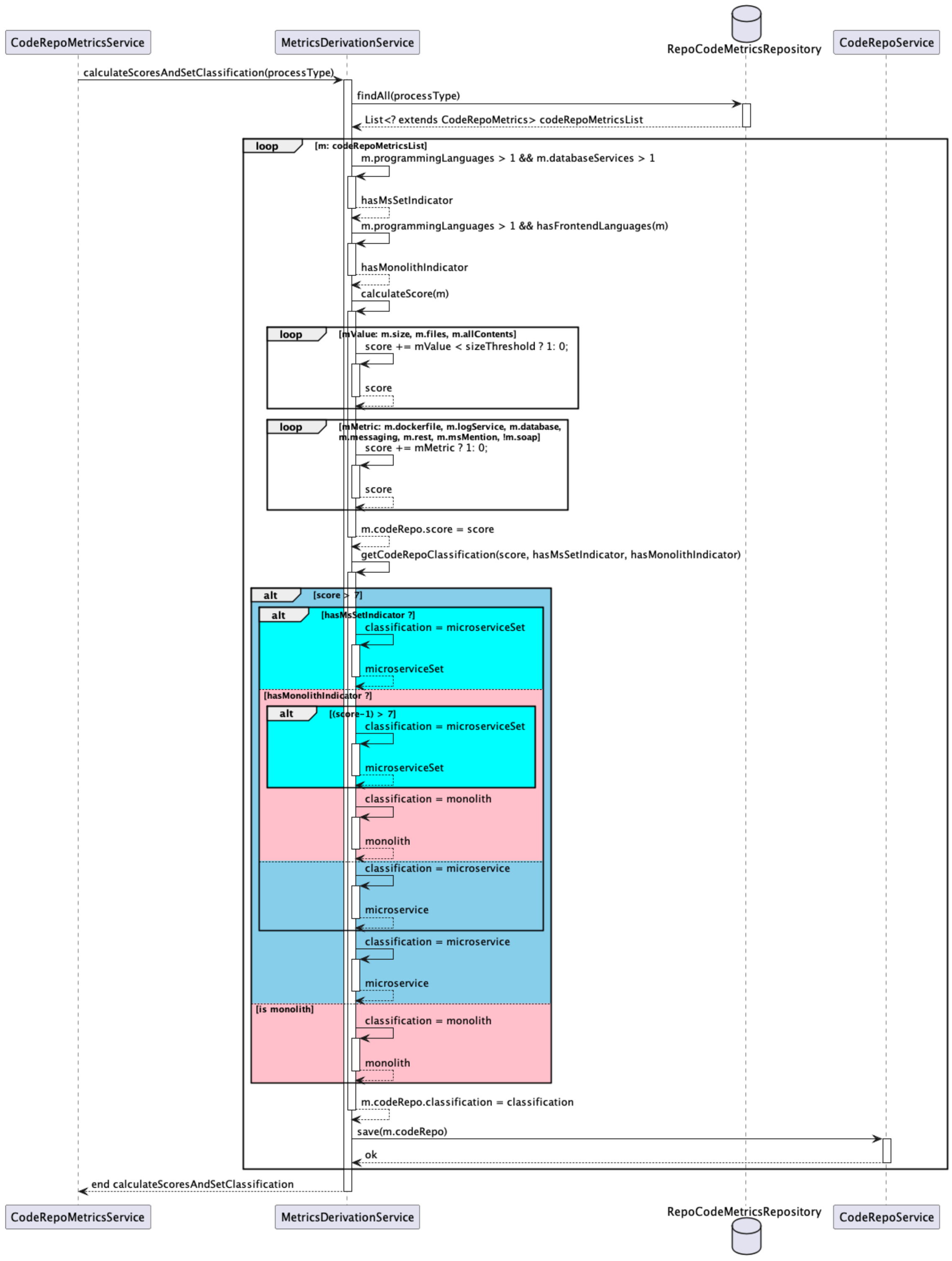

CodeRepoMetricsService
MetricsDerivationService
RepoCodeMetricsRepository
CodeRepoService
calculateScoresAndSetClassification(processType)
findAll(processType)
List<? extends CodeRepoMetrics> codeRepoMetricsList
loop [m: codeRepoMetricsList]
m.programmingLanguages > 1 && m.databaseServices > 1
hasMsSetIndicator
m.programmingLanguages > 1 && hasFrontendLanguages(m)
hasMonolithIndicator
calculateScore(m)
loop [mValue: m.size, m.files, m.allContents]
score += mValue < sizeThreshold ? 1: 0;
score
loop [mMetric: m.dockerfile, m.logService, m.database, m.messaging, m.rest, m.msMention, !m.soap]
score += mMetric ? 1: 0;
score
m.codeRepo.score = score
getCodeRepoClassification(score, hasMsSetIndicator, hasMonolithIndicator)
alt [score > 7]
alt [hasMsSetIndicator ?]
classification = microserviceSet
microserviceSet
[hasMonolithIndicator ?]
alt [(score-1) > 7]
classification = microserviceSet
microserviceSet
classification = monolith
monolith
classification = microservice
microservice
classification = microservice
microservice
[is monolith]
classification = monolith
monolith
m.codeRepo.classification = classification
save(m.codeRepo)
ok
end calculateScoresAndSetClassification

Figure 4.1: Sequence Diagram for the Classification Algorithm

Chapter 5

GitHub Microservice Mining Tool

In this chapter, we will introduce the solution overview of the proposed tool, as well as the implementation details with a set of Unified Modeling Language (UML) diagrams powered by PantUML[1]. This also includes the process of metric collection and threshold derivation shown in Chapter 3 in addition to the classification algorithm presented in Chapter 4.

5.1 Solution Design

The design of this tool addresses the goal of creating an MSA corpus and making it available to practitioners. Therefore, this tool's functionalities start from maintaining repositories, and their collected metrics, and then using the metrics in the algorithm designed for MSA identification. Finally, the identified MSAs are presented to practitioners with filtering options. Before that, we describe the role of GitHub in the implementation of this tool.

5.1.1 GitHub

There are many uses for GitHub as a collaborative tool for software development. Among other functionalities, GitHub is an internet-based code repository for projects developed using git as Source Control Management (SCM). According to Vidoni [29], GitHub is the most commonly used source for MSR. Its robust API which provides access to publicly available repositories is a reason for that. So there is a huge amount of data available from successful projects, which helps GitHub continue to be the most significant development platform to date [13]. For this reason, we chose GitHub to be the code repository platform for the implementation of this tool.

5.1.2 Solution Overview

The GitHub Microservice Mining Tool follows the monolith first design principle supported by Martin Fowler [11] and many other authors. This is due to the fact that this system is designed to

[1]https://plantuml.com/

be small and simple, and these characteristics favor a monolith rather than the case of more complex and bigger systems which would be a better fit with a microservice architecture. As shown in Fig. 5.1, the solution is composed of six internal components and one external, as described below:

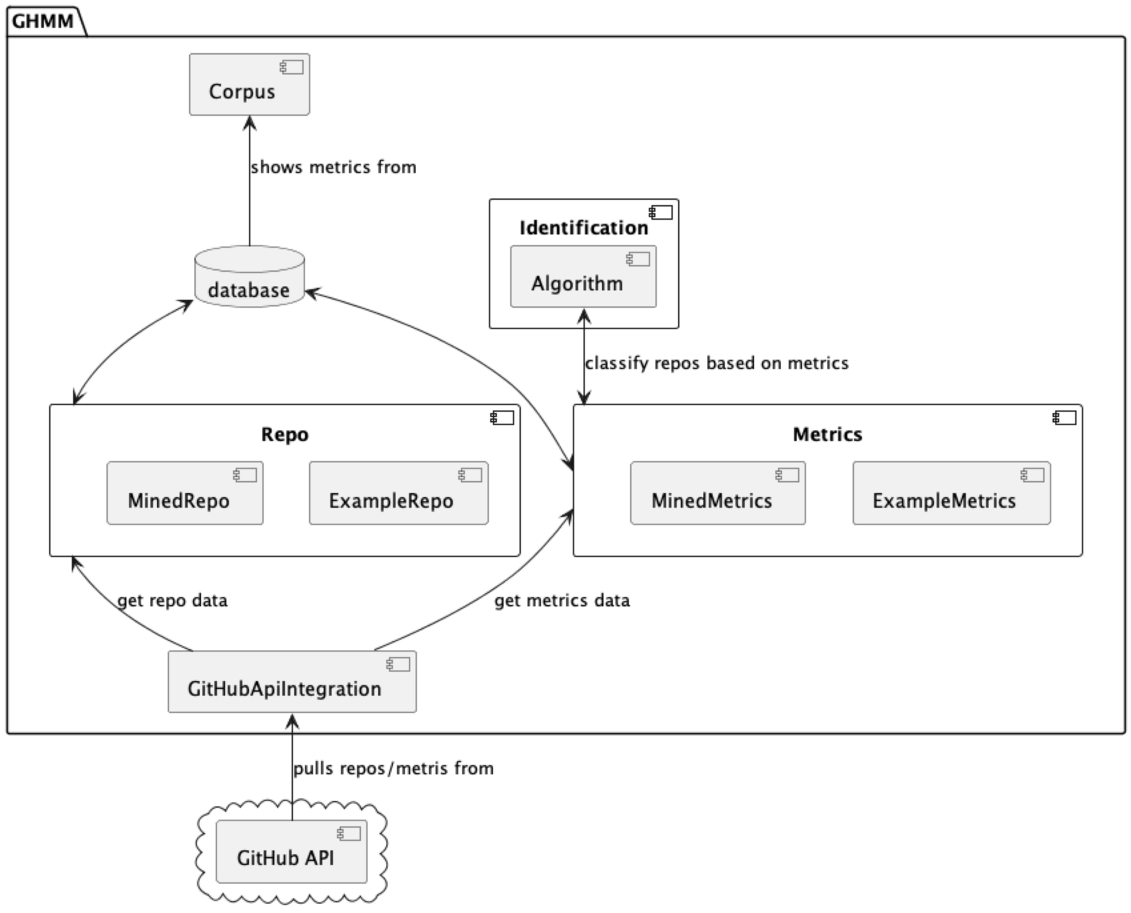


Figure 5.1: Component Diagram for GitHub Microservice Mining Tool

- **Metrics Component**: Generates metrics to help search for MSAs by:
  1. Pulling/reading metrics data from GitHub by using *GitHubApiIntegration* component.
  2. Maintaining metrics data about *ExampleMetrics* and *MinedMetrics* subcomponents in the *database* component.
  3. Sends the metrics data to be processed into the *Algorithm* subcomponent of the *Identification* component.
  4. Pushing/writing the results to the *database* component.
- **GitHubApiIntegration**: Implements the integration between the external *GitHub API* component and the system. This component can be replaced with other code repositories APIs.

- **Repo**: Maintains data for its subcomponents *ExampleRepo* and *MinedRepo* in the *database* component.
- **Identification Component**: Identify MSAs in GitHub throughout its search mechanism by:

    4. Pulling/reading MSAs metrics from the *Metrics* component.
    5. Processing those metrics to classified code repositories.
    6. Returning classification results to the *Metrics* component.

- **Corpus Component**: Shows the MSA corpus to the user and allows them to apply filters.

    7. Pulling/reading MSAs repositories URLs from the *database* component.

5.2 Architecture & Technologies

The tool was built on a monolith architecture using a Model-View-Controller design pattern for simplicity. The programming language of choice was Java 17[2] and the framework used was Spring Boot[3] for the Models and Controllers, whereas Thymeleaf[4] was the template engine for the views. Fig. 5.2 shows the system architecture with its main nodes and components:

- ClientMachine: This node is the user's machine which must have a web browser component for accessing the application.
- ServerMachine: This node is where the system is run and so as a classic monolith system, it has the frontend and backend components in it.
- GitHubServer: This is the representation of GitHub where the API for querying code repositories is.
- Database: This node represents an ideal database configuration in a node separated from the system. However, for this implementation, we used H2 Database Engine[5] embedded in the ServerMachine node for a simplified deployment on the Heroku Cloud Platform[6]. Still, it can be changed easily by modifying a configuration file.

5.3 Model

Fig. 5.3 shows the data model for the system which can be split into two major concepts: the first being the representation of code repositories with the class CodeRepo, and the second the representation of metrics collected for those repositories with the class CodeRepoMetrics. Each

[2]https://www.jcp.org/en/jsr/detail?id=392
[3]https://start.spring.io/
[4]https://www.thymeleaf.org/
[5]https://www.h2database.com/html/main.html
[6]https://www.heroku.com/

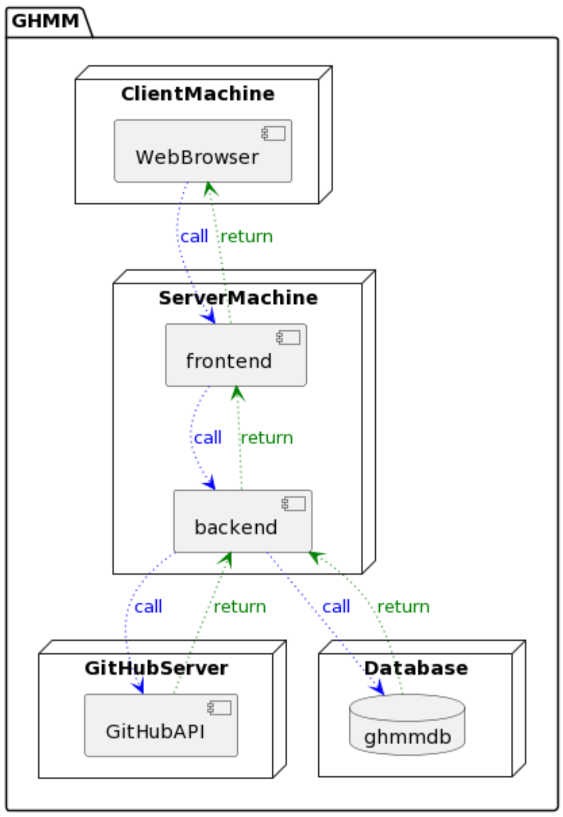


Figure 5.2: Deployment Diagram for GitHub Microservice Mining Tool

class has two specializations to separate examples of code repositories (those that we know their types and are using as a starting point: RepoExample) from mined code repositories (those that we what to search in GitHub and then classify them: RepoMined). The classes Language and Service support the collection of CodeRepoMetrics. The Threshold class stores the result of the process presented in Section 3.2 and the ProcessExecution class records each execution of metrics generation triggered by the user.

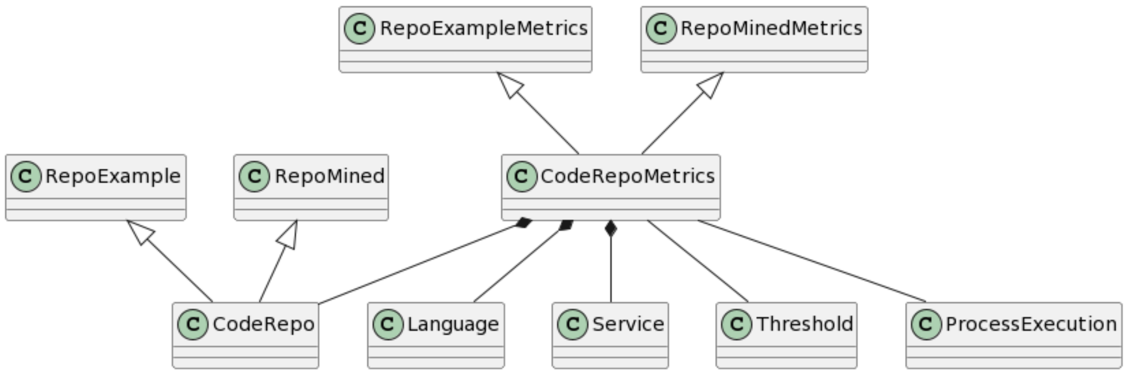


Figure 5.3: Class Diagram for GitHub Microservice Mining Tool

5.4 Use Cases

Fig. 5.4 shows the use case diagram for the solution. The only actor is the Software Engineer who represents any practitioner interested in an MSA corpus. Next, we present a brief description of each use case scenario in the format of user stories:

UC1 Maintain RepoExample: As Software Engineer, I want to add, list, or remove RepoExamples, so that I can process their metrics.

UC2 Process RepoExample Metrics: As Software Engineer, I want to process RepoExample Metrics, so that I can use the generated thresholds from continuous metrics to help classify search/uploaded repositories.

UC3 List/Review RepoExample Metrics: As Software Engineer, I want to list the generated RepoExample metrics, so that I can review and check their correctness.

UC4 View RepoExample Metrics Statistics: As Software Engineer, I want to view statistical data about the generated metrics, so that I can make inferences about my dataset.

UC5 Search/Upload MinedRepo: As Software Engineer, I want to upload a list of MinedRepos (unclassified repositories) or search them through GitHub API, so that I can process their metrics.

UC6 Process MinedRepo Metrics: As Software Engineer, I want to process MinedRepo Metrics, so that I can see their classification which is based on Algorithm 1.

UC7 List/Review MinedRepo Metrics: As Software Engineer, I want to list the MinedRepo metrics, so that I can better understand their classification.

UC8 List/Filter Classified MinedRepos: As Software Engineer, I want to list and/or filter classified MinedRepos, so that I can use them in my research work or any other I may find suitable.

Additionally, Appendix D shows the basic scenario for each use case, their pre, and post-conditions, and respective views, such as the example below in the system as well as a sequence diagram for the most important ones. Also, the code for the solution is available at GitHub Microservice Mining Tool.

UC1 Maintain RepoExample: As Software Engineer, I want to add, list, or remove RepoExamples.
Scenario: Add one repo example - the view is shown on Fig. 5.5

1 - On the main menu, click on Repo Examples > Add/Upload.

2 - Type the owner of the code repository in the field Owner.

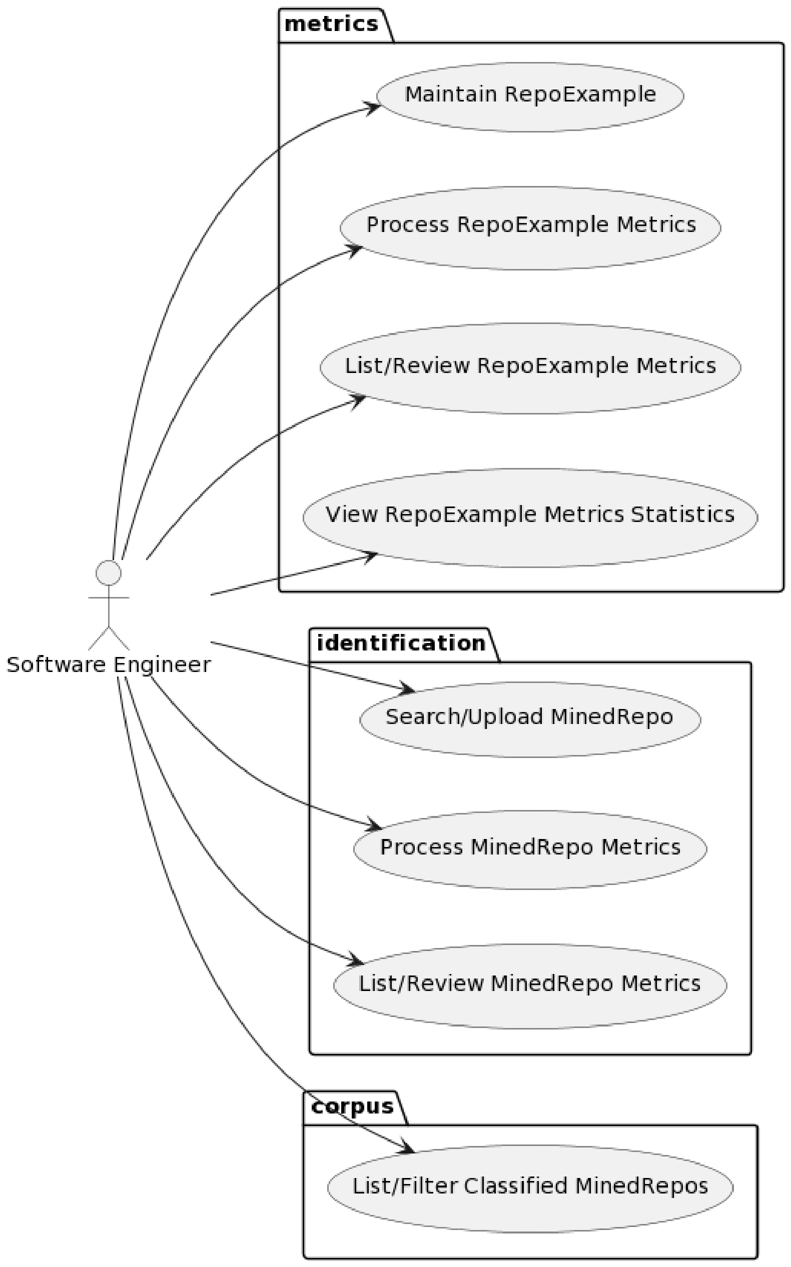

metrics
Maintain RepoExample
Process RepoExample Metrics
List/Review RepoExample Metrics
View RepoExample Metrics Statistics
Software Engineer
identification
Search/Upload MinedRepo
Process MinedRepo Metrics
List/Review MinedRepo Metrics
corpus
List/Filter Classified MinedRepos

Figure 5.4: Use Case Diagram for GitHub Microservice Mining Tool

3 - Type the name of the code repository in the field Name.

4 - Type the Url of the code repository in the field Url.

5 - Click on the checkbox Microservice if the entered repository is an MSA.

6 - Click on the button Add.

Precondition: N/A.

Post-condition: The repository example is saved in the database.

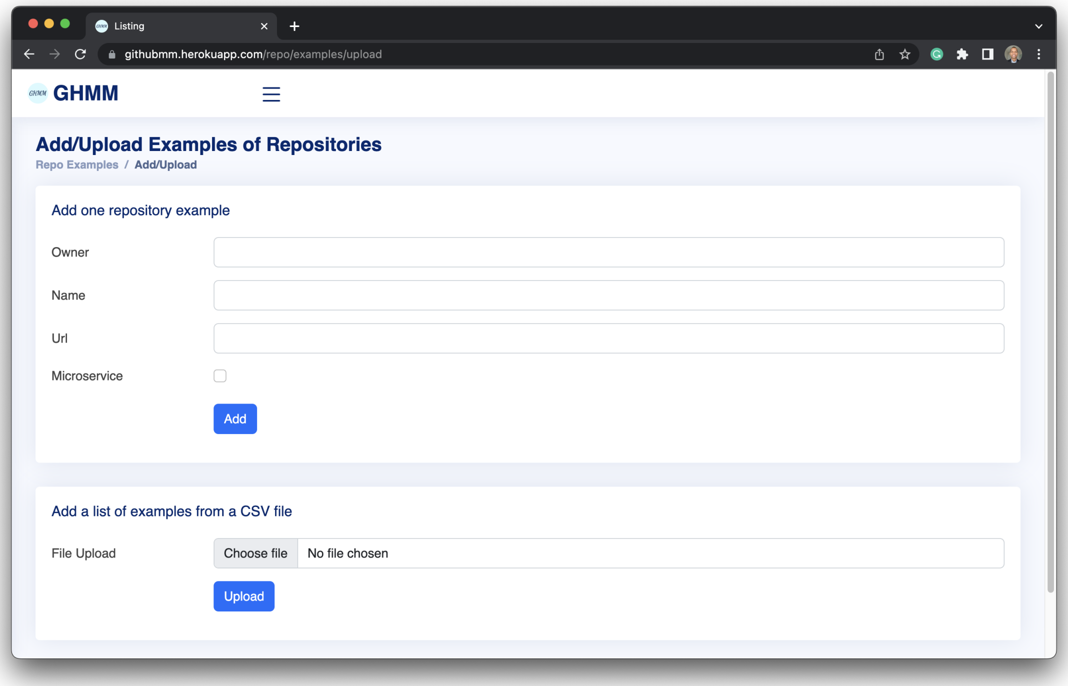
Scenario: List repo examples - the view is shown on Fig. 5.6

1 - On the main menu, click on Repo Examples > List Examples.

2 - Click on Page Size to choose how many repositories you want to see per page.

3 - The list of repositories is shown.

Precondition: There have been repositories examples added previously to the system.
Post-condition: The repository examples present in the database are listed to the user and can be filtered by name, owner, or URL.



Add/Upload Examples of Repositories

Repo Examples / Add/Upload

Add one repository example

Owner	
Name	
Url	
Microservice	☐

Add

Add a list of examples from a CSV file

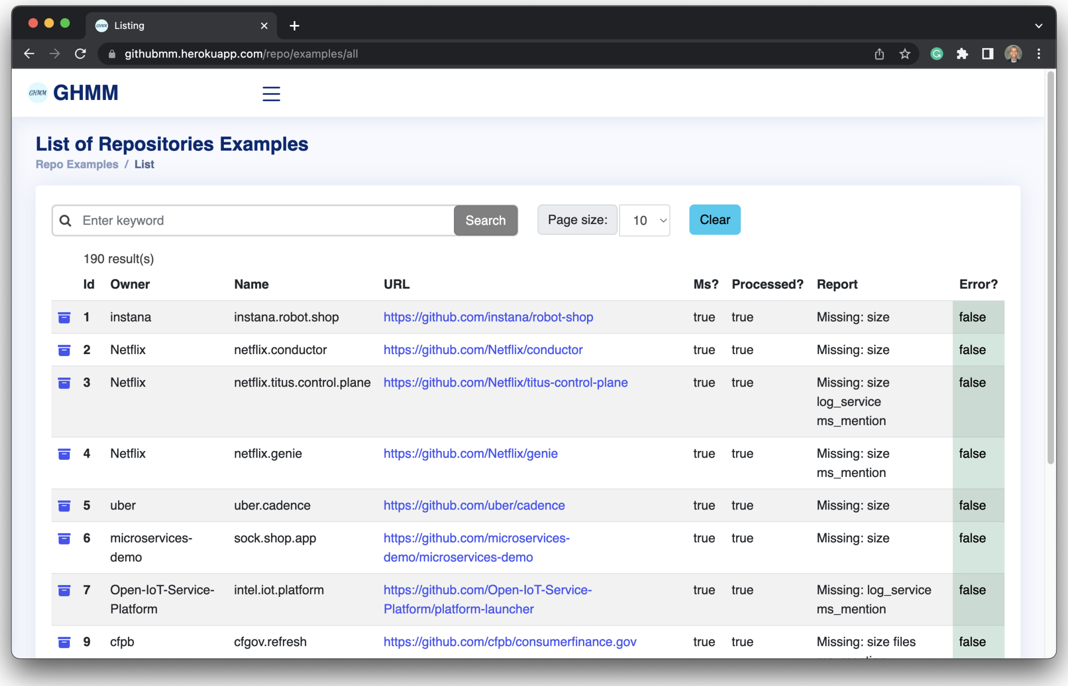
File Upload: Choose file No file chosen

Upload

Figure 5.5: Example View 1

In conclusion, the GitHub Microservice Mining Tool is a materialization of the concepts discovered in Chapter 2. Nonetheless, it is important to note that although these MSA identification concepts were implemented with GitHub API in this case, the same concepts still apply for any other platform having an API, such as Atlassian Bitbucket[7] or GitLab[8].

[7]https://bitbucket.org/
[8]https://about.gitlab.com/



Listing

githubmm.herokuapp.com/repo/examples/all

GHMM

List of Repositories Examples

Repo Examples / List

190 result(s)

	Id	Owner	Name	URL	Ms?	Processed?	Report	Error?
	1	instana	instana.robot.shop	https://github.com/instana/robot-shop	true	true	Missing: size	false
	2	Netflix	netflix.conductor	https://github.com/Netflix/conductor	true	true	Missing: size	false
	3	Netflix	netflix.titus.control.plane	https://github.com/Netflix/titus-control-plane	true	true	Missing: size log_service ms_mention	false
	4	Netflix	netflix.genie	https://github.com/Netflix/genie	true	true	Missing: size ms_mention	false
	5	uber	uber.cadence	https://github.com/uber/cadence	true	true	Missing: size	false
	6	microservices-demo	sock.shop.app	https://github.com/microservices-demo/microservices-demo	true	true	Missing: size	false
	7	Open-IoT-Service-Platform	intel.iot.platform	https://github.com/Open-IoT-Service-Platform/platform-launcher	true	true	Missing: log_service ms_mention	false
	9	cfpb	cfgov.refresh	https://github.com/cfpb/consumerfinance.gov	true	true	Missing: size files	false

Figure 5.6: Example View 2

Chapter 6

Evaluation

This chapter consolidates the research work by presenting the evaluation results produced throughout the GitHub Microservice Mining Tool shown in Chapter 5. The tool itself implements the solution proposed in Section 5.1 with the classification algorithm from Chapter 4, which in turn uses the metrics displayed in Chapter 3 that were gathered based on the study from Chapter 2.

We divide this evaluation into two distinct phases. The first (Section 6.2) was fundamental to iteratively evaluate and improve the algorithm. In the second phase (Section 6.3), we assumed the algorithm was stable, and the goal was to evaluate its capability to correctly identify MSAs and MSA sets. The links presented are paths from the tool's repository available at https://github.com/domingospanta/ghmm. Additionally, we also present a brief analysis of the results in Section 6.4, and possible threats to its validity in Section 6.5.

6.1 Evaluation Process

The evaluation process followed to validate whether or not the classification algorithm was able to correctly identify microservices was the following:

1. Upload a list of known MSAs to the tool.
2. Upload a list of known monoliths to the tool.
3. Process their metrics.
4. Run the classification algorithm.
5. Collect the classification results.
6. Create a confusion matrix from the results produced against the known results.
7. Analyze the generated metrics compared to the classification results and adjust the classification algorithm if necessary.
8. Repeat.

This process was executed several times with the primary goal of minimizing false positives. In the next sections, we will share details about the different phases of this process.

6.2 Phase 1

In this phase, the goal was to classify the same repositories from which the threshold had been generated. In other words, we wanted to verify that given several repositories to the algorithm as examples, the thresholds generated for the continuous metrics in combination with static binary metrics were enough for the algorithm to correctly classify those same examples with their correct type. We executed the evaluation process at least ten times, always making some changes, fixing some bugs, and improving the algorithm after each iteration. Following, we share the last iteration of the process in this phase:

1. We uploaded to the tool the complete curated list of 137 MSAs from the work of Podolski et al. [22]. After removing duplicates, 101 remained: microservices.csv[1]

2. We uploaded to the tool a list of 101 randomly selected monoliths from the work of Brito et al. [4]. And there were no duplicates, so all of them remained: monoliths.csv[2].

3. We processed their metrics and removed invalid entries (that is, those repositories which did not return results to the GitHub API). As a result, 94 MSAs and 96 monoliths' metrics were successfully collected: phase1_processed_metrics.csv[3]. In this file one can see the values collected for each of the metrics (7 binaries, and 3 continuous), as well as the auxiliary metrics to defined the indicators, such as DATABASE_SERVICES and PROGRAMMING_LANGUAGES used to set the MsSet indicator.

4. We ran the classification algorithm.

5. We collected the classification results: phase1_processed_classification.csv[4]. In this file, in addition to the 3 possible classifications of MICROSERVICE, MICROSERVICE_SET, AND MONOLITH, we can notice the score of each repository and a MESSAGE column reporting the missing metrics which is directly linked to the calculated score.

6. We created a confusion matrix for these results shown in Table 6.1.

7. We analyzed the classification results of this 10^{th} iteration and stopped the process.

[1]`https://github.com/domingospanta/ghmm/blob/main/data/sample/microservices.csv`

[2]`https://github.com/domingospanta/ghmm/blob/main/data/sample/monoliths.csv`

[3]`https://github.com/domingospanta/ghmm/blob/main/data/sample/phase1_processed_metrics.csv`

[4]`https://github.com/domingospanta/ghmm/blob/main/data/sample/phase1_processed_classification.csv`

Results As we can notice in Table 6.1, from the 94 MSAs input, the algorithm considered 59 to be MSA and 35 to be MSAs set. From those 59 MSAs, 31 were correctly identified as such, and while no MSA set was classified as MSA, 5 monoliths were classified as MSA. With that, the Class Precision for MSA was 0,86. It is essential to notice that even though the algorithm missed 28 out of 59 MSAs, it only classified 5 false positives, which means only a 14% rate. From those 35 MSA sets, 24 were correctly identified, giving it a Class Precision of 1 since none of the others was classified with this type. The other 11 MSA sets were wrongly classified as monoliths.

Table 6.1: Phase 1 Classification Results

	MSA	MSA set	Other	Class Precision
MSA Classification	**31**	0	5	0,86
MSA set Classification	0	**24**	0	1,00
Other Classification	28	11	**91**	0,70

6.3 Phase 2

In this phase, the goal was to classify a new dataset of known MSAs and applications known not to follow such an architecture utilizing the devised algorithm. Therefore, the evaluation process was as follows:

1. We uploaded to the tool 60 MSAs from the work of Rahman et al. [24][5]: mined_microservices.csv[6].
2. We uploaded to the tool 37 monoliths selected by a researcher outside of this project who did not have any knowledge about this identification approach: mined_monolitics.csv[7].
3. We processed their metrics and removed invalid entries. As a result, 35 MSAs and 33 monoliths' metrics were successfully collected: phase2_processed_metrics.csv[8].
4. We ran the classification algorithm.
5. We collected the classification results: phase2_processed_classification.csv[9].
6. We created a confusion matrix for these results shown on Table 6.2.
7. We analyzed the generated metrics compared to the classification results.

[5]The current list of MSAs from this work is available at `https://github.com/davidetaibi/Microservices_Project_List` and contains 60 projects (visited on 2023/04/27).

[6]/data/sample/mined_microservices.csv

[7]`https://github.com/domingospanta/ghmm/blob/main/data/sample/mined_monolitics.csv`

[8]`https://github.com/domingospanta/ghmm/blob/main/data/sample/phase2_processed_metrics.csv`

[9]`https://github.com/domingospanta/ghmm/blob/main/data/sample/phase2_processed_classification.csv`

Results In Table 6.2, we can see that the results of this second phase were similar to the first phase. Our algorithm correctly classified 22 out of 35 MSAs (16 as MSAs and 6 as MSA sets). There were 4 false positives, so the Class Precision was 0,85, close to 0,86 from the first phase. The other 13 microservices were classified as other.

Table 6.2: Phase 2 Classification Results

	MSA	MSA set	Other	Class Precision
MSA Classification	**16**	0	3	0,84
MSA set Classification	0	**6**	1	0,86
Other Classification	13	0	**29**	0,69

6.4 Analysis

The evaluation showed that the designed algorithm identifies actual microservice repositories with a precision of 85%, producing only 15% of false positives. This shows that the method utilized demonstrated an elevated rate of precision. Although there are many false negatives, this is not a concern. Indeed, our goal was to find repositories containing MSAs, and our algorithm can do it. What the many false negatives mean is that the algorithm will have to look into much more repositories that would be necessary to find the correct ones (MSA). However, given the enormous amount of MSAs available at GitHub, this does not threaten the attempt to create corpora of arbitrary sizes of MSAs.

6.5 Threats to Validity

Although the designed algorithm worked well for the used datasets in both validation phases, it may not produce similar results given a completely different dataset. We can affirm, however, that 70% of the metrics utilized are stable (binary metrics). In contrast, the other 30% (continuous metrics) are more affected by changes in the dataset, and so influence the final results. This also implies that they (and the algorithm) can evolve given more correctly classified software projects. Moreover, in phase 2 of the evaluation, we gave the tool a new set containing monoliths and MSAs in similar amounts, and it achieved a precision of 85%, which can be considered high.

There is the risk of overfitting our algorithm. However, in the second phase of our evaluation, we gave the algorithm a new dataset containing more than 60 projects, and it produced a precision similar to the results of phase 1. Thus, we expect the results to hold.

The validation process, as well as the result report, was manually executed by the authors, which is error-prone. Nevertheless, we make our source code available, a deployment for testing, and all the datasets used for further validation.

Chapter 7

Conclusions and Future Work

In this closing chapter of the dissertation, we will share our conclusions derived from the performed work. Additionally, we will explore potential research directions that might emerge as a result of this study.

7.1 Conclusions

From a thorough literature review, we have identified and analyzed a range of works related to the microservices architectural style. Based on this analysis, we have constructed a comprehensive set of characteristics that can effectively distinguish microservices from other architectural approaches.

Furthermore, we have developed a set of metrics to provide a quantifiable framework for identifying microservices in code repositories. The metrics encompass various aspects, such as lightweight protocol, independent deployability, database ownership, message broker usage, documentation, protocol weight, logging practices, size considerations, the number of files, and overall content analysis. These metrics serve as reliable indicators for identifying microservices.

Additionally, we have devised a microservice identification algorithm. This algorithm utilizes the set of metrics developed, enabling the accurate identification of actual microservice repositories with a high precision rate of 85%. This algorithm represents a valuable method for software engineers and researchers to efficiently identify microservices in code repositories.

Finally, as a practical utilization of the concepts and techniques proposed in this research, we have also implemented the GitHub Microservice Mining Tool. This tool integrates the set of metrics and the microservice identification algorithm, providing a user-friendly interface to mine microservices from code repositories. The tool leverages the insights gained from our literature review and algorithm development, offering a valuable resource for practitioners seeking to analyze and extract microservices from existing codebases.

7.2 Future Works

Although this thesis dissertation has made significant contributions to the identification and mining of microservices, there are some routes for further research and development in this domain. The following suggestions outline potential directions for future work:

Metrics Refinement and Extension: The set of metrics developed in this research serves as a solid foundation for microservice identification. However, further refinement and extension of these metrics could enhance the accuracy of microservice recognition.

Algorithm Enhancement: Although our algorithm achieves a meritorious precision rate, ongoing research can focus on optimizing and enhancing the algorithm's performance. Incorporating machine learning techniques, such as neural networks or clustering algorithms, may offer opportunities for more accurate and automated identification of microservices within code repositories.

Mixed Codebases Evaluation: The current evaluation of our algorithm and tool was performed on a specific set of code repositories. Conducting evaluations on diverse codebases, containing more programming languages, domains, and project sizes, could provide a more complete understanding of the algorithm's effectiveness and applicability.

By pursuing these future research directions, we can continue to advance the field of microservices architecture and empower software engineers with improved tools and insights for the effective development and maintenance of microservice-based systems.

©©

References

[1] Tiago L Alves, Christiaan Ypma, and Joost Visser. Deriving metric thresholds from benchmark data. In *2010 IEEE international conference on software maintenance*, pages 1–10. IEEE, 2010.

[2] Alan Bandeira, Carlos Alberto Medeiros, Matheus Paixao, and Paulo Henrique Maia. We need to talk about microservices: An analysis from the discussions on stackoverflow. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 255–259. IEEE Press, 2019.

[3] Pearl Brereton, Barbara A Kitchenham, David Budgen, Mark Turner, and Mohamed Khalil. Lessons from applying the systematic literature review process within the software engineering domain. *Journal of systems and software*, 80(4):571–583, 2007.

[4] Miguel Brito, Jácome Cunha, and João Saraiva. Identification of microservices from monolithic applications through topic modelling. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, SAC '21, page 1409–1418, New York, NY, USA, 2021. Association for Computing Machinery.

[5] Andrea Caracciolo, Andrei Chis, Boris Spasojevic, and Mircea Lungu. Pangea: A workbench for statically analyzing multi-language software corpora. In *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation*, pages 71–76. IEEE, 2014.

[6] Rui Chen, Shanshan Li, and Zheng Li. From monolith to microservices: A dataflow-driven approach. In *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*, pages 466–475. IEEE, 12 2017.

[7] José Correia and António Rito Silva. Identification of monolith functionality refactorings for microservices migration. *Software: Practice and Experience*, n/a, 2022.

[8] Mohamed Daoud, Asmae El Mezouari, Noura Faci, Djamal Benslimane, Zakaria Maamar, and Aziz El Fazziki. A multi-model based microservices identification approach. *Journal of Systems Architecture*, 118:102200, 2021.

[9] Thatiane de Oliveira Rosa, João Francisco Lino Daniel, Eduardo Martins Guerra, and Alfredo Goldman. A method for architectural trade-off analysis based on patterns: Evaluating microservices structural attributes. In *Proceedings of the European Conference on Pattern Languages of Programs 2020*. Association for Computing Machinery, 2020.

[10] Umesh Deshpande, Nick Linck, and Sangeetha Seshadri. Self managed data protection for containers. In *Proceedings of the 14th ACM International Conference on Systems and Storage*. Association for Computing Machinery, 2021.

[11] Martin Fowler. Bliki: Monolithfirst. https://martinfowler.com/bliki/MonolithFirst.html, Jun 2015.

[12] Paolo Di Francesco. Architecting microservices. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. IEEE, 4 2017.

[13] GitHub. Where the world builds software. https://github.com, 2023.

[14] Giona Granchelli, Mario Cardarelli, Paolo Di Francesco, Ivano Malavolta, Ludovico Iovino, and Amleto Di Salle. Towards recovering the software architecture of microservice-based systems. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, pages 46–53. IEEE, 4 2017.

[15] Ahmed E. Hassan. The road ahead for mining software repositories. In *2008 Frontiers of Software Maintenance*, pages 48–57, 2008.

[16] Sara Hassan, Rami Bahsoon, and Rick Kazman. Microservice transition and its granularity problem: A systematic mapping study. *Software: Practice and Experience*, 50:1651–1681, 2020.

[17] Hadi Hemmati, Sarah Nadi, Olga Baysal, Oleksii Kononenko, Wei Wang, Reid Holmes, and Michael W Godfrey. The msr cookbook: Mining a decade of research. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 343–352. IEEE, 2013.

[18] Wuxia Jin, Ting Liu, Yuanfang Cai, Rick Kazman, Ran Mo, and Qinghua Zheng. Service candidate identification from monolithic systems based on execution traces. *IEEE Transactions on Software Engineering*, 47:987–1007, 5 2021.

[19] Staffs Keele and other. Guidelines for performing systematic literature reviews in software engineering. Technical report, Technical report, ver. 2.3 ebse technical report. ebse, 2007.

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

[21] Giuseppe Antonio Pierro, Roberto Tonelli, and Michele Marchesi. Smart-corpus: an organized repository of ethereum smart contracts source code and metrics. *arXiv preprint arXiv:2011.01723*, 2020.

[22] Vladimir Podolskiy, Maria Patrou, Panos Patros, Michael Gerndt, and Kenneth B Kent. The weakest link: Revealing and modeling the architectural patterns of microservice applications. In *Proceedings of the 30th Annual International Conference on Computer Science and Software Engineering*, pages 113–122. IBM Corp., 2020.

[23] Wouter Poncin, Alexander Serebrenik, and Mark van den Brand. Process mining software repositories. In *2011 15th European Conference on Software Maintenance and Reengineering*, pages 5–14, 2011.

[24] Mohammad Imranur Rahman, Sebastiano Panichella, and Davide Taibi. A curated dataset of microservices-based systems. *SSSME-2019*, 2019.

[25] Chris Richardson. *Microservices patterns: with examples in Java*. Simon and Schuster, 2018.

[26] Yamina Romani, Okba Tibermacine, and Chouki Tibermacine. Towards migrating legacy software systems to microservice-based architectures: a data-centric process for microservice identification. In *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*, pages 15–19. IEEE, 3 2022.

[27] Khaled Sellami, Ali Ouni, Mohamed Aymen Saied, Salah Bouktif, and Mohamed Wiem Mkaouer. Improving microservices extraction using evolutionary search. *Information and Software Technology*, 151:106996, 2022.

[28] Jacopo Soldani, Giuseppe Muntoni, Davide Neri, and Antonio Brogi. The μtosca toolchain: Mining, analyzing, and refactoring microservice-based architectures. *Software: Practice and Experience*, 51(7):1591–1621, 2021.

[29] M Vidoni. A systematic process for mining software repositories: Results from a systematic literature review. *Information and Software Technology*, page 106791, 2021.

[30] Evgeny Volynsky, Merlin Mehmed, and Stephan Krusche. Architect: A framework for the migration to microservices. In *2022 International Conference on Computing, Electronics & Communications Engineering (iCCECE)*, pages 71–76. IEEE, 8 2022.

Appendix A

List of Included Publication on the Systematic Literature Review

Table A.1: List of Included Publications

Type	Authors	Title	Year	Publisher
Conference Paper	Bandeira A,Medeiros CA,Paixao M,Maia PH	We Need to Talk about Microservices: An Analysis from the Discussions on StackOverflow	2019	IEEE
Conference Paper	Wei Y,Yu Y,Pan M,Zhang T	A Feature Table Approach to Decomposing Monolithic Applications into Microservices	2020	ACM
Conference Paper	Santos A,Paula H	Microservice Decomposition and Evaluation Using Dependency Graph and Silhouette Coefficient	2021	ACM
Conference Paper	Eski S,Buzluca F	An Automatic Extraction Approach: Transition to Microservices Architecture from Monolithic Application	2018	ACM
Miscellaneous	Josélyne MI,Tuheirwe-Mukasa D,Kanagwa B,Balikuddembe J	Partitioning Microservices: A Domain Engineering Approach	2018	ACM
Conference Paper	Sellami K,Saied MA,Ouni A	A Hierarchical DBSCAN Method for Extracting Microservices from Monolithic Applications	2022	ACM

Conference Paper	Morais G,Bork D,Adda M	Towards an Ontology-Driven Approach to Model and Analyze Microservices Architectures	2021	ACM
Conference Paper	Santos N,Salgado CE,Morais F,Melo M,Silva S,Martins R,Pereira M,Rodrigues H,Machado RJ,Ferreira N,Pereira M	A Logical Architecture Design Method for Microservices Architectures	2019	ACM
Miscellaneous	Márquez G,Astudillo H	Identifying Availability Tactics to Support Security Architectural Design of Microservice-Based Systems	2019	ACM
Conference Paper	Auer F,Felderer M,Lenarduzzi V	Towards Defining a Microservice Migration Framework	2018	ACM
Conference Paper	Carrasco A,van Bladel B,Demeyer S	Migrating towards Microservices: Migration and Architecture Smells	2018	ACM
Conference Paper	Yang Z,Wu S,Zhang C	A Microservices Identification Approach based on Problem Frames	2022	IEEE
Conference Paper	Schroer C,Wittfoth S,Gomez JM	A Process Model for Microservices Design and Identification	2021	IEEE
Conference Paper	Zaragoza P,Seriai AD,Seriai A,Shatnawi A,Derras M	Leveraging the Layered Architecture for Microservice Recovery	2022	IEEE
Journal Article	Furda A,Fidge C,Zimmermann O,Kelly W,Barros A	Migrating Enterprise Legacy Source Code to Microservices: On Multitenancy, Statefulness, and Data Consistency	2018	IEEE
Conference Paper	Chen R,Li S,Li Z	From Monolith to Microservices: A Dataflow-Driven Approach	2017	IEEE
Conference Paper	Selmadji A,Seriai AD,Bouziane HL,Mahamane RO,Zaragoza P,Dony C	From Monolithic Architecture Style to Microservice one Based on a Semi-Automatic Approach	2020	IEEE
Conference Paper	Amiri MJ	Object-Aware Identification of Microservices	2018	IEEE

Conference Paper	Zhang Y,Liu B,Dai L,Chen K,Cao X	Automated Microservice Identification in Legacy Systems with Functional and Non-Functional Metrics	2020	IEEE
Conference Paper	Volynsky E,Mehmed M,Krusche S	Architect: A Framework for the Migration to Microservices	2022	IEEE
Miscellaneous	Francesco PD	Architecting Microservices	2017	IEEE
Conference Paper	Romani Y,Tibermacine O,Tibermacine C	Towards Migrating Legacy Software Systems to Microservice-based Architectures: a Data-Centric Process for Microservice Identification	2022	IEEE
Conference Paper	Daoud M,Mezouari AE,Faci N,Benslimane D,Maamar Z,Fazziki AE	Towards an Automatic Identification of Microservices from Business Processes	2020	IEEE
Conference Paper	Yedida R,Krishna R,Kalia A,Menzies T,Xiao J,Vukovic M	Lessons learned from hyper-parameter tuning for microservice candidate identification	2021	IEEE
Journal Article	Jin W,Liu T,Cai Y,Kazman R,Mo R,Zheng Q	Service Candidate Identification from Monolithic Systems Based on Execution Traces	2021	IEEE
Journal Article	Daoud M,Mezouari AE,Faci N,Benslimane D,Maamar Z,Fazziki AE	A multi-model based microservices identification approach	2021	ScienceDirect
Journal Article	Li S,Zhang H,Jia Z,Li Z,Zhang C,Li J,Gao Q,Ge J,Shan Z	A dataflow-driven approach to identifying microservices from monolithic applications	2019	ScienceDirect
Journal Article	Christoforou A,Andreou AS,Garriga M,Baresi L	Adopting microservice architecture: A decision support model based on genetically evolved multi-layer FCM	2022	ScienceDirect
Miscellaneous	Correia J,Silva AR	Identification of monolith functionality refactorings for microservices migration	2022	Wiley
Miscellaneous	Trabelsi I,Abdellatif M,Abubaker A,Moha N,Mosser S,Ebrahimi-Kahou S,Guéhéneuc YG	From legacy to microservices: A type-based approach for microservices identification using machine learning and semantic analysis	2022	Wiley

Journal Article	Balalaie A,Heydarnoori A,Jamshidi P,Tamburri DA,Lynn T	Microservices migration patterns	2018	Wiley
Conference Paper	Brito M,Cunha J,Saraiva J	Identification of Microservices from Monolithic Applications through Topic Modelling	2021	ACM
Conference Paper	Oliveira J,Pinheiro D,Figueiredo E	JExpert: A Tool for Library Expert Identification	2020	ACM
Conference Paper	Carvalho L,Garcia A,Colanzi TE,Assunção WK,Lima MJ,Fonseca B,Ribeiro M,Lucena C	Search-Based Many-Criteria Identification of Microservices from Legacy Systems	2020	ACM
Conference Paper	Costa DI,e Silva Filho EP,da Silva RF,de C. Quaresma Gama TD,Cortés MI	Microservice Architecture: A Tertiary Study	2020	ACM
Conference Paper	Granchelli G,Cardarelli M,Francesco PD,Malavolta I,Iovino L,Salle AD	Towards Recovering the Software Architecture of Microservice-Based Systems	2017	IEEE
Journal Article	Schiewe M,Curtis J,Bushong V,Cerny T	Advancing Static Code Analysis With Language-Agnostic Component Identification	2022	IEEE
Journal Article	Sellami K,Ouni A,Saied MA,Bouktif S,Mkaouer MW	Improving microservices extraction using evolutionary search	2022	ScienceDirect
Journal Article	Abdellatif M,Shatnawi A,Mili H,Moha N,Boussaidi GE,Hecht G,Privat J,Guéhéneuc YG	A taxonomy of service identification approaches for legacy software systems modernization	2021	ScienceDirect
Journal Article	Soldani J,Muntoni G,Neri D,Brogi A	The µTOSCA toolchain: Mining, analyzing, and refactoring microservice-based architectures	2021	Wiley
Journal Article	Hassan S,Bahsoon R,Kazman R	Microservice transition and its granularity problem: A systematic mapping study	2020	Wiley

Conference Paper	Podolskiy V,Patrou M,Patros P,Gerndt M,Kent KB	The Weakest Link: Revealing and Modeling the Architectural Patterns of Microservice Applications	2020	ACM
Conference Paper	Deshpande U,Linck N,Seshadri S	Self Managed Data Protection for Containers	2021	ACM
Miscellaneous	de Oliveira Rosa T,Daniel JF,Guerra EM,Goldman A	A Method for Architectural Trade-off Analysis Based on Patterns: Evaluating Microservices Structural Attributes	2020	ACM

Appendix B

List of Included Microservices Code Repositories

	URL
1	https://github.com/instana/robot-shop
2	https://github.com/Netflix/conductor
3	https://github.com/Netflix/titus-control-plane
4	https://github.com/Netflix/genie
5	https://github.com/uber/cadence
6	https://github.com/microservices-demo/microservices-demo
7	https://github.com/Open-IoT-Service-Platform/platform-launcher/
8	https://github.com/apache/incubator-airflow/
9	https://github.com/cfpb/cfgov-refresh
10	https://github.com/Yelp/Tron
11	https://github.com/Yelp/paasta
12	https://github.com/Yelp/kafka-utils
13	https://github.com/Yelp/task_processing
14	https://github.com/Yelp/casper
15	https://github.com/linkedin/Burrow
16	https://github.com/linkedin/WhereHows
17	https://github.com/DataDog/integrations-core
18	https://github.com/DataDog/dd-trace-rb
19	https://github.com/DataDog/dd-trace-dotnet
20	https://github.com/DataDog/integrations-extras
21	https://github.com/DataDog/trace-examples
22	https://github.com/dotnet-architecture/eShopOnContainers
23	https://github.com/NrgXnat/xnat-docker-compose
24	https://github.com/square/shuttle

25	https://github.com/apache/ignite
26	https://github.com/apache/beam
27	https://github.com/apache/arrow
28	https://github.com/apache/pulsar
29	https://github.com/apache/incubator-skywalking
30	https://github.com/apache/james-project
31	https://github.com/apache/bookkeeper
32	https://github.com/apache/flink
33	https://github.com/apache/myfaces-tobago
34	https://github.com/apache/predictionio
35	https://github.com/apache/incubator-griffin
36	https://github.com/apache/metron
37	https://github.com/apache/trafficcontrol
38	https://github.com/spotify/bigtable-autoscaler
39	https://github.com/andreaskoch/dockerized-magento
40	https://github.com/arvatoSCM/dockerize-magento2
41	https://github.com/sameersbn/docker-gitlab
42	https://github.com/SAP/InfraBox
43	https://github.com/zammad/zammad-docker-compose
44	https://github.com/lucirr/docker-compose-portal
45	https://github.com/reportportal/reportportal
46	https://github.com/onap/portal
47	https://github.com/datosgobar/portal-base
48	https://github.com/cBioPortal/cbioportal
49	https://github.com/WeblateOrg/docker-compose
50	https://github.com/ONLYOFFICE/Docker-CommunityServer
51	https://github.com/owncloud-docker/server
52	https://github.com/zalando/zally
53	https://github.com/metabrainz/listenbrainz-server
54	https://github.com/pingcap/tidb-docker-compose
55	https://github.com/ansjin/terminus
56	https://github.com/cloudfoundry-incubator/stratos
57	https://github.com/spryker/docker-shop-suite/
58	https://github.com/devicehive/devicehive-docker
59	https://github.com/FriendsOfREDAXO/redaxo-mit-docker
60	https://github.com/okfn/docker-ckan
61	https://github.com/adaptdk/drupal-docker-compose
62	https://github.com/theodorosploumis/drupal-docker
63	https://github.com/poldracklab/open_fmri
64	https://github.com/eoinsha/node-seneca-base

65	https://github.com/Mogtofu33/docker-compose-drupal
66	https://github.com/inspirehep/inspire-next
67	https://github.com/pardahlman/docker-rabbitmq-cluster
68	https://github.com/alexellis/faas-example-voting-app
69	https://github.com/simplesteph/kafka-stack-docker-compose
70	https://github.com/cms-sw/cms-docker/
71	https://github.com/bitrixdock/bitrixdock
72	https://github.com/enonic-cloud/docker-compose-enonic-cms
73	https://github.com/cortex-cms/cortex-starter
74	https://github.com/icecrime/vossibility-stack
75	https://github.com/GeoNode/geonode
76	https://github.com/jazzband/website
77	https://github.com/Landoop/fast-data-connect-cluster
78	https://github.com/xchem/fragalysis-stack
79	https://github.com/sprintcube/docker-compose-lamp
80	https://github.com/mgcrea/docker-compose-tick-stack
81	https://github.com/damsonn/node-docker-compose
82	https://github.com/cytopia/devilbox/
83	https://github.com/phaldan/compose-lampn
84	https://github.com/StackStorm/st2-docker
85	https://github.com/dashersw/cote-workshop
86	https://github.com/Accenture/reactive-interaction-gateway
87	https://github.com/uc-cdis/compose-services
88	https://github.com/testdrivenio/flask-microservices-main
89	https://github.com/nklmish/microservice-demo
90	https://github.com/kbastani/spring-cloud-microservice-example
91	https://github.com/sivaprasadreddy/spring-boot-microservices-series
92	https://github.com/cer/microservices-examples
93	https://github.com/ewolff/microservice
94	https://github.com/apssouza22/java-microservice
95	https://github.com/mjhea0/microservice-movies
96	https://github.com/launchany/microservices-nginx-gateway
97	https://github.com/markglh/composing-microservices-with-sbt-docker
98	https://github.com/microservices-patterns/ftgo-application
99	https://github.com/callistaenterprise/blog-microservices
100	https://github.com/spring-petclinic/spring-petclinic-microservices
101	https://github.com/sczyh30/vertx-blueprint-microservice

Appendix C

List of Included Monoliths Code Repositories

	URL
1	https://github.com/miansen/Roothub
2	https://github.com/huanglu20124/invoice
3	https://github.com/Lab41/Dendrite
4	https://github.com/pibigstar/parsevip
5	https://github.com/OCR4all/OCR4all
6	https://github.com/moocss/EasyCMS
7	https://github.com/Vino007/javaEEScaffold
8	https://github.com/GdeiAssistant/GdeiAssistant
9	https://github.com/purang-fintech/seppb
10	https://github.com/muralibasani/kafkawize
11	https://github.com/qianqianjun/Educational-management
12	https://github.com/wsk1103/movie-boot
13	https://github.com/JoeyBling/bootplus
14	https://github.com/forTribeforXuanmo/sword-forum
15	https://github.com/justinscript/travel.b2b
16	https://github.com/superman544/JavaOJSystem
17	https://github.com/514840279/danyuan-application
18	https://github.com/iminto/baicai
19	https://github.com/krishagni/openspecimen
20	https://github.com/justinscript/shopping.plat
21	https://github.com/cym1102/nginxWebUI
22	https://github.com/Jannchie/biliob_backend
23	https://github.com/INCF/eeg-database
24	https://github.com/Lotharing/SDIMS

25 https://github.com/kanban/kanban-app
26 https://github.com/Frodez/BlogManagePlatform
27 https://github.com/finallysmile3/ExamSystem
28 https://github.com/cloudfoundry-attic/login-server
29 https://github.com/atlasapi/atlas
30 https://github.com/metasfresh/metasfresh-webui-api-legacy
31 https://github.com/tangdu/smh2
32 https://github.com/qiao-zhi/jwxt
33 https://github.com/jiangzongyao/kettle-master
34 https://github.com/jdmr/mateo
35 https://github.com/shuxianfeng/movision
36 https://github.com/leluque/university-site-cms
37 https://github.com/ghostxbh/uzy-ssm-mall
38 https://github.com/OpenGeoportal/OGP2
39 https://github.com/litblank/hammer
40 https://github.com/choerodon/agile-service-old
41 https://github.com/kai8406/cmop
42 https://github.com/gliderwiki/glider
43 https://github.com/hslooooooool/form_flow
44 https://github.com/mozammel/mNet
45 https://github.com/easy-ware/api-manager
46 https://github.com/lvr1997/ershoujiaoyi
47 https://github.com/Ryan–Yang/CBoard-boot
48 https://github.com/GZzzhsmart/P2Pproj
49 https://github.com/HIIT/dime-server
50 https://github.com/doooyo/Weixin_Server
51 https://github.com/justinbaby/my-paper
52 https://github.com/edgexfoundry/core-data
53 https://github.com/mofadeyunduo/online-judge
54 https://github.com/MiniPa/cjs_ssms
55 https://github.com/AURIN/online-whatif
56 https://github.com/fishstormX/fishmaple
57 https://github.com/opendevstack/ods-provisioning-app
58 https://github.com/parasoft/parabank
59 https://github.com/MaritimeConnectivityPlatform/IdentityRegistry
60 https://github.com/zhangyanbo2007/youkefu
61 https://github.com/starqiu/RDMP1
62 https://github.com/yunchaoyun/active4j-flow
63 https://github.com/yorkmass/Yark-AdminMS
64 https://github.com/vector1989/EMAS

65	https://github.com/768330962/poet_ready_system
66	https://github.com/EUSurvey/EUSURVEY
67	https://github.com/suyeq/steamMall
68	https://github.com/zlren/noah-health
69	https://github.com/Prasad108/TutesMessanger
70	https://github.com/busing/circle_web
71	https://github.com/UDA-EJIE/udaLib
72	https://github.com/bbaibb1009/wxcrm
73	https://github.com/khasang/delivery
74	https://github.com/TexnologiaLogismikou/Fiz
75	https://github.com/uq-eresearch/oztrack
76	https://github.com/wang007/live-server
77	https://github.com/zxwgdft/paladin-boot
78	https://github.com/shenshaoming/byte_easy
79	https://github.com/nimble-platform/business-process-service
80	https://github.com/codemky/uni
81	https://github.com/ushahidi/SwiftRiver-API
82	https://github.com/softservedata/lv257
83	https://github.com/aramsoft/aramcomp
84	https://github.com/bao17634/Warehouse-system
85	https://github.com/shigenwang/membership
86	https://github.com/SafeExamBrowser/seb-server
87	https://github.com/Seenck/jeecg-bpm-3.8
88	https://github.com/GraffiTab/GraffiTab-Backend
89	https://github.com/surajcm/Poseidon
90	https://github.com/loongw513029/buscloud
91	https://github.com/tcrct/duang
92	https://github.com/ZFGCCP/ZFGC3
93	https://github.com/WilsonHu/sinsim
94	https://github.com/crypto-coder/open-cyclos
95	https://github.com/sfx478076717/goldenarches
96	https://github.com/zndo/oss-admin-parent
97	https://github.com/dp2-g56/Dp2-L02
98	https://github.com/cable5881/Fund
99	https://github.com/wangwang1230/te-empl
100	https://github.com/ElectiveTeam/elective_system
101	https://github.com/Rocklee830630/WMSystem

Appendix D

Use Case Descriptions And Views for GitHub Microservice Mining Tool

UC1 Maintain RepoExample: As Software Engineer, I want to add, list, or remove RepoExamples.
Scenario: Add one repo example - the view is shown on Fig. D.1

1 - On the main menu, click on Repo Examples > Add/Upload.
2 - Type the owner of the code repository in the field Owner.
3 - Type the name of the code repository in the field Name.
4 - Type the Url of the code repository in the field Url.
5 - Click on the checkbox Microservice if the entered repository is an MSA.
6 - Click on the button Add.

Precondition: N/A.
Post-condition: The repository example is saved in the database.

Scenario: Add a list of repo examples - the view is shown on Fig. D.1 and the sequence diagram on Fig. D.10

1 - On the main menu, click on Repo Examples > Add/Upload.
2 - Click on Choose file.
3 - Select the CSV file.
4 - Click on Upload.

Precondition: The user has a CSV file in the format of this example[1]containing the repositories to be added.

[1]https://github.com/domingospanta/ghmm/blob/main/data/sample/microservices.csv

Post-condition: The list of repository examples are saved in the database.

Scenario: List repo examples - the view is shown on Fig. D.2

1 - On the main menu, click on Repo Examples > List Examples.

2 - Click on Page Size to choose how many repositories you want to see per page.

3 - The list of repositories is shown.

Precondition: There have been repositories examples added previously to the system.
Post-condition: The repository examples present in the database is listed to the user and can be filtered by name, owner, or URL.

Scenario: Remove repo example - the view is shown on Fig. D.2

1 - On the main menu, click on Repo Examples > List Examples.

2 - Click on Page Size to choose how many repositories you want to see per page.

3 - On the left side of each line, click on the archive icon to delete the line.

Precondition: There have been repositories examples added previously to the system.
Post-condition: The repository example is removed from the database.

UC2 Process RepoExample Metrics: As Software Engineer, I want to process RepoExample Metrics.
Scenario: Start RepoExamples Metrics Processing - the view is shown on Fig. D.3 and the implementation details on sequence diagrams illustrated by Fig. D.11 and Fig. D.12

1 - On the main menu, click on Repo Examples > Process Metrics.

2 - Click on the Start button.

Precondition: There have been repositories examples added previously to the system.
Post-condition: The system fetched data about the repositories on GitHub to generate their metrics.

UC3 List/Review RepoExample Metrics: As Software Engineer, I want to list the generated RepoExample metrics.
Scenario: List RepoExample Metrics - the view is shown on Fig. D.4

1 - On the main menu, click on Repo Examples > List Metrics.

2 - Click on Page Size to choose how many repositories you want to see per page.

Precondition: The RepoExamples Metrics have been previously processed.
Post-condition: The repository examples metrics present in the database are listed to the user and can be filtered by name, owner, or URL.

UC4 View RepoExample Metrics Statistics: As Software Engineer, I want to view statistical data about the generated metrics.
Scenario: Show RepoExample Metrics Statistics - the view is shown on Fig. D.5

1 - On the main menu, click on Repo Examples > Metrics Statistics.

Precondition: The RepoExamples Metrics have been previously processed.
Post-condition: Statistics about MSAs and monoliths are shown.

UC5 Search/Upload MinedRepo: As Software Engineer, I want to upload a list of MinedRepos (unclassified repositories) or search them through GitHub API.
Scenario: Search MinedRepo - the view is shown on Fig. D.6

1 - On the main menu, click on Identification > Search/Upload.

2 - Type the search term on the field Search String according to GitHub API search query[2].

3 - On the field Programming Languages, hold down the control key and click on the desired programming languages for the search.

4 - On the field Quantity, type the number of code repositories you want to find.

5 - Click on the Clear current list checkbox if you want to erase any previous search/upload.

6 - Click on Search.

Precondition: N/A
Post-condition: The desired quantity of code repositories is searched on GitHub and saved to the database.

Scenario: Upload MinedRepo - the view is shown on Fig. D.6

[2]https://docs.github.com/en/rest/search?apiVersion=2022-11-28#search-repositories

1 - On the main menu, click on Identification > Search/Upload.

2 - Click on Choose file.

3 - Select the CSV file.

4 - Click on Upload.

Precondition: The user has a CSV file in the format of this example[3] containing the mined repositories to be added.
Post-condition: The list of mined repositories are saved in the database.

UC6 Process MinedRepo Metrics: As Software Engineer, I want to process MinedRepo Metrics.
Scenario: Start MinedRepos Metrics Processing - the view is shown on Fig. D.7

1 - On the main menu, click on Identification > Process Metrics.

2 - Click on the Start button.

Precondition: There have been mined repositories added previously to the system.
Post-condition: The system fetched data about the repositories on GitHub to generate their metrics.

UC7 List/Review MinedRepo Metrics: As Software Engineer, I want to list the generated MinedRepo metrics.
Scenario: List MinedRepo Metrics - the view is shown on Fig. D.8

1 - On the main menu, click on Identification > List Metrics.

2 - Click on Page Size to choose how many repositories you want to see per page.

Precondition: The MinedRepos Metrics have been previously processed.
Post-condition: The repository examples metrics present in the database are listed to the user and can be filtered by name, owner, or URL.

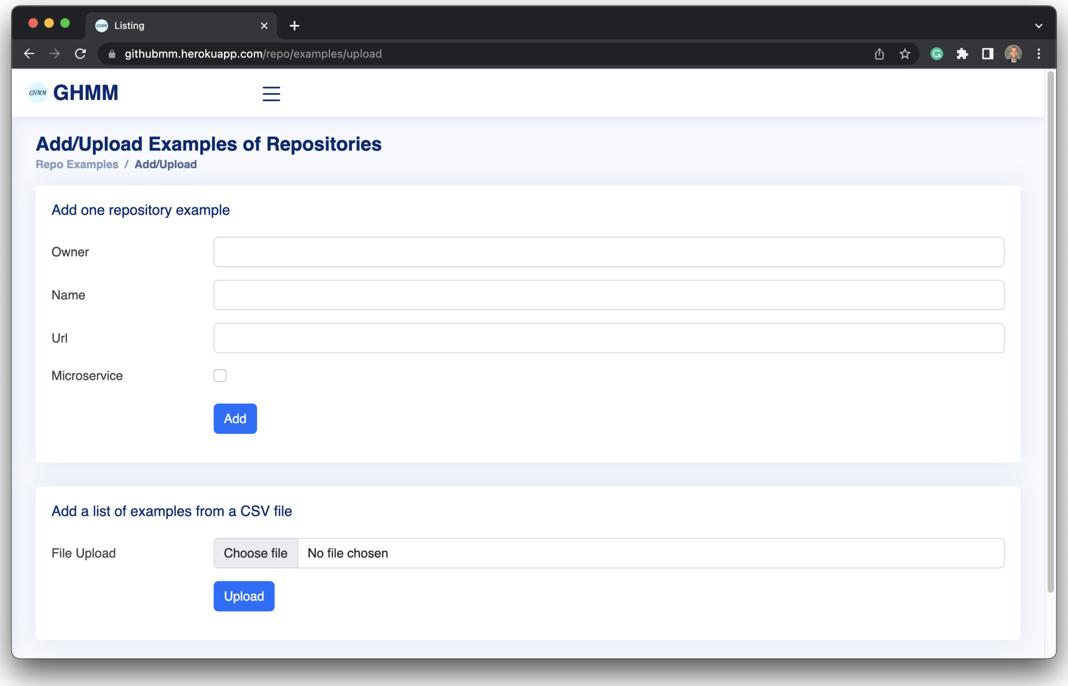
UC8 List/Filter Classified MinedRepos: As Software Engineer, I want to list and/or filter classified MinedRepos.
Scenario: Show Classified MinedRepos - the view is shown on Fig. D.9

1 - On the main menu, click on Corpus > Mined Repos.

2 - Click on Page Size to choose how many mined repositories you want to see per page.

[3]https://github.com/domingospanta/ghmm/blob/main/data/sample/mined_microservices.csv

Precondition: The MinedRepos Metrics have been previously processed.
Post-condition: The mined repositories present in the database are listed to the user with their respective classification and can be filtered by name, owner, or URL.



Add/Upload Examples of Repositories

Repo Examples / Add/Upload

Add one repository example

Owner	
Name	
Url	
Microservice	☐

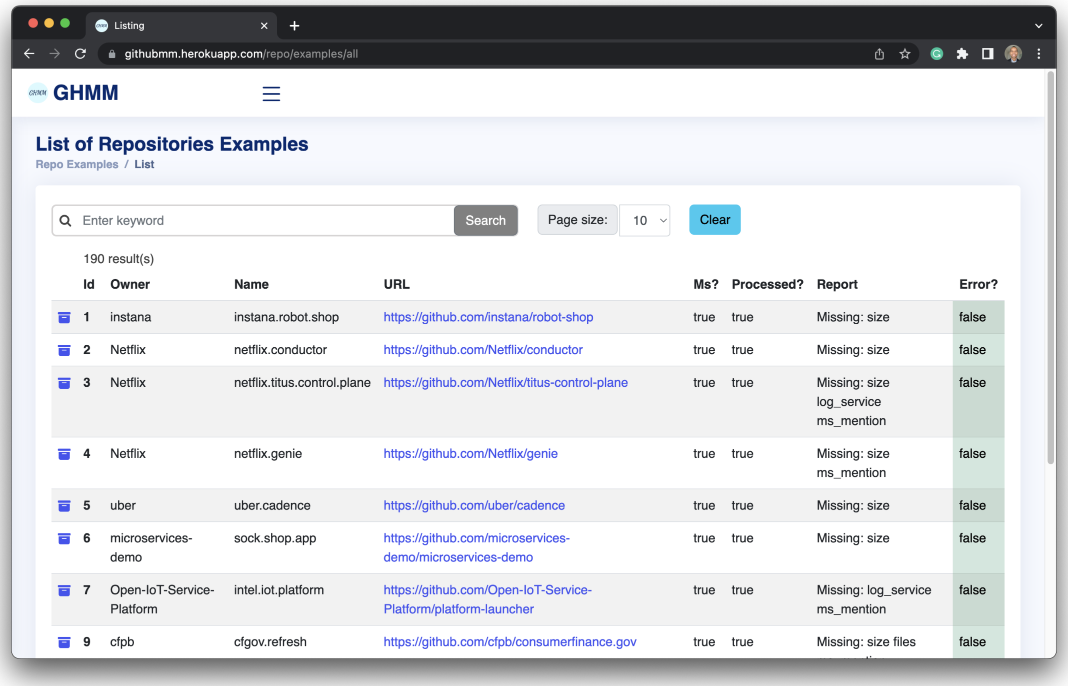
Add

Add a list of examples from a CSV file

File Upload: Choose file No file chosen

Upload

Figure D.1: Add/Upload Examples of Repositories View



Listing

githubmm.herokuapp.com/repo/examples/all

GHMM

List of Repositories Examples

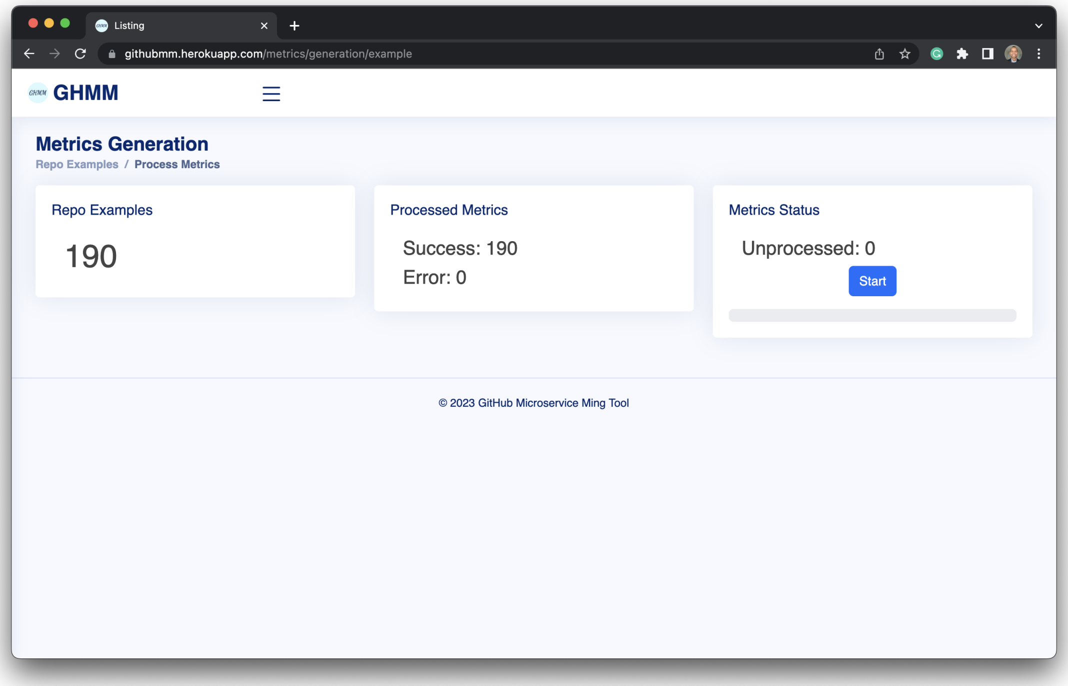
Repo Examples / List

Enter keyword | Search | Page size: 10 | Clear

190 result(s)

	Id	Owner	Name	URL	Ms?	Processed?	Report	Error?
	1	instana	instana.robot.shop	https://github.com/instana/robot-shop	true	true	Missing: size	false
	2	Netflix	netflix.conductor	https://github.com/Netflix/conductor	true	true	Missing: size	false
	3	Netflix	netflix.titus.control.plane	https://github.com/Netflix/titus-control-plane	true	true	Missing: size log_service ms_mention	false
	4	Netflix	netflix.genie	https://github.com/Netflix/genie	true	true	Missing: size ms_mention	false
	5	uber	uber.cadence	https://github.com/uber/cadence	true	true	Missing: size	false
	6	microservices-demo	sock.shop.app	https://github.com/microservices-demo/microservices-demo	true	true	Missing: size	false
	7	Open-IoT-Service-Platform	intel.iot.platform	https://github.com/Open-IoT-Service-Platform/platform-launcher	true	true	Missing: log_service ms_mention	false
	9	cfpb	cfgov.refresh	https://github.com/cfpb/consumerfinance.gov	true	true	Missing: size files	false

Figure D.2: List of Repositories Examples View



Listing

githubmm.herokuapp.com/metrics/generation/example

GHMM

Metrics Generation

Repo Examples / Process Metrics

Repo Examples

190

Processed Metrics

Success: 190

Error: 0

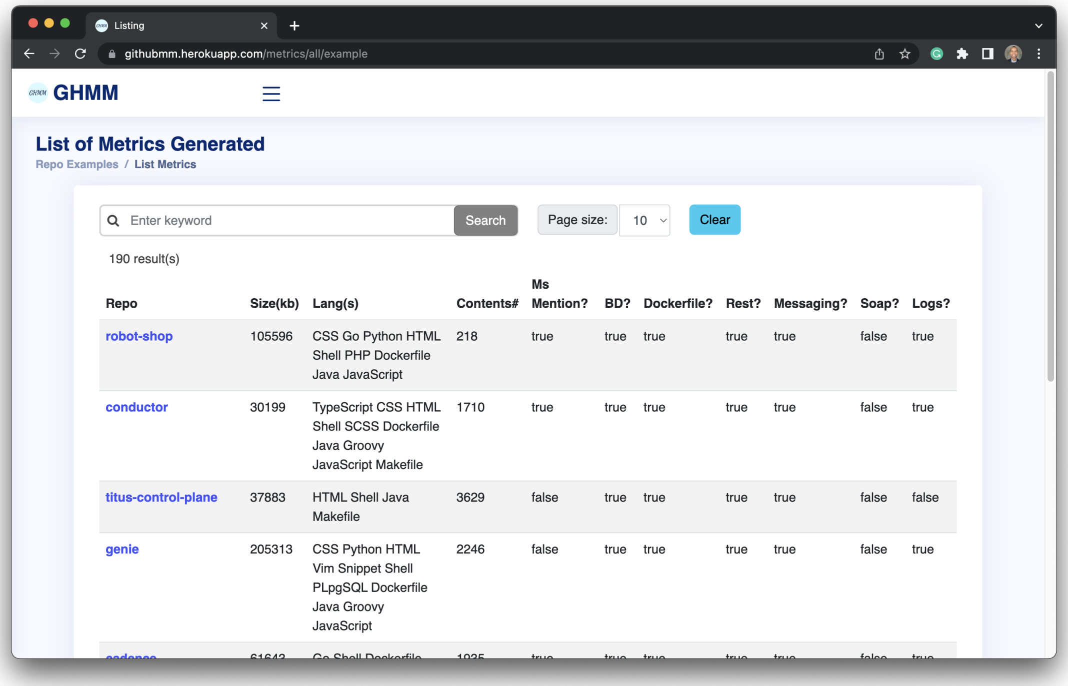
Metrics Status

Unprocessed: 0

Start

© 2023 GitHub Microservice Ming Tool

Figure D.3: Repo Examples Metrics Generation View


Listing

githubmm.herokuapp.com/metrics/all/example

GHMM

List of Metrics Generated

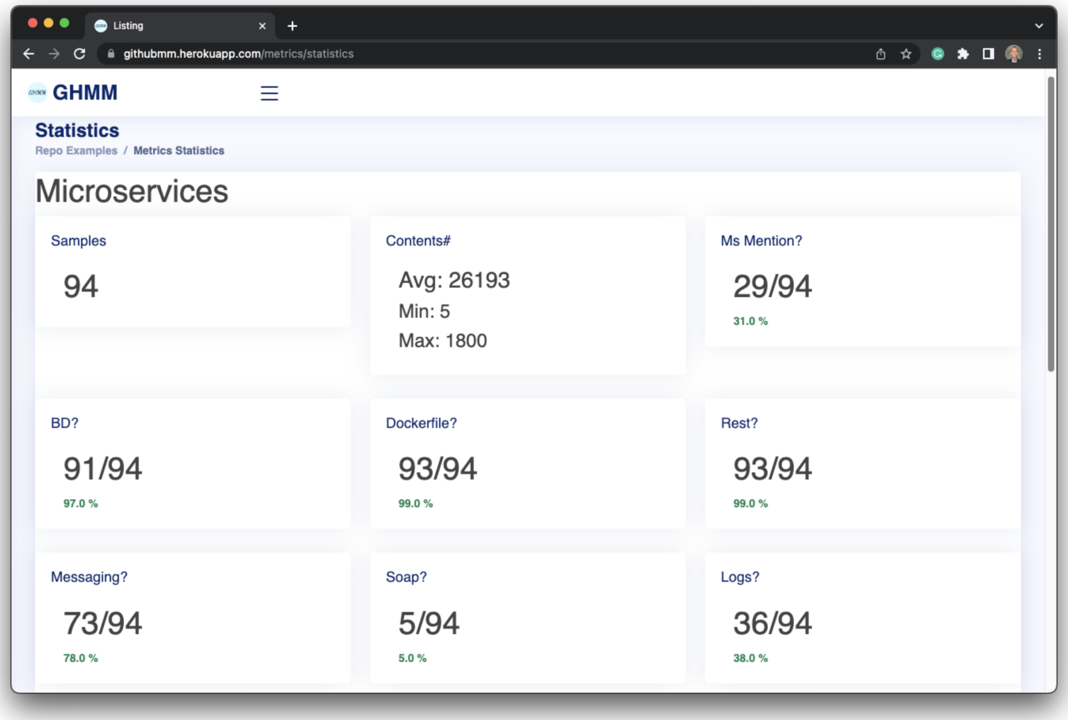
Repo Examples / List Metrics

Enter keyword | Search | Page size: 10 | Clear

190 result(s)

Repo	Size(kb)	Lang(s)	Contents#	Ms Mention?	BD?	Dockerfile?	Rest?	Messaging?	Soap?	Logs?
robot-shop	105596	CSS Go Python HTML Shell PHP Dockerfile Java JavaScript	218	true	true	true	true	true	false	true
conductor	30199	TypeScript CSS HTML Shell SCSS Dockerfile Java Groovy JavaScript Makefile	1710	true	true	true	true	true	false	true
titus-control-plane	37883	HTML Shell Java Makefile	3629	false	true	true	true	true	false	false
genie	205313	CSS Python HTML Vim Snippet Shell PLpgSQL Dockerfile Java Groovy JavaScript	2246	false	true	true	true	true	false	true

Figure D.4: Repo Examples List of Metrics Generated View


Listing

githubmm.herokuapp.com/metrics/statistics

GHMM

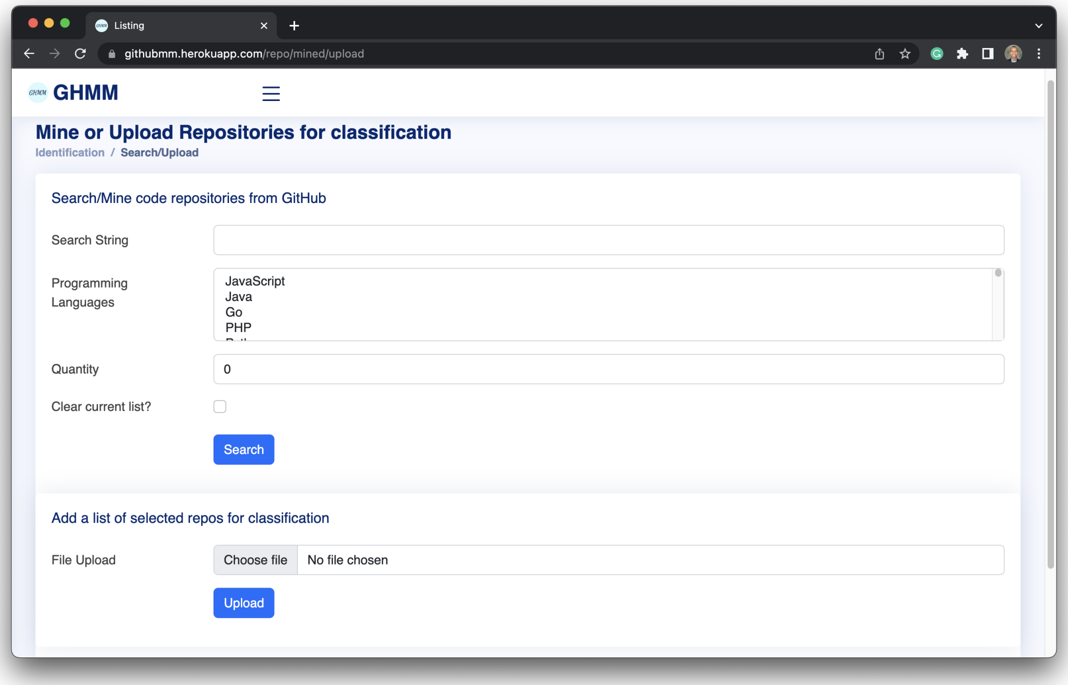
Statistics

Repo Examples / Metrics Statistics

Microservices

Samples 94	Contents# Avg: 26193 Min: 5 Max: 1800	Ms Mention? 29/94 31.0 %
BD? 91/94 97.0 %	Dockerfile? 93/94 99.0 %	Rest? 93/94 99.0 %
Messaging? 73/94 78.0 %	Soap? 5/94 5.0 %	Logs? 36/94 38.0 %

Figure D.5: Repo Examples Metrics Statistics View



Mine or Upload Repositories for classification

Identification / Search/Upload

Search/Mine code repositories from GitHub

Search String

Programming Languages: JavaScript, Java, Go, PHP

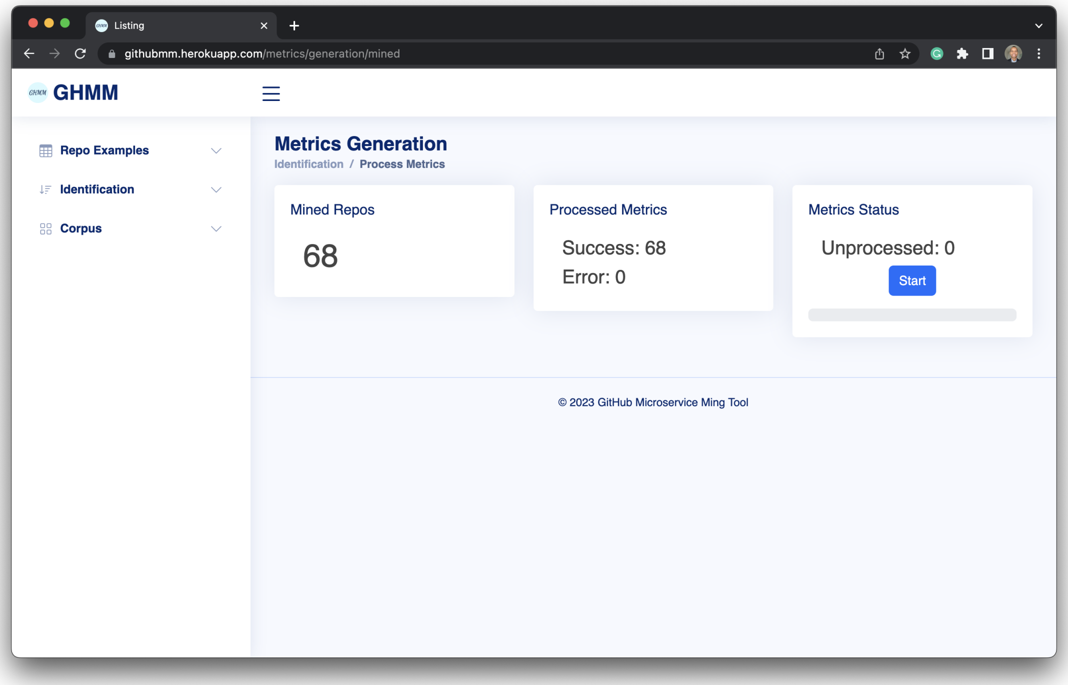
Quantity: 0

Clear current list?

Add a list of selected repos for classification

File Upload: Choose file — No file chosen

Figure D.6: Search/Mine code repositories from GitHub View



Metrics Generation

Identification / Process Metrics

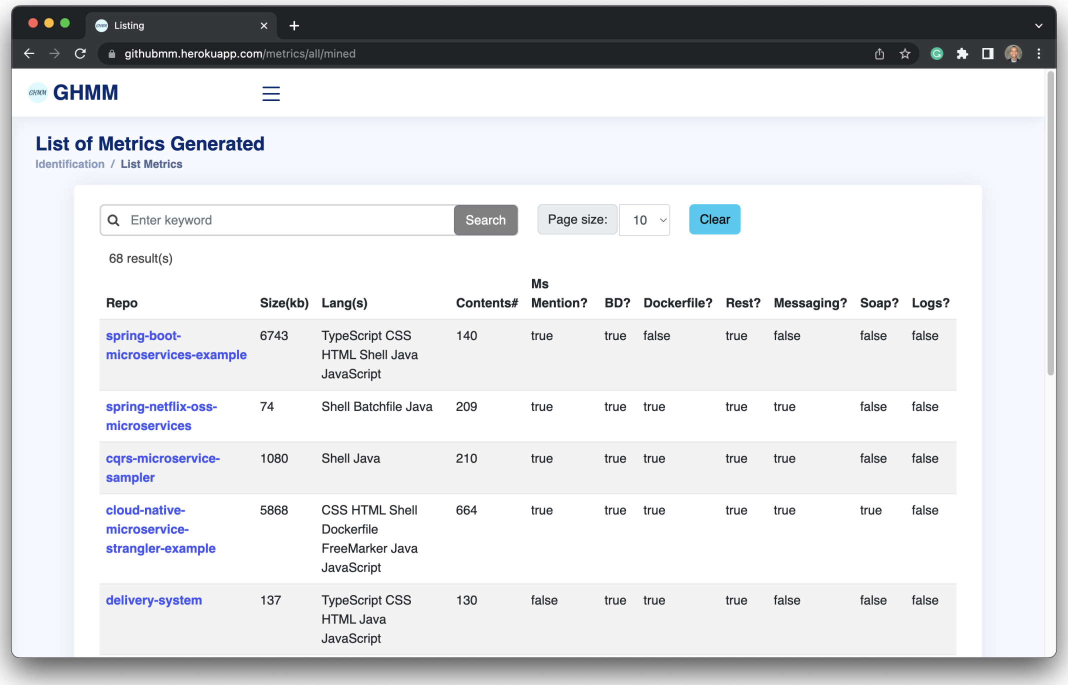
Mined Repos: 68

Processed Metrics: Success: 68, Error: 0

Metrics Status: Unprocessed: 0

© 2023 GitHub Microservice Ming Tool

Figure D.7: Process Mined Repos Metrics View



Listing

githubmm.herokuapp.com/metrics/all/mined

GHMM

List of Metrics Generated

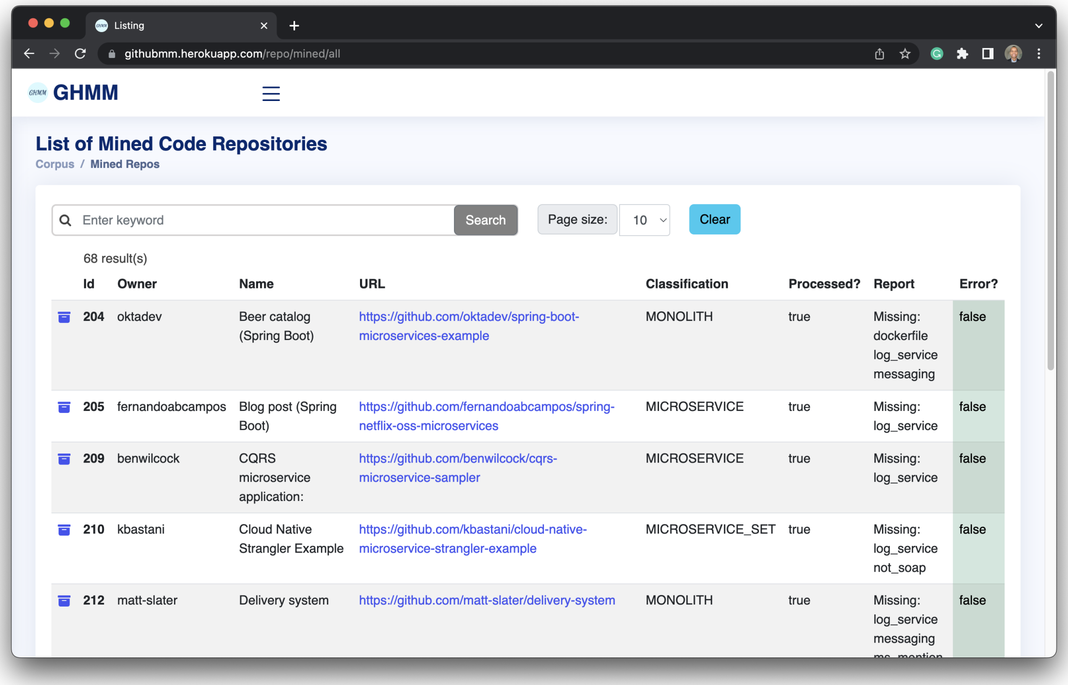
Identification / **List Metrics**

Enter keyword | Search | Page size: 10 | Clear

68 result(s)

Repo	Size(kb)	Lang(s)	Contents#	Ms Mention?	BD?	Dockerfile?	Rest?	Messaging?	Soap?	Logs?
spring-boot-microservices-example	6743	TypeScript CSS HTML Shell Java JavaScript	140	true	true	false	true	false	false	false
spring-netflix-oss-microservices	74	Shell Batchfile Java	209	true	true	true	true	true	false	false
cqrs-microservice-sampler	1080	Shell Java	210	true	true	true	true	true	false	false
cloud-native-microservice-strangler-example	5868	CSS HTML Shell Dockerfile FreeMarker Java JavaScript	664	true	true	true	true	true	true	false
delivery-system	137	TypeScript CSS HTML Java JavaScript	130	false	true	true	true	false	false	false

Figure D.8: List of Mined Repos Generated Metrics View



Listing

githubmm.herokuapp.com/repo/mined/all

GHMM

List of Mined Code Repositories

Corpus / **Mined Repos**

Enter keyword | Search | Page size: 10 | Clear

68 result(s)

	Id	Owner	Name	URL	Classification	Processed?	Report	Error?
	204	oktadev	Beer catalog (Spring Boot)	https://github.com/oktadev/spring-boot-microservices-example	MONOLITH	true	Missing: dockerfile log_service messaging	false
	205	fernandoabcampos	Blog post (Spring Boot)	https://github.com/fernandoabcampos/spring-netflix-oss-microservices	MICROSERVICE	true	Missing: log_service	false
	209	benwilcock	CQRS microservice application:	https://github.com/benwilcock/cqrs-microservice-sampler	MICROSERVICE	true	Missing: log_service	false
	210	kbastani	Cloud Native Strangler Example	https://github.com/kbastani/cloud-native-microservice-strangler-example	MICROSERVICE_SET	true	Missing: log_service not_soap	false
	212	matt-slater	Delivery system	https://github.com/matt-slater/delivery-system	MONOLITH	true	Missing: log_service messaging	false

Figure D.9: List of Mined Code Repositories View

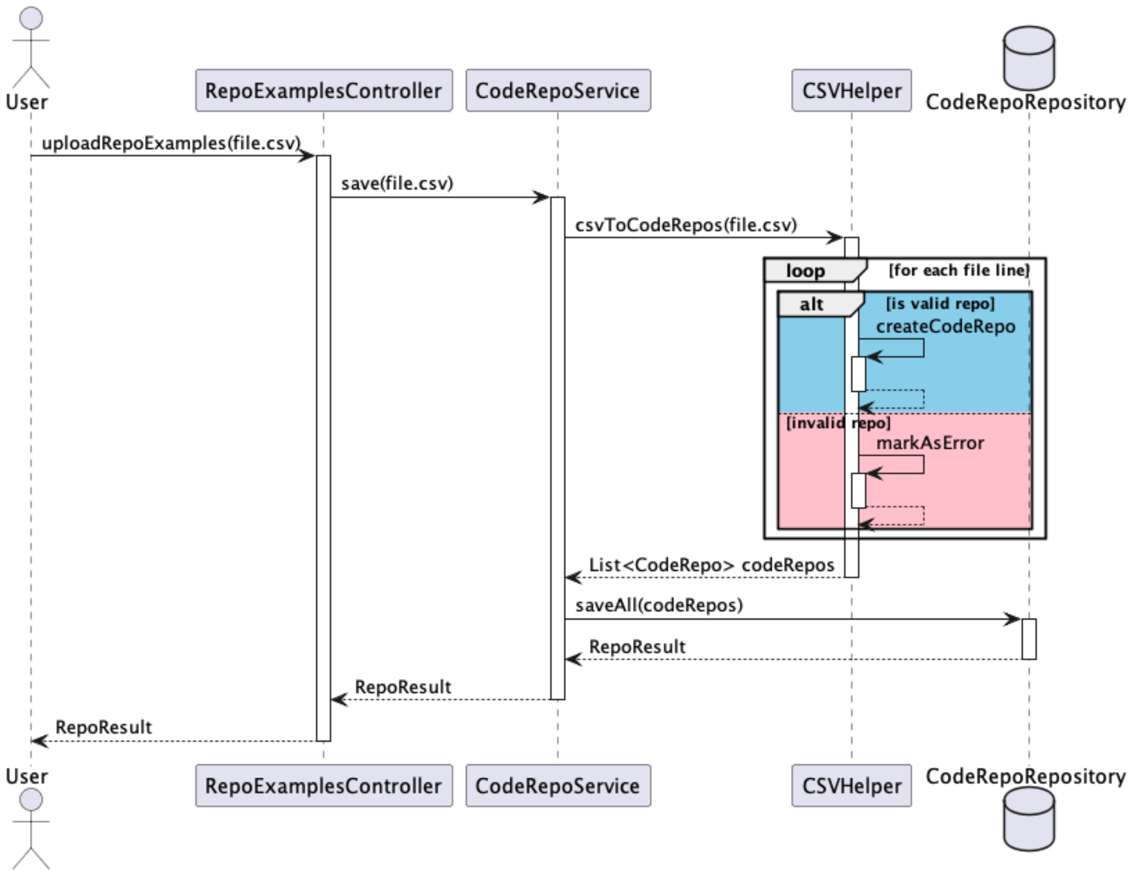


Figure D.10: Upload Repo Examples Diagram

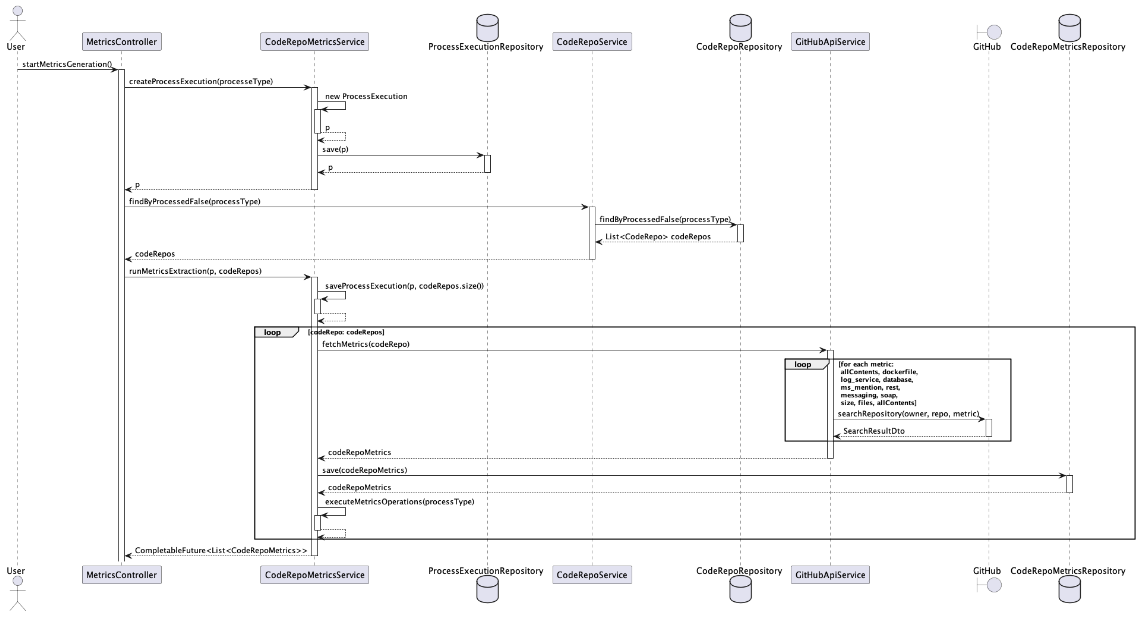

User
MetricsController
CodeRepoMetricsService
ProcessExecutionRepository
CodeRepoService
CodeRepoRepository
GitHubApiService
GitHub
CodeRepoMetricsRepository
startMetricsGeneration()
createProcessExecution(processeType)
new ProcessExecution
p
save(p)
p
p
findByProcessedFalse(processType)
findByProcessedFalse(processType)
List<CodeRepo> codeRepos
codeRepos
runMetricsExtraction(p, codeRepos)
saveProcessExecution(p, codeRepos.size())
loop [codeRepo: codeRepos]
fetchMetrics(codeRepo)
loop [for each metric: allContents, dockerfile, log_service, database, ms_mention, rest, messaging, soap, size, files, allContents]
searchRepository(owner, repo, metric)
SearchResultDto
codeRepoMetrics
save(codeRepoMetrics)
codeRepoMetrics
executeMetricsOperations(processType)
CompletableFuture<List<CodeRepoMetrics>>

Figure D.11: Start Metrics Generation Sequence Diagram

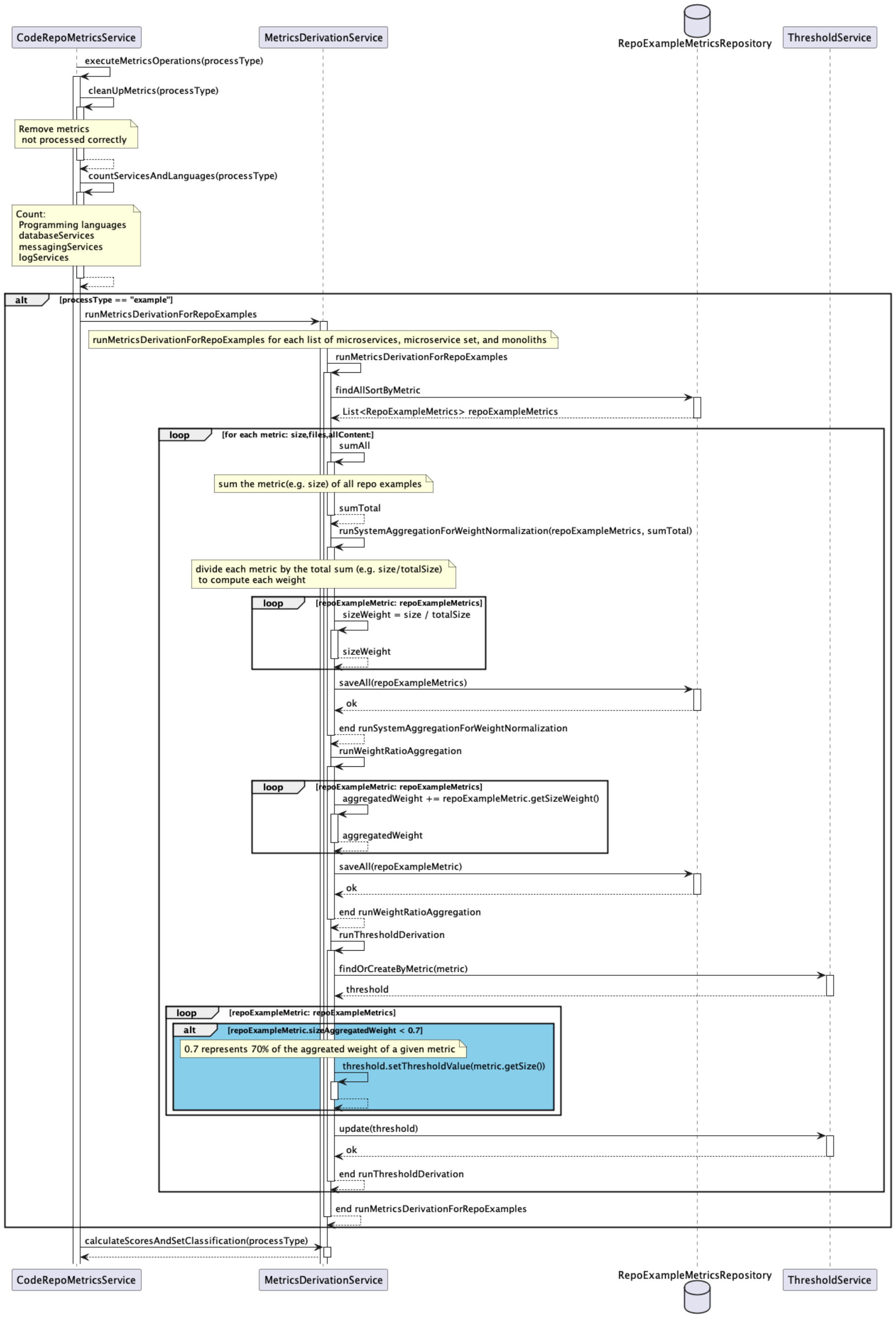

CodeRepoMetricsService
MetricsDerivationService
RepoExampleMetricsRepository
ThresholdService
executeMetricsOperations(processType)
cleanUpMetrics(processType)
Remove metrics
not processed correctly
countServicesAndLanguages(processType)
Count:
Programming languages
databaseServices
messagingServices
logServices
alt
[processType == "example"]
runMetricsDerivationForRepoExamples
runMetricsDerivationForRepoExamples for each list of microservices, microservice set, and monoliths
runMetricsDerivationForRepoExamples
findAllSortByMetric
List<RepoExampleMetrics> repoExampleMetrics
loop
[for each metric: size,files,allContent:]
sumAll
sum the metric(e.g. size) of all repo examples
sumTotal
runSystemAggregationForWeightNormalization(repoExampleMetrics, sumTotal)
divide each metric by the total sum (e.g. size/totalSize)
to compute each weight
loop
[repoExampleMetric: repoExampleMetrics]
sizeWeight = size / totalSize
sizeWeight
saveAll(repoExampleMetrics)
ok
end runSystemAggregationForWeightNormalization
runWeightRatioAggregation
loop
[repoExampleMetric: repoExampleMetrics]
aggregatedWeight += repoExampleMetric.getSizeWeight()
aggregatedWeight
saveAll(repoExampleMetric)
ok
end runWeightRatioAggregation
runThresholdDerivation
findOrCreateByMetric(metric)
threshold
loop
[repoExampleMetric: repoExampleMetrics]
alt
[repoExampleMetric.sizeAggregatedWeight < 0.7]
0.7 represents 70% of the aggreated weight of a given metric
threshold.setThresholdValue(metric.getSize())
update(threshold)
ok
end runThresholdDerivation
end runMetricsDerivationForRepoExamples
calculateScoresAndSetClassification(processType)
CodeRepoMetricsService
MetricsDerivationService
RepoExampleMetricsRepository
ThresholdService

Figure D.12: Execute Metrics Operations Sequence Diagram