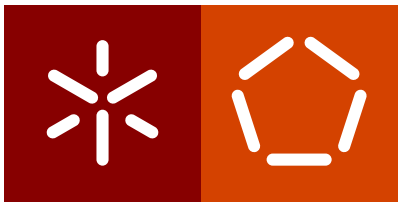
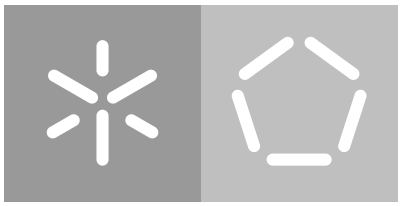


**Universidade do Minho**
Escola de Engenharia
Departamento de Informática

Miguel António Ferrão Brito

# Identification of Microservices from Monolithic Applications through Topic Modelling

January 2021



**Universidade do Minho**
Escola de Engenharia
Departamento de Informática

Miguel António Ferrão Brito

# Identification of Microservices from Monolithic Applications through Topic Modelling

Master dissertation
Master in Informatics Engineering

Dissertation supervised by
**Prof. Dr. Jácome Cunha**
**Prof. Dr. João Saraiva**

January 2021

## AUTHOR COPYRIGHTS AND TERMS OF USAGE BY THIRD PARTIES

## STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

Miguel Brito

Assinado por : **Miguel António Ferrão Brito**
Num. de Identificação: BI15155110
Data: 2021.01.26 19:58:35 +0000

# ACKNOWLEDGEMENTS

I would like to thank my adviser Dr. Jácome Cunha and co-adviser Dr. João Saraiva, for their continued and consistent support and guidance through this enthusiastic experience over the last year.

A special thanks to Wuxia Jin from Xi'an Jiantong University for providing some of the metrics implementation and for helping out on solving some unclear concepts on the remaining metrics.

Last but not least, a special thanks to my family and friends for supporting me through this thesis, as well as my entire education that ultimately lead to this work.

## ABSTRACT

Microservices emerged as one of the most popular architectural patterns in the recent years given the increased need to scale, grow and flexibilize software projects accompanied by the growth in cloud computing and DevOps. Many software applications are being submitted to a process of migration from its monolithic architecture to a more modular, scalable and flexible architecture of microservices. This process is slow and, depending on the project's complexity, it may take months or even years to complete.

This dissertation proposes a new approach on microservices identification by resorting to topic modelling in order to identify services according to domain terms. This approach in combination with clustering techniques produces a set of services based on the original software. The proposed methodology is implemented as an open-source tool for exploration of monolithic architectures and identification of microservices. An extensive quantitative analysis using the state of the art metrics on independence of functionality and modularity of services was conducted on 200 open-source projects collected from GitHub. Cohesion at message and domain level metrics showed medians of roughly 0.6. Interfaces per service exhibited a median of 1.5 with a compact interquartile range. Structural and conceptual modularity revealed medians of 0.2 and 0.4 respectively. Further analysis to understand if the methodology works better for smaller/larger projects revealed an overall stability and similar performance across metrics.

Our first results are positive demonstrating beneficial identification of services due to overall metrics' results.



**Keywords:** microservice architecture, monolithic decomposition, topic modelling, software clustering

# RESUMO

Os microserviços emergiram como um dos padrões arquiteturais mais populares na atualidade dado o aumento da necessidade em escalar, crescer e flexibilizar projetos de *software*, acompanhados da crescente da computação na *cloud* e *DevOps*. Muitas aplicações estão a ser submetidas a processos de migração de uma arquitetura monolítica para uma arquitetura mais modular, escalável e flexivel de microserviços. Este processo de migração é lento, e dependendo da complexidade do projeto, poderá levar vários meses ou mesmo anos a completar.

Esta dissertação propõe uma nova abordagem na identificação de microserviços recorrendo a modelação de tópicos de forma a identificar serviços de acordo com termos de domínio de um projeto de *software*. Esta abordagem em combinação com técnicas de *clustering* produz um conjunto de serviços baseado no projeto de *software* original. A metodologia proposta é implementada como uma ferramenta *open-source* para exploração de arquiteturas monolíticas e identificação de microserviços. Uma análise quantitativa extensa recorrendo a métricas de independência de funcionalidade e modularidade de serviços foi conduzida em 200 aplicações *open-source* recolhidas do *GitHub*. Métricas de coesão ao nível da mensagem e domínio revelaram medianas em torno de 0.6. Interfaces por serviço demonstraram uma mediana de 1.5 com um intervalo interquartil compacto. Métricas de modularidade estrutural e conceptual revelaram medianas de 0.2 e 0.4 respetivamente. Uma análise mais aprofundada para tentar perceber se a metodologia funciona melhor para projetos de diferentes dimensões/características revelaram uma estabilidade geral do funcionamento do método.

Os primeiros resultados são positivos demonstrando identificações de serviços benéficos tendo em conta que os valores das métricas são de uma forma global positivos e promissores.

**Palavras-chave:** arquitetura de microserviços, decomposição de monólitos, modelação de tópicos, *clustering* de *software*

# CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# 1

# INTRODUCTION

One of the main problems during the development of a large scale application, arises during the increasing difficulty in addition of new functionality and maintenance of the current one given the increased complexity of the software project (Visser, 2016). The increase in complexity typically results in functionality being spread across different sections of the application, making the process of identification and correction of *bugs* harder. In order to deal with such problem in monolithic architectures abstractions are created with the goal to ensure greater cohesion between similar functionally, striving to follow the Single Responsibility Principle (SRP) - merge things that change for the same reason, and segregate things that change for different reasons (Newman, 2015). However, abstractions themselves with evolution increase in degree of complexity and become harder to reason about. At this stage, advantages of a monolithic architecture are inferior to its disadvantages (Chen et al., 2017).

The decomposition of systems into modules began to be systematised and debated by Parnas (1972) long before the massification of software systems. Parnas intended to demonstrate that the efficiency of the modularisation of a system depends on the criteria used, contrary to the pure fragmentation of systems into small modules.

The relevance of the functional decomposition of systems initially mentioned by Parnas was reinforced by the demand to distribute complex systems through network infrastructures such as web services and remote objects resulting from efforts to deal with systems of greater dimension and complexity (Kamimura et al., 2018).

From the difficulty to manage the inherent complexity due to the evolution of large scale projects, greater difficulty in applying changes, maintaining the project, fixing *bugs* and unnecessarily big deployments the microservice architecture (MA) arised: small autonomous and cohesive services, with the goal to ensure and maintaining the Single Responsibility Principle. Each service is in charge of a small set of tasks and responsible for doing them well (Newman, 2015). Microservices allow greater heterogeneity of technologies granting the ability for each service to be programmed in a different language as long as there is a common technology to communicate. Microservices allow for better resilience, given the existence of more processes reducing the possibility of a localised system failure to crash the entire system. Microservices can be scaled more efficiently focusing the resources

on specific services that receive the most frequent and taxing tasks. Apart from the more technical aspects, microservices provide better agility and ease to the process of developing software since the services can be distributed across multiple teams and a higher level of isolation is ensured. Overall, microservices provide answers to multiple challenges that arise throughout the evolution of a software system into more complex versions.

Microservices have/are been quickly adopted to develop new software. There are, however, many legacy software systems that were developed before MA were introduced, but that can benefit from the agility and flexibility MA software development offers (Newman, 2015). This is particularly relevant when we consider the usual maintenance and evolution processes required in a modern software life-cycle. In order to benefit from MA such legacy software systems - that we call monolithic software systems - need to be refactored into a semantically equivalent microservice-based one. Performing such refactoring manually is both complex/time consuming and prone to errors, and its quality is often strongly linked to the experience and knowledge of the specialist leading the said refactoring (Kamimura et al., 2018; Pahl and Jamshidi, 2016). The refactoring is typically done following a Strangler pattern, that is, incrementally migrating and replacing modules of the system into a new architecture until the migrated systems overcome the old system.

The transition process from a monolithic application into a more modular microservices architecture is expensive, laborious and filled with unique challenges according to each system (Kazanavičius and Mažeika, 2019), which might oblige to put aside current resources (i.e. software development teams) to work on the migration in parallel or completely stop the development of further functionalities in more extreme cases. Furthermore, the process is limited by the knowledge and experience of the expert carrying out such migration (Pahl and Jamshidi, 2016), typically resulting in the introduction of *anti-patterns* during the decomposition of the system (Fritzsch et al., 2019).

Given the considerable investments needed to migrate a monolith to a MA, there is a strong necessity for methodologies, processes and tools that reduce the time and amount of work needed and make the whole process more efficient and less error prone.

The identification of microservices in legacy monolithic systems is still an open problem with just a few proposed approaches (Mazlami et al., 2017; Kamimura et al., 2018; Jin et al., 2018, 2019; Gysel et al., 2016; Chen et al., 2017). Most of these proposals, however, use their own quality metrics to assess the quality of the achieved transformation. Moreover, they are not supported by a tool that can be automatically applied to a legacy system, and as consequence the approaches are *validated* in a small (less than ten) number of monolithic systems. The exception is Jin et al. (2019), which uses a dynamic analysis approach: it runs the legacy software system, to infer the microservices so to migrate it into a MA one. Although it provides good results, it has the disadvantage of requiring inputs or test cases to

properly execute the system. Unfortunately, this is not the case in most legacy systems (Petrić et al., 2018).

In this dissertation we propose a static analysis technique to identify microservices in a legacy software system based on topic models. Topic models (Kherwa and Bansal, 2018) allow to mine a set of topics across a collection of documents. Thus, by applying topic modelling to a monolithic software system we identify the systems' topics, which correspond to domain terms, and represent the microservices implemented by that legacy systems. Such topic models are inferred from the collection of lexical information in the source code, namely method declarations, method invocations, variables and class names, etc. For instance, considering a software for managing stocks, one would expect to find terms related to products, suppliers, etc. The collection of components with related names are likely part of the same microservice. We explore in detail this idea in this dissertation. The mined topics can then be combined into the structural information of the source code into a graph. Such graph is then clustered in order to identify microservices.

To assess the quality of the identified microservices we use the MA metrics proposed in Jin et al. (2019) that evaluate the independence of functionality and modularity of microservices. Furthermore, the proposed methodology is implemented as an open-source tool[1] for exploration of monolithic applications. This tool was validated by performing a quantitative analysis study on 200 open-source monolithic software systems collected from GitHub. The results obtained concerning MA metrics' are positive. Regarding independence of functionality, CHM and CHD presented a median of roughly 0.6; IFN presented a median of 1.5. Modularity metrics of SMQ and CMQ demonstrated median values of 0.2 and 0.4 respectively. Overall the results are positive, showing relevant proposals of microservices and a promising first step to refactor monolithic applications.

The remainder of this dissertation is structured as follows: Section 2 introduces the reader to the concepts of monolithic and microservices architectures; Section 3 describes the current state of the art on microservice identification and the overall process of migration; Section 4 presents the methodology we devised for microservices identification; Section 5 describes the implementation of the proposed methodology into a prototype; Section 6 discusses a case study and a walkthrough of the methodology applied to an example project; Section 7 describes the steps taken to quantitatively analyse our methodology and concludes with our results; Finally, Section 8 ends with some conclusions and future work.

1 https://github.com/miguelfbrito/microservice-identification

# 2

# BACKGROUND

In order to identify microservices from a monolithic architecture, it is necessary to firstly understand the core ideas behind each architecture, their benefits and challenges inherited by the way they are structured and how their components interact and its impact on the software development process.

## 2.1 MONOLITHS

Many of the web applications we interact with nowadays were built according to a monolithic architecture.

This architectural style is characterised by an application composed of all the core logic related to the domain of the problem contained in a single process (Newman, 2015). This type of architecture is typically responsible for handling all the functionality related to persistence and manipulation of data according to the business domain. The outermost layer, may also be responsible for serving the interface for interaction with the user, or, exposing its functionality through an API.

Although described as an architectural style that is not very modular compared to a microservice architecture, its modularity is achieved using underlying programming techniques, such as abstractions, implementations and extensions, thus allowing the much needed modularity to handle more complex problems and adding new features.

Inherent to the way this architecture is structured, some advantages of its usage can be identified: rapid and simple development at the beginning of the project; simplicity in the process of deployment given the existence of a single component and simpler to scale horizontally.

The mentioned advantages are especially appealing when starting a project, or it is a short duration or low complexity project, however, for other situations long-term thinking is essential, especially since it is estimated that about 50 % of the development cost of software is allocated to maintenance (Alija, 2017). Representing such a significant percentage of the cost of the project, keeping the project open to new functionality, *bug* fixes and technical debt under control is essential, and also the main challenges developers face given the continued

growth in project complexity and size. The deployment, although simpler in procedural terms, updating deploys to new versions requires the *re-deploy* of the application completely. The fact that the deployment is done in relation to a single process, which, in case of failure in one of the application modules, can propagate the failure throughout the process will increase unreliability and decrease resilience to failures.

Overall, monolithic architectures accompanied by good design practices at a structural level, which seek to reduce coupling and increase cohesion are sufficient to deal with a wide range of standard solutions. Most of the disadvantages are associated with the high granularity at which the deploy process is conducted, the way they deal with failures and the lack of flexibility to make changes in advanced stages of the project.

## 2.2 MICROSERVICES

Microservices are described by Fowler and Lewis (2014) as an application architecture composed of a set of small services, each one being independent and self-contained, executed in its process, with the objective of performing a small set of tasks well. With its isolation, there is a need for a communication layer. Typically, each of the services exposes its API, which will receive requests and respond according to the logic defined in the service.

Inspired by the *Service Oriented Architecture (SOA)*, which aims to deal with some of the difficulties present in managing monolithic architectures of high complexity by fragmenting the application in various services, the same philosophy of application fragmentation was continued. Although in a finer granularity, with the intent to share as little as possible between microservices. The principle of single responsibility is one of the main focuses in defining the segregation of services.

### 2.2.1 *Advantages*

The advantages of micro-services are varied compared to monolithic architectures. Several of the advantages associated with micro-services are the result of the adopted characteristics of distributed systems and by taking the philosophy of service-oriented architectures to a greater extreme.

#### *Heterogeneity of technologies*

Communication between micro-services is typically done over HTTP using a technology totally independent of the language, one of the most popular being the *REST (Representational State Transfer)* APIs that serve information generally by *JSON (JavaScript Object Notation* or *XML (eXtensible Markup Language)*.

Thereby, the existence of multiple languages across application's technological stack is possible. If a particular service needs performance improvements, the change to a technology that allows working with lower level and optimise the functionality of a specific service, will have no impact on the rest of the system at a functional level, as long as the interfaces exposed keep the same contract and functionality.

*Flexibility in deploys*

No matter how small the change made to the application, a monolithic system requires the deployment to encompass the entire application. To decrease the time of *downtime* the *deploys* are heavier since they cannot be as frequent. However, accumulating changes made to a single deploy will bring a higher risk as it will be more error prone. Several changes also make it difficult to identify any problems as they are spread across several application's components.

In contrast, micro-services seek to maintain a close relationship with *DevOps* culture and the philosophies of *Continuous Integration (CI)* and *Continuous Delivery (CD)*, philosophies that streamline the process of *deployment*, by integrating automated pipelines responsible for performing pre-defined code verification and validation tasks, and consequently performing deployment to production. Following this philosophy, *deploys* go from infrequent, containing several features spread over several modules, to very frequent, with each *release* strictly associated with its service. Frequent *deploys* and specific to a service, reduce the risk of problems, decrease downtime and make it easier to identify the source of problems that arrive in production (Newman, 2015).

Frequent *deploys* allow for a closer relationship with cycles of agile methodologies, which place great importance on constant interaction and feedback with customers and end users.

*Scalability*

When the need to scale a monolithic application by necessity presented by a given module arises, the entire application must also be scaled. The level of granularity to scale an application of this kind is quite high, and as such, quite inefficient, resulting in a huge waste of computational resources and extra costs. In contrast, micro-services allow for considerably fine-grained scalability, providing scalability only for services that are overloaded. Thus, there will be greater efficiency of resources in use and cost reduction. The fine granularity of services also allows for optimisations at an architectural level

*Resilient and fault-tolerant*

Dealing with failures, especially when said system is a large and complex system is inevitable. The distributed systems nature of micro-services, provide the ability to isolate

and limit failures to certain sections of the application, contrary to what would happen in a monolithic application, where the failure in a functionality can result in the complete failure of the entire application. Transitioning from a monolithic architecture to a MA does not automatically solve all these problems by itself (Behara, 2018), in an initial phase, micro-services will be particularly vulnerable to failures, given the addition of network communication being more prone to failures than pure memory communication. However, increasing from a single point of failure to several is the key feature in making a system based on distributed systems resilient and fault tolerant.

Using standards such as the *Bulkhead* pattern, an analogy to ships' waterproofing systems, which allow water to enter the hull to be contained and isolated by compartments, it is also possible to isolate faults and keep the rest of the service functional, using groups of services that will be self-contained, without affecting the availability of the remaining services (Bulkhead Pattern).

*Organisational structure*

The work carried out by large teams on complex projects is in itself a complex and difficult task to manage. Usually, teams are kept in small groups of elements to facilitate this management. This fragmentation into teams is aligned in a natural way with what is expected from micro-services (Newman, 2015), that is, the work carried out by a team will typically be associated with a specific segment that will be represented by a set of micro-services. According to Conway (1967), organisations that design systems, produce systems that reflect the organisational structure of the organisation itself. The occurrence of team segregation according to the business' domain and its functionality causes the number of communication channels between the different teams to be optimised and reduced and the impact of the organisational structure on the software is also reduced.

### 2.2.2 *Challenges*

Several of the challenges of using microservices are actually challenges inherent in the use of distributed systems. These must be taken into account in the design of a microservice architecture.

The segregation of the various services to different processes requires communication over the network, contrary to what happens in monoliths, where communication between components is carried out in memory. Communications using *HTTP* in *REST APIs* or Advanced Message Queuing Protocol (AMQP) are typically used (Wenzel et al., 2020). This type of communication adds a layer of overhead not found in monolithic applications, requiring the analysis of its use in components that need to have very short response times and are primarily focused on performing operations quickly.

In an initial phase and in simpler systems, access to services will be made by consulting configuration files composed by the respective locations of the services, allowing for simpler and direct access to the desired service, however, in order to deal with failures and automatically scale the services, the usage of dynamic assignments is imperative, adding another level of complexity. Typically, the said layer is composed of a *discovery pattern* in order to identify available services to where the communication can be established. The need to add this pattern contributes to increasing the complexity of the infrastructure due to the need for its correct configuration and the addition of another component that has to be monitored and taken into account when failures in the system are identified.

The identification of *bugs* and failures by performing high-level tests, such as *end-to-end* tests and integration tests, becomes more complex, since it is necessary to create *stubs* of the remaining services to ensure isolation of the fraction that is tested against the rest of the system (Newman, 2015).

Similarly to the testing of the system as a whole, performing debug of the system becomes more difficult, since there is a high amount of *logs* created by the different services making it tougher to isolate failures being propagated across services (Nemer, 2019). The use of *log aggregators* and other systems is essential in trying to mitigate this problem.

# 3

# STATE OF THE ART

Decomposition of systems into modules has a topic started to be systematised and debated by Parnas (1972) long before the massification of software systems. In his work, Parnas intends to demonstrate that the efficiency of the modularisation of a system depends on the criterion used, with fragmentation in small modules not being the path to success, but rather the way in which the segregation criterion is chosen.

The relevance of the functional decomposition of systems initially mentioned by Parnas, was reinforced by the large necessity to distribute complex systems over network infrastructures such as *web services* and remote objects resulting from efforts to deal with larger and more complex systems (Kamimura et al., 2018) and the adaptation to cloud infrastructure.

The main challenges of said decomposition are mainly the manual and tedious work accompanied by the difficulty in identifying the functional units given the need for a detailed analysis of the various dimensions of the software architecture. The quality of the final result is often strongly linked to the experience and knowledge of the specialist who performs said decomposition (Kamimura et al., 2018; Pahl and Jamshidi, 2016). It is common for such migrations to introduce common anti-patterns of microservices design (Fritzsch et al., 2019).

## 3.1 MANUAL MIGRATION FROM MONOLITHS TO MICRO-SERVICES

In this section, an analysis of the bibliography is performed regarding the migration processes from monolithic architectures to microservices carried out manually, by specialists, in industrial scale projects. An attempt is made to identify knowledge about which elements of the transition process can be identified and systematised in order to collect useful information with the goal to automate specific sections of the process.

With the rising popularity in usage of microservices architectures as an alternative to some of the problems inherent in monolithic architectures, the need to formalise such migration and the identification of methodologies and processes to improve and increase its efficiency raises as well.

Many software architects responsible to provide design solutions to complex software systems see the transition of their legacy application to microservices as an advantage in

order to improve and make the software development process more efficient. However, migrating architectures is a complex process, very dependent of the domain of the problem and the specificities of said software, being necessary many iterations until said migration could be deemed as finished (Dehghani, 2018).

In order to combat the *ad-hoc* component carried out so far in the migration process, Balalaie et al. (2018) collected a set of design patterns identified empirically by analysing migration processes of industrial level applications. Their work seeks to make an analysis that encompasses the entire migration process, from the identification of the current architecture and its decomposition in different services, to processes related to *DevOps* and migration to the cloud by processes of *CI*, *CD*, introduction of *service discovery*, *load balancing*, orchestration of *containers*, *etc*.

Since a rigid, *one-size-fits-all* style would never be functional given the high uniqueness resulting from complex and domain specific solutions, and the variety of how software systems are composed, Balalaie et al. (2018) use an approach called *Situational Method Engineering (SME)* (Henderson-Sellers et al., 2014). In a concise way, a method can be designed according to the specific situation through a set of migration patterns (Balalaie et al., 2018). Each migration pattern is used according a specific phase of the process and deals with things such as: continuous integration, architecture recovery, monolithic decomposition, service registry, load balancers, *etc*. In this dissertation we will be particularly focused on the monolithic decomposition phase.

From his empirical study and the application of *SME*, a repository of 16 patterns accompanied by a migration plan was proposed (illustrated in Figure 8) which seeks throughout the process to deal with possible situational dependencies between the various steps.

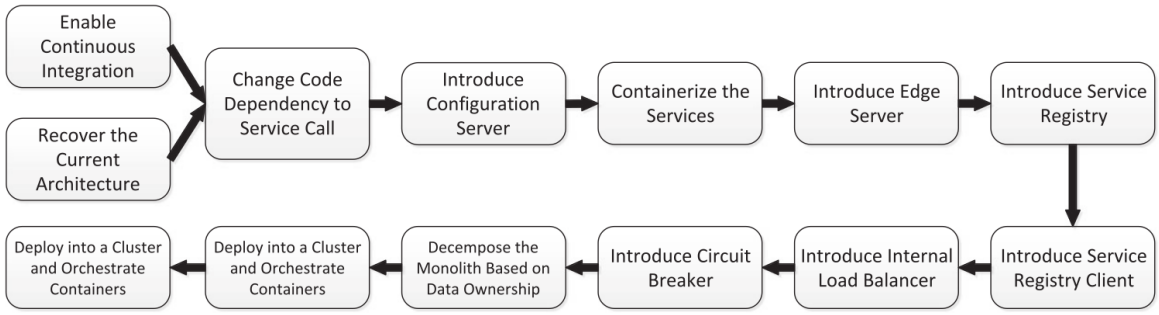



Figure 1: Proposed migration plan

Source: Balalaie et al. (2018)

In interaction with three companies that carried out the migration of their large-scale monolith, they found that of the 16 patterns identified, 85 % were used in the migration to micro-services.

With a similar objective of investigating and systematising the migration process, Fritzsch et al. (2019) carried out a qualitative study regarding the intentions, strategies and challenges in the migration process of 14 large-scale systems with expected migration periods between 1.5 and 3 years) from different domains (public transport systems, hotel management, retail, etc.), using detailed interviews from 16 experienced professionals from 10 different companies.

From the carried out analysis, they conclude that the difficulty in dealing with the maintenance of the systems and the need for more efficient scalability continues to be the crucial factors in originating the transition to microservices. One of the main challenges reported in the migration process was how to decompose the systems. Seven participants chose a functional decomposition approach, described by Fowler and Lewis (2014), of which, four resorted to *domain-driven design* techniques. The remaining participants opted for non-systematic approaches (Fritzsch et al., 2019).

None of the participants in the study considered or was aware of the existence of auxiliary techniques that could be used in the decomposition of their systems (eg. static code analysis, *execution traces* analysis) (Fritzsch et al., 2019). Six of the participants made proposals for micro-services with anti-patterns (*Wrong Cuts, Shared Persistency, Service Intimacy*, patterns identified by Taibi et al. (2018)), resulting from inappropriate decompositions.

Despite the massive adaptation of microservice architectures, there are specialists, such as Fowler (2015), who argue that a software system will be more successful if it is initially built according to a monolithic architecture and later migrated to microservices once complexity starts to gain relevance and its properties make more sense. Thus, an initial monolithic architecture allows to better explore the complexity of the system and the limits of the components (Fowler, 2015).

There is already a high need to formalise this migration from the large amount of applications being migrated to microservices, ideas like the one presented by *Fowler* reinforce the relevance of the continued need for research and formalisation of methods that make the migration from monolithic architectures to micro-services more efficient and effective.

The state of the art of the most promising ares of knowledge regarding the process of identification of microservices is described below.

## 3.2 Source-code oriented solutions

Considering that source code represents the functionality and the domain of the problem at the most detailed level and from it a large amount of information can be extracted, this is one of the most evident approaches as basis of information collection in order to infer microservices. In the following sections approaches resorting to source-code as input are identified and analysed.

### 3.2.1 *Static analysis solutions*

Static analysis is widely used in the software testing areas regarding bugs identification, code complexity analysis, among others. This technique is also promising in this area given the need to work on the source code and perform its manipulation.

With the aim of a static approach, Mazlami et al. (2017) proposes in their work 3 formal coupling strategies later processed according to *clustering* algorithms. The proposed strategies are based on the combination of the source code of the monolithic application and its meta-data, collected from the *Git* version system.

Of the coupling strategies proposed by the authors, logical coupling, is based on the premise that changes made to a monolith are made only to a specific module, in this way, the history of changes is analysed taking into account that the classes that are changed together should also be together in a micro-service. Semantic coupling is based on grouping classes that have code on the same things, that is, by calculating the similarity of domain terms between two classes, a measure can be obtained to identify how semantically close the classes are. Finally, coupling by contribution, based on the law of Conway (1967), which indicates that the structure of software represents the structure of the organisation, and as such, specific modules that receive changes from a single set of elements of the team must remain in the same micro-service. Each of the coupling techniques is used in the initial extraction process, and a dependency graph is then built to be used for the application of clustering algorithms.

The performed clustering operates only on the *minimum spanning tree* [1] to ensure that each removal of an edge results in an increase in the number of connected components. However, this operation introduces the disadvantage of leaving out some of the connections initially obtained by the coupling method, limiting the service proposal (Mazlami et al., 2017). Another limiting factor of this work is expressed by the use of classes as the atomic unit of identification and extraction of micro-services, making it impossible to segregate components of a class for several micro-services. Another apparent fault on their method is identified on the work of Jin et al. (2019), where they observed a high occurrence of services containing a large chunk of the application which goes against microservices' core ideas.

Kamimura et al. (2018) also resort to *c*lustering techniques to carry out their identification and proposal of microservices. The extraction of the classes is initiated by the identification of the *endpoints* of the API of the application in question, using annotations of specific frameworks (eg. *@Controller* in *Spring*), which identify the endpoints exposed in *REST APIs*. To extract the remaining knowledge, annotations such as *@Entity* and *@Table* are used to identify the classes responsible for defining the persistence and data manipulation. The clustering method mentioned in this work has as one of the main objectives to deal with

1 *Subset* of the vertices of an undirected graph to which all vertices are connected by the lowest possible total weight of the edges, without the cycles.

omnipresent modules, that is, modules with a high number of relationships with other modules and present in a high number of modules. Proposed by Kobayashi et al. (2012), this strategy aims to address this problem by weakening the importance attributed to modules with this particularity, facilitating the process of clustering of the components (Kobayashi et al., 2013).

### 3.2.2 *Dynamic analysis solutions*

Dynamic analysis techniques emerged as an alternative to static analysis using the analysis of program execution (eg. logs) in order to obtain extra information regarding how the software behaves under user interaction.

According to Candela et al. (2016), techniques that process code analysis based on their syntactic relationships, using metrics such as coupling and cohesion or naming conventions, might not be sufficient for optimal identification, given that the code-level relationship may not be the same in terms of functionality (Jin et al., 2018). In order to deal with this gap, Jin et al. (2018) propose the use of execution traces collected during the execution of certain test cases created by the user, which according to Dit et al. (2013) might represent a closer relationship with the true functionality of a given software system. The method by them proposed, named *Functionality-oriented microservice extraction (FoME)*, is characterised by three essential steps. Firstly, a clustering is performed at the class level from the *execution traces* to obtain a basic skeleton of services. Then, given that many of the classes of this skeleton will be associated with too many services, it is necessary to perform a process of selection and cleaning. Generally, the option is to assign the class to the cluster on which it is most dependent, which is not the only alternative since extracting this class for a different service may in certain situations be an appropriate solution. In order to consider both approaches, the dependence between cluster and the class is calculated: if the class has a high level of dependency on a service, it is moved to that service; if the class is weakly dependent on the tested clusters, a new service (Jin et al., 2018) is created. Finally, service candidates identified by clusters are generated.

They later extend their work by proposing *Functionality-oriented Service Candidate Identification (FoSCI)* (Jin et al., 2019) resorting to a search-based functional atom grouping algorithm based on Non-dominated Sorting Genetic Algorithm-II (NSGA-II) using as optimisation objectives both intra and inter structural connectivity and inter and intra conceptual connectivity. Their work also results in the extension and proposition of new metrics for quantitative evaluation of proposed microservices.

In the application of those methods, the quality of the tests created is essential to ensure that there is a good coverage of system's functionality (Jin et al., 2018). Although initially referred to as an advantageous approach to static analysis, as stated by Candela et al. (2016),

the creation of "real" tests in an automated way that reproduce the behaviour of a user, and that provide good test coverage is complex and a process far from trivial.

## 3.3 MODEL-ORIENTED SOLUTIONS

The importance of models in the development of software systems and Model-driven-development (MDD) enhance the use of model-based approaches since similar to others, an analysis of the interactions between components of a system can be made, although at a different level of abstraction and detail.

Gysel et al. (2016) propose *ServiceCutter*, which follows a model-oriented approach, using artifacts such as domain models and use cases to extract a graph representation in order to later identify proposals of microservices. Weights are added to the graph's edges according to a set of criteria in order to identify clusters, and consequently good candidates for micro-services. In view of the good selection of criteria, a survey of 16 coupling criteria was carried out, according to an analysis of the literature and author's experience, for later combination with clustering techniques, **Figure 2**.

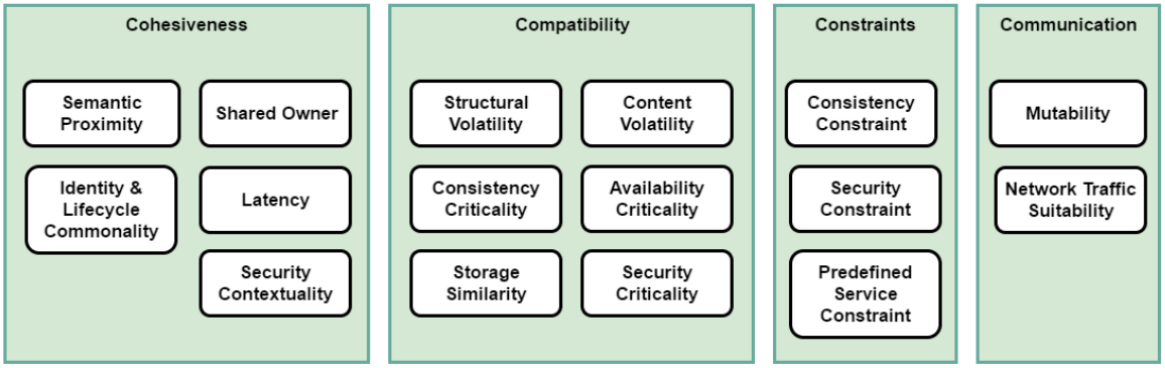



Figure 2: Coupling criteria proposed by Gysel et al. (2016)

Source: Gysel et al. (2016)

Gysel's methodology currently supports two clustering techniques: *Epidemic Label Propagation* proposed by Leung et al. (2009) and Girvan and Newman (2001) algorithm.

However, *ServiceCutter* does not have the capacity to extract the baseline information necessary for its operation from a *software* project, being highly dependent on the software artifacts provided by the user. This fact can be limiting in several ways. Firstly, considering that the documentation is often considered as scarce and outdated, especially in current Agile methodologies, which are based mainly on the user's feedback and not on documentation development. Second, due to the need of compatibility between the artifacts produced and the artifacts expected by *Service Cutter*.

Chen et al. (2017) also followed a model-oriented approach using *data-flow diagrams (DFD)*. Their approach is described in three phases: an analysis of business requirements is carried out by the engineers together with the users, to obtain the *data-flow diagram*; then, their algorithm combines the collected diagram with the same operations in a virtual abstract diagram; and finally, it uses this virtual diagram to extract proposals for micro-services. Chen et al. (2017) show a comparison with *Service Cutter* given the similarity in the model approach of both works, arguing that theirs is easier to operate, by using simpler diagrams; by not performing unnecessary decompositions leaving entities that make sense to be together in different services; and easier to understand by presenting the extracted services in a more intuitive way. This comparison was, however, carried out for a very limited case studies and may enhance the characteristics of one tool relative to another.

Although all these improvements presented relative to *Service Cutter*, this approach to models is limited by some crucial points. Firstly, the combination between the models and the operations is performed based on the names of the operations, forcing *DFD* to strictly follow the naming convention used in CRUD operations (Chen et al., 2017). Secondly, this process mostly performs the service proposal using data operations carried out in the business logic, however, if there is a need to deal with other capabilities that are not strictly linked to the data and its operations, the service proposals will be of higher granularity. Finally, the comparison made with *Service Cutter* using very simple test cases limits the validation regarding the ability to handle the extraction of services in large-scale applications.

Similarly to *Service Cutter*, the proposal made by *DFD* is also highly dependent on the quality of the artifacts provided by its user. Artifacts may need to be converted to the appropriate formats supported by the tools; process that is not automated and can result in degradation of the quality of the artifacts. Overall, there's a huge amount of work placed on the users in order to prepare the execution of the proposed tool.

## 3.4 SUMMARY

Based on the investigation and analysis of the bibliography presented in the previous sections, a discussion about the presented approaches: statically-oriented, dynamically-oriented and model-oriented.

Starting with the latter, model-oriented identification of services, Gysel et al. (2016) and Chen et al. (2017) present two promising approaches by the techniques that they address however very limited by the fact that they are highly dependent on user's input. This input of artifacts must follow specific rules and comply with the formats and rules imposed by their software. This process of pre-processing input and conversion to comply with the established rules can in itself be a time-consuming task.

Source-code oriented solutions according to a dynamic approach allow to obtain extra information that static approaches do not allow. Jin et al. (2018) and Jin et al. (2019) resort to execution *logs* to make them more reliable in identifying functionality. This type of approach is however limited by two factors: the first is the extra load placed on the system to collect *logs* during execution, which can be a problem in large-scale systems; secondly, their approach is limited by the quality and coverage of test cases carried out to execute and collect the respective *logs*. The creation of tests in an incomplete way and with an inadequate coverage will also result in a proposal of inadequate microservices. The generation of tests that portray the user's behaviour in an automated way is also a complex task, for which concrete solutions do not yet exist.

Lastly, static source-code oriented solutions are considered. They stand out right from the start because they are approaches that use strictly the source code, without the need for other artifacts or a high investment of pre-processing by the end user. Compared to dynamic approaches, they do not require any type of code execution or *logging* which makes them more promising for analysing large-scale applications.

Overall, most of the techniques proposed use lexical terms in one way or another as part of their methodology. After all, the software is ultimately being read and created by developers and the way modules are grouped together to fit nicely with the domain of the project is one of the main ways to have better software comprehension. Although, most techniques rely at some level on processing such identified lexical terms as a mean to identify domain terms, the techniques being used (ie. Jaccard distance and Tf-IDF (Term frequency-inverse document frequency)) are simple techniques that might not yield the best results. Considering that, it is our goal to expand on other techniques that could increase the importance and impact of how code is written, to better identify domain terms ultimately yielding better microservices proposals.

# 4

# A METHODOLOGY TOWARDS THE IDENTIFICATION OF MICROSERVICES

In this chapter, we describe the proposed methodology to identify microservices. Figure 3 illustrates an overview of the steps composing the identification process.

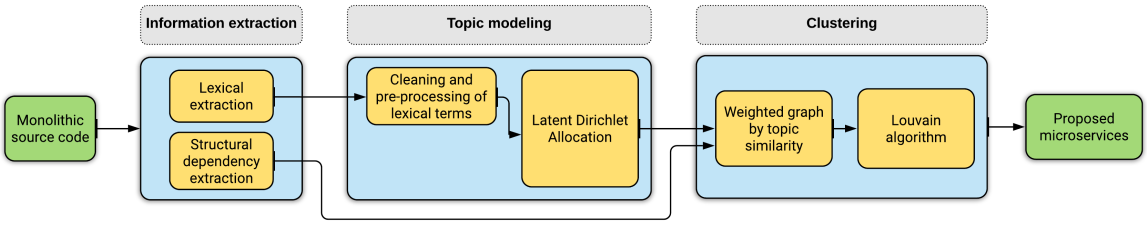



Figure 3: Overview of the architecture of the proposed method and developed tool

The methodology can be briefly summarised by the following: First, we extract lexical and structural information from the source code of the monolithic system being migrated to MA, described in Section 4.1.

Next, we use the extracted information to fit a topic modelling technique allowing the identification of topics and its distributions for each component of the software project.

Finally, the topic distribution and the structural information are combined and fed into a clustering algorithm identifying microservices proposals.

The next three sections describe these steps in detail.

Note that the process we describe is generic and not specific for a particular language or paradigm. However, since our tool is instantiated for the Java language, and in particular for the Spring Framework, the examples presented are written in this language and framework.

## 4.1 INFORMATION EXTRACTION

The building blocks of our microservice identification methodology are the lexical and structural dependency information occurring in the source code of the monolithic software system under analysis. Next, we describe how such information is computed.

LEXICAL EXTRACTION Lexical extraction is defined as the extraction of all the lexical/textual terms from source code that are relevant to identify what a given component represents in the context of the domain of the project.

We perform this extraction in a structured version of the source code: its underlying Abstract Syntax Tree (AST). The main reason to extract textual terms from the AST instead of its textual representation (and applying filters, stop words and other kinds of preprocessing) is to have better control of the information being extracted. This greatly reduces the amount of analysis needed to identify the most relevant terms. With a pure Natural Language Processing (NLP) approach applied to the source code, filtering keywords from the language would be simple. However with the addition of external abstraction (in the case of languages supporting it) and external libraries that introduce terms that could be completely unrelated to the domain of the problem, a pure NLP approach would produce worst results.

To handle the presented problem, only terms (such as variable types) referenced to the project are taken into account. Without losing generality, let us consider the following line of code written in Java (Spring Framework):

```
return new ResponseEntity<>(user, responseHeaders, HttpStatus.OK);
```

A project parsing approach gives us the possibility to only extract a certain parameter, for instance `user`, much more related to the domain than the whole expression that is mostly composed by Spring Framework terms and abstractions. For instance, given that `ResponseEntity` is so popular in projects based on the Spring Framework, an NLP approach could calculate that the addition of it to a list of stop words would solve the problem, however, that would only work for the common terms or require significant manual work when dozens of libraries are used in a project. Any addition of libraries that represent a strong connection to the domain, contrary to abstraction and helpers libraries, can still be included by an include list of types. Overall, our approach works by trying to filter out abstractions related to external libraries and frameworks and focus on the terms more related to the domain of the project. This is done by extracting textual terms from components' names, variable declarations, method/functions declarations and its parameters, and method/functions invocations.

STRUCTURAL DEPENDENCY EXTRACTION Most of the dependencies are straightforward to obtain working on the AST. However, other tasks such as the identification of types of expressions or finding the usage of a symbol are not so simple as they involve significant work over the AST. For those, and considering the particular case of the Java language, we used *JavaParser Symbol Solver* sym, which resolves what are expressions referring to.

The use of structural dependencies enables a better representation of the software architecture at hand, as it fits nicely into a graph representation and presents interaction between modules.

## 4.2 TOPIC MODELLING

Topic modelling techniques allow to identify latent semantic structures from a set of documents, similarly to how a developer would analyse a software project and identify domain terms to grasp how it can be decomposed on a semantic level.

The quality of topics proposed by these techniques is highly dependant on the quality of the inputs fed into them. Accordingly, textual terms $t$ are pre-processed going through tokenization, stop word removal and stemming in order to remove terms without much significance as domain terms and reduce variations of the same root words. Extremes of very common and rare terms are filtered and then collected as a bag of words as the final form.

Having computed the lexical and structural information, we can now use a topic modelling classificator to group such information in clusters, which will then form the microservices identified in the legacy system.

In an exhaustive and comprehensive state of the art review on topic modelling done by Kherwa and Bansal (2018), four major groups of topic modelling classificators are identified: Probabilistic Latent Semantic Analysis (PLSA), Latent Dirichlet Allocation (LDA), Latent Semantic Analysis (LSA) and Non-negative Matrix Factorisation (NMF). This work also presents a detailed quantitative analysis comparing LDA *versus* LSA concluding with the superiority of LDA: it yields higher coherence values across topics and less overlap between topics. In an exploratory work done by Stevens et al. (2012) comparing NMF, LDA and LSA and analysing its weaknesses and strengths, they conclude in favour of LDA due to its flexibility and coherence advantages over others. Sun et al. (2017) propose a technique of clustering classes in packages in order to increase program comprehension and reducing large packages resorting to LDA, PLSA and Latent Semantic Indexing (LSI) as a base of clustering methods. From the case studies conducted, LDA resulted in better clustering results and the topics identified were more useful for comprehension by developers.

Overall, LDA is widely used and the most popular on the topic modelling field, being the core of evolution and extension to other models, such as Dynamic topic model, Author topic model, Multilingual Topic Model (Kherwa and Bansal, 2018; Sun et al., 2016). Thus, we also adopt the LDA classificator in our microservice identification approach, describing it next.

### 4.2.1 *Latent Dirichlet Allocation*

LDA categorises documents by topics via a generative probabilistic model (Blei et al., 2003). It treats each document as a random mixture of latent topics, and each topic as a distribution of words of the corpus. The words with higher probabilities that represent a topic usually give a good overview of what is the topic describing and talking about, hence

allowing to discover a set of concepts representing the entire corpus (Jelodar et al., 2019). LDA is an unsupervised model requiring only the corpus of the documents without any extra metadata. LDA does not consider the order of the words in the documents or their semantic importance being only fed with a bag of words (BoW) – a simplified representation of a corpus containing count occurrence for each word. These features allow LDA to be scalable to thousands or millions of documents (Sun et al., 2017).

The components being used as the basis of work to the LDA model are formally described as follows:

1. A word represents the basic unit extracted from the source code of a software project representing the textual terms denoted as $t = \{w_1, w_2, ..., w_n\}$.

2. A document, identified as a component (*eg.* class in Java, module in C) in the context of a software project is a collection of words and described as $d = \{w_1, w_2, ..., w_n\}$.

3. A corpus is a collection of documents identified as $c = \{d_1, d_2, ..., d_n\}$.

Choosing the number of $K$ topics to be identified by the LDA model is a challenge on itself. Ideally, the number of topics should be selected after an analysis of the domain terms of the project at hand complemented with an analysis of inter-topic distance (Sievert and Shirley, 2014), in order to assess how well defined are the topics, its independence from each other and the amount of overlap. The distribution of terms per topic should also be taken into consideration to avoid topics from being composed of a large chunk of concepts across the domain, resulting in lower service cohesion.

Figure 4 and Figure 5 illustrate inter-topic distance and term distribution per topic for a model of 8 and 11 topics respectively. By comparing both figures it can be concluded that the model shown on Figure 5 represents an excessive number of defined topics given the amount of overlap introduced by the increase in number of topics.

An automated approach to identify an adequate amount of topics, yet computationally more expensive, can be simply done by creating models for a wide range of topics, measuring the coherence for each one and deciding upon the best topic. Measuring coherence of topics has as a main goal to verify if a *"set of facts support each other*" (Röder et al., 2015) and refer to a specific domain of knowledge. Among the multiple metrics proposed and the extensive analysis of the state of the art on coherence measurements done by Röder et al. (2015), the $c_v$ metric (which results from the combination of previous metrics) is the one having the highest correlation to human ratings on topic coherence.

With multiple values of $c_v$ measured for a range of number of topics, the best topic is automatically selected by identifying the knee point as shown on Figure 6.

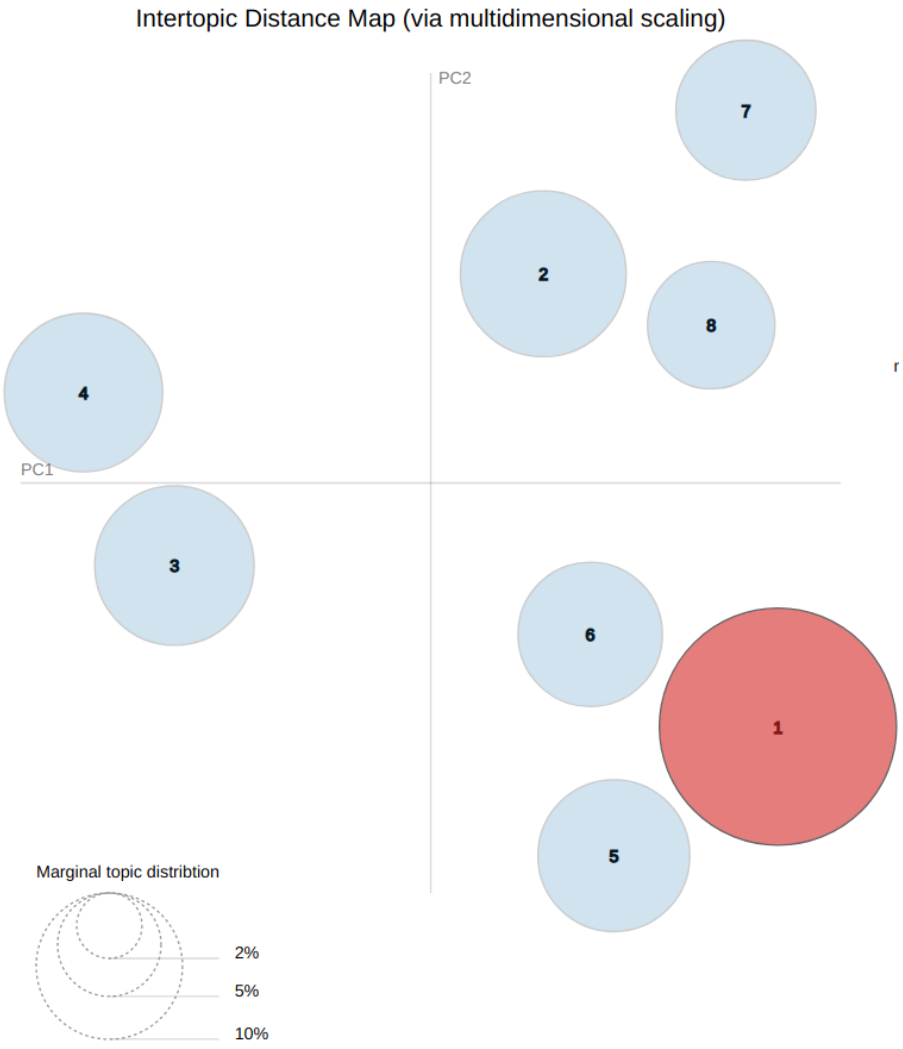



Figure 4: Intertopic distance with number of topics set to 8

## 4.3 CLUSTERING

The fitting of the LDA model against $K$ topics produces a distribution of topics across documents. That distribution can be used on its own to cluster components into groups of proposed services. However, that would mostly take into account the domain aspect of the software, ignoring the structural relationship and dependencies between classes. Thus, we combine the previously extracted structural dependencies with the distribution of topics into an edge-weighted graph $G$. In the graph $G = (E, V)$, the vertices $v_i \in V$ correspond to a component $c_i \in C$ from the monolithic project. Each edge $e_i \in E$ is weighted by a weight function determining how strong is the relationship according to topic distribution. The higher the value the stronger the relationship between topics identified are. Each component $c$ is then identified by a vector $\vec{v}$ to represent the distribution of probability across topics $t$,

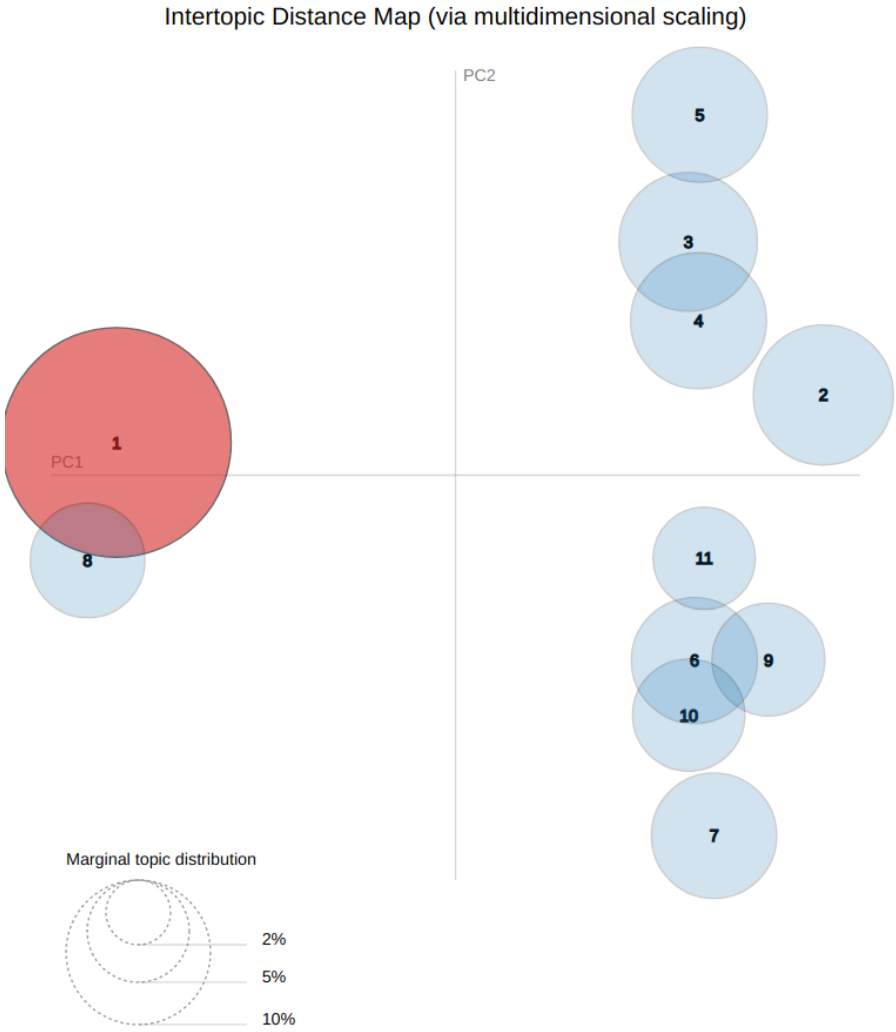



Figure 5: Intertopic distance with number of topics set to 11

$\vec{v} = \langle t_1, t_2, ..., t_n \rangle$. The cosine similarity is then used to find how similar two vectors $\vec{v_1}, \vec{v_2}$ are, as follows:

$$\cos(\theta) = \frac{\vec{v1} \cdot \vec{v2}}{\|\vec{v1}\|\|\vec{v2}\|} = w(e_i) \in G \tag{1}$$

An illustration of a formal definition of graph $G$ is presented in Figure 7.

We conducted an analysis of the state of the art of clustering/community detection algorithms to decide how to cluster the graph into proposals of services. Rahiminejad et al. (2019) performed a topological and functional comparison of community detection algorithms in biological networks. Six algorithms are analyzed: Combo, Conclude, Fast Greedy, Leading Eigen, Louvain and Spinglass. The main criteria of evaluation for those algorithms were: appropriate community size (neither too small nor too large), performance

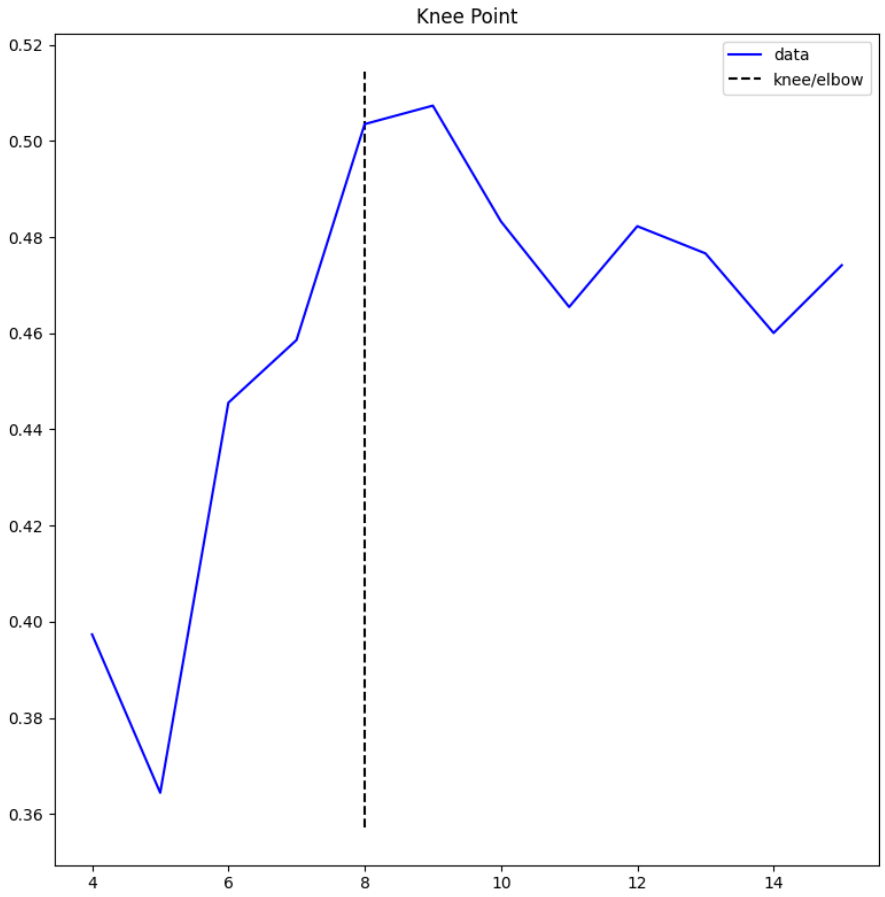



Figure 6: Identification of knee point for coherence over number of topics

in terms of speed and two other features regarding gene and biological functions. From the conducted evaluation on two distinct data sets they conclude favouring Louvain given that the communities found were very similar to the top methods and Louvain was the fastest community detection method.

The Louvain algorithm presented by Blondel et al. (2008) is a heuristic algorithm based on modularity maximisation. Its main goal is to maximise network modularity. Modularity is a measure of strength of division of a network into clusters/communities. Higher modularity represents dense connections within nodes in a community but sparse connections between different communities (Newman, 2006). It is an unsupervised algorithm not requiring the number of communities to be identified nor their sizes.

At its core the algorithm is divided into two main steps repeated iteratively (Blondel et al., 2008; Mishra):

- Step 1 - Each node in the network is assigned to its own community. The number of communities is equal to the number of nodes $N$; For each neighbour $j$ of node $i$, it is tested if the modularity increases by moving it from community $i$ to community $j$. If

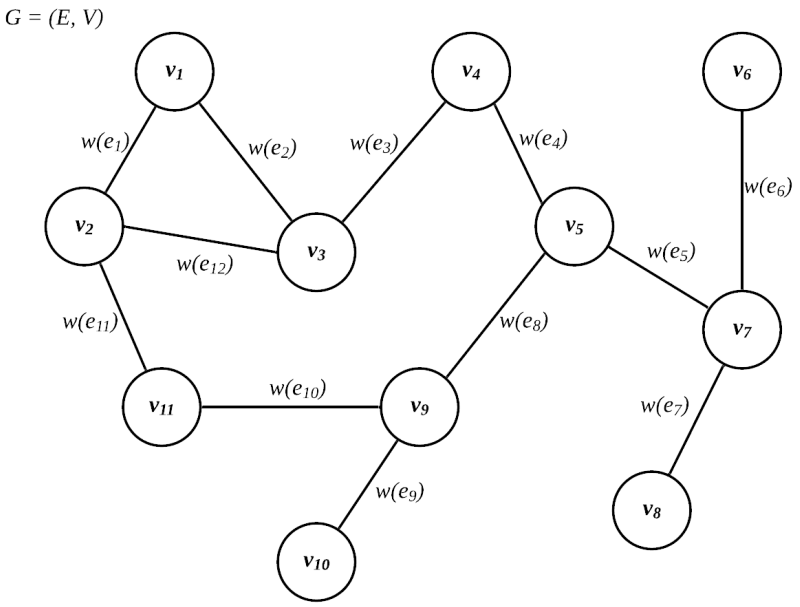



Figure 7: Weighted graph representation

there is an increase, the node is moved, otherwise it stays in the original community. This step is repeated for all the nodes in a sequential order and repeated until no improvements in modularity can be achieved.

- Step 2 - The network is rebuilt by merging the nodes in the same community.

Steps 1 and 2 are executed iteratively until the merging of communities does not change and a maximum modularity is reached. The majority of the computational work is done on the first iterations. Step 2 exponentially reduces the amount of work as it gets closer to the final iterations.

A common problem identified as a limitation of the Louvain algorithm and other algorithms that use modularity as its core is that it may fail to identify modules smaller than a given scale (Fortunato and Barthélemy, 2007). That problem was observed initially on projects more complex and bigger in number of components (classes in our case – Java). In order to avoid that, the resolution parameter is manipulated allowing to discover clusters at different scales (Lambiotte et al., 2014). A higher resolution results in more iterations of the merging step of the algorithm, resulting in less but bigger clusters. Similarly a lower resolution results in less merging, meaning more clusters of smaller size. For each resolution the project is clustered and executed against metrics of independence of functionality and modularity presented in Section 7. From that metric execution the best resolution can be selected. Although a resolution is chosen and consequently its proposed services, we provide the user all the proposed services for other resolutions. With different granularities

of proposed microservices the user can do a more informed and qualitative identification of what represents the best solution for the current project.

# 5

# IMPLEMENTATION

In this section the implementation of a prototype previously described according to a given methodology is presented. The prototype serves as a proof of concept to validate the methodology and should not be thought as a tool that provides the identification of the absolute best microservices but as a guiding tool to the expert employing such migration. It should however, present the expert with possible cohesive and loosely coupled versions of microservices given an initial monolithic architecture depending on a set of parameters provided by the user. The full implementation can be found at `https://github.com/miguelfbrito/microservice-identification`.

Figure 8 represents an overall flow of the prototype implemented.

## 5.1 INFORMATION EXTRACTION

The information extraction stage is responsible to process the input provided by the user as a software project and process it in order to extract and identify all the relevant information for microservice identification. The identified relevant information are the lexical terms that compose the source code of the project as well as the structural dependencies between components of the system (ie. classes, modules, etc.). Both stages are closely connected considering that the method relies on the information parsed from the project.

### 5.1.1 *Project parsing*

Given the nature of the methodology being related to NLP and data science methods, the usage of a language such as Python provided that a parser to parse Java projects was available and fulfilled the requirements would be a nice fit as it would be easier to handle the project and avoid extra communication layers. The research pointed to the usage of *javalang*[1], an open-source project providing an intuitive API to parse Java projects and according to the Java 8 language specifications. An initial implementation to extract all the relevant

1 https://github.com/c2nes/javalang

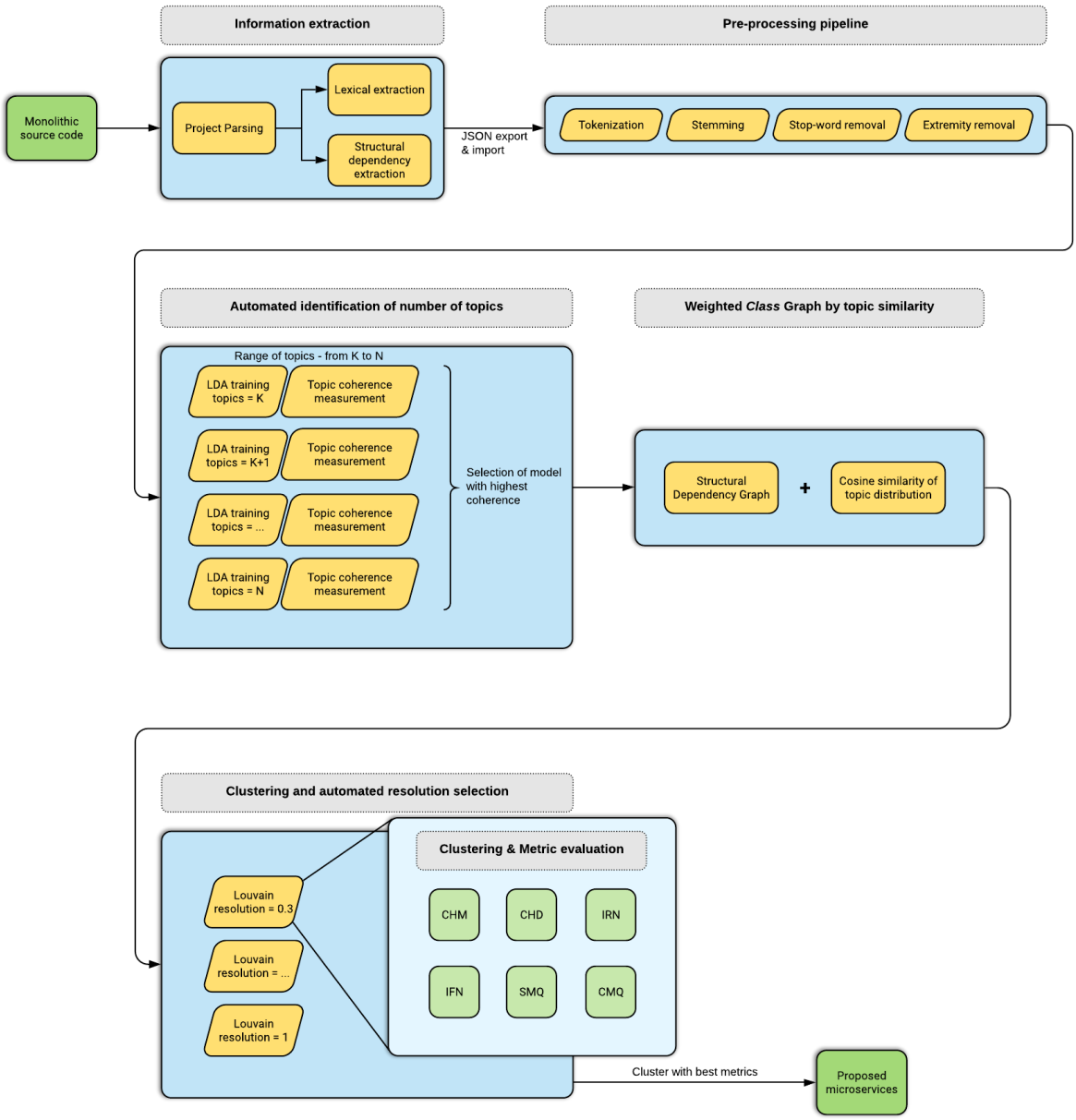



Figure 8: Overall flow of the tool

information was conducted and used to perform preliminary tests on topic modelling techniques. However, as the requirements regarding the parsing of the project increased, *javalang* proved to be insufficient as the iterator it provided had limitations and became harder to obtain specific information and provided very limited capabilities to resolve the declarations connected to a given element (eg. variable declaration types). Those limitations

forced a refactor in the tool and resorting to the official *Java Parser*[2] which merged with the *Java Symbol Solver*[3] project allowing to resolve specific expressions and method calls (sym).

The information extraction is made by the following steps:

- Firstly, the *Java Parser* identifies the source root of the project and from it parses a list of `CompilationUnits` which is a representation of each java file.
- For each of the `CompilationUnits` classes are identified as well as direct dependencies of inheritance and implementation.
- A set of visitors is then created to identify: variable declarations (*VariableDeclaratorVisitor*), method declarations (*MethodDeclarationVisitor*), method calls (*MethodCallExprVisitor*) and annotations (*AnnotationVisitor*). *VariableDeclaratorVisitor* identifies all the variables declared on a given class, which later are resolved and dependencies between the original and the resolved class are created. *MethodDeclarationVisitor* identifies all the methods declared for a given class, which later are processed, resolving and identifying all dependencies for all of its parameters and return type. *MethodCallExprVisitor* is responsible to identify all the method calls, which later are resolved and dependencies are created. *AnnotationVisitor* simply identifies annotations attached to a given class. Apart from the dependencies, the actual lexical terms describing variables, methods, etc. are collected as they represent the base of knowledge for semantic work. All classes are referenced as its fully qualified name in order to guarantee the identity of each class.
- The information identified and collected by the Java Parser and iteratively built into an object containing all information related to a class on the original project is then exported to a *JSON* (JavaScript Object Notation) file.

## 5.2 TOPIC MODELLING

The previously extracted JSON file containing all the dependencies as well as the lexical terms for each class is now read from a Python program to proceed with further processing and the application of topic modelling.

For each class an object is instantiated according to previously saved annotations, variables, dependencies, methods and its parameters and return types. This object is then responsible to provide a merge of all the semantic terms for further processing.

At this stage the loaded structure goes through a step of pre-processing. This includes multiple transformations resorting to the *Natural Language Toolkit* (NLTK), such as: tokenization, stemming, stop word removal, for both common dictionary terms and a custom set

2 https://github.com/javaparser/javaparser
3 https://github.com/javaparser/javaparser/wiki/About-the-Symbol-Solver

of words that the user can include in order to avoid terms very common in the application that do not necessarily represent good domain terms. With the Gensim[4] library the cleaned text is used to build a dictionary, from which extreme terms are removed (eg. top 20% most common words are not taken into consideration) and finally used to create a corpus by transforming each document into a BoW.

The application of topic modelling techniques is done through the APIs provided by the Gensim library, which provides a variety of techniques for topic modelling and text processing. Training the LDA model is done by providing the corpus and a correspondent dictionary, the number of topics and optional parametrizations regarding the model itself. During the research for better parametrizations and ways to optimise the model being trained, multiple references to the usage of Mallet[5] over Gensim were found (Rafferty, Prabhakaran). From the experimental information gathered, Mallet appears to provide better coherence values over Gensim, however it is a slower method. Considering that we prefer higher quality over fast results, and the difference is not immense Mallet is used to train the LDA model. Gensim provides however an interface to interact with Mallet as it is a Java package.

The selection of the best model depends on the user input. If the user chooses to provide a number of topics, the model is trained according to the extracted information and provided number of topics. If the user does not provide the number of topics, measures to identify the best number of topics are performed.

The identification of the best number of topics is made by identifying a range of possible number of topics (eg. 5-25). For each trained model its coherence values are measured according to the $c_v$ metric identified by Röder et al. (2015) and the correspondent implementation provided in Gensim[6]. From the plot of the coherence of topics over number of topics the knee is identified. The knee represents the point at which the coherence of topics stabilises over number of topics and further increase in number of topics might not be worth it due to a low increase in coherence. Another reason is that the increase in number of topics indefinitely has a tendency to increase coherence values considering that each topic will be composed of lesser words and therefore there is greater probability of a given topic being more cohesive.

The final result of the trained LDA model is a list of probability distributions across topic per document (ie. java classes).

4 https://radimrehurek.com/gensim/
5 http://mallet.cs.umass.edu/
6 https://radimrehurek.com/gensim/models/coherencemodel.html

## 5.3 CLUSTERING

Clustering represents the last main stage before proposing microservices. In order to apply the previously obtained semantic knowledge and combine it with the structural information of dependencies a graph structure as to be created. Such endeavours were done resorting to the networkx library[7], which provides easy creation and manipulation of graphs as well as good interaction with other libraries because it is one of the most used libraries on graph manipulation on Python. The initial graph of classes is built according to dependencies between each other and then the cosine similarity between vectors of topic probability are applied for each edge.

With the graph created and weights set for each edge, the graph can be fed into the clustering algorithm. Similarly as the selection of number of topics, the resolution used to identify clusters' fragmentation can be defined by the user or chosen between an arbitrary range. The selection of best cluster in an arbitrary range is achieved by the evaluation of five metrics regarding independence of functionality and modularity presented in Chapter 7.

Regarding the clustering algorithm itself, multiple libraries and implementations were tested (ie. NetworkX Communities, NetworkX Clustering, igraph, scikit-learn). In the end, the Python-Louvain library[8] was utilised due to straightforward interaction with networkx and slightly better results on overall modularity on some experimental cases.

The execution of the Louvain algorithm against a directed *networkx* graph yields a list of clusters and consequently proposals of microservices. In some rare cases where there are clusters composed of 1 or 2 classes some post-processing is employed and such cluster is merged with the closest regarding semantic proximity. If the semantic distance is zero the structural distance of number of dependencies between clusters is considered.

## 5.4 METRICS

With the goal to keep objectivity and impartiality on metric evaluation there was an attempt to obtain the implementation of metric evaluation done by the work of Jin et al. (2019). Out of the five metrics, *CHM* and *CHD* were used exactly as provided by the authors, however, the remaining metrics had an implementation closely attached to their technique of microservice identification and a new implementation had to be done. The implementation was done following the formal definition presented in the paper, and with contact with authors to clarify a few unclear concepts.

7 https://networkx.github.io/
8 https://github.com/taynaud/python-louvain

# 6

## CASE STUDY

In this section we present a walkthrough of the proposed methodology on a Java Spring application named JPetStore[1]. JPetStore is a shopping application themed as pet selling. Our goal is to present the most relevant steps, namely the application of the topic modelling as a way to analyse the identified topics and words that compose the given topics. Since it is a relatively small project, steps concerning identification of $K$ topics are not included, however the most frequent identified resolution is used. Skipping the initial information extraction and pre-processing of the textual terms the LDA model is inferred according to the pre-processed textual terms against a number of topics of $K$=4. For this particular case we also considered a resolution range between 0.6 and 1.1 with 0.1 increments. In Table 1 the top 10 words in its stemmed version are presented for each topic:

- Topic 0 - The first five top words represent strong domain concepts referring to users/accounts.
- Topic 1 - Strong domain relation to cart management.
- Topic 2 - Strong domain relation to catalogue: products, category, item.
- Topic 3 - Strong domain relation to order execution: line, status, total, price, cart, stock.

| *Topic* | *Top 10 words representing a topic* |
|---|---|
| 0 | account username password signon profile clear resolut status version serial |
| 1 | item cart line quantiti big total price stock increment inventori |
| 2 | product catalog categori item resolut serial clear profil total name |
| 3 | order sequenc line statu usernam total price version cart stock |

Table 1: Top 10 stemmed words belonging to each topic

1 Repository found at https://github.com/mybatis/jpetstore-6

The topic distribution for each class of the project is presented in Table 2. The vast majority of the classes have a strong association to one of the topics (highlighted in bold face). The class `web.actions.AbstractActionBean` stands out as having a very similar distribution across topics because it represents an abstraction extended by the main entities exposing the application functionality as beans. Although there is still no way to deal with abstraction classes being split from the classes that directly require them (on the implemented tool), this kind of information may be useful in the future as a way to alert and guide the final user when conducting the migration to a MA.

Lastly, with the calculation of cosine similarity between the structural dependencies between classes resorting to topic distribution, the clusters of classes as potential microservices can be identified. At this stage multiple resolutions of the *Louvain* algorithm are tested and executed against metrics in order to identify which granularity provides the best metrics. Although metrics provide the user with an informed quantitative view of the microservices being presented, the adequate level of granularity of the services is dependent on the project's domain and architecture and the subjective perspective by the developer on how small or how many interactions should exist between services. Developer *A* might prefer smaller services which inherently leads to more connections to external services while Developer *B* might prefer slightly larger services resulting in less interactions between services. Such occurrence can be observed in this case study by Figure 9. The plot on the left represents the summation of all normalised metric values, while the plot on the right represents the evolution of each normalised metric across resolution.

According to the summation of all metrics, resolution 1.0 and 1.1 represent the highest values and equal proposed clusters. However by analysing each individual metric across resolutions setting different weights on specific aspects that a user prefers above others could result in different choices. For instance, if a user values more lower interaction between services he/she should prioritise the evolution of *IRN* more than the other metrics. In this particular case, *IRN* improves with larger services and so the user should analyse the proposals in the higher range of resolution. On the other side of the spectrum, if a user prioritises smaller and more cohesive services inherently resulting in higher interaction with other services, he should prioritise *CMQ* and *SMQ*. In this particular case it is important to note that the high values for *CMQ* and *SMQ* in the lower end of resolution are heavily impacted by the low amount of classes per service (observable on Appendix A.1.1) and are also affected by data normalisation as the variance between values is relatively close (0.13 to 0.21 for *SMQ* and 0.3 to 0.45 for *CMQ*). *CHM* and *CHD* stay at the same value across resolution because those metrics are measured concerning only interfaces exposed by each microservice. If such interfaces stay equal it is expected to behave as shown in Figure 9.

Figure 10 illustrates the clustering performed and the connections between classes for the solutions presented in the higher range of resolution (1.0 and 1.1). Four clusters are

| *Class* | *Topic distribution* | | | |
|---|---|---|---|---|
| *(org.mybatis.jpetstore.*)* | 0 | 1 | 2 | 3 |
| web.actions.AccountActionBean | **0.79** | 0.03 | 0.14 | 0.04 |
| domain.Account | **0.82** | 0.04 | 0.09 | 0.05 |
| service.AccountService | **0.72** | 0.09 | 0.09 | 0.1 |
| mapper.AccountMapper | **0.81** | 0.06 | 0.06 | 0.07 |
| web.actions.AbstractActionBean | 0.26 | 0.25 | 0.25 | 0.23 |
| mapper.LineItemMapper | 0.1 | **0.61** | 0.1 | 0.18 |
| domain.Item | 0.05 | **0.73** | 0.18 | 0.05 |
| domain.LineItem | 0.04 | **0.82** | 0.04 | 0.11 |
| domain.CartItem | 0.06 | **0.83** | 0.06 | 0.06 |
| domain.Cart | 0.04 | **0.89** | 0.04 | 0.04 |
| web.actions.CartActionBean | 0.12 | **0.71** | 0.13 | 0.04 |
| mapper.ItemMapper | 0.09 | **0.66** | 0.16 | 0.09 |
| web.actions.CatalogActionBean | 0.03 | 0.08 | **0.86** | 0.03 |
| domain.Category | 0.13 | 0.13 | **0.62** | 0.12 |
| mapper.ProductMapper | 0.11 | 0.11 | **0.68** | 0.11 |
| mapper.CategoryMapper | 0.14 | 0.14 | **0.57** | 0.14 |
| domain.Product | 0.1 | 0.1 | **0.7** | 0.1 |
| service.CatalogService | 0.05 | 0.17 | **0.74** | 0.05 |
| mapper.SequenceMapper | 0.15 | 0.15 | 0.15 | **0.55** |
| mapper.OrderMapper | 0.11 | 0.1 | 0.1 | **0.69** |
| domain.Sequence | 0.19 | 0.18 | 0.19 | **0.44** |
| domain.Order | 0.08 | 0.16 | 0.02 | **0.73** |
| service.OrderService | 0.06 | 0.24 | 0.06 | **0.64** |
| web.actions.OrderActionBean | 0.2 | 0.08 | 0.05 | **0.66** |

Table 2: Topic distribution for each class on the JPetStore project (The distribution is expected to sum to 1, however due to rounding there are cases where that does not happen)

identified as proposals of microservices (each cluster is coloured differently in the figure). The remaining clusters of classes can be consulted in Appendix A.1.1.

In this example the clusters are very similar and are according to the topics identified and its distribution. For larger and more complex applications the clustering will have a more

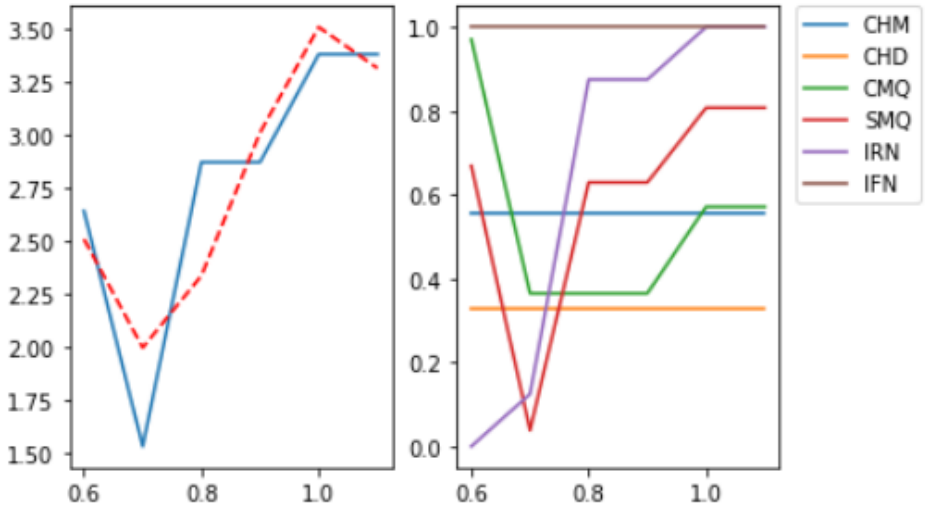



Figure 9: JpetStore's metrics regarding independence of functionality and modularity

significant impact, given higher number of topics and its distribution, and an increased number of dependencies between components.

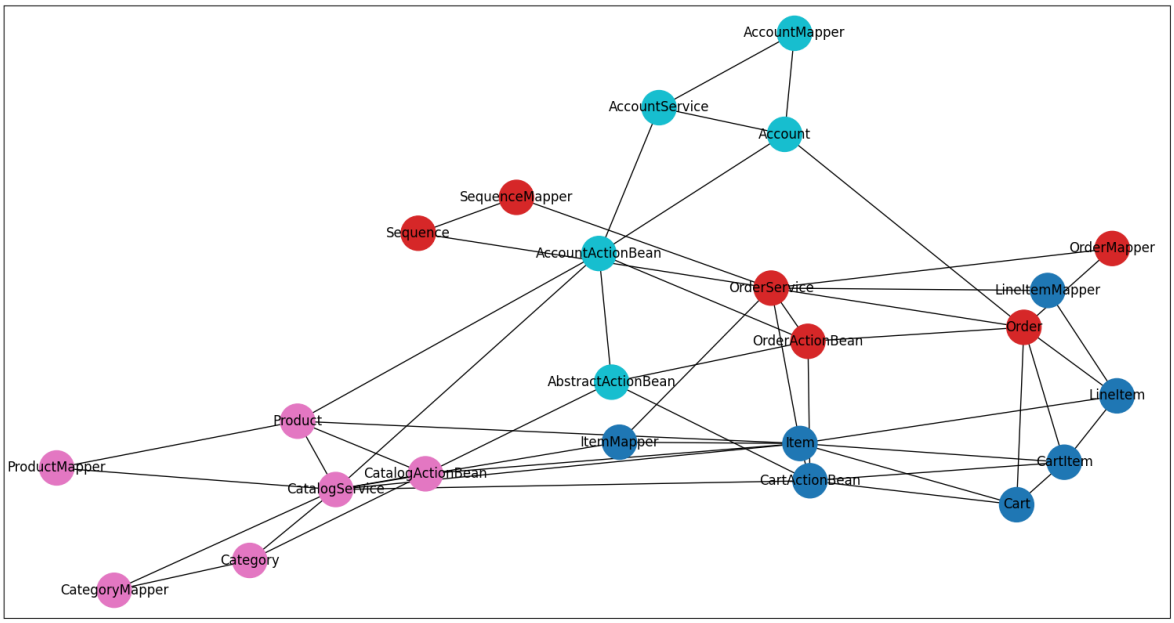



Figure 10: Clustering using topic distribution similarity as weight

In the context of the example, the identified clusters propose four different services related to: accounts/users, cart management, catalogue management and order execution.

# 7

# EVALUATION

In order to quantitatively assess the quality of the microservices proposed by our approach, we collected 200 projects from *GitHub* and computed the MA metrics proposed by the work of Jin et al. (2019) regarding independence of functionality and modularity. In this section we describe in detail this evaluation, starting by the metrics.

## 7.1 INDEPENDENCE OF FUNCTIONALITY

Independence of functionality refers to external independence, meaning how independent and well-defined the services are. The following metrics are calculated resorting to the interfaces $I$ of a service. An interface is any class that exposes functionality as an endpoint. For each interface, methods are considered as operations $O$.

The value **ifn** (*interface number*) quantifies the number of interfaces for a given service. It is based on the Single Responsibility principle. A smaller *ifn* represents a higher likelihood of any given service assuming a single responsibility Jin et al. (2019). *IFN* represents the average of all *ifn*.

$$IFN = \frac{1}{N}\sum_{i=1}^{N} ifn_i \tag{2}$$

$$ifn_i = |I_i| \tag{3}$$

The value **chm** (*cohesion at message level)* quantifies the cohesiveness at the message level of the interfaces of a given service. A higher *chm* represents a higher cohesiveness of the service. *CHM* represents the average of all *chm*. Messages are composed by the terms of method parameters (*par*) and method returns (*ret*).

$$CHM = \frac{1}{N}\sum_{i=1}^{N} chm_i \tag{4}$$

$$chm_i = \begin{cases} \frac{\sum_{(k,m)} f_{msg}(opr_k, opr_m)}{\frac{1}{2}|O_i| \times (|O_i|-1)}, & \text{if } |O_i| \neq 1 \\ 1, & \text{if } |O_i| = 1 \end{cases} \tag{5}$$

Jaccard index is used as function of similarity both for similarity of return terms and parameter terms.

$$f_{msg}(opr_k, opr_m) = \frac{\left(\frac{|ret_k \cap ret_m|}{|ret_k \cup ret_m|}\right) + \left(\frac{|par_k \cap par_m|}{|par_k \cup par_m|}\right)}{2} \tag{6}$$

The value **chd** (*cohesion at domain level*) quantifies the cohesiveness at domain level of the interfaces of a given service. It is quantified very similarly to *chm* varying only on the function of similarity. Instead of using only message terms, all domain terms are considered.

$$CHD = \frac{1}{N} \sum_{i=1}^{N} chd_i \tag{7}$$

$$chd_i = \begin{cases} \frac{\sum_{(k,m)} f_{dom}(opr_k, opr_m)}{\frac{1}{2}|O_i| \times (|O_i|-1)}, & if |O_i| \neq 1 \\ 1, & if |O_i| = 1 \end{cases} \tag{8}$$

$$f_{dom}(opr_k, opr_m) = \frac{|f_{term}(opr_k) \cap f_{term}(opr_m)|}{|f_{term}(opr_k) \cup f_{term}(opr_m)|} \tag{9}$$

The value **IRN** (*interaction number*) quantifies the number of method calls across two different services. The smaller the *IRN* the better (Jin et al., 2018).

$$IRN = \sum_{(opr_j, opr_k)} w_{j,k} \tag{10}$$

## 7.2 MODULARITY

Modularity evaluates how cohesive are the services in its internal interactions and how loosely coupled are the interactions across services.

**SMQ** (*Structural Modularity Quality*) quantifies modularity from a structural viewpoint. Higher *SMQ* represents better modularized services.

$$SMQ = \frac{1}{N} \sum_{i=1}^{N} scoh_i - \frac{1}{N(N-1)/2} \sum_{i \neq j}^{N} scop_{i,j} \tag{11}$$

*SMQ* quantification is divided into the quantification of intra-connectivity and inter-connectivity. The value ***scoh*** quantifies cohesiveness of a given service while ***scop*** quantifies coupling between services. High *scoh* and low *scop* represent a cohesive and loosely coupled architecture. $\mu_i$ represents the total edges for a service. An edge is counted when there is

a structural call dependency between $N$ entities. $\sigma_{i,j}$ is similar to $\mu$ but acts on a service to service level, meaning that an edge occurs when there is a dependency between service $i$ and service $j$.

$$scoh_i = \frac{\mu_i}{N_i^2}, \quad scop_{i,j} = \frac{\sigma_{i,j}}{2(N_i \times N_j)} \tag{12}$$

**CMQ** (*Conceptual Modularity Quality*) quantifies modularity from a conceptual viewpoint. Higher CMQ represents better modularity.

$$CMQ = \frac{1}{N}\sum_{i=1}^{N} ccoh_i - \frac{1}{N(N-1)/2}\sum_{i\neq j}^{N} ccop_{i,j} \tag{13}$$

$$ccoh_i = \frac{\mu_i}{N_i^2}, \quad ccop_{i,j} = \frac{\sigma_{i,j}}{2(N_i \times N_j)} \tag{14}$$

*CMQ* is very similar to *SMQ* but textual terms are used instead of call dependencies. Therefore, an edge is considered if the interception between terms is not empty.

## 7.3 SCOPE OF ACTION

Given the high complexity of creating a generalised solution for multiple languages and *frameworks* the scope of action for the solution had to be reduced to something more specific and manageable.

Starting with the language choice, languages for the web such as *PHP, Javascript, Java, C# and Python* stand out as the most common languages being used on the web. Despite the widespread use of *PHP* given the so popular LAMP and its use in content management systems like *Wordpress*, this is a language that has not been particularly adapted in the world of microservices because of its declining popularity. The use of *PHP* would make more sense if *transpiling* was also carried out for a language with better adaptation to the world of microservices. *Javascript* and in particular *f*rameworks like *node* have become highly popular in recent years, being used massively in both small and large scale projects that need to be easily scalable. Although this combination of *Javascript* and *node* is quite popular and makes sense in the context of using microservices, many of the applications have already started being developed during the adoption of microservice architectures. Of the remaining identified languages, *Java* stands out because it is a language widely used in the industry, especially for systems that require the language to allow the development of robust applications. The fact that it has remained over the years as a solid and popular choice for *web* applications and that there are a large number of monolithic applications is also a point in its favour.

The choice of the *framework* is also important, given the diversity of development methods and directions taken when developing for a specific framework. The specificity of a *framework* also enables to identify extra context regarding architecture through things such as annotations and other metadata. In a study carried out by RebelLabs (2017) involving 2060 participants in the software development area, they demonstrated, as in previous years, the high use and adoption of *Spring* compared to other *frameworks*, Figure 11.

Given the popularity and high availability of Java Spring applications that is the combination being used for the conducted evaluation.

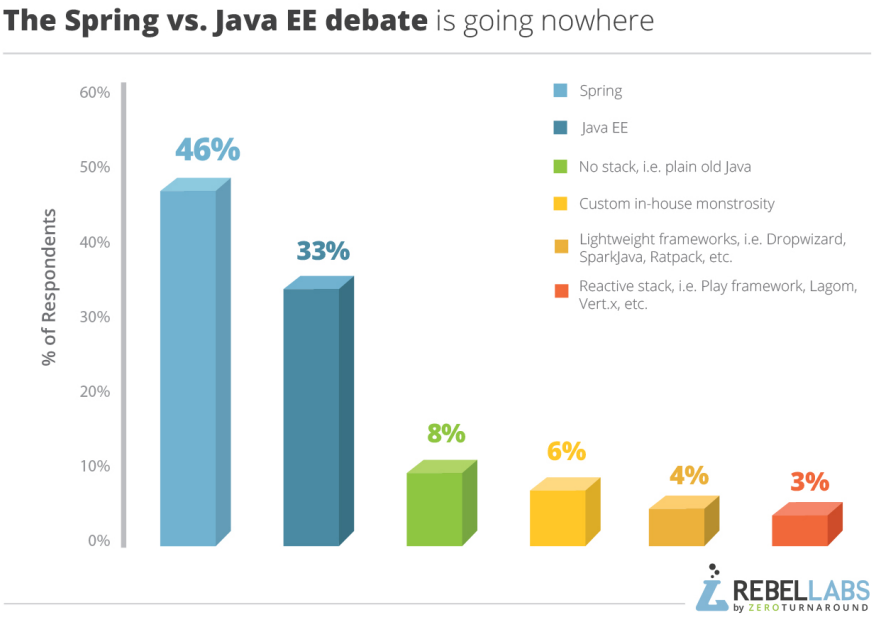



Figure 11: Survey on Java frameworks usage

Source: RebelLabs (2017)

## 7.4 PROJECT COLLECTION

With the goal of evaluating the quality of the services proposed by our methodology, we collected 200 Java Spring applications from *GitHub* using its search API (v3) and executed against the state of the art metrics proposed by Jin et al. (2019) on independence of functionality and modularity.

We used the GitHub search API for code as a mean to identify repositories using the terms *"RequestMapping"* and *"Controller"*, as those are very common and unique to applications built using the Spring Framework. Since any request to a GitHub endpoint is bound to 1000 results per each search, we used the parameter of file size to be able to find more repositories

with Java Spring projects. Thus, we created a set of queries by increasing the size interval (`min_size` and `max_size`) by 200 bytes and bound to a starting point of 500 bytes and final point of 200000 (around 200 KB). The query (template) used was as follows:

```
https://api.github.com/search/code?q=RequestMapping+Controller+language:
java+size:{min_size}..{max_size}
```

Executing this set of queries we identified 104024 results, however, many of them represent results on the same repository something that the API does not allow to exclude. These results were then filtered by uniqueness, removing duplicate references to the same repository and forks of the same repository. Filters were also applied to some very common repositories, such as Spring-Boot forks of the framework, demo and test projects by using the following stop words *{'release', 'framework', 'learn', 'source', 'spring', 'study', 'demo', 'test', 'practice', 'practise'}*. That reduced the original 104024 results to 29368.

Based on the criteria taken by Borges et al. (2016) and Ma et al. (2016) on works regarding criteria collection of GitHub repositories we selected the top repositories based on GitHub stars. Thus, for every repository, we parsed the number of stars as main decider and other relevant information such as open issues, subscribers and forks.

In order to guarantee that the projects are monolithic applications, only projects containing one "src" folder are considered. Any project with less than 30 classes is also discarded as a project with that dimension may represent just a "toy" Java project. From the final projects the top 200 are selected as evaluation data set.

The histogram of the projects collected by the number of classes is presented in Figure 12. In perspective, the biggest project considered is roughly around 2500 classes.

## 7.5 SETUP

We implemented the presented methodology in a proof of concept and tested the collected projects against state of the art metrics. Our main goal is, from a quantitative point of view, to understand if the microservices being proposed are relevant regarding independence of functionality and modularity.

Regarding the number of topics, we selected a range of arbitrary values according to quantity of classes. The ranges are wide enough in order to allow the identification of a knee point on coherence values, but not too large as that would slow down the process in a meaningful way.

Clustering resolution was also arbitrarily set. From our analysis a range from 0.3 to 1 should be able to deal with most applications in the hundreds of components: a resolution of 1 can identify clusters of larger sizes in small applications (in the tens) and resolution of 0.3 can identify smaller clusters in large applications (in the hundreds). The set range guarantees

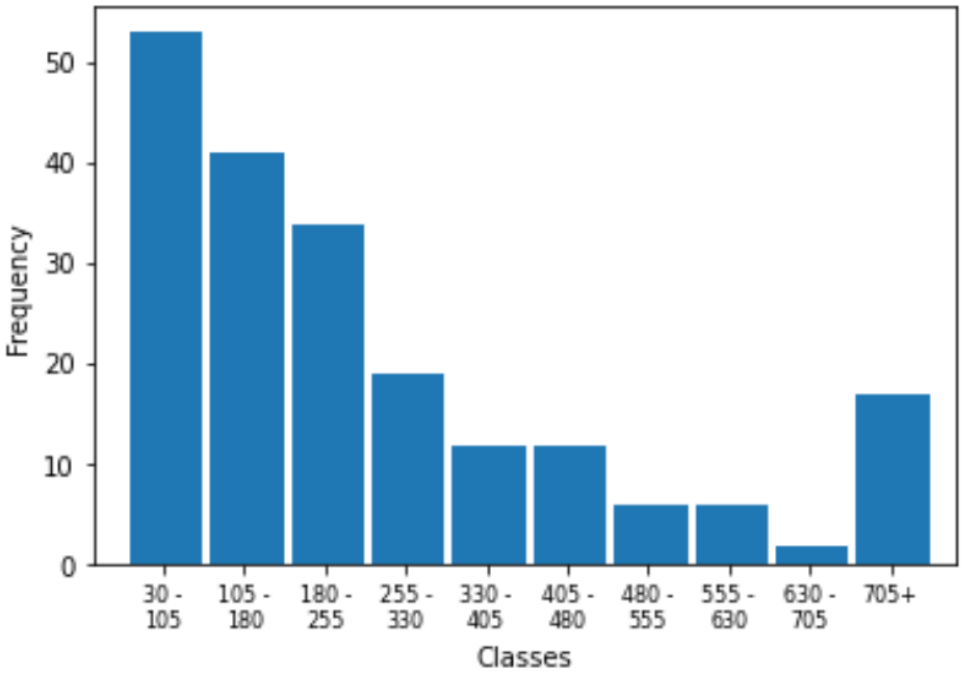



Figure 12: Histogram of collected projects by class count

us that we are able to identify smaller or larger microservices if they quantitatively represent the best cohesion and loosely coupling.

The selection factor of the best proposal of microservices is done by selecting the combination of *CHM*, *CHD*, *SMQ*, *CMQ*, *IRN* and *IFN* with best total absolute value. Regarding operation identification as a mean to measure CHM and CHD we collected all the methods present in controllers' classes. We also applied a threshold to calculate CHD since only tight terms related to the domain should be considered, requiring extensive cleaning and manual labour, something not feasible given the significant set of applications. In fact this is a similar process to the one present in the work proposing the metrics (Jin et al., 2019). The use of stop words for pre-processing was also very generic without any specificity per project. Ideally an analysis and addition of common terms irrelevant to the domain should be added despite the effort we put in information extraction stage to reduce the inclusion of external abstraction.

## 7.6 RESULTS

The respective individual results for each project can be found at Appendix A.2. For each project the following data is presented: project name on GitHub, number of classes, number of stars on GitHub, *CHM*, *CHD*, *IFN*, *CMQ* and *SMQ*, and total number of identified services. Considering that the study involves a large amount of projects, we resort heavily to the

usage of box plots, which provide a good perspective on data distribution. The box plot of the results obtained across all 200 applications are presented in Figure 13.

CHM and CHD which represent independence of functionality at operation level (methods exposed by services as interfaces) both show medians roughly around 0.6 which represent positive proposals regarding independence and well-defined services.

SMQ and CMQ which represent modularity of services and are bound between -1 and 1 presented medians roughly around 0.2 and 0.4 respectively. Some negative outliers regarding SMQ were observed, however CMQ remained positive across all projects.

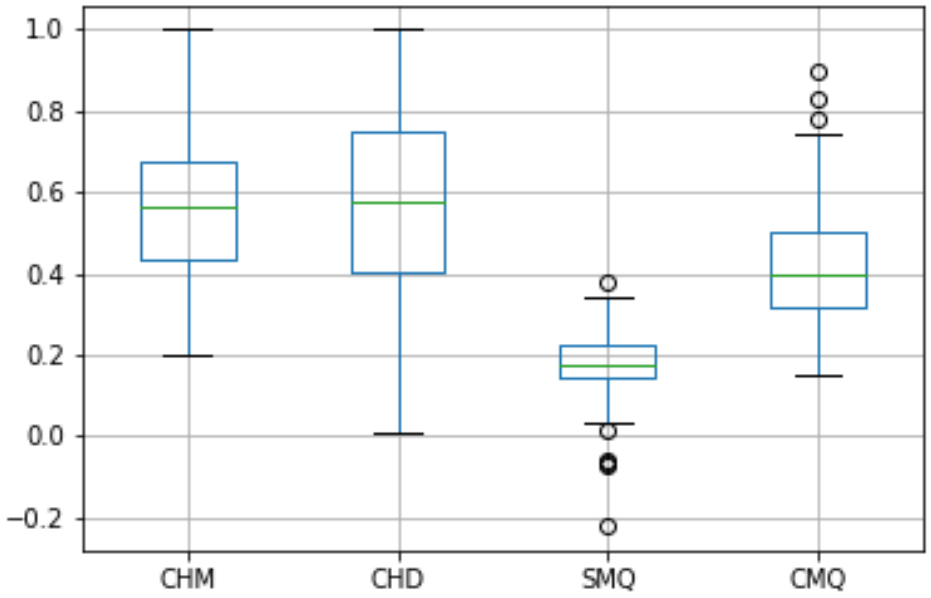



Figure 13: Metrics' box plot across 200 projects

IFN quantifies the interfaces exposed by a given interface, hence, a smaller IFN represents a higher likelihood of a service having a single responsibility. The median and both Q1 and Q3 on the box plot presented in Figure 14 are quite compact and close to values representing a single responsibility service (*ie.* IFN of 1).

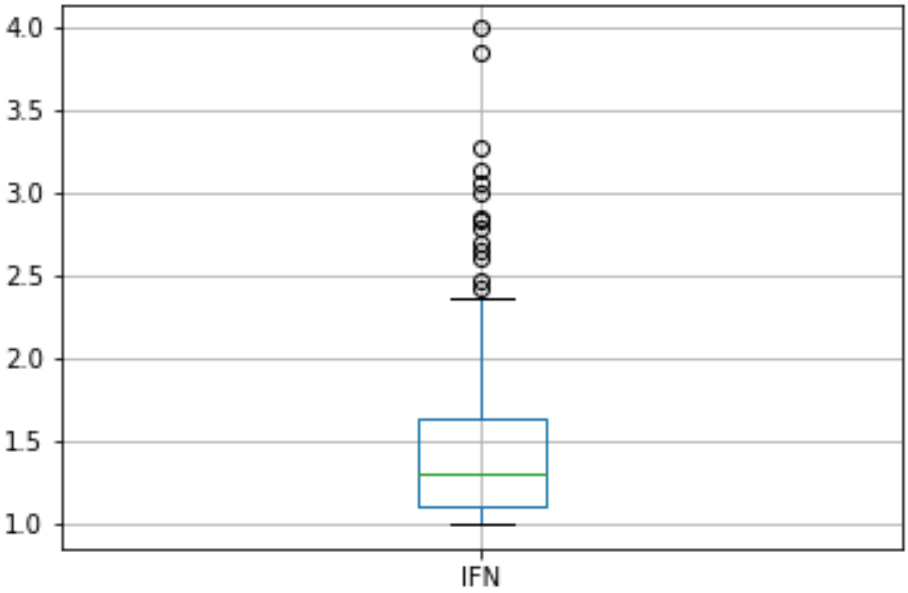



Figure 14: IFN's box plot across the 200 projects

## 7.7 ANALYSIS

Although the box plot of metrics shows an overall picture of how the tool performs quantitatively, the data set considered for study is composed of projects with wildly different domains, a wide range of total classes per project as well as different aspects of modularity either by the granularity of how classes are composed regarding methods or how often external calls to other classes are made. In order to get some insight on those aspects of modularity we introduce a more extensive analysis on the following section.

### 7.7.1 *Correlation analysis with metrics*

In this section a correlation analysis to metrics is conducted with the goal to understand if particular modularity properties of a project can be directly correlated to the obtained metrics. For instance, if bigger classes with more method declarations produce better results, or how impactful are external invocations to other classes, does having less method invocations result in better metrics? Or vice versa? For such analysis two ratios are considered: ratio of total methods declarations over total classes, and ratio of total methods invocations to external classes over total classes.

The main goal of each ratio is to further investigate other aspects of project modularity other than number of classes. For instance, the first ratio allows us to understand how the methodology behaves with different quantity of method declarations which provides some perspective on *class* size. The second ratio aims to show how does the methodology work for different levels of cohesion, in other words, assess if there is any significant difference in projects that have more or less external calls between *classes*.

Regarding the statistic method used to find the correlation both parametric and non-parametric methods were initially considered. The ultimate decision was made by analysing prerequisites of both methods (namely Pearson correlation vs Spearman correlation) (Sarmento, 2017). Firstly, the outliers can have a huge impact on Pearson's (parametric) method. Removing some outliers did not result in significant changes in correlation values. Secondly, Pearson's requires that the data being analysed follows a normal distribution or very close to it. Upon normality evaluation through the Shapiro-Wilk method, both ratios and *IFN* failed the normality test for a $p = 0.05$.

Considering that Spearman (non-parametric) method relies on the same assumptions of Pearson's except the normality of data, Spearman's method is the most appropriate for the analysis.

Another point taken into consideration for proper correlation analysis is the percentage of change for each series of data, otherwise, the correlation could yield high results even though both series evolved with similar trend but completely different levels of change. In other words, if the trend is similar, without considering percentage of change, it is likely that a high correlation will be found, even though there is not a direct influence of one variable over the other.

The obtained correlation values for both ratios to each metric is presented on Table 3.

| | Method Declarations / Classes | Method Invocations / Classes |
|---|---|---|
| CHM | -0.080260 | -0.092911 |
| CHD | -0.078329 | -0.027260 |
| IFN | 0.078123 | 0.308083 |
| CMQ | -0.283848 | -0.109619 |
| SMQ | -0.043258 | -0.198207 |

Table 3: Correlation between metrics and both ratios.

No strong correlations between metrics and presented ratios were found, however, there are two weak correlations (*CMQ* with *Method Declarations / Classes* and *IFN* and *Method Invocations / Classes*) that can be discussed. From the *CMQ* correlation (darker shade) it can be hypothesised that an increase in method declarations results in lower *CMQ*, hence worse results. This could happen due to harder identification of topics, or the classes themselves

not being as cohesive from a domain perspective. This correlation could also be highly affected by projects composed of classes disproportionately large compared to other classes.

The second occurrence of an weak correlation is between *IFN* and *Method Invocations / Classes*. An increase in method invocations results in an increase of *IFN*. Such occurrence could be a result of worse overall cohesion on the project making it difficult to identify services that are also more cohesive at an interface level.

Considering that no strong correlations were found, further analysis regarding both ratios was conducted. For each ratio a box plot is calculated and according to the distribution of projects along ratios groups were formed. In short, for each ratio, a group was created for the following intervals: minimum to first quartile (Q1), first quartile to median, median to third quartile (Q3) and finally third quartile to maximum value. The main goal is to assess if certain ranges of a given ratio perform better for metrics than the others. Box plot of both ratios is illustrated by Figure 15. Each group represents roughly 25% of the applications. Given the huge variability of values in the last group, creating other groups from that was considered, but since that last group did not showed any particular relevant variability to others such action was not executed.

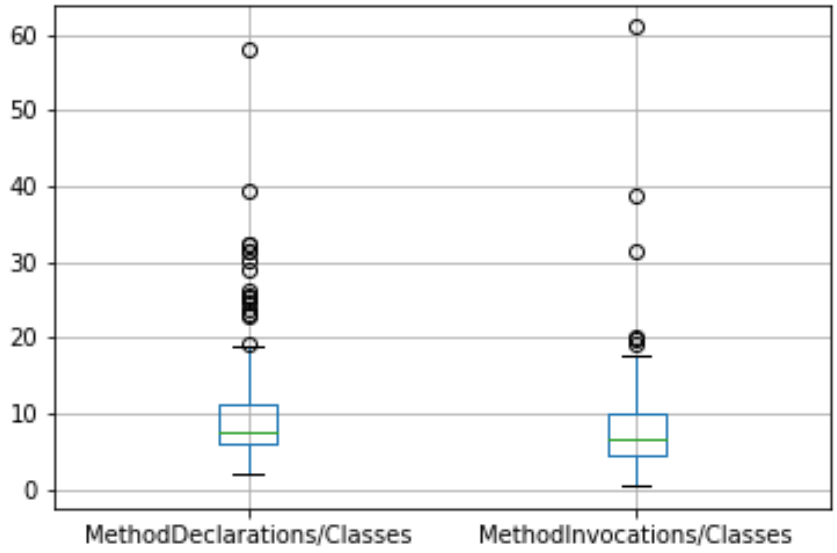



Figure 15: Box plot of both ratios

In the following section the box plot of metrics for both ratios are presented, each box plot is associated to a group of roughly 25% of the applications previously described. For instance, *CHM1* represents the first group, *CHM2* the second, and so on.

*Method Declarations / Classes*

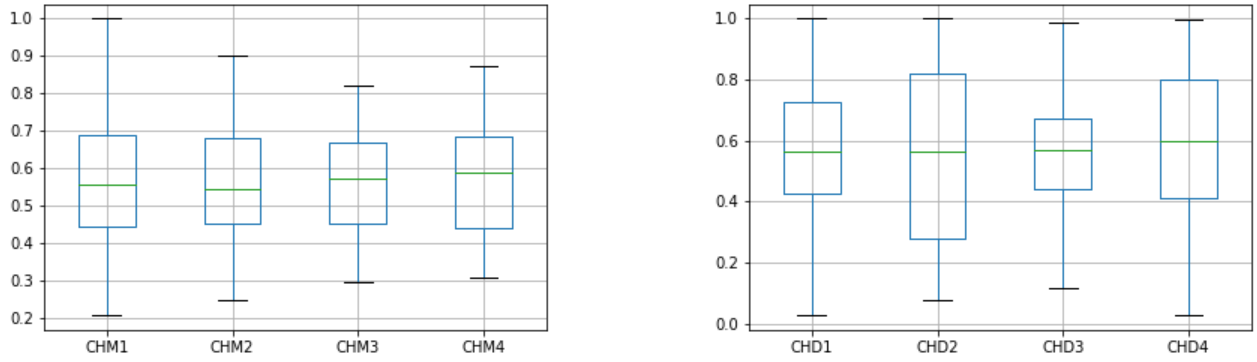



Figure 16: Box plot for CHM (left) and CHD (right) across ranges of *Method Declarations / Classes*

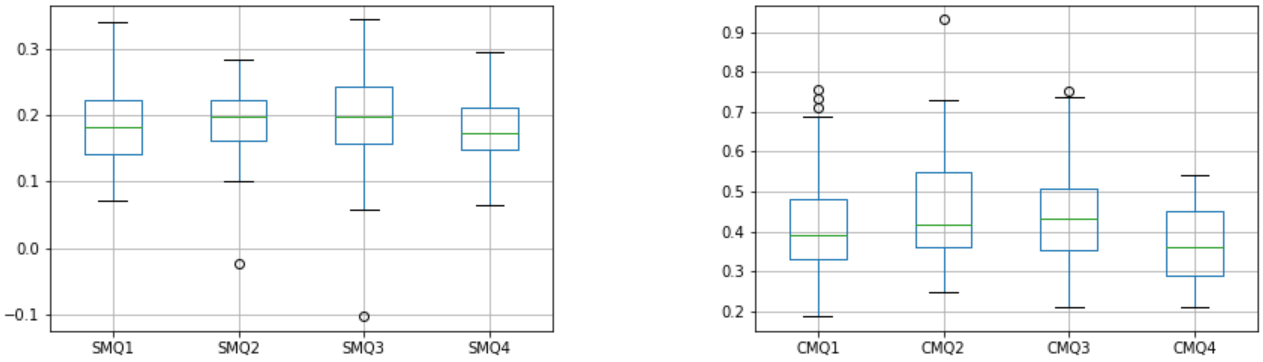



Figure 17: Box plot for SMQ (left) and CMQ (right) across ranges of *Method Declarations / Classes*

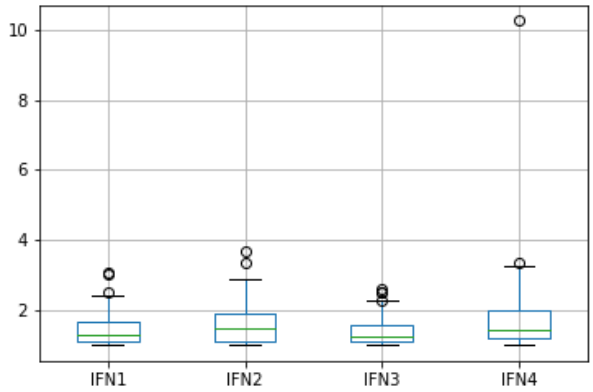



Figure 18: Box plot ranges of IFN across ranges of *Method Declarations / Classes*

*Method Invocations / Classes*

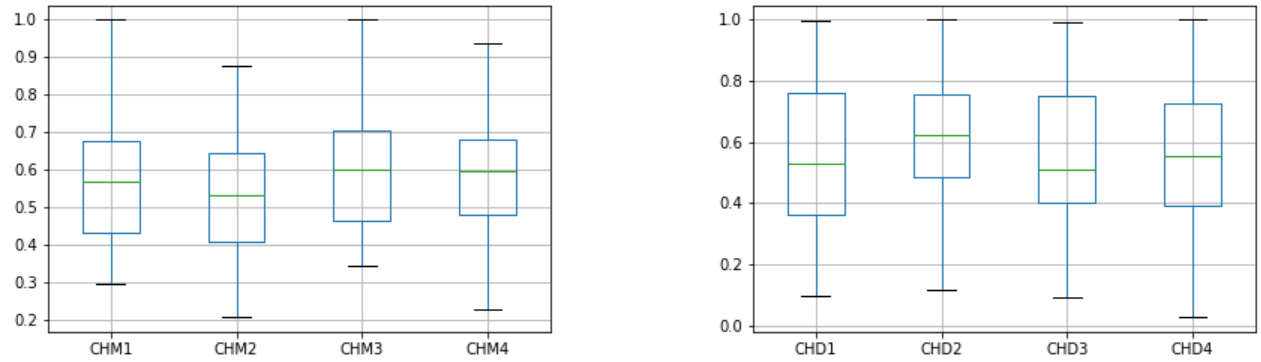



Figure 19: Box plot for CHM (left) and CHD (right) across ranges of *Method Invocations / Classes*

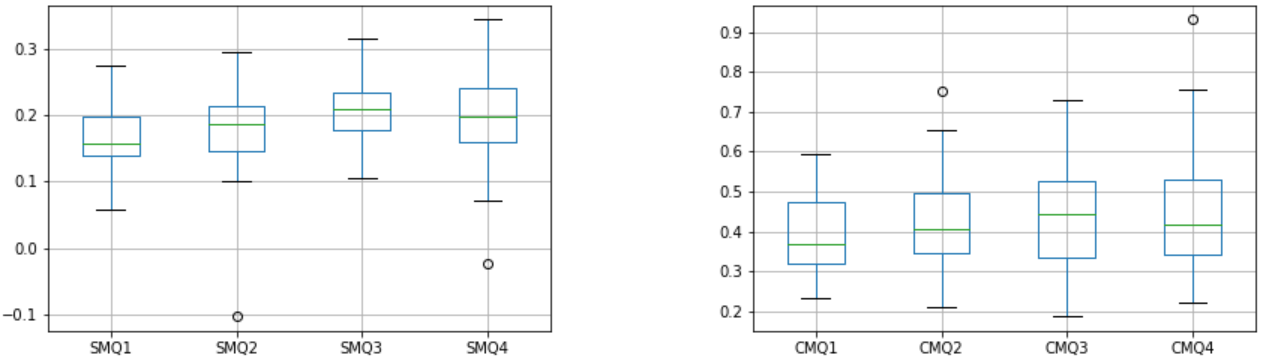



Figure 20: Box plot for SMQ (left) and CMQ (right) across

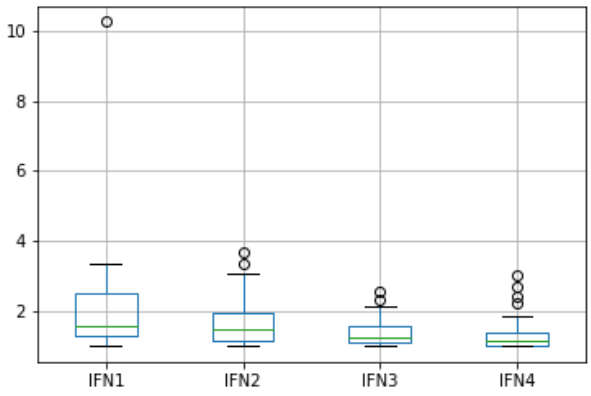



Figure 21: Box plot ranges of IFN across ranges of *Method Invocations / Classes*

Overall both ratios exhibits stable results across metrics and no differences were observed that could point in a considerable better performance of one ratio over another. Nevertheless, there are some minor behaviours that can be observed and contribute to further conclusions.

RATIO OF METHOD DECLARATIONS / CLASSES Regarding *Method Declarations / Classes*, *CHM's* median increases slightly towards higher values of the ratio. *CHD's* median shows a similar effect, although through a smoother increment. *SMQ's* best results are found at *SMQ2* and *SMQ3*, meaning that could be a range where *SMQ* performs slightly better and eventually has a drop in performance. *CMQ* performs similarly, there is a slightly improvement towards *CMQ3* in median and then a slightly drop in performance.

RATIO OF METHOD INVOCATIONS / CLASSES Regarding *Method Invocations / Classes*, *CHM* shows slightly better medians towards the higher range. *SMQ* shows an improvement towards *SMQ3* with a moderate improvement in median compared to *SMQ1*. *CMQ* follows a similar pattern an increase towards *CMQ3* resulting in a moderate increase compared to *CMQ1*. *IFN* exhibits a more cohesive box plot towards *IFN4*.

NUMBER OF CLASSES Similarly to the analysis previously conducted on two ratios, we now conduct an analysis on the number of classes. The goal is to understand if projects of a particular range of classes behave better than others or if there is a trend regarding one of the metrics. The analysis is divided into groups of projects each group being composed of applications in a range of classes. For instance, *SMQ150* contains all projects with less than 150 classes and more than 30 (being the minimum), *SMQ300* contains projects between 150 and 300 classes. The last group, *SMQ2600* includes everything beyond 1050 classes and up to 2600 classes.

Considering that, out of the 200 applications previously taken into study there was a lack of projects on the higher range of classes (groups with less than 10 applications), 45 applications over 800 classes were executed and added to this analysis. The results are shown in Figures 22, 23, 24. Both *CHM* and *CHD* show considerable variance across groups and no further conclusions can be made. That is however expected, considering that *CHM* and *CHD* are measured at a different level of granularity (interfaces) instead of total classes. Total number of interfaces on each project might be more impactful compared to number of classes. *SMQ* shows a moderate drop from *SMQ150* to *SMQ300* followed by some variance roughly between 0.15 and 0.2 for other groups. *CMQ* shows a tendency in decrease towards *CMQ600* followed by a recovery in its values. *IFN* exhibits a tendency to worse results towards *IFN900* although followed by recovery in the last groups. Further discussion to this analysis is presented on Section 7.7.2.

*Metrics aggregated by number of classes*

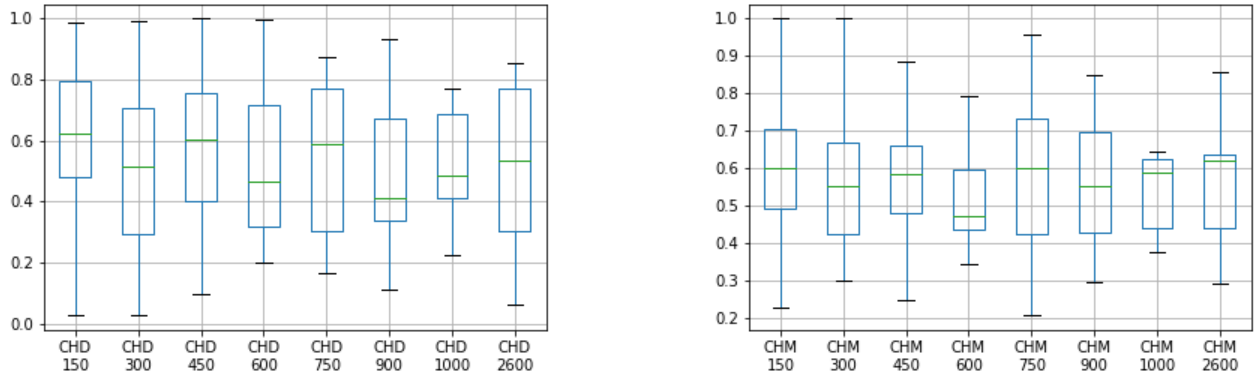



Figure 22: Box plot for CHM (left) and CHD (right) across ranges of classes

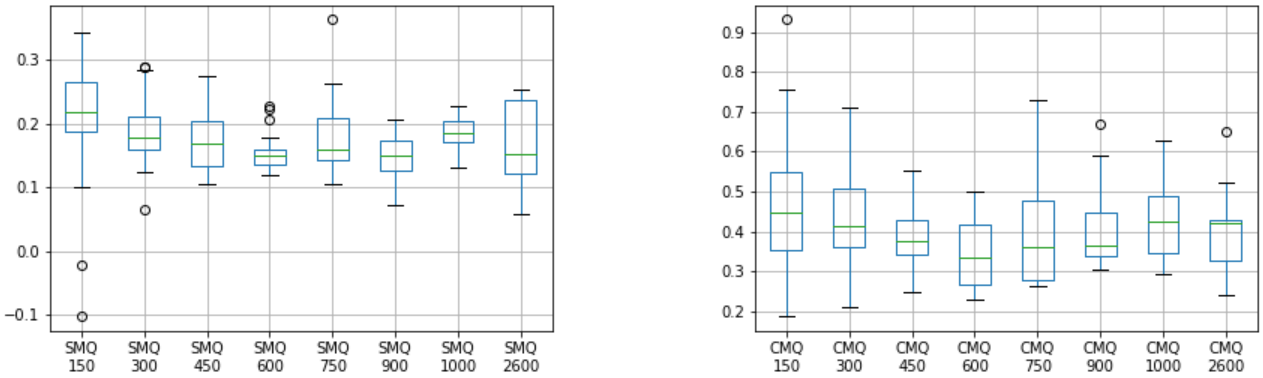



Figure 23: Box plot for SMQ (left) and CMQ (right) across ranges of classes

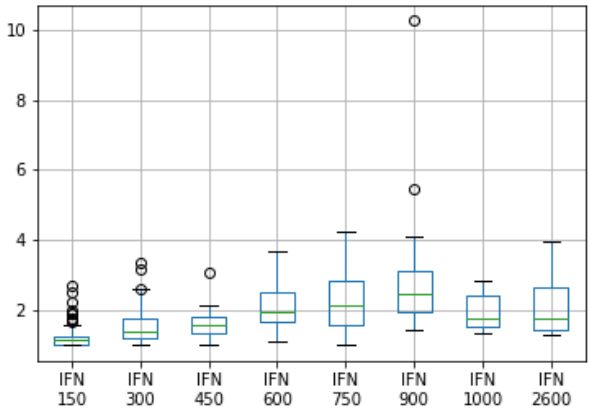



Figure 24: Box plot ranges of IFN across ranges of classes

### 7.7.2 *Resolution selection and analysis*

In this section some examples of the obtained metrics are presented as a way to identify what went well, what went wrong, what are the causes of specific metrics and what further information can we identify that could help users make educated decisions on choosing the most appropriate proposal for their perception of good microservices.

*Common patterns*

Firstly, a collection of the most common patterns of metric evolution across resolution are presented.

This collection represents some examples of possible plots of resolution across metrics, that are the easiest to select the best proposition of microservices according to the results in quantitative metrics. A strong tendency to a range of resolutions is also common, meaning higher quantitative metrics on that section.

Figure 25 presents a strong tendency regarding resolution identification. Its start has some instability but a peek is identified between 0.5 and 0.6 and after that the metrics represent a strong tendency towards worse services given the big jump from an absolute value of 3 to a final 1.25.

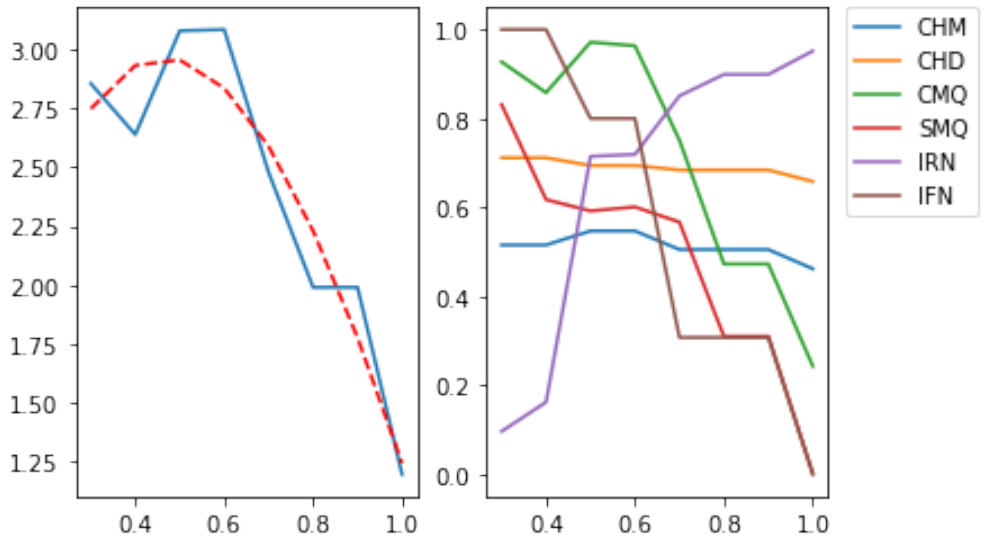



Figure 25: Metrics across resolution for 'biliob-backend' (200 classes)

Figure 26 illustrates a similar pattern, however the peek range of the best metrics is more pronounced given both higher and lower edges of resolution representing worse metrics. The maximum is in accordance to a resolution of 0.7, however, a range between 0.5 and 0.8 could be taken into account given different subjective perspectives to the domain in question.

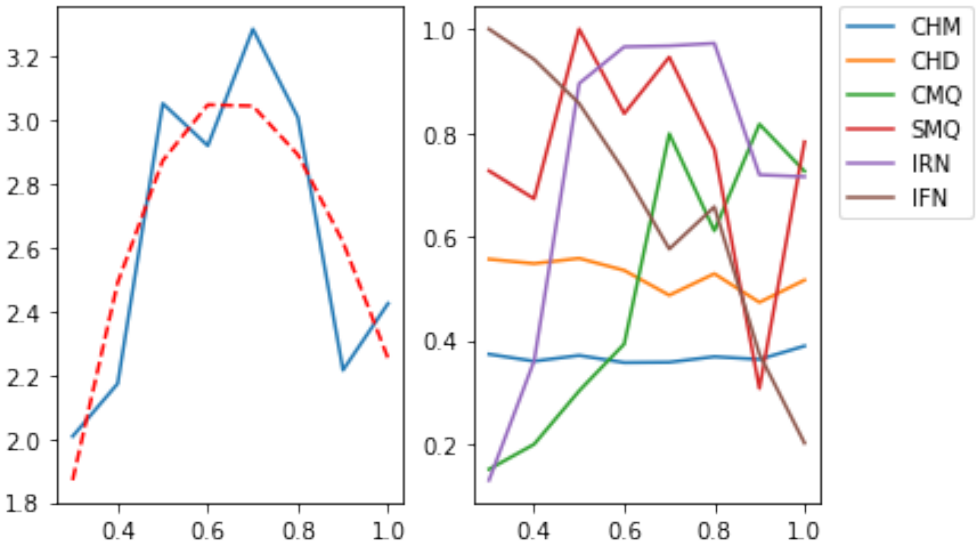



Figure 26: Metrics across resolution for 'jeecg-bpm' (810 classes)

Figure 27 represents a strong tendency towards lower resolutions. This can be explained by the total number of 2514 classes composing the project. Lower resolutions guarantee that services are smaller and are not composed of big chunks of the application. The increase in metrics at a resolution of 1 can be explained by the growth of *IRN*, *CHD* and the stabilisation of *CMQ*, remaining metrics stayed at lower values. *IRN* is expected to increase given less services. *CMQ* increase is not too significant considering that it had a drop on previous resolutions and it stays at around the same values.

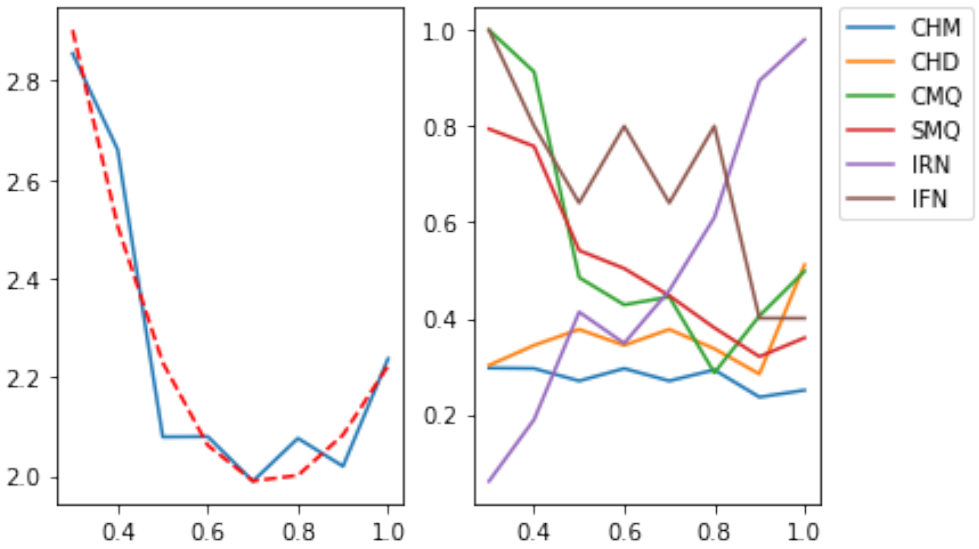



Figure 27: Metrics across resolution for 'open-cyclos' (2514 classes)

Figure 28 represents an interesting example because there is a range (ie. 0.6 - 0.7) where most metrics are stable meaning possible similar proposals of services both in quantitative metrics as well as in size.

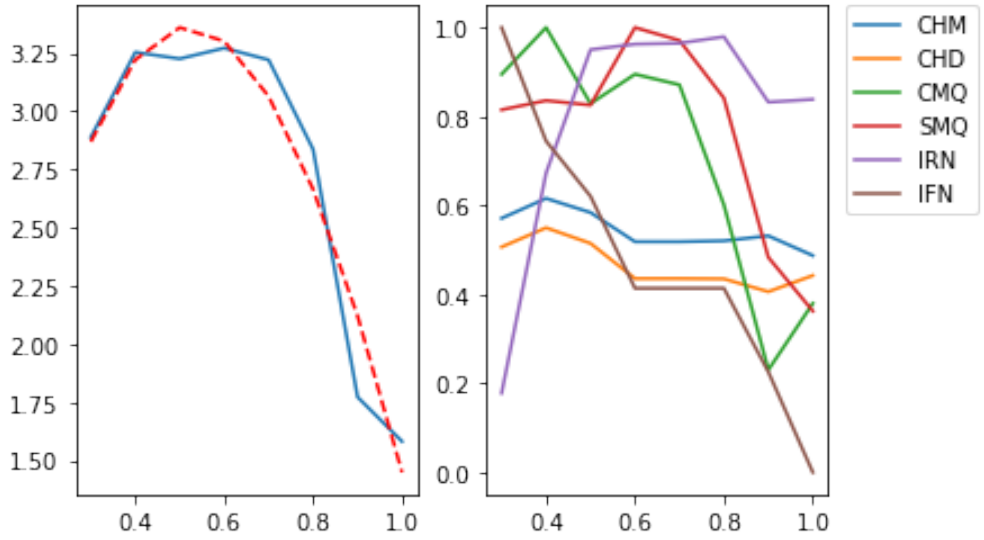



Figure 28: Metrics across resolution for 'oztrack' (213 classes)

*Outliers*

In this section examples harder to understand and infer adequate resolution are presented. The main reason for such occurrence could be noise, either by the randomness of the clustering algorithm causing unstable results or improper identification of topics resulting in some cases in higher variability of metrics.

Figure 29 illustrates a typical case of noise, where there is a tendency for better metrics towards the middle of the range, however there is a big gap at 0.6 caused by a significant drop in *CMQ* and *SMQ*.

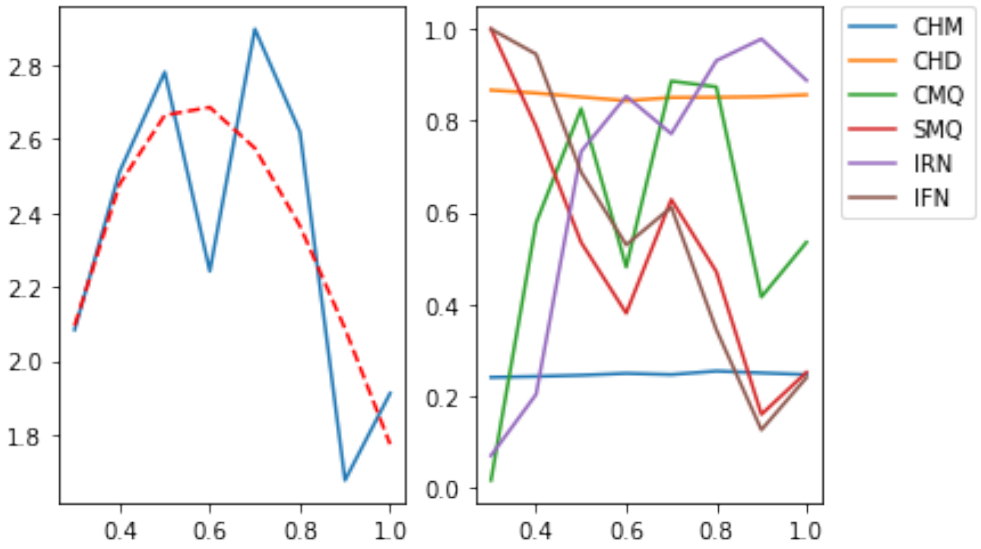



Figure 29: Metrics across resolution for 'ActivationCodeMall' (307 classes)

Figure 30 illustrates an example where at first it does not seem intuitive why there are two peeks, but by analysing each metric individually an informed decision can be made. The first peek, resolution 0.3, represents an extreme case where *IRN* is at its worse while *CMQ* and *SMQ* are at its best, which does not represent microservices of quality given such extremes. However, at the second peek, resolution of 0.8, overall metrics diverge to a similar point and represent proposals of microservices of higher quality and overall more cohesive and loosely coupled than the first peek.

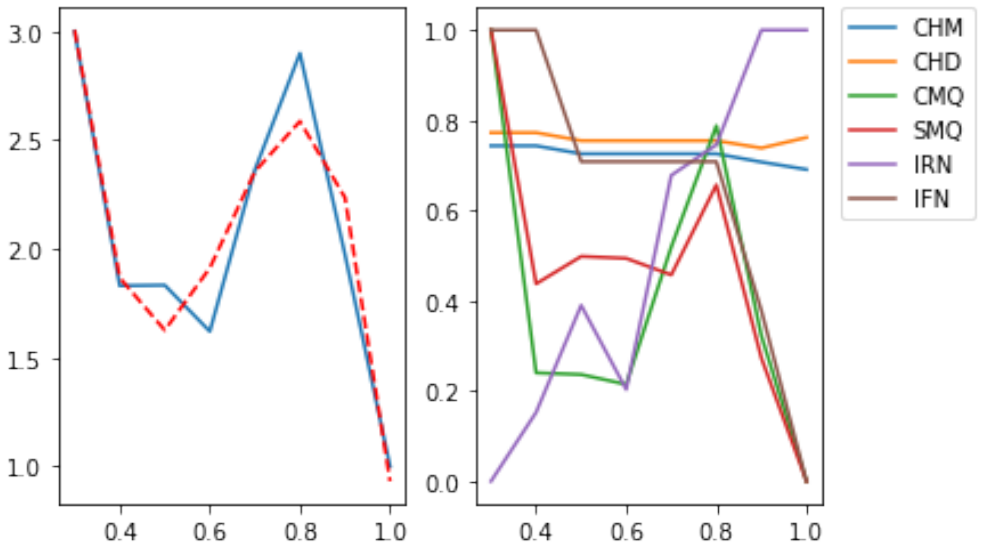



Figure 30: Metrics across resolution for 'api-manager' (181 classes)

Both Figure 31 and Figure 32 illustrate examples with multiple peeks on absolute values. Tipically examples with such variability and huge peeks are resultant of side effects of data normalisation and not necessarily hugely different services. On cases like this original absolute values of each metric should be analysed. For instance, *SMQ* for 'paladin-boot' has a minimum of 0.092 and a maximum of 0.129 despite such huge variability. Similarly *CMQ* has a minimum 0.231 of and a maximum of 0.287.

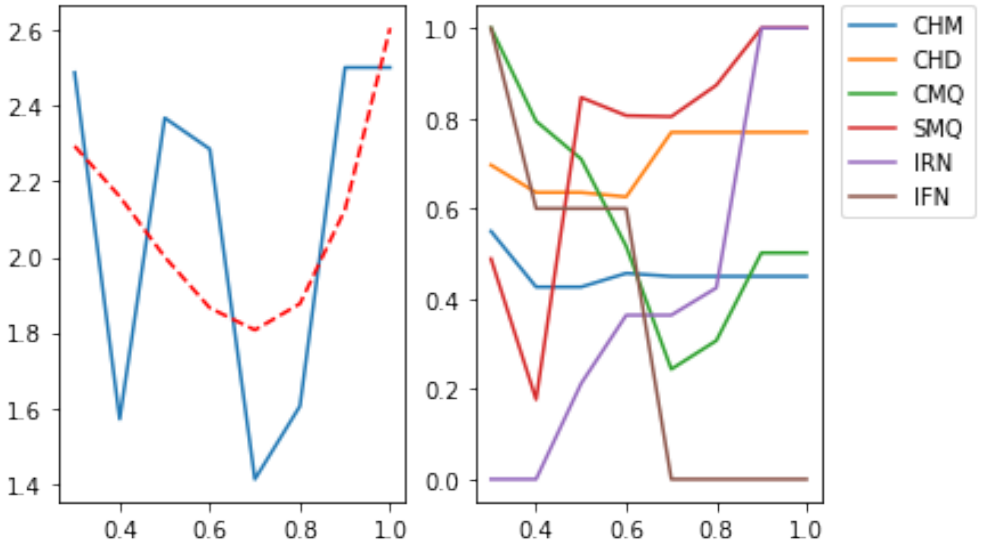



Figure 31: Metrics across resolution for 'CSC191' (32 classes)

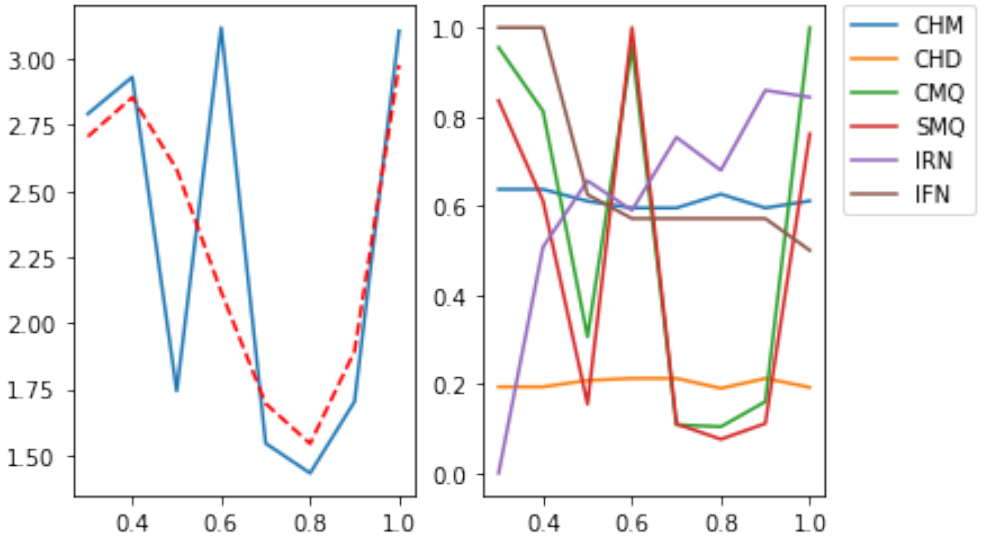



Figure 32: Metrics across resolution for 'paladin-boot' (328 classes)

Figure 33 represents patterns typically happening on small projects. Its reduced size results in equal proposals across multiple resolutions hence the constant metrics. The huge variation presented is a side effect of data normalisation instead of huge differences across resolutions.

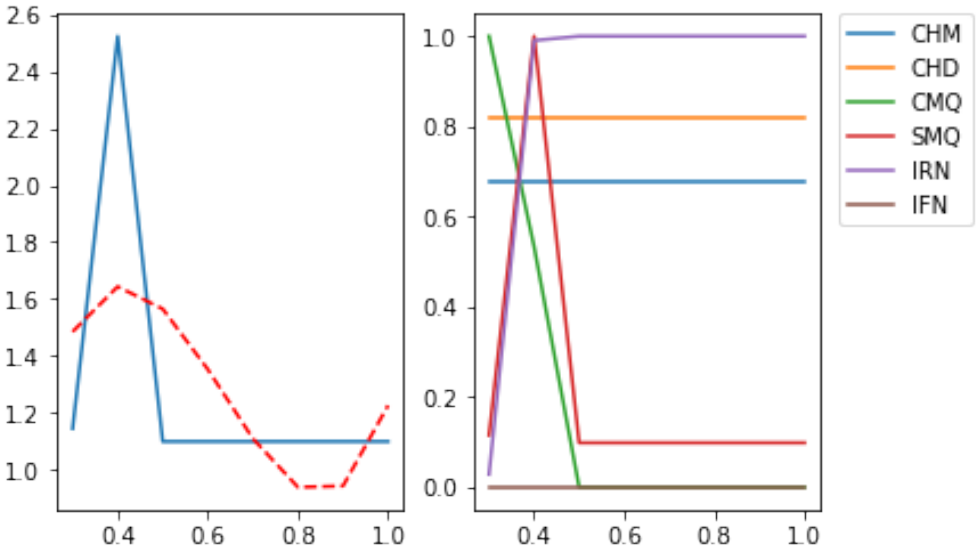



Figure 33: Metrics across resolution for 'H5APP-java' (39 classes)

Similar to Figure 30, Figure 34 represents a similar pattern where the metrics start high at low resolution, probably due to the tendency of higher values towards lower resolutions but the second peek represents a strong proposal given overall metrics.

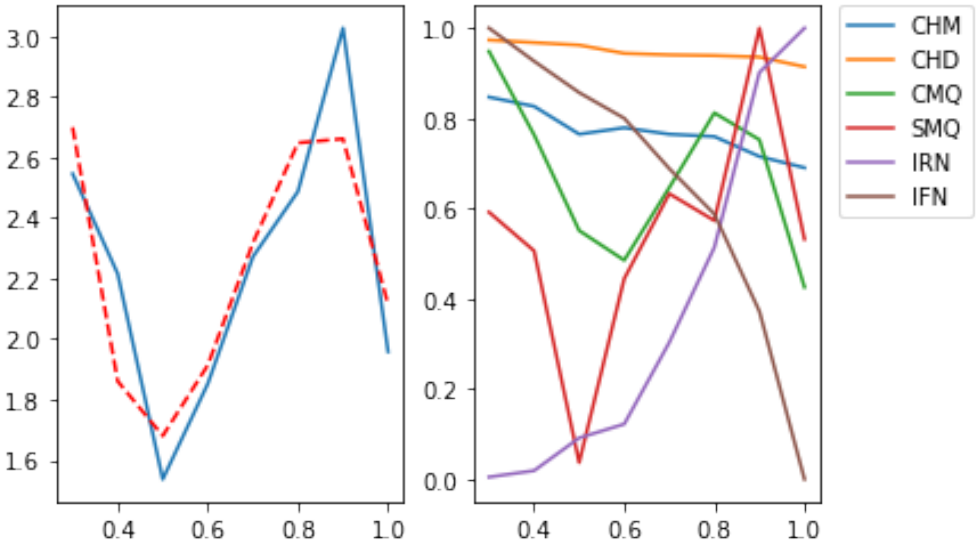



Figure 34: Metrics across resolution for 'WMSystem' (816 classes)

One of the biggest takeaways taken from this analysis is that metrics should be analysed individually as a mean to understand what is the best proposal as a whole. Picking the best proposal purely based on the best absolute value might be an obvious thing to do on strong tendencies, however each metric represents a different aspect of how components interact both internally and externally. Having strong metrics regarding an aspect while totally ignoring others does not provide strong proposals of microservices.

Another takeaway from this analysis is that *CHM* and *CHD* do not bring much value when identifying best resolution. The most probable cause of this is that *CHM* and *CHD* are based on the identified interfaces of each service. The identification of interfaces on its own is limited by the automated identification of endpoints across a large range of different applications. For better granularity on interface identification they should ideally be manually provided by the user. This does not mean *CHM* and *CHD* do not provide good information as an overall assessment of quality of microservices, bad segregation of interfaces still leads to low *CHM* and *CHD*. It is also important to note that the creation of new services will require the future addition of endpoints or a similar layer of communication. Another common pattern that the user should take into consideration is when *SMQ* and *CMQ* present a large variance between lower and higher resolutions compared to other metrics. This could mean that the range of resolution used is wider than it should be, and adapting it to a more appropriate range could help reasoning about the results. It is also important to note that the presented metrics are normalised across all clusters created, and sometimes big spikes are just a consequence of data normalisation and not necessarily big metric change. Apart from this analysis the user should be aware of the non normalised metrics to avoid this pitfall.

One of the first questions that might arise is why a comparison with the current state of the art solutions was not conducted? There are essentially to approaches that we could follow regarding that question: either follow a comparison with the current proposed solution with a very limited amount of projects (6 projects to be exact) and analysed with its own proposed metrics, or use those metrics and conduct a more extensive study that could in the future ideally serve as a comparison to other proposed solutions. We opted to follow the second path as we believe it presents more value to the state of the art and discussion in this field.

Regarding metrics of independence of functionality, both *CHM* and *CHD* presented good median values, however there is a big interval between upper and lower whisker. The upper whisker results might be more frequent on smaller applications and applications better designed with strong domain concepts. The lower whisker could be described as projects of larger complexity, with weaker domain concepts or an overall lack of pre-processing and cleaning of domain terms.

The higher values of *CMQ* compared to *SMQ* could be justified by the usage of an approach mainly based on the extraction of lexical terms. The underlying structure of the graph for clustering is based on the structural dependencies, however, edge weights when based on semantics will privilege classes semantically closer. Both *SMQ* and *CMQ* are bound between -1 and 1, despite some negative outliers on *SMQ* the results obtained are positive and are relevant regarding the modularity of proposed services. Higher values of *CMQ* over *SMQ* are expected given the semantic-focused approach of our methodology. Another reason for lower SMQ is higher levels of abstraction which are discussed later.

Through further analysis on correlation of two ratios to metrics, and metrics analysis on ranges of classes some considerations were identified. There is not a particular group that represents major difference in results to others since our method works with similar performance across a wide variety of variants regarding project structure. Nevertheless, some tendencies and groups with slightly better/worse performance were identified. The most significant are observed in the comparison of metrics across classes specially regarding *SMQ* and *CMQ*. Both show better results in the first group composed of less classes. This is expected considering those projects are less complex, hence identification of coherent topics is easier as well as the maximisation of modularity on Louvain clustering. It is however important to note that the downtrend stabilises at some point instead of continuously decreasing towards negative values. We took measures to mitigate preliminary results since some groups had a low amount of projects. We increased each group to a minimum of 15 projects, however that might not be enough to mitigate some of the instability resultant of wide amount of variants introduced by projects of different domains and differences in design.

Regarding IFN, our proposed tool seems to have some tendency to choose smaller services (due to metrics combination) resulting in better *IFNs* and worse *SMQ* (*i.e.* with smaller microservices it is expected that more external connections exist).

An interesting point inherited by the topic modelling technique employed is that it is language agnostic. We observed projects of different spoken languages (i.e. English, Spanish, Portuguese, etc.) and, as long as the vocabulary stays coherent and consistent across the domain, it should not make a difference on how it performs. Going even further, some projects were heavily composed of abbreviations to the point where it becomes hard to read and understand its domain. Similarly, as long as those abbreviations form a consistent vocabulary the method should perform similarly.

Considering that we did not apply customised pre-processing for each project, when there are levels of abstraction composed of multiple components, there is a possibility that those same abstractions are identified as a topic and eventually result in a service. Better pre-processing would definitely help, however there will always be cases where identified topics do not represent the reality of the domain. Allowing the user to discard topics that

purely represent abstractions or are composed by very scattered terms could bring benefits to the process of service identification.

Our approach uses classes as the unit of decomposition of a project. Even though we can identify multiple classes from a java file, such granularity might be too high to identify cross-cutting concepts and segregate them into their unique services. Using finer units of decomposition such as methods might help the identification of such cross-cutting concepts and improve the overall identification of domain terms and consequently better cohesion and loosely coupling.

The current value of our proposed tool for developers is mainly to explore an architecture and guide the user to identify microservices according to metrics of independence of functionality and modularity. Although we identify a resolution at which the independence of functionality and modularity are at their highest, choosing the adequate number of services is dependent on the subjective understanding of what represents good microservices to the expert. Considering that each cluster of classes results in a direct proposition of a microservice the trade-off between cohesion and coupling is an important feature that the expert should evaluate. In short, choosing lower resolution, hence smaller microservices, results inherently in higher cohesion with higher coupling, while higher resolutions results in the opposite. The appropriate balance between cohesion and coupling at a class level is not that straight-forward to identify, hence the ultimate decision should be made by the expert.

We hypothesise that the main cause of success or failure to the methodology is abstraction, considering that it hinders topic identification and in some cases the identification of abstraction as a topic might result in isolating such abstraction into an independent proposal of a service. Although abstraction usually increases with bigger projects, our data set is composed of applications wildly varying in their domain, hence, a smaller but more complex application could have more abstraction than a large application more focused on the domain. Further analysis to measure the level of abstraction of projects and its correlation to metrics would have to be conducted to confirm our hypothesis.

Overall, all the metrics demonstrate promising results towards microservice identification regarding independence of functionality and modularity across a wide set of projects of different domains and complexity.

## 7.8 PERFORMANCE

Undergoing migration to microservices is a time consuming process that can take months or even years to complete. The sole process of identification of possible services is helpful and may speed up the whole process. Considering that the process is extensive there is no need to provide the user with real-time proposals. It will be perfectly acceptable for the user

to run the tool for a couple of hours or days as long as quality services are being proposed. In any case, we want to give a picture of the time our tool takes to compose such proposal.

The machine we used for such execution is composed of an `i7 8550U` processor and `16GB` of RAM. In Figure 35 the box plots of execution time distributed by classes is presented for the 200 applications previously considered for analysis under the same parameters. Each box plot represents a range of projects by quantity of classes. For instance, the first box plot is composed of projects containing a minimum of 30 classes and a maximum of 499 classes.

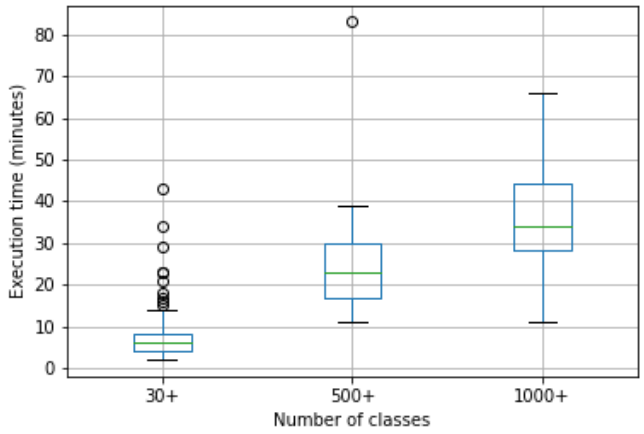



Figure 35: Box plot of execution time in minutes across groups of classes

Although box plots by number of classes are shown and a pattern can be identified regarding performance over number of classes, to be more precise, the properties that most affect the performance are the total number of lexical terms (influencing duration of topic identification) and complexity of project. Regarding the latter there are moderate improvements that can be employed. The tool was initially built as a unit to perform topic modelling for a specific number of topics and clustering for a specific resolution. Further improvements on optimal topic identification through coherence metrics and finding of optimal resolution were built on top of the existing solution. Although most of the data being reused on different iterations of the process is being cached and used efficiently, the process of collecting metrics out of each proposed service based on resolution requires parsing the project from the ground. Improving such procedure will definitely improve performance, however it should not be a drastic change as identifying topics and measuring their coherence remains as the slowest step of the process.

## 7.9 THREATS TO VALIDITY

In this section we discuss the threats to the proposed method organised according to Wohlin et al. (2012).

CONCLUSION VALIDITY. A possible threat is related to the reliability of measures, in this case of the metrics. *CHM* and *CHD* implementations were provided by their authors, however the remaining were implemented according their original publication, and when in doubt we have contacted the authors to discuss and clarify them. We have also performed extended tests to guarantee the correctness of the results.

INTERNAL VALIDITY. Another threat to validity refers to how the parameters of the study are selected. Ideally we should do an extensive analysis of how the number of topics and resolution affect the results directly, however, that would require a vast amount of work given the high number of applications and possible permutations. To mitigate such threat, we defined arbitrary ranges for each parameter. Regarding the resolution parameter, the selected arbitrary range demonstrated a level of granularity capable of identifying small microservices in large applications as well as larger microservices in smaller applications, in other words, it should have the capability to identify different levels of granularity in different sized applications. Ultimately, resolution is tested against metrics and we can understand how it performs. Unfortunately the same cannot be done with the number of topics. To identify the appropriate number of topics we resort to a metric of coherence which evaluates if the terms of each topic make sense together. This is applied on an arbitrary range of number of topics. However, the ultimate contribution of the number of topics to the method can only be evaluated after clustering and no further individual conclusions can be achieved.

CONSTRUCT VALIDITY. A construct validity threat relates to the quality of microservices being proposed. Even though we used the state of the art metrics regarding microservices, it is theoretically possible, however unlikely, to achieve good metrics that do not necessarily represent good proposals of microservices. The amount of projects taken into consideration should decrease such possibility, however, a qualitative analysis of the metrics used would have to be conducted in order to make further conclusions. Bringing experts to conduct an analysis of the decomposition we propose would also help understand the quality of such proposals, and help identifying possible improvements to the overall process.

EXTERNAL VALIDITY. An external threat relates to the architectures of projects we used. Our goal was to take monolithic applications and thus it was necessary to filter out

projects composed of other architectures such as SOA and MA. Repositories built upon such architectures are often composed of multiple projects, hence multiple `src` folders. To mitigate such occurrences only projects containing one `src` folder were considered given the definition of monolithic applications as being composed of a single program. However, there is no absolute guarantee that all the projects considered follow the definition of a monolith. A second threat relates to the fact that we use only open-source projects. Nevertheless, it is now common to find companies and other entities making their code available. For instance, our list of projects includes a project by the Australian Government (`AtlasOfLivingAustralia/biocache-service`). However, it is possible the results may vary for proprietary software.

# 8

# CONCLUSION

We present a methodology to identify microservices from monolithic software architectures. The methodology proposed is agnostic of the programming language and paradigm. We have implemented the methodology in a proof of concept and tested against the state of the art quantitative metrics on independence of functionality and modularity of microservices. The evaluation was conducted against the collection of 200 open-source Java Spring applications from GitHub.

The proposed methodology performed well regarding independence of functionality with medians roughly close to 0.6 for both *CHM* and *CHD* and low values of *IFN* representing relevant proposals of microservices. The results concerning modularity are also positive, with better performance regarding the *CMQ* over *SMQ* given the nature of our semantic based approach. The overall results are positive and lay a foundation for the usage of topic modelling techniques on microservices identification.

## 8.1 CONTRIBUTIONS

The field of automated microservice identification and migration is relatively new with a few proposed methodologies and even less working prototypes or proper extensive validation by metrics or qualitative analysis. In order to mitigate some of those gaps the following contributions were carried out:

- Improvements on the weak consideration of domain terms on the current state of art through the usage of topic modelling.
- A methodology capable of identifying proposals of microservices with a very low need for user input and language agnostic.
- Implementation of a prototype capable of applying the proposed methodology for Java Spring web applications, however extendable to other languages as long as the proper data is fed into the algorithm.

- Extensive study on 200 open-source applications and five quantitative state of the art metrics regarding microservices' independence of functionality and modularity.

## 8.2 FUTURE WORK

The present work has its limitations, hence the realisation of specific analysis and improvements would be beneficial to improve its quality. Some points for improvement and future work are presented below:

- Conduct an in-depth analysis of topic modelling methods extended from LDA in order to understand if there is a benefit in using them. For instance, the Author Topic Modelling, extended from LDA, could be used to infer topics based on the authoring of code by a specific developer, and according to those identifications and Parnas thesis infer clusters of services?
- Conduct a qualitative study in order to proper understand the impact of metrics regarding the quality of the proposed services. Such study should also allow to identify to what extent each metric contributes to the proposal of services and for the improvement of the selection of the proposal being presented to the user.
- For ultimate evaluation the proposals of microservices should be taken into consideration and a migration conducted to a working version of a microservice architecture.

# BIBLIOGRAPHY


About the symbol solver · javaparser/javaparser wiki. https://github.com/javaparser/javaparser/wiki/About-the-Symbol-Solver. (Accessed on 14/07/2020).

Nexhati Alija. Justification of software maintenance costs. *International Journal of Advanced Research in Computer Science and Software Engineering*, 7:15–23, 03 2017. doi: 10.23956/ijarcsse/V7I2/01207.

Armin Balalaie, Abbas Heydarnoori, Pooyan Jamshidi, Damian Tamburri, and Theodore Lynn. Microservices migration patterns. *Software: Practice and Experience*, 48, 07 2018. doi: 10.1002/spe.2608.

Samir Behara. Making your microservices resilient and fault tolerant, Aug 2018. URL https://samirbehara.com/2018/08/06/making-your-microservices-resilient-and-fault-tolerant/. (Accessed on 20/12/2019).

David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3(null):993–1022, March 2003. ISSN 1532-4435.

Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, oct 2008. doi: 10.1088/1742-5468/2008/10/p10008. URL https://doi.org/10.1088%2F1742-5468%2F2008%2F10%2Fp10008.

H. Borges, A. Hora, and M. T. Valente. Understanding the factors that impact the popularity of github repositories. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 334–344, 2016.

Bulkhead Pattern. Bulkhead pattern - cloud design patterns. URL https://docs.microsoft.com/en-us/azure/architecture/patterns/bulkhead. (Accessed on 20/07/2020).

Ivan Candela, Gabriele Bavota, Barbara Russo, and Rocco Oliveto. Using cohesion and coupling for software remodularization: Is it enough? *ACM Transactions on Software Engineering and Methodology*, 25:1–28, 06 2016. doi: 10.1145/2928268.

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

Rui Chen, Shanshan Li, and Zheng (Eddie) Li. From monolith to microservices: A dataflow-driven approach. pages 466–475, 12 2017. doi: 10.1109/APSEC.2017.53.

Melvin Conway. Conway's law, 1967. URL http://www.melconway.com/Home/Conways_Law.html. (Accessed on 27/12/2019).

Zhamak Dehghani. How to break a monolith into microservices. https://martinfowler.com/articles/break-monolith-into-microservices.html, 2018. (Accessed on 26/12/2019).

Bogdan Dit, Meghan Revelle, Malcom Gethers, and Denys Poshyvanyk. Feature location in source code: A taxonomy and survey. *Journal of Software Maintenance and Evolution: Research and Practice*, 25, 01 2013. doi: 10.1002/smr.567.

Santo Fortunato and Marc Barthélemy. Resolution limit in community detection. *Proceedings of the National Academy of Sciences*, 104(1):36–41, 2007. ISSN 0027-8424. doi: 10.1073/pnas.0605965104. URL https://www.pnas.org/content/104/1/36.

Martin Fowler. Monolith first. https://martinfowler.com/bliki/MonolithFirst.html, June 2015. (Accessed on 26/12/2019).

Martin Fowler and James Lewis. Microservices. https://martinfowler.com/articles/microservices.html, March 2014. (Accessed on 27/12/2019).

Jonas Fritzsch, Justus Bogner, Stefan Wagner, and Alfred Zimmermann. Microservices migration in industry: Intentions, strategies, and challenges. 10 2019. doi: 10.1109/ICSME.2019.00081.

Michelle Girvan and Mark Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences of the United States of America*, 99:7821–7826, 11 2001.

Michael Gysel, Lukas Kölbener, Wolfgang Giersche, and Olaf Zimmermann. Service cutter: A systematic approach to service decomposition. pages 185–200, 09 2016. doi: 10.1007/978-3-319-44482-6_12.

Brian Henderson-Sellers, Jolita Ralyte, Pär Ågerfalk, and Matti Rossi. *Situational Method Engineering*. 01 2014. ISBN 978-3-642-41466-4. doi: 10.1007/978-3-642-41467-1.

Hamed Jelodar, Yongli Wang, Chi Yuan, Xia Feng, Xiahui Jiang, Yanchao Li, and Liang Zhao. Latent dirichlet allocation (lda) and topic modeling: Models, applications, a survey. *Multimedia Tools Appl.*, 78(11):15169–15211, June 2019. ISSN 1380-7501. doi: 10.1007/s11042-018-6894-4. URL https://doi.org/10.1007/s11042-018-6894-4.

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

W. Jin, T. Liu, Y. Cai, R. Kazman, R. Mo, and Q. Zheng. Service candidate identification from monolithic systems based on execution traces. *IEEE Transactions on Software Engineering*, pages 1–1, 2019. ISSN 1939-3520. doi: 10.1109/TSE.2019.2910531.

M. Kamimura, K. Yano, T. Hatano, and A. Matsuo. Extracting candidates of microservices from monolithic application code. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, pages 571–580, Dec 2018. doi: 10.1109/APSEC.2018.00072.

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

Pooja Kherwa and Poonam Bansal. Topic modeling: A comprehensive review. *ICST Transactions on Scalable Information Systems*, 7:159623, 07 2018. doi: 10.4108/eai.13-7-2018.159623.

K. Kobayashi, M. Kamimura, K. Kato, K. Yano, and A. Matsuo. Feature-gathering dependency-based software clustering using dedication and modularity. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pages 462–471, Sep. 2012. doi: 10.1109/ICSM.2012.6405308.

Kenichi Kobayashi, Manabu Kamimura, Keisuke Yano, Koki Kato, and Akihiko Matsuo. Sarf map: Visualizing software architecture from feature and layer viewpoints, 2013.

Renaud Lambiotte, Jean-Charles Delvenne, and Mauricio Barahona. Random walks, markov processes and the multiscale modular organization of complex networks. *IEEE Transactions on Network Science and Engineering*, 1(2):76–90, Jul 2014. ISSN 2327-4697. doi: 10.1109/tnse.2015.2391998. URL http://dx.doi.org/10.1109/TNSE.2015.2391998.

Ian X. Y. Leung, Pan Hui, Pietro Liò, and Jon Crowcroft. Towards real-time community detection in large networks. *Physical review. E, Statistical, nonlinear, and soft matter physics*, 79 6 Pt 2:066107, 2009.

W. Ma, L. Chen, Y. Zhou, and B. Xu. What are the dominant projects in the github python ecosystem? In *2016 Third International Conference on Trustworthy Systems and their Applications (TSA)*, pages 87–95, 2016.

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

Abhishek Mishra. Demystifying louvain's algorithm and its implementation in gpu | by abhishek mishra | walmartlabs | medium. https://medium.com/walmartlabs/demystifying-louvains-algorithm-and-its-implementation-in-gpu-9a07cdd3b010. (Accessed on 16/07/2020).

Joe Nemer. Advantages and disadvantages of microservices architecture. https://cloudacademy.com/blog/microservices-architecture-challenge-advantage-drawback/, 2019. (Accessed on 26/12/2019).

M. E. J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences of the United States of America*, 103(23):8577–8582, Jun 2006. ISSN 0027-8424. doi: 10.1073/pnas.0601602103. URL https://pubmed.ncbi.nlm.nih.gov/16723398. 16723398[pmid].

S. Newman. *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, 2015. ISBN 9781491950333. URL https://books.google.pt/books?id=jjl4BgAAQBAJ.

Claus Pahl and Pooyan Jamshidi. Microservices: A systematic mapping study. pages 137–146, 01 2016. doi: 10.5220/0005785501370146.

D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12):1053–1058, December 1972. ISSN 0001-0782. doi: 10.1145/361598.361623. URL http://doi.acm.org/10.1145/361598.361623.

Jean Petrić, Tracy Hall, and David Bowes. How effectively is defective code actually tested? an analysis of junit tests in seven open source systems. In *Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering*, PROMISE'18, page 42–51, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450365932. doi: 10.1145/3273934.3273939. URL https://doi.org/10.1145/3273934.3273939.

Selva Prabhakaran. Gensim topic modeling - a guide to building best lda models. https://www.machinelearningplus.com/nlp/topic-modeling-gensim-python/. (Accessed on 10/05/2020).

Greg Rafferty. Lda on the texts of harry potter. topic modeling with latent dirichlet allocation. https://towardsdatascience.com/basic-nlp-on-the-texts-of-harry-potter-topic-modeling-with-latent-dirichlet-allocation-~:text=The%20difference%20between%20Mallet%20and,Latent%20Dirichlet%20Allocation%20via%20Mallet. (Accessed on 03/09/2020).

Sara Rahiminejad, Mano R. Maurya, and Shankar Subramaniam. Topological and functional comparison of community detection algorithms in biological networks. *BMC Bioinformatics*, 20(1):212, Apr 2019. ISSN 1471-2105. doi: 10.1186/s12859-019-2746-0. URL https://doi.org/10.1186/s12859-019-2746-0.

RebelLabs. Developer productivity report 2017: Java tools usage | rebel. https://www.jrebel.com/blog/java-development-tools-usage-stats, September 2017. (Accessed on 26/01/2020).

Michael Röder, Andreas Both, and Alexander Hinneburg. Exploring the space of topic coherence measures. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, WSDM '15, page 399–408, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450333177. doi: 10.1145/2684822.2685324. URL https://doi.org/10.1145/2684822.2685324.

David Sarmento. Chapter 22: Correlation types and when to use them. https://ademos.people.uic.edu/Chapter22.html, 10 2017. (Accessed on 26/09/2020).

Carson Sievert and Kenneth Shirley. LDAvis: A method for visualizing and interpreting topics. In *Proceedings of the Workshop on Interactive Language Learning, Visualization, and Interfaces*, pages 63–70, Baltimore, Maryland, USA, June 2014. Association for Computational Linguistics. doi: 10.3115/v1/W14-3110. URL https://www.aclweb.org/anthology/W14-3110.

Framework Spring. Spring framework. https://spring.io/. (Accessed on 30/07/2020).

Keith Stevens, Philip Kegelmeyer, David Andrzejewski, and David Buttler. Exploring topic coherence over many models and many topics. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 952–961, Jeju Island, Korea, July 2012. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/D12-1087.

X. Sun, X. Liu, B. Li, Y. Duan, H. Yang, and J. Hu. Exploring topic models in software engineering data analysis: A survey. In *2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pages 357–362, 2016.

Xiaobing Sun, Xiangyue Liu, Li Bin, Bixin Li, David Lo, and Lingzhi Liao. Clustering classes in packages for program comprehension. *Scientific Programming*, 2017:1–15, 01 2017. doi: 10.1155/2017/3787053.

Davide Taibi, Valentina Lenarduzzi, and Claus Pahl. Architectural patterns for microservices: A systematic mapping study. 03 2018. doi: 10.5220/0006798302210232.

Joost Visser. *Building Maintainable Software: Ten Guidelines for Future-proof Code*. O'Reilly Media, Incorporated, 2016. ISBN 9781491967423. URL https://books.google.pt/books?id=EFiYAQAACAAJ.

Maira Wenzel, John Parente, and Sabah Shariq. Communication in a microservice architecture | microsoft docs. https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/communication-in-microservice-architecture, 2020. (Accessed on 22/12/2019).

Claes Wohlin, Per Runeson, Martin Hst, Magnus C. Ohlsson, Bjrn Regnell, and Anders Wessln. *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated, 2012. ISBN 3642290434.

# A

# APPENDIX

## A.1 IDENTIFIED CLUSTERS FOR JPETSTORE BY RESOLUTION

### A.1.1 *Cluster resolution: 0.6*

```
Service 0
org.mybatis.jpetstore.mapper.LineItemMapper
org.mybatis.jpetstore.domain.LineItem

Service 1
org.mybatis.jpetstore.mapper.OrderMapper
org.mybatis.jpetstore.service.OrderService
org.mybatis.jpetstore.domain.Sequence
org.mybatis.jpetstore.domain.Order
org.mybatis.jpetstore.mapper.SequenceMapper

Service 2
org.mybatis.jpetstore.web.actions.CartActionBean
org.mybatis.jpetstore.domain.Cart
org.mybatis.jpetstore.domain.Item
org.mybatis.jpetstore.domain.CartItem
org.mybatis.jpetstore.mapper.ItemMapper

Service 3
org.mybatis.jpetstore.domain.Product
org.mybatis.jpetstore.service.CatalogService
org.mybatis.jpetstore.domain.Category
org.mybatis.jpetstore.mapper.ProductMapper
org.mybatis.jpetstore.mapper.CategoryMapper
```

```
org.mybatis.jpetstore.web.actions.CatalogActionBean

Service 4
org.mybatis.jpetstore.web.actions.AccountActionBean
org.mybatis.jpetstore.domain.Account
org.mybatis.jpetstore.mapper.AccountMapper
org.mybatis.jpetstore.service.AccountService

Service 5
org.mybatis.jpetstore.web.actions.OrderActionBean
org.mybatis.jpetstore.web.actions.AbstractActionBean
```

### A.1.2 *Cluster resolution: 0.7*

```
Service 0
org.mybatis.jpetstore.mapper.OrderMapper
org.mybatis.jpetstore.service.OrderService
org.mybatis.jpetstore.domain.Sequence
org.mybatis.jpetstore.domain.LineItem
org.mybatis.jpetstore.domain.Order
org.mybatis.jpetstore.mapper.SequenceMapper
org.mybatis.jpetstore.mapper.LineItemMapper

Service 1
org.mybatis.jpetstore.web.actions.CartActionBean
org.mybatis.jpetstore.domain.Cart
org.mybatis.jpetstore.domain.Item
org.mybatis.jpetstore.domain.CartItem
org.mybatis.jpetstore.mapper.ItemMapper

Service 2
org.mybatis.jpetstore.domain.Product
org.mybatis.jpetstore.service.CatalogService
org.mybatis.jpetstore.domain.Category
org.mybatis.jpetstore.mapper.ProductMapper
org.mybatis.jpetstore.mapper.CategoryMapper
org.mybatis.jpetstore.web.actions.CatalogActionBean
```

```
Service 3
org.mybatis.jpetstore.web.actions.AccountActionBean
org.mybatis.jpetstore.domain.Account
org.mybatis.jpetstore.mapper.AccountMapper
org.mybatis.jpetstore.service.AccountService

Service 4
org.mybatis.jpetstore.web.actions.OrderActionBean
org.mybatis.jpetstore.web.actions.AbstractActionBean
```

### A.1.3 *Cluster resolution: 0.8*

```
Service 0
org.mybatis.jpetstore.web.actions.CartActionBean
org.mybatis.jpetstore.domain.Cart
org.mybatis.jpetstore.domain.Item
org.mybatis.jpetstore.domain.LineItem
org.mybatis.jpetstore.domain.CartItem
org.mybatis.jpetstore.mapper.ItemMapper
org.mybatis.jpetstore.mapper.LineItemMapper

Service 1
org.mybatis.jpetstore.mapper.OrderMapper
org.mybatis.jpetstore.service.OrderService
org.mybatis.jpetstore.domain.Sequence
org.mybatis.jpetstore.web.actions.AbstractActionBean
org.mybatis.jpetstore.domain.Order
org.mybatis.jpetstore.mapper.SequenceMapper
org.mybatis.jpetstore.web.actions.OrderActionBean

Service 2
org.mybatis.jpetstore.domain.Product
org.mybatis.jpetstore.service.CatalogService
org.mybatis.jpetstore.domain.Category
org.mybatis.jpetstore.mapper.ProductMapper
org.mybatis.jpetstore.mapper.CategoryMapper
org.mybatis.jpetstore.web.actions.CatalogActionBean
```

```
Service 3
org.mybatis.jpetstore.web.actions.AccountActionBean
org.mybatis.jpetstore.domain.Account
org.mybatis.jpetstore.mapper.AccountMapper
org.mybatis.jpetstore.service.AccountService
```

### A.1.4 *Cluster resolution: 0.9*

```
Service 0
org.mybatis.jpetstore.web.actions.CartActionBean
org.mybatis.jpetstore.domain.Cart
org.mybatis.jpetstore.domain.Item
org.mybatis.jpetstore.domain.LineItem
org.mybatis.jpetstore.domain.CartItem
org.mybatis.jpetstore.mapper.ItemMapper

Service 1
org.mybatis.jpetstore.mapper.OrderMapper
org.mybatis.jpetstore.service.OrderService
org.mybatis.jpetstore.domain.Sequence
org.mybatis.jpetstore.web.actions.AbstractActionBean
org.mybatis.jpetstore.domain.Order
org.mybatis.jpetstore.mapper.LineItemMapper
org.mybatis.jpetstore.mapper.SequenceMapper
org.mybatis.jpetstore.web.actions.OrderActionBean

Service 2
org.mybatis.jpetstore.domain.Product
org.mybatis.jpetstore.service.CatalogService
org.mybatis.jpetstore.domain.Category
org.mybatis.jpetstore.mapper.ProductMapper
org.mybatis.jpetstore.mapper.CategoryMapper
org.mybatis.jpetstore.web.actions.CatalogActionBean

Service 3
org.mybatis.jpetstore.web.actions.AccountActionBean
org.mybatis.jpetstore.domain.Account
```

```
org.mybatis.jpetstore.mapper.AccountMapper
org.mybatis.jpetstore.service.AccountService
```

### A.1.5 *Cluster resolution: 1 & 1.1*

```
Service 0
org.mybatis.jpetstore.mapper.OrderMapper
org.mybatis.jpetstore.service.OrderService
org.mybatis.jpetstore.domain.Sequence
org.mybatis.jpetstore.domain.Order
org.mybatis.jpetstore.mapper.SequenceMapper
org.mybatis.jpetstore.web.actions.OrderActionBean

Service 1
org.mybatis.jpetstore.web.actions.CartActionBean
org.mybatis.jpetstore.domain.Cart
org.mybatis.jpetstore.domain.Item
org.mybatis.jpetstore.domain.LineItem
org.mybatis.jpetstore.mapper.LineItemMapper
org.mybatis.jpetstore.domain.CartItem
org.mybatis.jpetstore.mapper.ItemMapper

Service 2
org.mybatis.jpetstore.domain.Product
org.mybatis.jpetstore.service.CatalogService
org.mybatis.jpetstore.domain.Category
org.mybatis.jpetstore.mapper.ProductMapper
org.mybatis.jpetstore.mapper.CategoryMapper
org.mybatis.jpetstore.web.actions.CatalogActionBean

Service 3
org.mybatis.jpetstore.domain.Account
org.mybatis.jpetstore.web.actions.AbstractActionBean
org.mybatis.jpetstore.service.AccountService
org.mybatis.jpetstore.web.actions.AccountActionBean
org.mybatis.jpetstore.mapper.AccountMapper
```

## A.2 METRICS RESULTS OF GITHUB PROJECTS

*(Intentionally left blank. Content resumes on next page.)*

| GitHub Project | #Classes | #Stars | CHM | CHD | IFN | CMQ | SMQ | #Services |
|---|---|---|---|---|---|---|---|---|
| miansen/Roothub | 389 | 100 | 0.31 | 0.34 | 1.61 | 0.37 | 0.14 | 34 |
| huanglu20124/invoice | 73 | 95 | 0.69 | 0.55 | 1.0 | 0.28 | 0.14 | 9 |
| Lab41/Dendrite | 101 | 92 | 0.56 | 0.7 | 1.2 | 0.49 | 0.19 | 21 |
| pibigstar/parsevip | 90 | 87 | 0.76 | 0.43 | 1.36 | 0.22 | 0.38 | 15 |
| OCR4all/OCR4all | 47 | 86 | 0.81 | 0.99 | 1.27 | 0.64 | 0.26 | 16 |
| moocss/EasyCMS | 64 | 86 | 0.73 | 0.97 | 1.25 | 0.42 | 0.19 | 10 |
| Vino007/javaEEScaffold | 45 | 84 | 0.51 | 0.25 | 1.75 | 0.31 | 0.21 | 11 |
| GdeiAssistant/GdeiAssistant | 455 | 79 | 0.4 | 0.79 | 1.66 | 0.37 | 0.23 | 37 |
| purang-fintech/seppb | 430 | 79 | 0.43 | 0.65 | 1.43 | 0.4 | 0.18 | 33 |
| muralibasani/kafkawize | 99 | 70 | 0.57 | 0.63 | 1.0 | 0.38 | 0.08 | 14 |
| qianqianjun/Educational-management | 100 | 66 | 1.0 | 0.97 | 1.0 | 0.51 | 0.3 | 15 |
| wsk1103/movie-boot | 217 | 60 | 0.57 | 0.66 | 1.78 | 0.39 | 0.18 | 24 |
| JoeyBling/bootplus | 150 | 58 | 0.54 | 0.28 | 1.29 | 0.36 | 0.18 | 18 |
| forTribeforXuanmo/sword-forum | 79 | 57 | 0.36 | 0.44 | 1.0 | 0.19 | 0.13 | 14 |
| justinscript/travel.b2b | 515 | 51 | 0.47 | 0.31 | 2.21 | 0.25 | 0.11 | 31 |
| superman544/JavaOJSystem | 192 | 47 | 0.36 | 0.62 | 1.13 | 0.36 | 0.21 | 25 |
| 514840279/danyuan-application | 277 | 46 | 0.56 | 0.47 | 1.32 | 0.74 | 0.27 | 31 |
| iminto/baicai | 84 | 41 | 0.76 | 0.86 | 2.0 | 0.22 | 0.28 | 14 |
| krishagni/openspecimen | 1244 | 41 | 0.26 | 0.48 | 1.51 | 0.28 | 0.11 | 51 |
| justinscript/shopping.plat | 298 | 39 | 0.7 | 0.12 | 1.62 | 0.23 | 0.12 | 24 |
| cym1102/nginxWebUI | 82 | 39 | 0.46 | 0.52 | 1.13 | 0.25 | 0.27 | 16 |
| Jannchie/biliob_backend | 200 | 37 | 0.55 | 0.69 | 1.07 | 0.41 | 0.18 | 18 |
| INCF/eeg-database | 1007 | 36 | 0.35 | 0.28 | 2.71 | 0.39 | 0.1 | 46 |
| Lotharing/SDIMS | 112 | 34 | 0.67 | 0.93 | 1.15 | 0.62 | 0.04 | 17 |
| kanban/kanban-app | 140 | 34 | 0.4 | 0.64 | 2.0 | 0.48 | 0.2 | 18 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Frodez/BlogManagePlatform | 317 | 33 | 0.53 | 1.0 | 1.0 | 0.33 | 0.17 | 23 |
| finallysmile3/ExamSystem | 108 | 33 | 0.65 | 0.4 | 1.15 | 0.3 | 0.15 | 17 |
| cloudfoundry-attic/login-server | 133 | 32 | 0.55 | 0.48 | 1.0 | 0.26 | 0.21 | 13 |
| atlasapi/atlas | 1797 | 32 | 0.64 | 0.52 | 2.47 | 0.32 | 0.12 | 59 |
| metasfresh/metasfresh-webui-api-legacy | 1081 | 30 | 0.29 | 0.29 | 1.53 | 0.26 | 0.1 | 46 |
| tangdu/smh2 | 111 | 27 | 0.61 | 0.83 | 1.1 | 0.34 | 0.28 | 24 |
| qiao-zhi/jwxt | 595 | 26 | 0.49 | 0.72 | 1.82 | 0.26 | 0.15 | 60 |
| jiangzongyao/kettle-master | 70 | 24 | 0.71 | 0.55 | 1.09 | 0.27 | 0.24 | 15 |
| jdmr/mateo | 704 | 23 | 0.43 | 0.26 | 1.84 | 0.46 | 0.19 | 47 |
| shuxianfeng/movision | 886 | 21 | 0.86 | 0.45 | 2.62 | 0.32 | 0.19 | 47 |
| leluque/university-site-cms | 254 | 21 | 0.42 | 0.46 | 1.64 | 0.39 | 0.15 | 18 |
| zhangdaiscott/jeecg-nomaven | 596 | 19 | 0.35 | 0.52 | 2.06 | 0.23 | 0.12 | 41 |
| ghostxbh/uzy-ssm-mall | 89 | 19 | 0.59 | 0.61 | 1.36 | 0.55 | 0.08 | 17 |
| OpenGeoportal/OGP2 | 330 | 18 | 0.67 | 0.58 | 1.54 | 0.41 | 0.13 | 32 |
| litblank/hammer | 226 | 18 | 0.65 | 0.24 | 1.29 | 0.29 | 0.17 | 20 |
| choerodon/agile-service-old | 914 | 17 | 0.38 | 0.77 | 1.61 | 0.47 | 0.13 | 44 |
| kai8406/cmop | 237 | 17 | 0.55 | 0.34 | 1.68 | 0.33 | 0.16 | 23 |
| gliderwiki/glider | 329 | 15 | 0.61 | 0.59 | 1.39 | 0.39 | 0.17 | 31 |
| hsloooooooool/form_flow | 467 | 15 | 0.55 | 0.3 | 1.9 | 0.31 | 0.15 | 27 |
| mozammel/mNet | 148 | 15 | 0.57 | 0.45 | 1.0 | 0.43 | 0.25 | 22 |
| easy-ware/api-manager | 181 | 15 | 0.74 | 0.77 | 1.18 | 0.44 | 0.2 | 22 |
| lvr1997/ershoujiaoyi | 78 | 14 | 0.62 | 0.59 | 1.0 | 0.72 | 0.18 | 14 |
| Ryan–Yang/CBoard-boot | 228 | 13 | 0.52 | 0.28 | 1.5 | 0.42 | 0.19 | 23 |
| GZzzhsmart/P2Pproj | 275 | 13 | 0.37 | 0.37 | 1.29 | 0.48 | 0.17 | 37 |
| HIIT/dime-server | 91 | 13 | 0.38 | 0.71 | 1.33 | 0.27 | 0.12 | 16 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| doooyo/Weixin_Server | 303 | 13 | 0.8 | 0.88 | 1.58 | 0.34 | 0.22 | 25 |
| justinbaby/my-paper | 463 | 13 | 0.37 | 0.23 | 2.0 | 0.46 | 0.13 | 41 |
| edgexfoundry/core-data | 38 | 12 | 0.48 | 0.63 | 1.0 | 0.39 | 0.02 | 4 |
| mofadeyunduo/online-judge | 43 | 12 | 0.7 | 0.41 | 1.0 | 0.4 | 0.34 | 10 |
| MiniPa/cjs_ssms | 98 | 11 | 0.69 | 0.59 | 1.0 | 0.34 | 0.2 | 9 |
| AURIN/online-whatif | 429 | 11 | 0.47 | 0.75 | 1.14 | 0.35 | 0.12 | 25 |
| fishstormX/fishmaple | 173 | 11 | 0.54 | 0.55 | 1.2 | 0.42 | 0.29 | 23 |
| opendevstack/ods-provisioning-app | 146 | 11 | 0.44 | 0.08 | 1.0 | 0.39 | 0.27 | 19 |
| parasoft/parabank | 260 | 10 | 0.43 | 0.07 | 3.14 | 0.3 | 0.12 | 18 |
| MaritimeConnectivityPlatform/IdentityRegistry | 124 | 10 | 0.45 | 0.66 | 1.17 | 0.35 | 0.17 | 19 |
| zhangyanbo2007/youkefu | 830 | 10 | 0.59 | 0.87 | 3.28 | 0.31 | 0.17 | 46 |
| BCSquad/pmph | 953 | 10 | 0.57 | 0.5 | 1.52 | 0.5 | 0.19 | 59 |
| starqiu/RDMP1 | 60 | 10 | 0.9 | 0.86 | 1.25 | 0.24 | 0.21 | 7 |
| Propro-Studio/propro-server | 310 | 10 | 0.48 | 0.38 | 1.55 | 0.27 | 0.12 | 23 |
| yunchaoyun/active4j-flow | 219 | 9 | 0.43 | 0.57 | 1.28 | 0.35 | 0.15 | 27 |
| BCSquad/pmph_java_front | 369 | 9 | 0.72 | 0.76 | 1.47 | 0.47 | 0.17 | 42 |
| yorkmass/Yark-AdminMS | 69 | 9 | 0.47 | 0.7 | 1.1 | 0.33 | 0.19 | 15 |
| vector1989/EMAS | 265 | 9 | 0.51 | 0.27 | 1.24 | 0.22 | 0.16 | 32 |
| 768330962/poet_ready_system | 57 | 8 | 0.78 | 0.34 | 1.0 | 0.83 | 0.14 | 14 |
| EUSurvey/EUSURVEY | 294 | 8 | 0.57 | 0.82 | 2.12 | 0.23 | 0.03 | 23 |
| suyeq/steamMall | 120 | 8 | 0.72 | 0.63 | 1.09 | 0.52 | 0.25 | 20 |
| zlren/noah-health | 86 | 8 | 0.73 | 0.95 | 1.22 | 0.41 | -0.07 | 15 |
| Prasad108/TutesMessanger | 109 | 8 | 0.64 | 0.55 | 1.0 | 0.67 | 0.18 | 20 |
| busing/circle_web | 268 | 8 | 0.53 | 0.61 | 1.64 | 0.53 | 0.17 | 28 |
| UDA-EJIE/udaLib | 320 | 7 | 0.5 | 1.0 | 2.0 | 0.34 | 0.2 | 23 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| bbaibb1009/wxcrm | 95 | 7 | 0.59 | 0.83 | 1.08 | 0.59 | 0.2 | 16 |
| khasang/delivery | 218 | 6 | 0.78 | 0.74 | 1.0 | 0.68 | 0.18 | 22 |
| TexnologiaLogismikou/Fiz | 199 | 6 | 0.51 | 0.43 | 1.33 | 0.58 | 0.17 | 16 |
| uq-eresearch/oztrack | 213 | 6 | 0.62 | 0.55 | 2.8 | 0.43 | 0.2 | 18 |
| wang007/live-server | 238 | 6 | 0.51 | 0.4 | 1.36 | 0.5 | 0.17 | 21 |
| zxwgdft/paladin-boot | 328 | 6 | 0.64 | 0.19 | 1.0 | 0.28 | 0.11 | 22 |
| shenshaoming/byte_easy | 55 | 5 | 0.29 | 0.53 | 1.17 | 0.51 | 0.15 | 13 |
| nimble-platform/business-process-service | 222 | 5 | 0.5 | 0.62 | 1.79 | 0.36 | 0.18 | 16 |
| codemky/uni | 615 | 5 | 0.3 | 0.52 | 1.65 | 0.27 | 0.14 | 58 |
| ushahidi/SwiftRiver-API | 249 | 5 | 0.32 | 0.58 | 1.4 | 0.29 | 0.17 | 24 |
| softservedata/lv257 | 171 | 5 | 0.37 | 0.29 | 1.56 | 0.47 | 0.18 | 17 |
| joubin/CSC191 | 32 | 5 | 0.43 | 0.64 | 1.2 | 0.68 | 0.13 | 7 |
| aramsoft/aramcomp | 345 | 5 | 0.58 | 0.92 | 1.5 | 0.42 | 0.27 | 43 |
| ba017634/Warehouse-system | 76 | 5 | 0.61 | 0.75 | 1.0 | 0.39 | 0.28 | 13 |
| shigenwang/membership | 153 | 5 | 0.67 | 0.78 | 1.17 | 0.26 | 0.18 | 20 |
| SafeExamBrowser/seb-server | 680 | 5 | 0.2 | 0.27 | 2.0 | 0.24 | 0.1 | 27 |
| Seenck/jeecg-bpm-3.8 | 810 | 5 | 0.36 | 0.55 | 1.91 | 0.26 | 0.14 | 42 |
| simbest/simbest-cores | 401 | 5 | 0.55 | 0.25 | 1.62 | 0.31 | 0.13 | 30 |
| GraffiTab/GraffiTab-Backend | 188 | 5 | 0.57 | 0.73 | 1.14 | 0.41 | 0.17 | 23 |
| surajcm/Poseidon | 122 | 4 | 0.7 | 0.71 | 1.0 | 0.51 | 0.2 | 14 |
| loongw513029/buscloud | 347 | 4 | 0.65 | 0.77 | 2.0 | 0.36 | 0.21 | 32 |
| tcrct/duang | 377 | 4 | 0.69 | 0.1 | 4.0 | 0.35 | 0.1 | 21 |
| ZFGCCP/ZFGC3 | 471 | 4 | 0.79 | 0.93 | 1.4 | 0.29 | 0.16 | 51 |
| WilsonHu/sinsim | 243 | 4 | 0.56 | 0.92 | 1.45 | 0.55 | 0.17 | 32 |
| crypto-coder/open-cyclos | 2514 | 4 | 0.3 | 0.3 | 1.75 | 0.29 | 0.06 | 66 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| sfx478076717/goldenarches | 56 | 4 | 0.54 | 0.48 | 1.0 | 0.52 | 0.2 | 11 |
| zndo/oss-admin-parent | 106 | 4 | 0.94 | 0.94 | 1.17 | 0.62 | 0.09 | 17 |
| dp2-g56/Dp2-Lo2 | 245 | 4 | 0.74 | 0.27 | 1.89 | 0.36 | 0.16 | 22 |
| cable5881/Fund | 201 | 4 | 0.41 | 0.48 | 1.56 | 0.25 | 0.16 | 33 |
| wangwang1230/te-empl | 156 | 4 | 0.65 | 0.94 | 2.0 | 0.5 | 0.29 | 24 |
| ElectiveTeam/elective_system | 97 | 4 | 0.51 | 0.7 | 1.09 | 0.37 | 0.19 | 17 |
| Rocklee830630/WMSystem | 816 | 4 | 0.8 | 0.96 | 3.85 | 0.28 | 0.08 | 41 |
| 75193982/HyLMS | 535 | 4 | 0.74 | 1.0 | 1.91 | 0.47 | 0.2 | 47 |
| sunxingtm/FPMS | 721 | 4 | 0.8 | 0.67 | 1.53 | 0.44 | 0.26 | 69 |
| next-step/jwp-jdbc | 117 | 4 | 0.65 | 0.53 | 1.25 | 0.73 | 0.22 | 12 |
| KongZouXiang/TradeSteward | 232 | 4 | 0.37 | 0.39 | 1.18 | 0.56 | 0.25 | 23 |
| Glamdring/welshare | 241 | 4 | 0.65 | 0.43 | 1.47 | 0.3 | 0.12 | 20 |
| skyisbule/nanfeng | 71 | 4 | 0.57 | 0.2 | 1.0 | 0.24 | 0.22 | 15 |
| OHDSI/ArachneCentralAPI | 832 | 4 | 0.29 | 0.57 | 1.55 | 0.36 | 0.07 | 41 |
| mingslife/LightCMS | 103 | 4 | 0.54 | 0.12 | 1.18 | 0.41 | -0.07 | 12 |
| linolee/class4 | 187 | 4 | 0.68 | 0.63 | 1.36 | 0.4 | 0.29 | 23 |
| yaowuya/lprapm | 105 | 4 | 0.65 | 0.8 | 1.15 | 0.4 | 0.28 | 20 |
| yShen868/ssmo3 | 94 | 4 | 0.48 | 0.7 | 1.09 | 0.43 | 0.29 | 15 |
| dagilmore/Riddlin | 51 | 4 | 0.68 | 0.58 | 2.2 | 0.52 | 0.25 | 11 |
| dovier/coj-web | 453 | 4 | 0.55 | 0.32 | 2.17 | 0.23 | 0.13 | 30 |
| chenzhq/WsMonitor | 246 | 4 | 0.87 | 0.49 | 1.5 | 0.32 | 0.13 | 24 |
| parkkyoung/f4mall | 59 | 4 | 0.71 | 0.74 | 1.14 | 0.49 | 0.27 | 9 |
| zhouweiwei18/HospitalSystem | 210 | 3 | 0.42 | 0.84 | 1.0 | 0.22 | 0.18 | 38 |
| uq-eresearch/lorestore | 85 | 3 | 0.68 | 0.03 | 1.0 | 0.37 | 0.14 | 13 |
| momoplan/dataanalysis | 195 | 3 | 1.0 | 0.98 | 1.56 | 0.4 | 0.18 | 23 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| isa-group/ideas-studio | 115 | 3 | 0.66 | 0.47 | 1.5 | 0.39 | 0.21 | 16 |
| smartcommunitylab/smartcampus.vas.corsi.web | 65 | 3 | 0.38 | 0.86 | 1.09 | 0.37 | 0.27 | 14 |
| xiongzhenhai-zh/produce-project-managerment | 113 | 3 | 0.71 | 0.99 | 1.11 | 0.54 | 0.24 | 17 |
| flamefire33/uckefu | 609 | 3 | 0.62 | 0.84 | 2.85 | 0.26 | 0.16 | 38 |
| XMFBee/AuthPlatform1 | 281 | 3 | 0.3 | 0.71 | 1.32 | 0.5 | 0.16 | 39 |
| scrumtracker/scrumtracker2017 | 44 | 3 | 0.61 | 0.68 | 1.57 | 0.7 | 0.17 | 10 |
| kingslayer15/mutual | 170 | 3 | 0.6 | 0.54 | 1.12 | 0.57 | 0.24 | 19 |
| AtlasOfLivingAustralia/biocache-service | 151 | 3 | 0.36 | 0.4 | 1.8 | 0.31 | 0.15 | 22 |
| yunchaoyun/active4j-oa | 436 | 3 | 0.4 | 0.58 | 1.52 | 0.38 | 0.14 | 36 |
| racem-cherni/KinderGartenProject | 209 | 3 | 0.56 | 0.16 | 2.38 | 0.35 | 0.19 | 17 |
| peonycmsTeam/peonytancms | 280 | 3 | 0.66 | 0.78 | 1.4 | 0.49 | 0.21 | 31 |
| danbaixidanbai/OCproject | 77 | 3 | 0.87 | 0.8 | 1.33 | 0.58 | 0.29 | 12 |
| wh4585hai/OnlineSchool | 259 | 3 | 0.6 | 0.16 | 1.4 | 0.25 | 0.2 | 25 |
| c2s/telegram-bot-admin | 138 | 3 | 0.52 | 0.47 | 1.21 | 0.39 | 0.15 | 19 |
| thinksgroup/Niceschool | 165 | 3 | 0.41 | 0.48 | 1.15 | 0.51 | 0.25 | 22 |
| CrazyZhao/zblog | 63 | 3 | 0.6 | 0.6 | 1.3 | 0.55 | 0.27 | 12 |
| tlkzzz/xpjfx | 478 | 3 | 0.44 | 0.57 | 2.43 | 0.27 | 0.15 | 36 |
| nds1993/OpenMPS | 922 | 3 | 0.65 | 0.97 | 2.21 | 0.36 | 0.24 | 75 |
| shangtech/WeiXinPlatform | 88 | 3 | 0.65 | 0.94 | 1.88 | 0.46 | 0.21 | 14 |
| jlu-linshuhang/goods-master | 87 | 3 | 0.75 | 0.97 | 1.2 | 0.34 | 0.23 | 11 |
| tanzhb/zhgj-project | 508 | 3 | 0.44 | 0.41 | 1.21 | 0.25 | 0.14 | 46 |
| moxiaohei/OPMS | 130 | 3 | 0.53 | 0.62 | 1.0 | 0.46 | 0.24 | 17 |
| holagoldfish/H5APP-java | 39 | 3 | 0.68 | 0.82 | 1.0 | 0.51 | 0.23 | 10 |
| okfarm09/JYLAND | 68 | 3 | 0.53 | 0.69 | 1.0 | 0.4 | 0.22 | 10 |
| kimki1124/MetelSOS | 61 | 3 | 0.8 | 0.8 | 1.12 | 0.43 | 0.27 | 13 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| cdcchain/cdc-browser | 58 | 3 | 0.5 | 0.21 | 1.5 | 0.55 | 0.17 | 14 |
| orgqy88cckz/AccumulationFund | 134 | 3 | 0.63 | 0.17 | 1.0 | 0.9 | 0.26 | 7 |
| forwardNow/javaee_pkui | 253 | 3 | 0.37 | 0.9 | 1.43 | 0.45 | 0.18 | 33 |
| LiJiuRi/jie_you_ba | 102 | 3 | 0.45 | 0.59 | 1.12 | 0.59 | 0.24 | 16 |
| RMHM/miniMap | 83 | 3 | 0.7 | 0.6 | 1.0 | 0.49 | 0.18 | 11 |
| tom1994/CEM | 404 | 3 | 0.58 | 0.09 | 1.29 | 0.52 | 0.16 | 40 |
| choerodon/agile-service | 815 | 3 | 0.35 | 0.73 | 1.41 | 0.44 | 0.13 | 38 |
| nieyue/ActivationCodeMall | 307 | 3 | 0.24 | 0.86 | 1.37 | 0.54 | 0.22 | 33 |
| kdirector1990/WeRPNetwork | 116 | 2 | 0.75 | 0.77 | 1.0 | 0.33 | 0.25 | 10 |
| longyzkd/wj-web-ext-enhancer | 154 | 2 | 0.54 | 0.65 | 1.2 | 0.15 | 0.2 | 17 |
| VecJunZhi/NewZSWBEM | 475 | 2 | 0.72 | 0.47 | 1.3 | 0.46 | 0.22 | 31 |
| 598605338/yikao | 607 | 2 | 0.59 | 0.54 | 1.21 | 0.52 | 0.22 | 45 |
| nullcodeexecutor/pts | 174 | 2 | 0.65 | 0.01 | 1.47 | 0.39 | 0.15 | 20 |
| kinorsi/mykided-api | 710 | 2 | 0.62 | 0.75 | 1.08 | 0.34 | 0.11 | 32 |
| YouAreOnlyOne/CommunityInformationForWeb | 163 | 2 | 0.44 | 0.55 | 1.69 | 0.31 | 0.18 | 17 |
| CeaMYHBK/AppCeaM | 35 | 2 | 0.49 | 0.53 | 1.0 | 0.78 | -0.06 | 8 |
| gvSIGAssociation/gvsig-web | 127 | 2 | 0.54 | 0.54 | 1.22 | 0.62 | 0.18 | 14 |
| dr-thomashartmann/phd-thesis | 36 | 2 | 0.35 | 0.63 | 1.0 | 0.71 | -0.22 | 13 |
| GreaterLondonAuthority/GLA-OPS | 625 | 2 | 0.75 | 0.17 | 1.0 | 0.28 | 0.14 | 33 |
| qiemengyan/videoconfigserver | 214 | 2 | 0.71 | 0.27 | 1.47 | 0.37 | 0.15 | 19 |
| immime/shop-2 | 292 | 2 | 0.42 | 0.22 | 2.0 | 0.44 | 0.13 | 24 |
| hongqiang/shopb2b | 465 | 2 | 0.36 | 0.23 | 1.97 | 0.48 | 0.13 | 46 |
| AaronSum/hotel-mgr-sys | 498 | 2 | 0.33 | 0.43 | 1.93 | 0.21 | 0.14 | 39 |
| superman7/AccountManagement | 67 | 2 | 0.89 | 0.51 | 1.0 | 0.71 | 0.26 | 13 |
| amit-an/webapp_war_sample | 334 | 2 | 0.35 | 0.56 | 3.07 | 0.4 | 0.15 | 22 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| SupermePower/zammc-manage | 119 | 2 | 0.41 | 0.82 | 1.06 | 0.4 | 0.2 | 18 |
| dtdhehe/ptu-life | 66 | 2 | 0.36 | 0.75 | 1.25 | 0.73 | 0.17 | 12 |
| lancelee98/PeopleMange | 182 | 2 | 0.59 | 0.25 | 1.21 | 0.37 | 0.18 | 20 |
| YXG520/onlineExamSystem | 41 | 2 | 0.37 | 0.75 | 1.2 | 0.6 | 0.16 | 9 |
| jeezhau/ec-server | 149 | 2 | 0.56 | 0.18 | 1.06 | 0.43 | 0.23 | 21 |
| HuangBear/TheaterProject | 155 | 2 | 0.55 | 0.55 | 1.3 | 0.48 | 0.22 | 18 |
| litbo/hospitalzj | 185 | 2 | 0.41 | 0.61 | 1.14 | 0.58 | 0.25 | 26 |
| fawks96/pet-hospital | 74 | 2 | 0.4 | 0.6 | 1.0 | 0.5 | 0.29 | 17 |
| heaptrip/heaptrip | 483 | 2 | 0.6 | 0.69 | 1.38 | 0.32 | 0.13 | 33 |
| 398907877/AppPortal | 470 | 2 | 0.52 | 0.2 | 2.84 | 0.34 | 0.16 | 32 |
| xabaohui/zis | 553 | 2 | 0.46 | 0.48 | 2.65 | 0.37 | 0.14 | 29 |
| RoyZeng/gmhx | 402 | 2 | 0.67 | 0.45 | 1.8 | 0.42 | 0.19 | 41 |
| eea/eionet.webq | 196 | 2 | 0.56 | 0.34 | 1.0 | 0.48 | 0.16 | 19 |
| liudexiang3218/CMSLite | 221 | 2 | 0.68 | 0.22 | 1.3 | 0.37 | 0.19 | 31 |
| Russel-JX/OUC-Family | 276 | 2 | 0.83 | 0.77 | 1.2 | 0.53 | 0.16 | 27 |
| Maxcj/Maxcj | 177 | 2 | 0.62 | 0.21 | 1.47 | 0.39 | 0.19 | 21 |
| qwe7783131/CareerDevelopment | 193 | 2 | 0.84 | 0.56 | 1.61 | 0.56 | 0.17 | 19 |
| assertmyself/gweb-v2 | 365 | 2 | 0.63 | 0.42 | 1.8 | 0.28 | 0.2 | 35 |
| Lewage59/design2019 | 33 | 2 | 0.7 | 0.86 | 1.0 | 0.62 | 0.33 | 9 |
| Ganweizhi/Ruanzhu02 | 142 | 2 | 0.82 | 0.64 | 1.24 | 0.55 | 0.3 | 19 |
| VN-Lf/GitNo3 | 126 | 2 | 0.43 | 0.4 | 1.15 | 0.45 | 0.23 | 22 |
| ssolutiondev/ssolution | 164 | 2 | 0.88 | 0.95 | 1.0 | 0.52 | 0.24 | 22 |
| findmyapp/findmyapp | 139 | 2 | 0.68 | 0.21 | 1.09 | 0.41 | 0.21 | 15 |
| Jeanwin/disrec | 333 | 2 | 0.65 | 0.64 | 1.3 | 0.34 | 0.26 | 40 |
| zhaowei520/CDCXH | 236 | 2 | 0.37 | 0.62 | 1.2 | 0.43 | 0.06 | 26 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| hyperaeon/CrazyAndOptimize | 1390 | 2 | 0.67 | 0.53 | 3.0 | 0.52 | 0.16 | 35 |
| HUTchengxi/Tutor | 169 | 2 | 0.74 | 0.99 | 1.0 | 0.74 | 0.24 | 23 |