Co-funded by the
Erasmus+ Programme
of the European Union

Martin Mayr

Identifying microservice candidates in service-oriented architectures based on runtime-data

Bachelor's Thesis

to achieve the university degree of

Bachelor of Science

Bachelor's degree programme: Software Engineering and Management

submitted to

Graz University of Technology

Supervisor

Dipl.-Ing. Dr.techn. Roman Kern

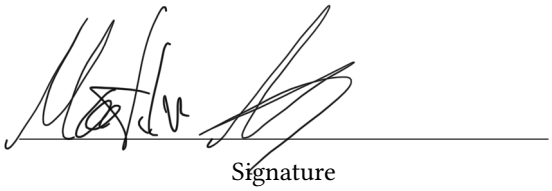
Institute for Interactive Systems and Data Science
Head: Univ.-Prof. Dipl.-Inf. Dr. Stefanie Lindstaedt

Graz, February 2021

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present bachelor's thesis.

Date



Signature

Acknowledgements

I would first like to thank my thesis advisor Dipl. -Ing. Dr. techn. Roman Kern of the Institute of Interactive Systems and Data Science at the Technical University of Graz. The door to Prof. Kern office was always open whenever I ran into a trouble spot or had a question about my research or writing.

I would also like to thank the people who inspired me to write my thesis about such an interesting and complex topic, allowed me to do it in cooperation with their company and were involved in the validation for this research project: Wieser Dietmar, Schöpf Edmund and Thaler Georg at the Raiffaisen Information Service. Without their participation and input, this thesis would never have been possible.

A special thank goes to my parents for providing me with unfailing support and continuous encouragement throughout my years of study.

Finally, I want to thank the Erasmus+ Program for supporting this thesis following the european spirit for development and innovation.

Thank you.

Abstract

When developing enterprise software applications, the goal is to expand the products life-span which is often done by developing a state of the art base to ensure the future expandability, integration of third-party applications and maintenance. Many companies started developing their tailored software in the early 2000s, but over the last 20 years, requirements for software products in terms of SLA, Testing and Fault Tolerance, to name a few, increased drastically as more and more issues came to light. As a result of this, a new architectural style emerged: Microservices. Microservices try to reduce technical challenges when developing applications and achieve a more structured way to develop software solutions on the organisation level. Whereas organisational structures can be changed relatively easily, the migration of an existing legacy system towards microservices requires a lot of human work.

Algorithmic techniques exist, but they heavily relay on static code analysis and ignore crucial runtime behaviours of the system. This thesis tackles that problem by presenting an algorithmic way to extract microservice candidates in a refactoring scenario entirely based on run-time data of the system. For this, a large amount of runtime-data was acquired and modelled as a graph. To represent the runtime dynamics of the system, a set of weight functions were defined. The extraction of the microservice candidates was realized by applying graph-based clustering algorithms on the graph representing the system. In addition to this, a web-based user-interface was developed to provide architectural insights before and after the extraction process.

To assert and test the correctness of the developed approach the author entered a cooperation with the Raiffeisen Information Service, which tested and rated the output of the extraction process. Besides this, the correctness was verified via custom microservice-specific metrics. The results show that the described approach works very well for its structural simplicity and can be used to analyze the current state of the system and automate the extraction process arbitrary well.

Contents

1 Introduction

Developments in areas such as cloud computing, Development and Operations (DevOps) and event-based systems enabled the development of new architectural styles as an alternative to the widely used monolithic architectural style. In the last couple of years, we could observe several of the worlds largest companies (such as eBay, Netflix and Amazon) implement and use this new architectural style to their advantage. [4]

In contrast to established software architectures, such as monoliths or service-oriented systems, the microservice pattern aims to split the application into a set of so-called services, where each service has its responsibility and can be deployed independently. This key feature tackles some of the most crucial requirements of today's software industry such as scalability, fault tolerance and DevOps. Despite these benefits, many companies fear the migration process. Taibi et. al. describe in their work that companies consider effort overheads, the complexity of decoupling and lack in return of interests as the main issues when migrating a legacy application. [21]

In fact, the refactoring of existing systems is and has always been one of the most complicated and error-prone processes in software engineering [3]. This especially holds true when migrating to a new architecture model. As Taibi et. al. stated, the industry sees different challenges in the migration process. One of them is the identification of individual software components that can be divided into single responsible units - namely microservices[21], [15]. While tools for the structural analysis of the monolith exist [21], there is a lack of tools that can assist software architects in the migration process by identifying possible microservice candidates and taking into account complex and crucial runtime aspects of the system[22],[4].

This thesis was elaborated in collaboration with the Raiffeisen Information Service, which provides the IT infrastructure and the software used by over 40 cooperative

banks with nearly 174 subsidiaries (2019).

Their software product handles most of the banking activities such as online banking, card payments, ATM machines and EBA-Clearing to name a few. As the current software solution is used by over 70.000 users a stable, reliable and easily scalable IT-Infrastructure is key to success. In the past years, efforts have been made to disavow the in this sector widespread mainframe architecture. This was achieved by splitting up the initial monolithic application into a service-oriented architecture (SOA). This SOA application is now in charge of most of the business procedures.

In the past couple of years, the users and legal requirements for the banking sector increased significantly. SOA based architectures have some major downsides in terms of agility and scalability. To mention two examples: dependencies between services and components can introduce scaling challenges and the widely used development pattern of sharing dependencies limits management capabilities.

To keep up with the high demands and to facilitate development cycles the Raiffeisen Information Service decided to refactor and extract parts of the software to step towards a microservice-based application. To simplify migration processes and to not fully rely on domain knowledge, this thesis analyzes different migration approaches and derives an enhanced approach for decomposition of service-oriented architectures based on execution data. The described approach and the resulting tool were developed and validated at Raiffeisen Information Service and will be part of their software architecture road-map.

1.1 Motivation

In a monolithic architecture, all functions and modules which are needed at runtime are packed into one application. While this approach of packing everything into one application might be successful for a small projects, it will be insufficient for applications growing in size. Once an application has grown in size, the monolithic design of the application will become overwhelmingly complex, the incomprehensible code base may delay bug fixes and become an obstacle for continuous deployment. [4] In the last twenty years, applications evolved from tightly-coupled systems into a large collection of services [18]. This so-called service-oriented architecture (SOA) model allowed applications to become more flexible and to draw

sophisticated context boundaries [9]. Due to the market's high increasing demand for new application features, fault tolerance, and continuous integration, systems required to change in how they are structured and built [18].

Microservices are an architectural approach that helps to model distributed, fine grade, and independent systems. Each system is modelled around business capabilities and has its responsibility. All services communicate via a lightweight protocol (often HTTP). These key properties allow applications to build upon this patter to fully take advantage of features like Cloud Computing, DevOps strategies, and agile frameworks like Scrum. [15]

Neither academia nor industry agree whether the newly emerged microservice pattern is a new architectural style or an implementation approach to the existing SOA patterns [23],[4]. Despite this, both agree on the advantages of microservices in comparison to other architecture styles [4]. While providing huge benefits, microservices can not be seen as a silver bullet. In fact, an application build upon microservice principles brings a lot of complexity in terms of testing, security, and decomposition[15].

1.2 Research Questions

This thesis aims to give an overview of existing extraction methods for microservices and proposes an enhanced method for service decomposition. For this following research questions will be answered:

- **RQ1**: Are there existing best practices to decompose an existing monolithic system into microservices and where are their key differences?
- **RQ2**: To which extend is domain knowledge necessary the in creation of microservices given an existing system?
- **RQ3**: How can a complex monolithic legacy system be observed at runtime, modelled and split into smaller, independent services?
  - **RQ3.1**: In which way have results be presented to Software Architects and Domain Experts so that they can gain insights in their application and evaluate microservice candidates?

  - **RQ3.2**: Is it possible to extract the different domains of an existing legacy system to provide a Domain Driven Design alike approach for extracting microservices based on runtime-data of a legacy system?
  - **RQ3.3**:How much does the result of the developed tool differ from a decomposition provided by a domain expert?
  - **RQ3.4**: How can the quality of the created microservices be assessed?

1.3 Structure of the Thesis

This thesis is structured as follows:

- **Chapter 2 Theoretical Background**: This chapter provides an overview of the key features, benefits and challenges to overcome the different architectural styles.
- **Chapter 3 State of the art Decomposition Approaches**: In this chapter different approaches to identify potential microservice candidates will be discussed.
- **Chapter 4**: Proposes an approach for identifying potential microservices candidates based on runtime-data.
- **Chapter 5**: **Evaluates** the approach described.
- **Chapter 6**: **Conclusion** summarizes and .discusses the outcome of the thesis

2 Theoretical Background

This chapter provides an introduction to the three main architecture patterns used in enterprise architectures, as well as their benefits, associated challenges and delimitation between them.

2.1 Monolithic Application

Monolithic applications have come a long way and are the traditional method of developing software. In a monolithic architecture all functions and modules are encapsulated into one single application - therefore models are tightly coupled and self-contained. While this type of architecture often is used in an early prototyping phase due to the ease of development, test and deployment, once the application becomes large and the team size grows significant, drawbacks will be encountered [19]:

- The development of the application will slow down as a result of a growing and ever more complex codebase. [19]
- Continuous Deployment / Continuous Integration becomes difficult as the build and deployment time of the monolith grows. [19]
- Scaling can only be achieved by running many instances behind a load-balancer (horizontal-scaling) [19]

The literature agrees that it is a good idea to start a project as a monolithic system in order to explore its complexity and component boundaries.[7] Figure 2.1 illustrates the basic concepts of a monolithic architecture yielding a model view component based monolithic application.

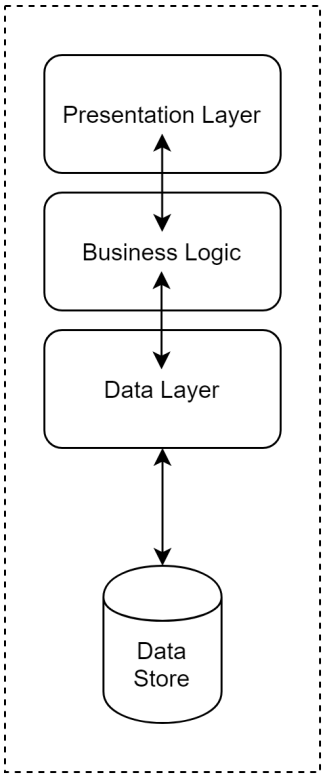


Figure 2.1: Example of an Monolithic Architecture scheme

2.2 Service Oriented Architecture

The Service-Oriented Architecture (SOA) pattern was first described by Roy W. Schulte and Yefim V. Natis in 1996 and was an attempt to break up existing monolithic applications into smaller subsystems modelled around business processes as well as easy integration of 3rd party systems[9]. Each business process can communicate with another or a 3rd party system using standardized messages (mostly XML) via an Enterprise Service Bus. By using a standardized communication protocol and well-defined interfaces, SOA can provide loosely couplings and high consistency. [9]

Although defined in 1996, SOA gained a lot of attention in recent years. Tackling challenges associated with maintenance and the growing Cloud Computing sector, it became a silver bullet for the migration of legacy systems. Different industries

gained a lot of benefits by building their application using a domain-specific adoption of SOA, but many companies have had a hard time and often failed at implementing and migrating legacy applications. A reason for this, as per Niknejad et. al., has often been the lack of technology and domain knowledge as well as information about critical success factors. [16] Figure 2.2 illustrates the basic concepts of a SOA architecture.

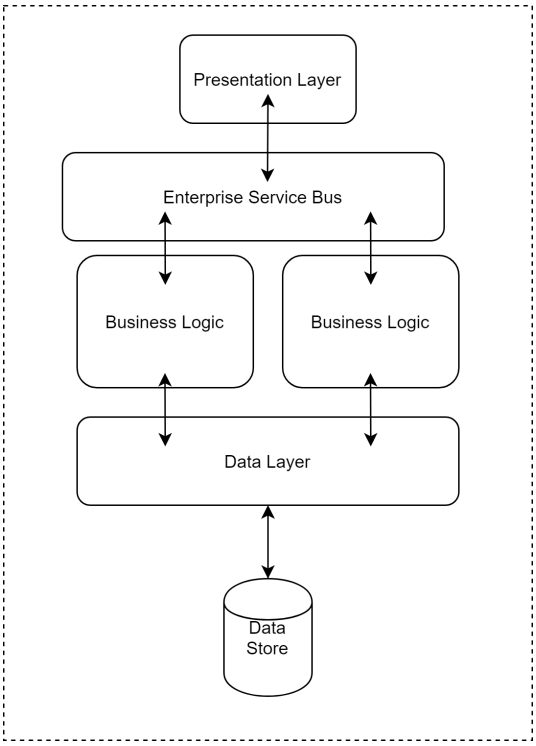


Figure 2.2: Example of an SOA Architecture scheme

2.3 Microservices

Lewis and Fowler in 2014 described the term microservices as an *"[...] approach to developing a single application as a suite of small services"*[12]. They stated that at this point (2014) there is no precise definition of this architectural style [12]. As of now neither academia nor developer movements, pushing the term microservice,

have come up with a formal definition of microservices architecture. Developers, pushing the term microservice claim it is a new architectural style. In contrast advocates of SOA insist in the fact that microservices are just an implementation approach to existing SOA. [23]

Even though there is no accurate definition of this architectural style, Lewis and Fowler[12] as well as Zimmerman[23], identified common characteristics and use-cases such as:

- **Single-responsibility** units encapsulate data, process logic and form service. Each service is modelled around business capabilities and communicates with other services, through a lightweight communication protocol (e.g. HTTP). [12]
- Services are identified by **business-driven development practices** such as domain-driven design.[15]
- **Multiple computing paradigms and languages** can be used to develop services. This allows the developer-team to unleash the full potential of different computing paradigms and technologies. [23],[12]
- **Lightweight container** technologies are used to scale and deploy services on demand.[23]
- **DevOps** approaches for automated configuration, performance and fault management are employed to extend agile practices and include service monitoring capabilities. [23]

Figure 2.3 illustrates a simplified microservice architecture.

2.3.1 Benefits and Challenges

In 2017 Taibi et. al. conducted a survey among software architects, migration consultants, project managers, developers and chief executive officers (CEOs) who successfully migrated their application to a set of microservices. The survey had the goal to analyze initial motivation, as well as the pros and cons of migrating from a monolithic architecture to microservice-based architectures in the industry. Respondents confirmed the benefits mentioned in literature such as [15], [12],[23], [21] but also stated that the migration process was linked with difficulties which had to be overcome.

Taking into account literature and industry, key benefits of a microservice-based architecture may be summarized as follows:

- **Modularity / Separation of Software Responsibilities**: The modular architecture allows to reduce the complexity of a monolithic system by breaking a system into independent and self-deployable services. This enables developer teams to test and deploy their software without having to wait for other teams. The reduced size of the code furthermore increases the understandability, and maintainability of the code. [21]
- **Scalability**: While scaling a monolithic as well as an SOA application can only be achieved by scaling the entire application(see Fig. 2.1 and Fig 2.2), microservices allow to scale only required parts, hence saving costs and configuration overhead. [12]
- **Fault tolerance**: The fault of a microservice might not impact the whole system, whereas the failure of a component in a monolithic application is highly likely to break the whole system. [21],[15]
- **Delegation of Team Responsibilities**: Since a microservice does not have any external dependencies, each development team can develop independently, reducing communication and coordination requirements among teams.[21],[12]
- **Decentralized Governance / Easy Technology Experimentation**: Since microservices are small by definition, they can be rewritten from scratch in a short amount of time. This allows companies to experiment with new technologies to see if they could fit their needs. In a survey participants mentioned that their microservice structure allowed them to easily (re-)write components in different programming languages using different technologies and integrate them flawlessly into the existing system landscape. A monolithic application would not have allowed this. [21],[12], [15]
- **DevOps**: Among the participants of the survey DevOps was considered as a crucial benefit allowing each development team to develop, test and deploy independently.[21]

When it comes to challenges, the literature and the results of the survey diverged. Whereas the literature stated challenging factors can be found mainly in technical areas, the survey participants further complained about economical and psychological limitations.

- **Decomposition**: Industry [21], as well as literature [15],[8],[4], mentioned that the decomposition process is one of the most challenging factors due to the lack of automated tools, complexity of the legacy system and unavailable domain knowledge.
- **Databases and Data Consistency**: In most legacy applications, one database is shared among different business procedures. The data in this database is often shared and accessed by different components. [15] Microservices incorporate the principle of decentralized data management. This requires to split up existing data-sets and assign the responsibility of each data-set to the concerned microservice.[12] Industry stated, that data governance and database migration is one of the biggest challenges when it comes to migrating existing legacy systems to microservice systems. [21]
- **Sizing**: The size of a single microservice is a question that has not been answered generally yet. According to Newman, trivial approaches like lines of code can be misleading. The reason is simple: different technologies require different implementation approaches. Each approach might be connected to more or fewer lines of code. Newman states, that in most cases domain experts do have a pretty good understanding if big is too big. [15]
- **Communication Overhead**: Each service needs to communicate with other services. This cannot be done via in-memory communication like in monoliths [12]. Communication in a microservice architecture is done over the network, adding complexity to each service and the network infrastructure. Among the participants of the survey conducted by Taibi et. al. , participants stated that this was a concerning factor but not a limiting one, since their systems were deployed on high performing cloud infrastructures. [21]
- **Service Discovery** describes the ability to detect new, destroyed and available services. This can become a challenge in an environment, where services are constantly created and destroyed. [15]
- **Migration effort estimation & Cost Overhead**: the cost estimation of a microservice-based system is often less accurate than that of an enclosed system. Among the participants of an industrial study, each respondent reported an effort overhead of nearly 20% more compared to the effort required developing a monolithic system. However, it must be noted that **each** respondent reported that the benefits of the new architecture such as maintainability and scalability highly compensate the extra effort.[21]

- **People's Minds**: For several developers changes in existing architectures are a general issue. Especially more senior developers do not always trust in recent technologies. They identify themselves with the legacy system and see it as part of their creation. This makes them reluctant to accept the importance of changes. [21]

Although only key benefits and challenges could be stated, it is easy to see that this new paradigm or architecture form entails some huge benefits in comparison to monolithic architectures. The delimitation to SOA is not clear, Zimmerman comes to the conclusion, that several characteristics of microservices mostly pertain to the development process, development culture and the process and physical viewpoints and not to the logical architectural patterns used. [23]

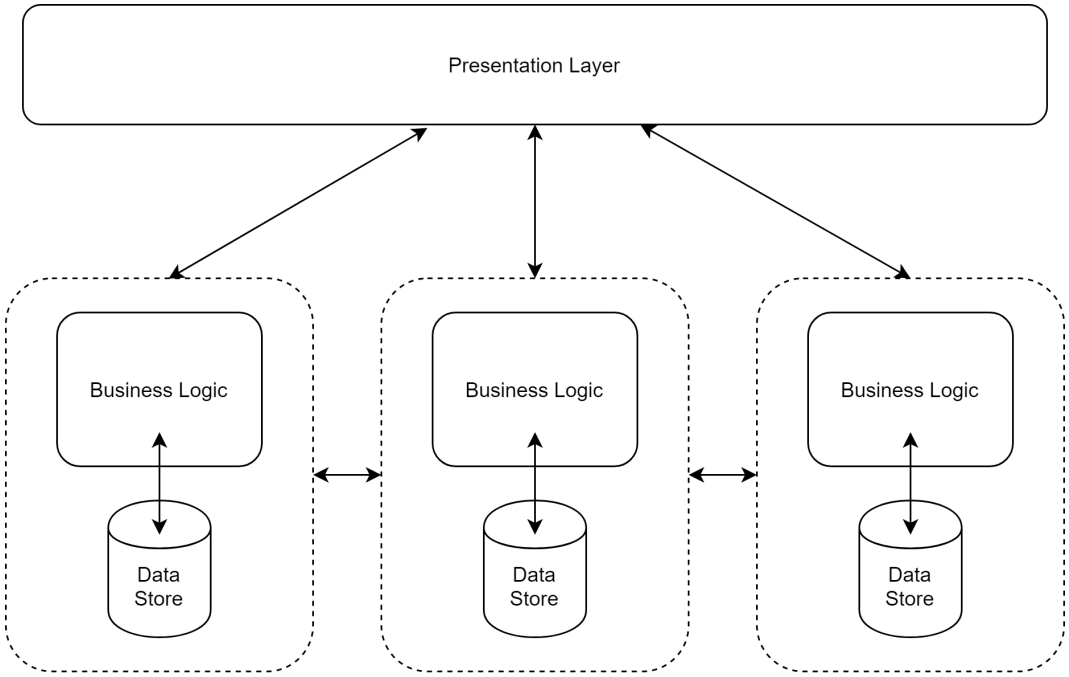


Figure 2.3: Example of an Microservice Architecture scheme

3 State of the art extraction methods

When speaking of automated microservice extraction, literature distinguishes between static and dynamic extraction methods. Static extraction describes the identification and extraction of microservices entirely based on the underlying source code. These approaches often highly rely on metrics and do not consider the complex runtime behaviour of applications. Dynamic approaches on the other hand can provide extraction advice by using data representing reality. They often use application traces or event logs. In the following state of the art approaches for microservice candidate identification and extraction are described.

3.1 Function-Splitting Heuristics for Discovery of Microservices in Enterprise Systems

De Alwis et. al. [5] described a heuristic that helps to identify suitable parts of an enterprise system which could be re-engineered as microservices. Their approach assesses the structural and behavioural properties of legacy enterprise systems and microservices. Analyzing patterns of system executions and business operation, De Alwis et. al. defined two heuristics namely the ”subtype” and the ”common subgraph”, which, they argue, help to identify prominent ”microserviceable” components.

3.2 Extraction of Microservices from Monolithic Software Architectures

Mazlami et. al. in [14] developed an algorithmic approach to tackle the extraction problem by presenting three formal coupling strategies and embedding those into a graph-based clustering algorithm. The coupling strategies rely on information gained from the monolithic codebase. The approach has two phases: First the codebase is analyzed. The result of this phase is a weighted graph where each node represents a class and each edge a connection among classes. The weight of the edges depends on the chosen coupling strategy. In the second phase, graph clustering algorithms extract possible microservice candidates.

3.3 Service Cutter: A Systematic Approach to Service Decomposition

Gysel et. al. in [8] propose a service decomposition framework based upon sixteen different coupling criteria derived from literature and industry. An entity-relationship model serves as the basis for the decomposition process. The process gets enhanced by so-called ”User Representations” where a user provides further information about the domain model by defining use cases, responsible roles and in a later step nano entities. A nano entity is a generalization of data, operation and artefact associated with a service. Based on this data an undirected graph is built, on which, based on coupling criteria, possible microservice candidates are proposed. To verify the correctness of the approach a tool named Service Cutter was developed. The tool can suggest candidates for service decomposition based on the above mentioned sixteen criteria.

3.4 Microservice Decomposition via Static and Dynamic Analysis of the Monolith

Krause et. al. present an approach that not only uses a static analysis of the legacy system but includes system runtime behaviour as well. Their approach combines established analysis techniques for microservice decomposition, such as bounded context. The migration approach was validated on a real-world example. Krause et. al. start the decomposition process with a domain analysis. Based on the outcome, they were able to identify bounded contexts of the monolith. After having gained essential insights into the application, the static software structure is analyzed and mapped to the previously identified context boundaries. This allows them to identify possible domain overlaps. In the last and final step service boundaries were refined by analyzing runtime data. [11]

3.5 A Decomposition Framework based on Process Mining

Taibi et. al. in [22] describe an approach which aims to extract microservice candidates entirely based on runtime behaviours of legacy applications. For this Taibi et. al. combined process-mining techniques and dependency analyses performed on log files collected from the monolith runtime behaviour. The decomposition process consists of 5 steps. In the first step, most frequently execution paths are identified by using a proprietary tool. In the second and third step, the frequency of execution path is analyzed and circular dependencies are removed. In the fourth step, different decomposition options were provided and evaluated manually. Finally step five ranks the elaborated options based on metrics. Taibi et. al. validated their approach in an industrial case study.

3.6 From Monolith to Microservices: A Dataflow-Driven Approach

Chen et. al. propose a "*[...] purified dataflow-driven mechanism that can guide rational, objective and easy-to-understand microservice-oriented decomposition.*" The described approach consists of three steps. In the first step a purified data flow diagram (DFD) is constructed manually. In the second step the purified DFD is converted into a decomposable DFD. In the third and final step the decomposable DFD is used to identity microservice candidates. [4]

3.7 Weaknesses of recent works

The above described methods all come with strength and weaknesses.

Mazlami et. al. [14] only use static code analysis to gain knowledge about the legacy system. The approach does not consider the complex runtime aspect of monolithic systems. Moreover, semantic coupling strategies, based on term-frequency inverse-document-frequency (TF-IDF) method to identify high-level topics or domain concepts, can be error-prone when considering the fact that especially in strongly regulated domains like the finance sector names for classes, methods and variables contain domain-specific information in domain language. Problems can occur when the internal coding guidelines of the application are in English but the domain-specific language is defined (by management or by law) as i.e. German or Italian.

Gysel et. al. [8] proposed an extraction method based on sixteen coupling criteria extracted from literature and industry experience. As [4] stated Gysel et. al. [8] used an undirected graph to model the system. Where dependencies in undirected graphs can be displayed, they do not contain any information about the direction of the dependency. For this reason and the defective determination of the edge weights, Chen et. al. [4] described the graph as non-objective. Aside from huge configuration overhead, Gysel et. al. [8] mention that their approach does not reflect any runtime aspects of the application such as throughput or bottlenecks.

4 A data-driven approach for microservice extraction

4.1 The proposed approach

The goal of the proposed approach is to model the software-system based on runtime-data. The result of this modelling process is a graph, which represents the software-system and it's runtime dynamics so that graph-clustering algorithms can be applied and ultimately microservice candidates can be extracted.

To model the graph some steps are required which are described in more detail in the following.

1. **Data acquisition**: In the first step runtime data is acquired using the Elastic Application Performance Monitor (APM) [1]. The Elastic APM can monitor each request to the system and all method-calls associated with it.
2. **Data filtering & distillation**: In the second step the extracted data is distilled.
3. **Graph creation**: In this step a weighted graph is build to represent the system on class level.
4. **Definition of a weight function**: This step applies a weight-function to the created graph to represent the complex dynamics of the system.
5. **Community detection**: The created graph is split up into communities using graph-clustering algorithms.

[1] https://www.elastic.co/apm

4.1.1 Data acquisition, filtering & distillation

To gather insight information on the application during runtime, log files or similar data-sources are required. The problem with relying heavily on log files while examining an existing system is that log-files are by nature not that precise as one relies on the usage or the implementation of the logging system. To overcome this problem this approach uses a so called Application Performance Monitor (APM), more precisely the APM Developed by Elastic. Elastic offers different Monitoring Agents for today's most used technologies. These Monitoring Agents connect to the application and can collect internal application states such as incoming requests to the system, function calls and database-queries.

Complex systems often have connections to 3rd party systems. These systems can be developed by the same company as well as by other companies and therefore be hosted in-house or not in-house. Since the to be examined application can not affect the performance of the 3rd party system, outgoing requests to these systems have been removed from the data-set. A simplified and distilled data-set can be seen in Table 4.1, where:

- Transactions are the highest level of work within a service (i.e. request to the application, background job or batch job). The field **Transaction ID** provides a unique identifier per request to the system.
- Spans contain information about the execution of a specific code path. They measure from the start to the end of an activity, and they can have a parent/child relationship with other spans. The **Span ID** is a unique identifier for each Span.
- The **Span Duration**, is measured in milliseconds and describes how long the system needs to complete the Span.
- The column **Span Name** provides the class name and the invoked method name spread by "#".

Timestamp[us]	Transaction ID	Span ID	Span Name	Span Duration[ms]	Parent ID
0	bf2722		RequestHandler#methodName		
1	bf2722	988fe4	ClassName#methodName	25	bf2722
2	bf2722	988fee	ClassName#methodName	45	988fee
3	bf27228		RequestHandler#methodName		

Table 4.1: Simplified distilled raw-data-set

4.1.2 Graph creation

Although the available dataset offers the granularity to construct a graph of how methods of classes interact with each other, this approach only evaluates the interactions on class level - which means how classes interact with each other. This could be seen as s weakness of the proposed approach but it is not since classes - driven by object-oriented principles - in each architecture model should encapsulate distinct and assignable logic.

The graph creation is done by traversing the filtered dataset and inserting classes as nodes and child-parent relationships as edges. The child-parent relationship is detected by interpreting the **Parent ID** field. Each inserted edge contains information about the transaction and the time the connection was used.

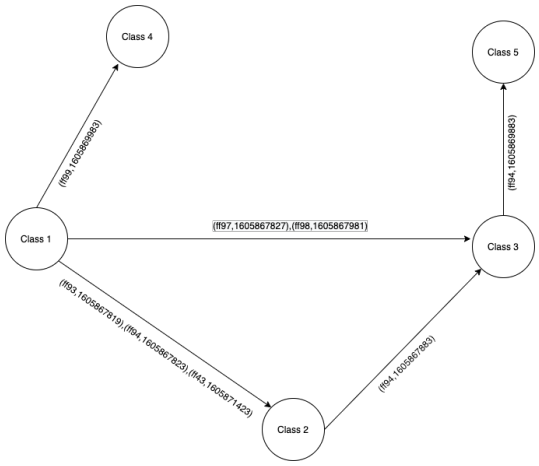


Figure 4.1: Each edge contains information about the transaction and the time of occurrence in the form: (transaction id, timestamp)

4.1.3 Weight factors

To not only model interaction between the different components of the system, but also the more complex dynamics between those components, a weight function is used. The proposed weight function consists of three factors:

- **Total calls**: Represents how often a function in class i is called by a function in class j.

$$total\ calls(i,j) = |calls(i,j)| \tag{4.1}$$

- **Average duration**: Represents the average computation time for a function call from calss i to a function in class j.

$$average\ duration(i,j) = \frac{|duration(i,j)|}{|calls(i,j)|},\ with\ |calls(i,j)| > 0 \tag{4.2}$$

- **Sample variance** : The sample variance (s^2) displays how regularly a function in class i invokes a function in class j . For this weight-factor the time frame in which the application has been monitored is divided into chunks of one minute before the recorded calls between i and j are summed and assigned to the corresponding chunk. After the assignment, the sample variance is computed by 4.3 . Table 4.2 provides an example on how the variance is calculated on a given data-set.

$$s^2 = \frac{\sum(X - \bar{X})^2}{N - 1} \tag{4.3}$$

Group Name	Timestamp	Number of requests
Class 1 - Class 2	2020-11-18T10:42:00+00:00	5
	2020-11-18T10:43:00+00:00	12
	2020-11-18T10:44:00+00:00	7
Variance : 8,6		

Table 4.2: Simplified example of how the set variance is calculated.

4.1.4 Weight function

Assigning each edge of the graph an appropriate weight given the above-defined weight components is all but complicated as different combinations result in differently sized and structured clusters. In the following two weight functions 4.4 and 4.5, which follow a basic intuition, are described. Each function consists of two parts, the *importance* and the *scale* part.

The *importance* part of the formula is used to determine how regularly a connection between two nodes i and j is invoked. To illustrate the problem to be solved, let's assume the system has some sort of a backup-function which's purpose it is to copy data from one database table to another and that starts every night at 11 pm. The implementation is done via two nodes i and j. i only reads a data-set, transmits it to j and j only writes the data-set. As the amount of data grows the $|calls(i,j)|$ will grow. This, besides being problematic from an implementational perspective, would lead to the false assumption, that the connection between i and j is one of the most used connections which is wrong. To correct this, the total amount of calls is multiplied with the set-variance of the corresponding connection between i,j. The set variance, normalized using the min-max scaler with a feature range between 0 and 1, represents, as the name suggests, how variable the connection is being used. In the above-described example, the variance would be relatively high due to the fact that the backup procedure starts every night at 11 pm and is inactive during the day.

The *scale* part of the equation follows the simple assumption that long-running requests should be bundled together to avoid spanning excessive service boundaries and benefit from features such as scalability.

$$w_1(i,j) = \underbrace{total\,calls(i,j) * s^2(i,j)}_{\text{importance}} * \underbrace{average\,duration(i,j)}_{\text{scale}} \tag{4.4}$$

$$w_2(i,j) = \frac{\overbrace{total\,calls(i,j) * s^2(i,j)}^{\text{importance}}}{\underbrace{average\,duration(i,j)}_{\text{scale}}} \tag{4.5}$$

4.1.5 Community Detection

In an ideal microservice architecture the components in each microservice are highly cohesive while microservices are coupled loosely among each other [15]. In this work, two different community detection algorithms are used to find the optimal clusters. Each algorithm uses the edge weights defined by 4.4 and 4.5 to compute the different clusters.

The Louvain Algorithm for community detection

The Louvain community detection algorithm was first described in 2008 by Bondel et. al. [1]. The algorithm aims to achieve a high density of edges within the clusters and a low density between the clusters which can be achieved by optimizing the modularity of each cluster.

The algorithm consists of two phases: In the first phase, each node is assigned to a different community. Then for each node i the neighbours j of i are considered by evaluating the gain of modularity that would take place by removing i from its community and by placing it in the community of j. The gain of modularity ΔQ is defined in Equation 4.6. If there is a possible positive gain in modularity for node i by placing it into the neighbour community, the node gets moved. If there is more than one possible positive gain in modularities, the node gets moved to the neighbourhood community which can provide the maximum gain in modularity. If no positive gain is possible, i stays in its original community. This process is repeated for all nodes until no further improvement can be achieved. In the second phase, a new network where nodes are communities from the previous phase is created. Any links between nodes of different communities are handled by connecting the communities via a weighted edge. Once the second phase has ended, the first phase can be re-applied to the network[1].

$$\Delta Q = \left[\frac{\Sigma_{in} + 2k_{i,in}}{2m} - \left(\frac{\Sigma_{tot} + k_i}{2m}\right)^2\right] - \left[\frac{\Sigma_{in}}{2m} - \left(\frac{\Sigma_{tot}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right] \quad (4.6)$$

with:

- Σ_{in}: The sum of all the weights of the links inside the community i is moving into
- Σ_{tot}: The sum of all the weights of the links to nodes in the community i is moving into
- k_i: the weighted degree of i
- $k_{i,in}$: The sum of the weights of the links between i and other nodes in the community that $i$is moving into
- m: The sum of the weights of all links in the network

The Infomap Algorithm community detection

The Infomap algorithm was described in 2009 by Rosvall and Bergstrom [20]. In contrast to Bondel et. al. [1], the ideal communities are calculated using the map-equation 4.7 and by minimizing description length of motion a random walk [2]. A random walker moves randomly from node to node in the network. The more the connection of a node to another node is weighted, the more likely the random walker will use that connection. The goal of the algorithm is to form clusters in which the random walker stays as long as possible and thus assuring that the weight of the connections within the cluster are greater than the weights between the clusters [20].

$$L(M) = w \curvearrowright log(w \curvearrowright) - 2\sum_{k=1}^{K} w_k \curvearrowright log(w_i \curvearrowright) - \sum_{i=1}^{N} w_i log(w_i) + \sum_{k=1}^{K} (w_k \curvearrowright + w_i) log(w_k \curvearrowright + w_k) \tag{4.7}$$

with:

- M: Network with N objects $(i = 1, \ldots, N)$ and K clusters $(k = 1, \ldots, K)$.
- w_i: The relative weight of all connections of a node i, that is the sum of the weights of all connections of a node divided by the sum of the weights of all connections of the network.
- $w_k = \sum_{i \in k} w_i$: The sum of the relative weights of all connections of the node of the cluster k.
- $w_k \curvearrowright$: The sum of the relative weights of all connections of the node of the cluster k leaving the cluster.
- $w \curvearrowright = \sum_{k=1}^{K} w_k$: The sum of the weights of all connections between node from different clusters.

4.2 Visualisation

Software Architects and Domain Experts are eager to know how the existing application performs under runtime conditions and what an eventual microservice architecture would look like. For this, a web-based user-interface (UI) was developed.

The developed user-interface provides an easy and intuitive way for exploring the current architecture, search for nodes, examine their connections and dependencies in the system during runtime and allows to visualize and analyze the proposed microservice architecture of the system based on each decomposition and graph weighting approach. In the following, each developed feature is briefly described and screenshots of the applications are provided. For reasons of data protection, the names of the nodes are anonymized.

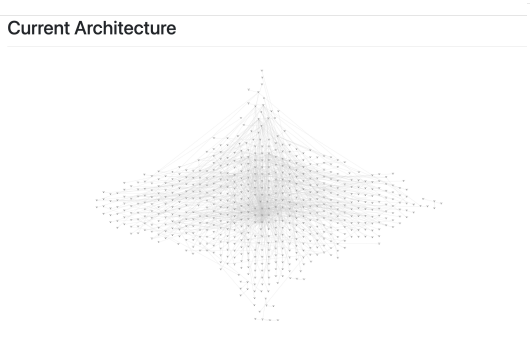
Browse Current Architecture

The use case of this feature is straight forward: Software Architects and Domain Experts can explore the existing structure of the system by moving freely through the graph. More information such as in and outgoing connections can be seen by clicking on the node. To gather information about the relation between two nodes the user can click on an edge between two nodes. Following information about the relationships can be given: average duration, total calls and variance. Figure 4.2 provides screenshots of the above discussed UI features.

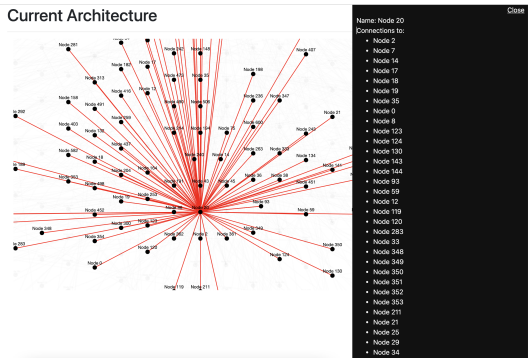
Explore Nodes

Exploring in and outgoing connections of a specific node can be difficult as the examined system grows. Therefore the UI provides a way to search for a node in the graph. If the requested node is found, the node and its descendants [2] will be visualised. Similar to the Browse Current Architecture functionality, the user can gather information about the relation between the nodes. Figure 4.3 provides screenshots of the above discussed UI features.

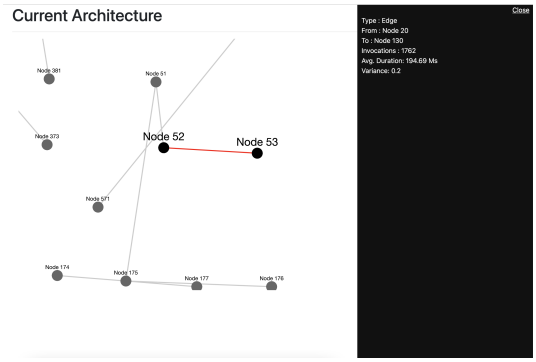
[2] A descendant of a node n is any node which is either the child of n or is (recursively) the descendant of any of the children of n.



(a) Current architecture in form of interactive graph.



(b) incoming or outgoing connections for node "Node 20"



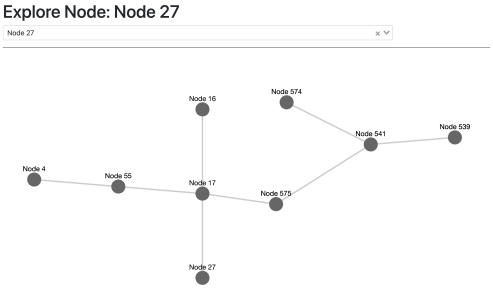
(c) Connection properties between "Node 52" and "Node 53"

Figure 4.2: UI section for exploring the current state of the architecture

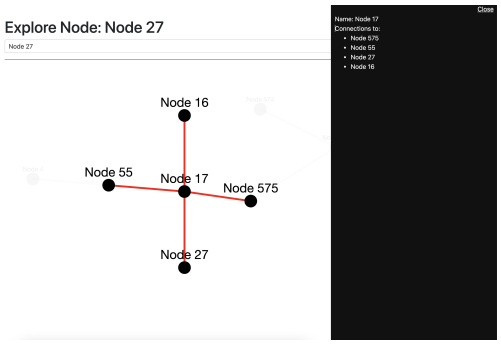
Browse Microservice Architecture

This functionality is designed to visualize the composed microservice candidates and consists of two views: a view in which the user can move freely across a high-level representation of the decomposed system and a view in which the user can explore each microservice individually.

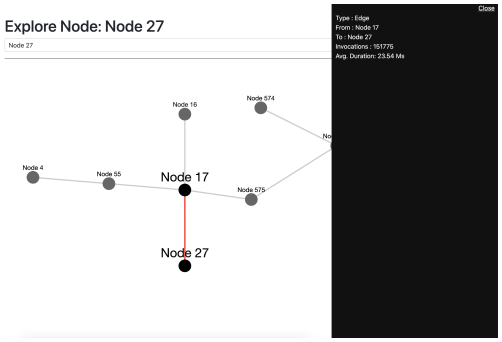
In the high-level representation of the system, each microservice is visualized as a box. Each box contains the nodes which form the microservice. If two microservices interact with each other, they are connected by a line with each other. Similar to the Browse Current Architecture view, the user can gather information about the relation between the microservices. In Figure 4.4 screenshots of these functionalities are provided.



(a) Descendants of "Node 27"



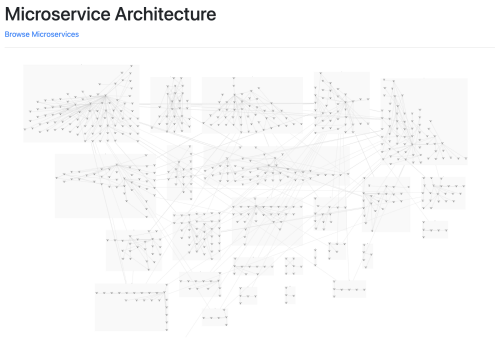
(b) Incoming or outgoing connections for node "Node 27"



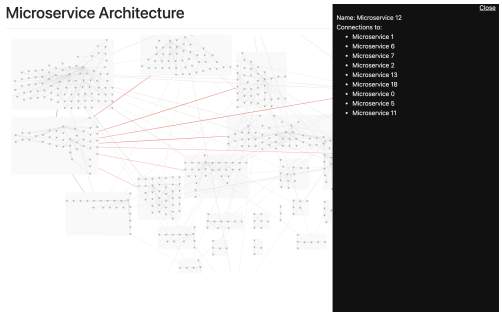
(c) Connection properties between "Node 17" and "Node 27"

Figure 4.3: UI section for assessing a specific node

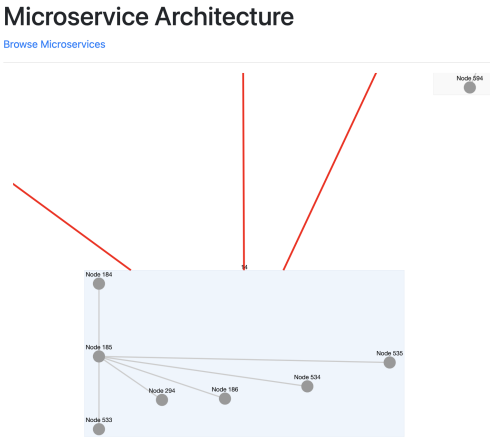
If the user wishes to collect deeper insight into a specific microservice, he can do so. The UI provides a view in which the user can explore a microservice. In this view, the components of the microservice, the connections inside the microservice, the incoming and outgoing connections of the microservice, the instability index as well as the enhanced instability index of the microservice can be seen. Figure 4.5 provides screenshots of the above discussed UI features.



(a) Overview of the proposed microservice architecture

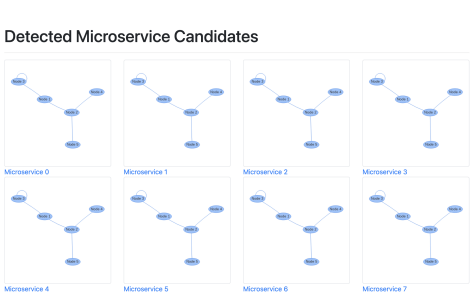


(b) Relations between the selected microservice and other microservices

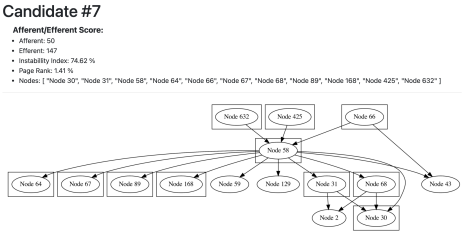


(c) In each microservice the corresponding nodes are visible

Figure 4.4: UI section for exploring the proposed microservice architecture



(a) List of individual microservices



(b) Detailed overview of a microservice. Nodes without a box, can are displayed as external dependencies which are provided by an other microservice..

Figure 4.5: UI section for exploring a microservice individually

5 Evaluation

This section aims to evaluate the described approach. The evaluation is done in cooperation with a software architect at the paring company. For the evaluation of this approach, the system has been monitored with the Elastic APM for two weeks. In these two weeks, 250 Gigabytes of raw data have been collected, distilled and transformed into a graph. The amount of data and the long observation time assure a reality near representation of the system and it's dynamics. The two clustering algorithms (Infomap & Louvain) as well as the two-weight functions 4.4 and 4.5 have been applied to the generated graph. The complete results can be found in Chapter 7 (Appendix).

The evaluation process is divided into two parts: in the first part, the quality of the discovered microservices is evaluated, based on in literature well-defined metrics. In the second part, the discovered microservice candidates were evaluated by a software architect at the paring company who evaluated how well the algorithms and the weight function performed in terms of domain detection and separation.

5.1 Evaluating the Results using metrics

5.1.1 Theoretical Background

Efferent and Afferent Distribution

To evaluate the quality of the generated clusters the efferent and afferent coupling can be taken into account. The **Efferent Coupling** measures the number of microservices on which a given microservice depends. The **Afferent Coupling** measures how many microservices depend on a given microservice. Kohring [10] describes that different studies evaluating large software systems have shown how

Efferent/Afferent Coupling behaves in ”well written” software. The afferent degree distribution should follow Zipf's Law. The efferent degree on the other hand should follow a log-normal distribution.

Instability Index

Metrics are index numbers which are often used to compare and determinate software quality. The chosen quality metric for this thesis is the Instability Index. Taken from object-oriented programming the metric helps to understand how tightly coupled a system is. Loosely coupled systems tend to be more stable than those which are tightly coupled.[13] The goal when designing microservice architectures is to have a high coupling inside the service but a loose coupling between the services [15]. To compute the Instability Index the Efferent/Afferent Coupling has to be determined for each microservice. Then the **Instabiliy Index** can be computed as follows:

$$I(n) = \frac{efferent(microservice_n)}{efferent(microservice_n) + afferent(microservice_n)} \tag{5.1}$$

Large values of $I(n)$ indicate an unstable microservice which is likely to change more often than a microservice with a relatively smaller value of $I(n)$. However as Kohring [10] stated, the importance of an element not only depends on the number of afferent or efferent links but also on the importance of the element from which the afferent link arises. Or in other words: small but central elements can be equally or even more important to the application depending on where from the afferent connection arises. Elements with a high degree of centrality immediately have a greater impact on development, maintenance and lastly on future architectural decisions. To respect this phenomenon one needs to quantify the importance of a node in a graph based on the importance of the nodes linking to it. Previous work has been done in this field i.e. in 1999 by Page et.al. [17]. The Page-Rank $P(n)$ is an iterative algorithm based on Markov-Chains which tries to identify more or less important elements in a network PageRank. Kohring [10] does not propose any enlargements of the Instability Index through the Page-Rank. To assign more significance to the Instability Index, in this work in the evaluation phase, not only the Instability Index but also the Page-Rank is taken into account. This enlargement allows drawing conclusions about how unstable and how central a cluster is to the system. The higher the Page-Rank, the more central and unstable a cluster is. This

i.e. affects fields like development and maintenance. Furthermore, a high value for $P(n)$ can help to predict how many active running instances for this cluster are needed.

To compute $P(n)$ a new graph is created based on the clusters found in the clustering step described in Chapter 4. When a node inside the cluster has a dependency to a node in another cluster, the clusters are connected. The weight required to compute the Page Rank is computed by taking the sum of nodes who have a dependency in a specific cluster.

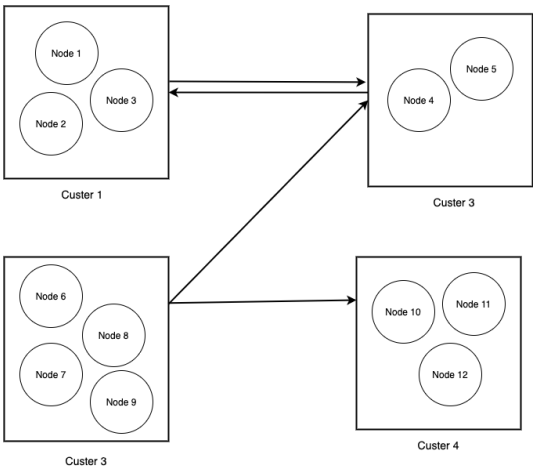


Figure 5.1: Graph created based on individual clusters

5.1.2 Evaluation Results

This part of the evaluation is done by comparing the two clustering algorithms as well the two-weight functions 4.4 and 4.5 against each other. To do so each algorithm has been applied the graph respective weighted with 4.4 and 4.5.

Furthermore, a requirement for $I(n)$ is defined: The Instability Index should follow a log-normal distribution. This requirement follows an easy intuition: This thesis describes an approach for decomposing legacy systems with - to a greater or lesser extent- business-centric modules. Therefore there will always be a few components which tend to be highly unstable. By adding this requirement as a quality metric, it can be assured that only a few microservice candidates tend to be more unstable whereas the rest tends to be stable.

In the following sections, the results are discussed. For ease of understanding each combination gets a short name. The short names are defined defined in 5.1. Each following distribution plots for Afferent/Efferent Coupling as well as for the Instability Index can be interpreted as a linear trend line. All results are listed in Chapter 7 (Appendix).

Short Name	Algorithm	Weightfunction
L-DIV	Louvain	4.5
L-MULT	Louvain	4.4
I-DIV	Infomap	4.5
I-MULT	Infomap	4.4

Table 5.1

Approach 1.1 - I-DIV

As proposed by Kohring et. al. [10], the Afferent Coupling should follow a log-normal distribution whereas the Efferent Distribution should behave like Zipf's law.

By plotting the Afferent Coupling Figure Figure (5.2), it can be seen, that R^2, which is a measure of how close the data-points are to the fitted trend line, is nearly 95%. This means, that the data Afferent Coupling heavily tends to a log-normal distribution. However, when looking at the Efferent Coupling Figure(5.3), R^2 is under 60% and therefore does not follow Zipf's Law.

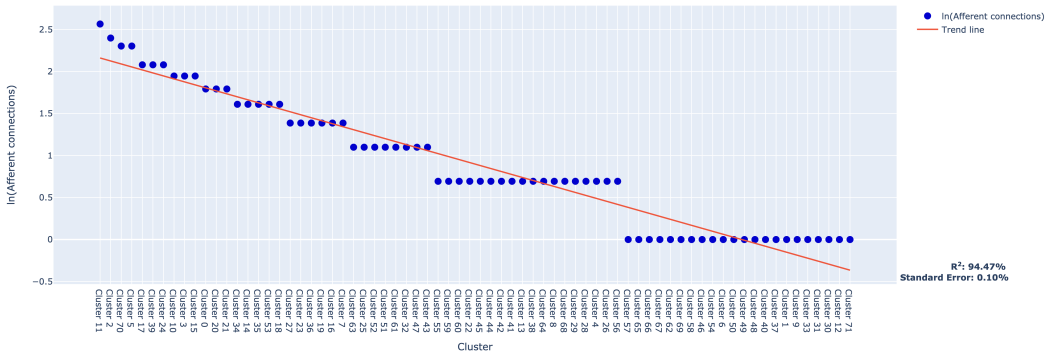


Figure 5.2: Afferent Coupling distribution for Approach 1.1

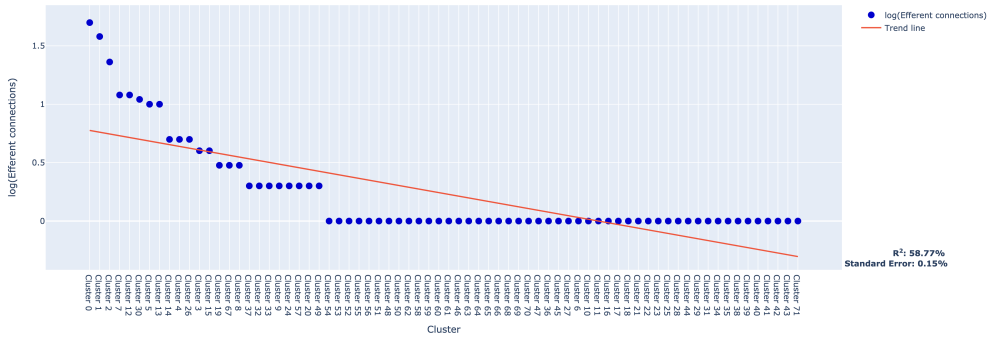


Figure 5.3: Efferent Coupling distribution for Approach 1.1

As defined above, the Instability Index, which provides insights into how stable and maintainable a cluster is, should follow a log-normal distribution. Although most of the clusters tend to high stability, with an R^2 of 55% approach 1.1 does not provide an optimal solution.

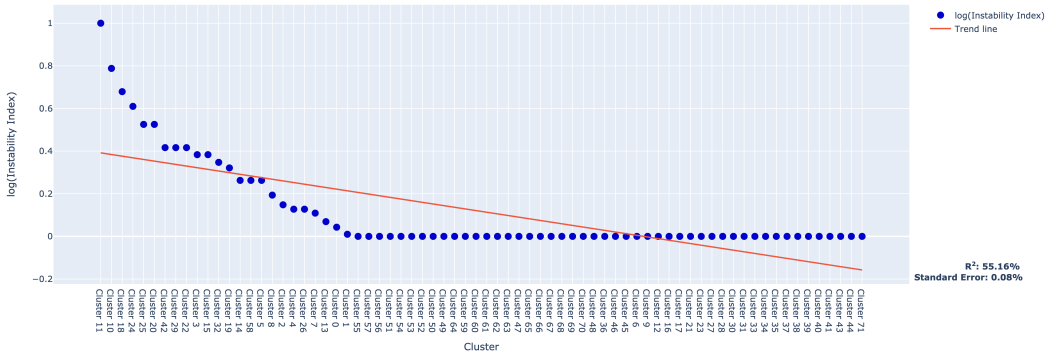


Figure 5.4: Instability Index for Approach 1.1

By altering the Instability Index by the Page Rank, we can observe that clusters which have a small $I(n)$ can be very centric to the system, whereas microservices which, in this approach, tend to have a high $I(n)$ are not core components of the system.

It is important to note, that this approach was able to extract a very centric module Figure (5.5 - Cluster 22) without causing the cluster to have a high $I(n)$.

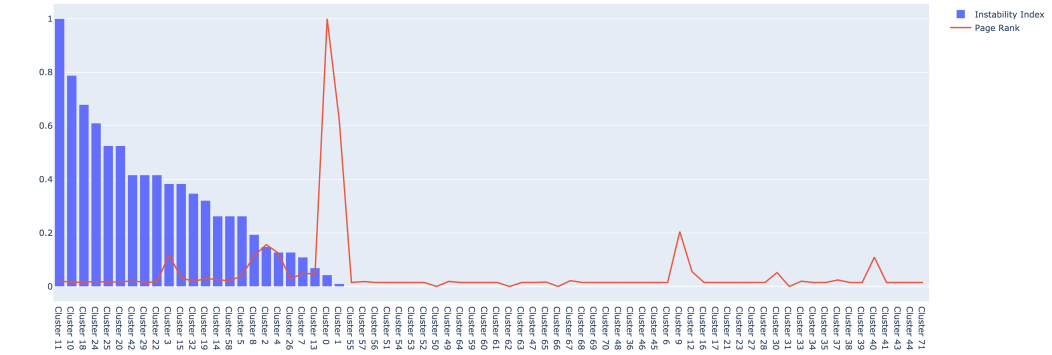


Figure 5.5: Page Rank and Instability Index in clusters

Approach 1.2 - I-MULT

In terms of Afferent and Efferent Coupling distribution, R^2 in approach 1.2 tends to be identical to approach 1.1. The main difference between I-DIV and I-MULT is that I-DIV has one cluster more than the current discussed approach.

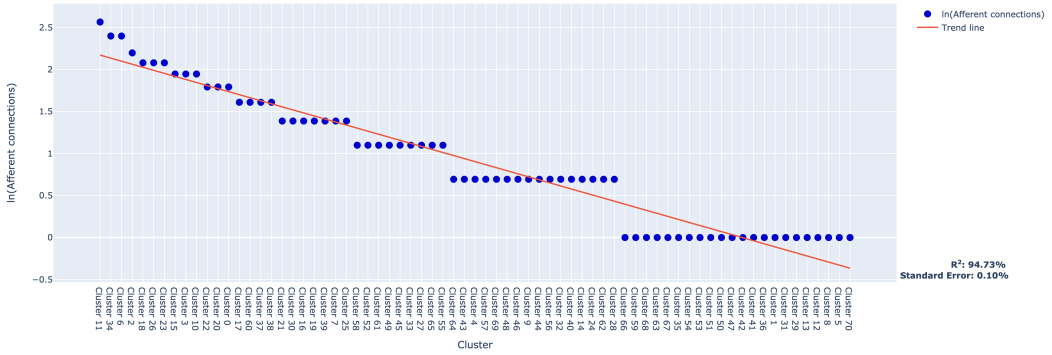


Figure 5.6: Afferent Coupling distribution for Approach 1.2

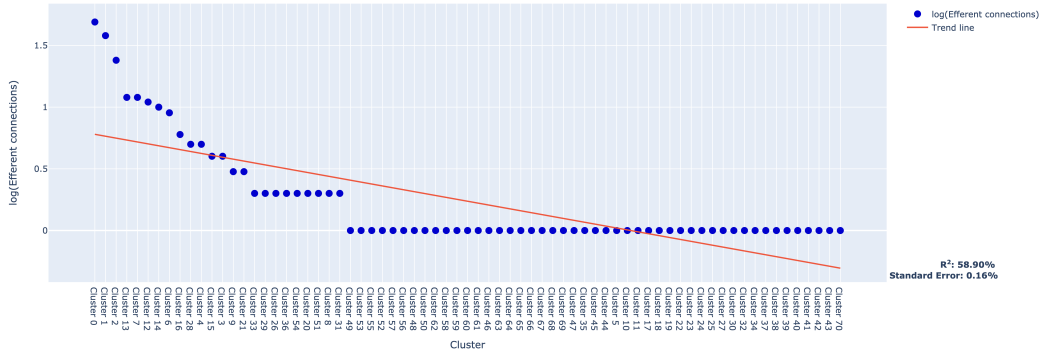


Figure 5.7: Efferent Coupling distribution for Approach 1.2

When evaluating the Instability Index for this approach, the results for R^2 show a slight decrease in accuracy from 55% to 52%.

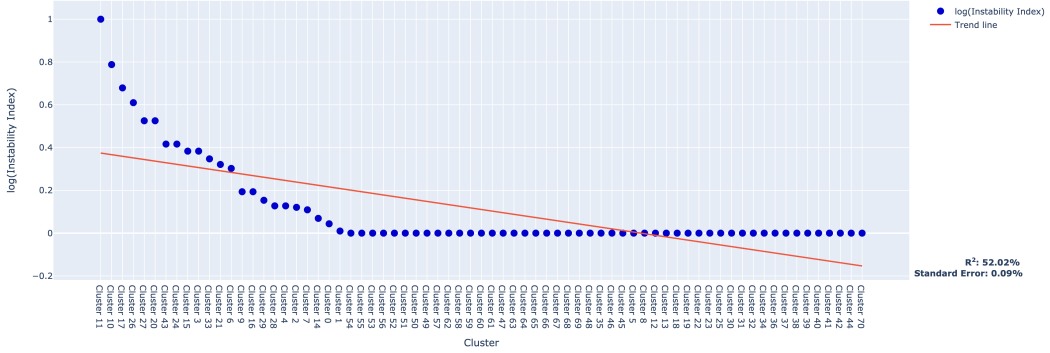


Figure 5.8: Instability Index for Approach 1.2

Also in this approach the clustering algorithm was able to extract a very centric module(5.9 - Cluster 21) without causing the cluster to have a high $I(n)$.

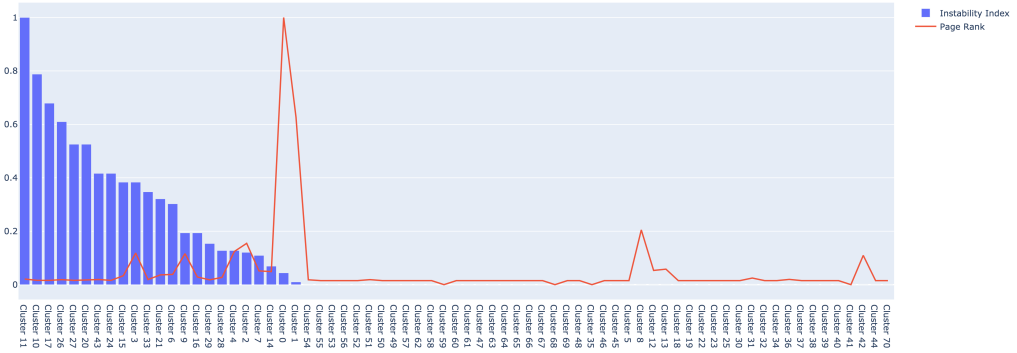


Figure 5.9: Page Rank and Instability Index in clusters

Approach 2.1 - L-DIV

When applying the Louvain Algorithm to the weighted graph, R^2 tends to be as high as in Approach 1.1 and 1.2 when evaluating the Afferent Coupling distribution. When evaluating the Efferent Coupling distribution R^2 is above 79%, which is far better in the above-described approaches.

By far the biggest difference can be observed in the number of clusters. The Louvain Algorithm weighted with 4.5 detects 27 clusters whereas the Infomap Algorithm in both cases detects above 70 clusters. Although the main reason for this is the way how these cluster algorithms work, Louvain (in this approach) tries to keep bigger and therefore more cohesive.

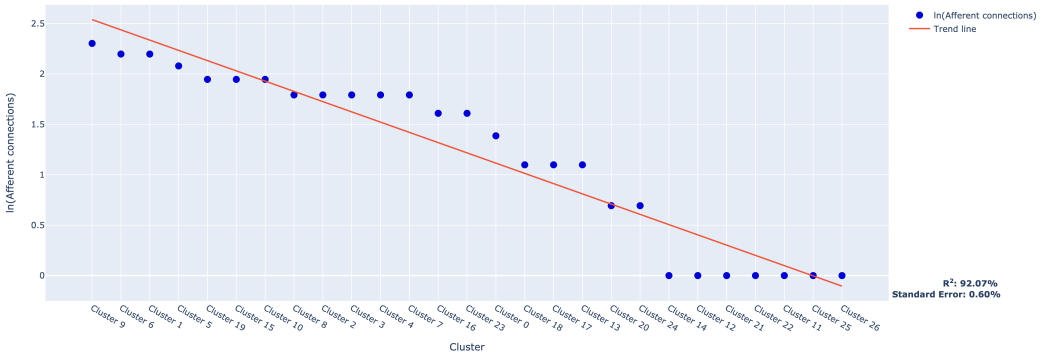


Figure 5.10: Afferent Coupling distribution for Approach 1.2

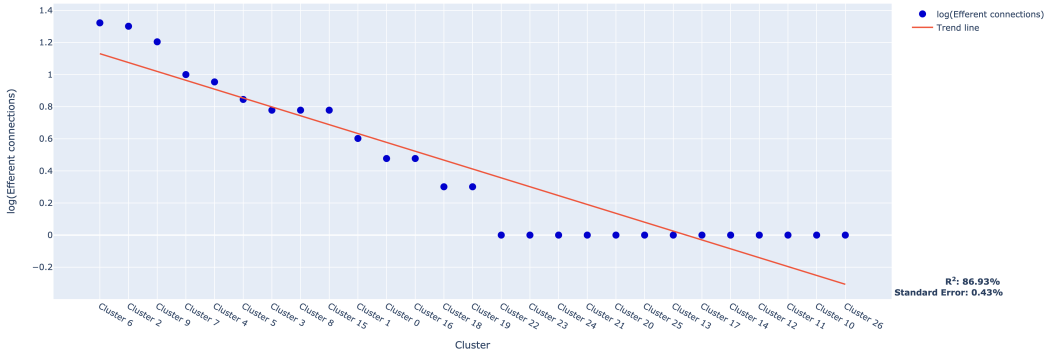


Figure 5.11: Efferent Coupling distribution for Approach 2.1

When evaluating the Instability Index for this approach, R^2 is above 95% which, taking into account the above-defined restriction and obtained results for the approaches using the Infomap Algorithm, is very good.

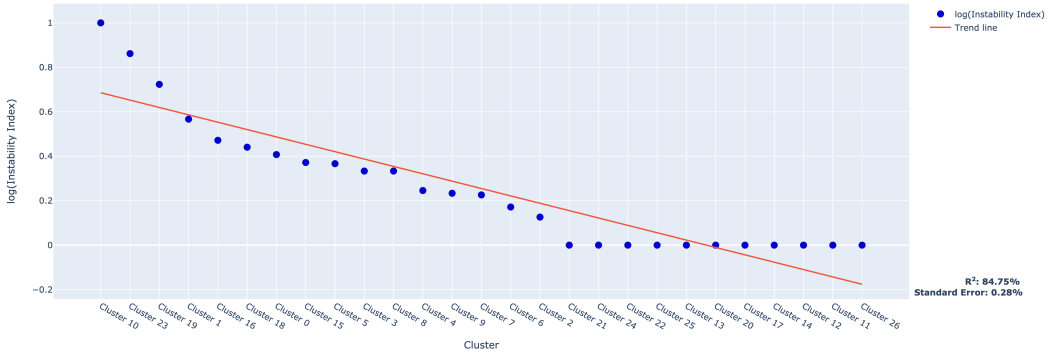


Figure 5.12: Instability Index for Approach 2.1

Similar to the above-evaluated approaches, the clustering algorithm was able to extract application-centric modules Figure (5.13) without causing the cluster to have respectively high $I(n)$.

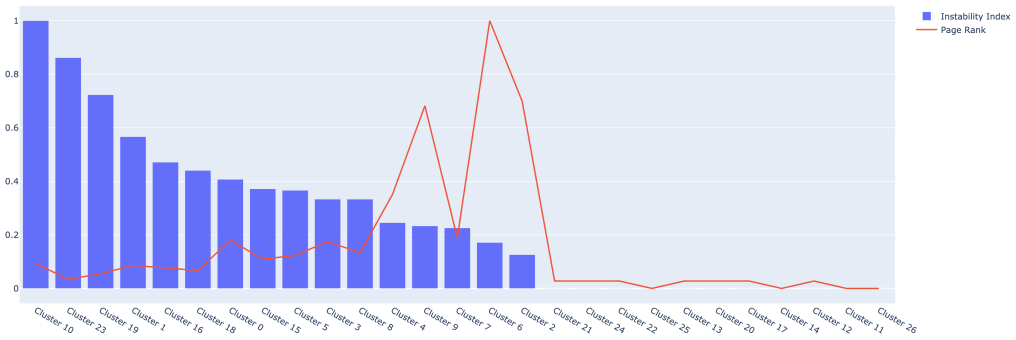


Figure 5.13: Page Rank and Instability Index in clusters

Approach 2.2 - L-MULT

Approach 2.2 behaves like the above described approach 2.1 when comparing R^2 of the Afferent/Efferent Coupling distributions. A notable difference is a slight decrease in both values of R^2. Further L-MULT has one cluster more than the above described L-DIV.

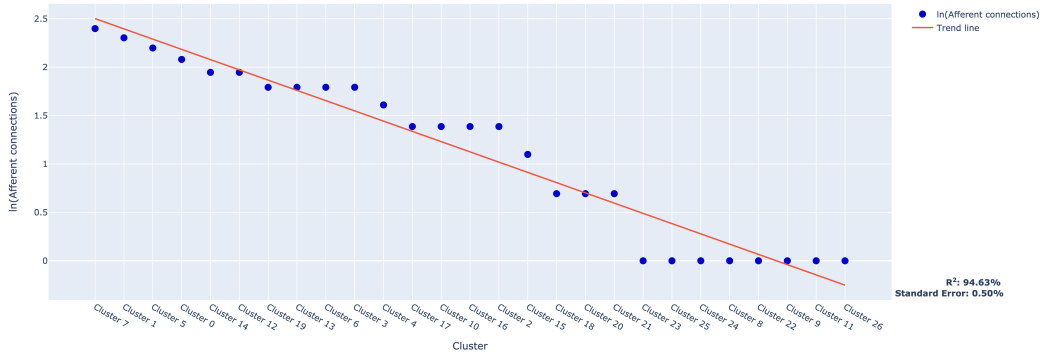


Figure 5.14: Afferent Coupling distribution for Approach 2.2

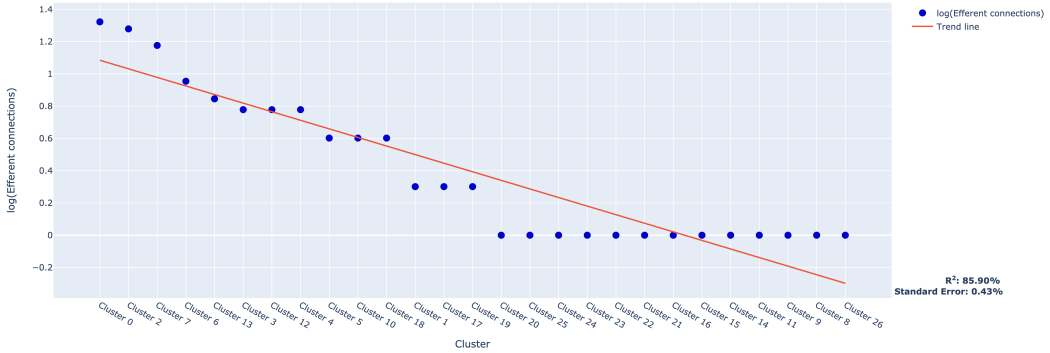


Figure 5.15: Efferent Coupling distribution for Approach 2.2

When evaluating the Instability Index for this approach and comparing it to L-DIV, R^2 decreases about 6%.

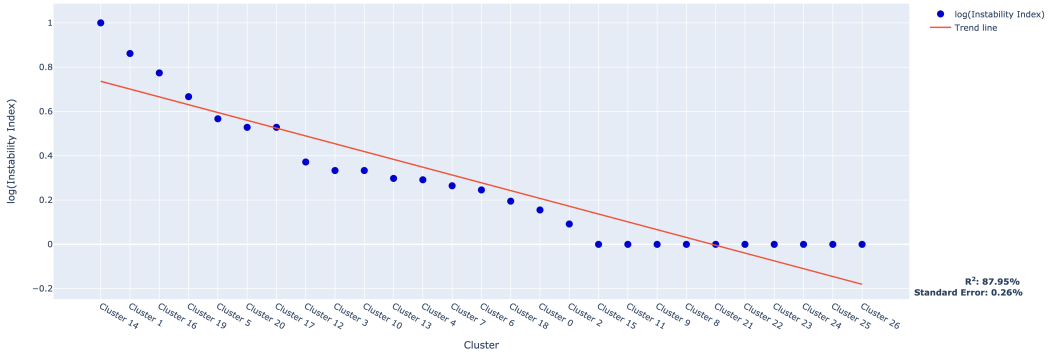


Figure 5.16: Instability Index for Approach 2.2

Also with this weight function, the algorithm was able to isolate application-centric without causing the clusters to have a high $I(n)$.

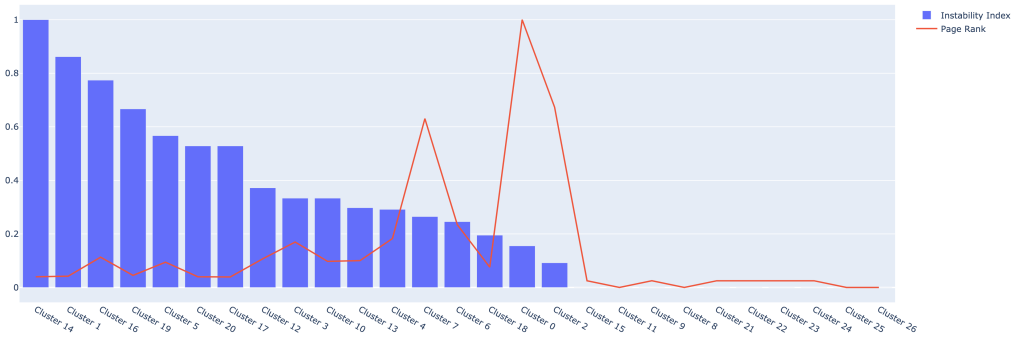


Figure 5.17: Page Rank and Instability Index in clusters

5.2 A domain expert's view

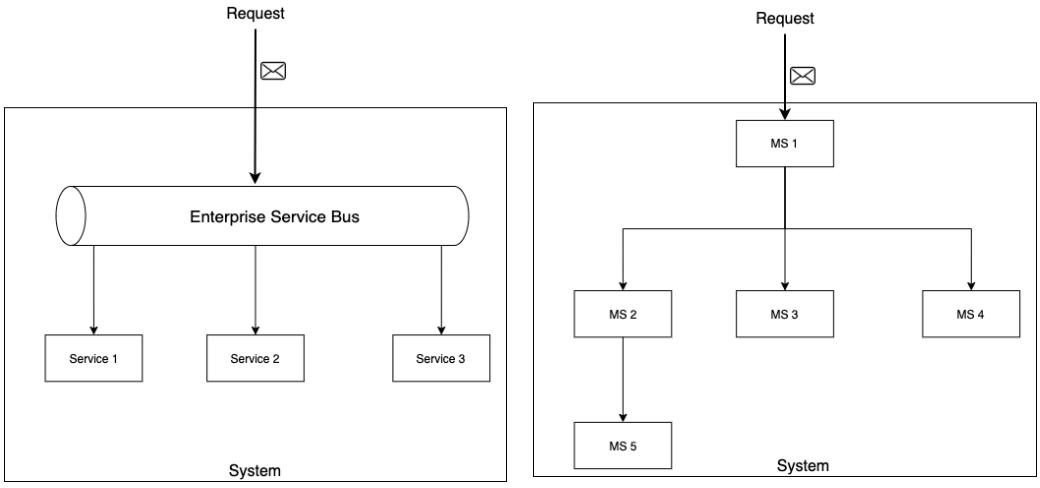
This part of the evaluation is done in cooperation with a software architect of the paring company and aims to answer the following research questions:

- To which extend is domain knowledge necessary for the creation of microservices given an existing system?
- How much does the result of the developed tool differ from a decomposition provided by a domain expert?
- Is it possible to extract the different domains of an existing legacy system to provide a Domain Driven Design approach for extracting microservices based on runtime-data of a legacy system?

Each above described approach comes along with different pros and cons. Although L-DIV and L-MULT have better metrics than I-DIV and I-MULT defined by literature, as described by the external evaluator, do not correctly identify and split domain boundaries. This not only leads to clear architectural no-go's such as the break of the - in the development guidelines defined - Model-View-Controller (MVC) pattern, but also leads to merges of different domains into one domain. These phenomena are especially linked to parts of the application which might overlap in terms of domain or technical implementation. This merge, as described by the software architect, might be logical and rational from a technical and mathematical point of view, but does not reflect any real-world scenarios. In contrast to this, Approach 1.1 and Approach 1.2 do not perform well metric-wise, but were able to detect domain boundaries successfully and provided a relatively good, but sometimes too granular, way of splitting the system while not breaking entirely defined development guidelines such as MVC.

Table 5.2 illustrates the results of the conducted random check performed by the external evaluator at the paring company in detail. While both described approaches perform well on easy to group clusters, mainly Approach 2.1 and Approach 2.2, both of which used the Louvain Algorithm, performed worse on the conducted random sample check. As described by the software architect, the main downside of Approach 1.1 and 1.2 is the high granularity. If the by the tool proposed architecture would be realized, each microservice, as by the Afferent/Efferent Metrics described, would have a high amount of dependencies. This, different when designing new microservice architectures, leads to a massive shift from dependencies allocated in memory to dependencies allocated in the network and could lead to the absurd

scenario in which for processing one request one microservice has dozens of dependencies to other microservices in the network Figure 5.18. This according to literature and the external evaluator, is not wrong but in practice it is not acceptable since it not only increases the infrastructural load such as network but also increases the call complexity of the system and its procedures. Besides this, the testing and deployment strategies of the paring company will be influenced when having a high amount of primary network-based dependencies. Josh Evans in his talk ”Mastering Chaos - A Netflix Guide to Microservices” [6] at the Infoq conference in 2016 explained this problem in detail.



(a) Existing SOA-Architecture: A request is processed by one service. All dependencies are in-memory dependencies.

(b) Mircorservice Architecture: A request can be processed by multiple services causing cascading network dependencies.

Figure 5.18: Shift from in-memory requests to recurring, cascading network requests.

The software architect states that when designing a new microservices based architecture, reoccurring network dependencies should be minimized. The presented approach and the underlying concepts are not designed to optimize such factors. Despite the high grade of granularity the Infomap Algorithm is more accepted by the external evaluator. This has a simple reason: From a human point of view, it’s easier to merge smaller, well-structured domains into bigger ones considering the technical challenge of such a merging process. While merging microservices domains will overlap but since it’s done by a domain expert who is evaluating each case individually, it will not cause any harm to the architecture.

As reported by the external evaluator, the described, unsupervised extraction approach can detect arbitrary well domains and their boundaries and allows Software Architects to gain insights about their system and explore a possible microservice architecture with a marginally overhead for combining services which are too granular.

Approach	L-DIV	L-MULT	I-DIV	I-MULT
Number of investigated microservices	10	10	10	10
of it totally correctly assigned microservices	2	5	6	8
of it at least one component in the microservice is assigned to a wrong domain	3	2	2	1
of it at least one component in the microservice brook a design constraints (i.e. MVC)	3	2	0	0
of it have other issues				
of it major (1)	2*	1*		
of it minor (0.5)			1†	1 †
Score	3 / 10	5 / 10	6.5 / 10	8.5 / 10

Table 5.2: Results of a conducted random check on the quality of the obtained microservices.
* Merged domains, † decoupling of core component which is planned to extract as library

6 Conclusion

This thesis focused on finding microservice candidates in monolithic applications based on collected runtime data. Chapter 2 introduced the terms Monolith, SOA and Microservice their strengths and their trade-offs. Chapter 3 discussed the state of the art semi-automated and automated approaches for microservice extraction. The developed approach was described in Chapter 4 and later evaluated in Chapter 5.

6.1 Outcomes

This section aims to answers the research questions.

RQ1: Are there existing best practices to decompose an existing monolithic system into microservices and where are their key differences?
Outcome: Different state of the art extraction methods have been found during the research. When looking at automated procedures for microservice extractions, methods use static code analysis, heuristics and process mining approaches. Non-algorithmic methods have not been part of the research.

RQ2: To which extend is domain knowledge necessary for the creation of microservices given an existing system?
Outcome: Domain knowledge is necessary when designing microservice architectures using a non-automated approach. As for the elaborated approach - based on the evaluation - one may assume that less is required, but that can not be assured since too few systems have been examined.

RQ3: How can a complex monolithic legacy system be observed at runtime, modelled and split into smaller, independent services?
Outcome: The observation of the system can be done using logs or an APM. While logs can be quite inaccurate as one relies on the usage and implementation of the logging system, APMs can provide information about the application state on every application-level (Controller-Level, Logic-Level, and Data Access Layer). As for the representation of the system the elaborated approach uses a directed graph which can represent the connection between different classes of the system. Splitting is done by applying a graph-clustering algorithm on the graph wichs represents the system.

RQ3.1: In which way have results be presented to software architects and domain experts so that they can gain insights into their application and evaluate microservice candidates?
Outcome: A web-based user interface has been developed where the user can explore and browse the existing monolithic architecture and explore the generated microservice architecture.

RQ3.2: Is it possible to extract the different domains of an existing legacy system to provide a Domain-Driven Design alike approach for extracting microservices based on runtime-data of a legacy system?
Outcome: Overall it can be said that the described approach can represent domains arbitrarily well in a monolithic legacy system.

RQ3.3: How much does the result of the developed tool differ from a decomposition provided by a domain expert?
Outcome: The results of the developed tool differ - to some extend - from the decomposition provided by a domain expert. Depending on the graph-clustering algorithm used, the tool splits the system into granular parts or breaks design patterns such as MVC. As stated by the external evaluator: Although some minor differences, the tool provides good insights and possible - not 100% accurate - microservice decomposition of the observed monolithic system.

RQ3.4: How can the quality of the created microservices be assessed?
Outcome: The quality of a microservice architecture can be assessed via specified metrics such as Afferent/Efferent distribution, Instability Index distribution or by a domain expert. Further, this thesis elaborates a more sophisticated way to detect the influence and the dependencies between microservices via the Page Rank.

6.2 Limitations & Future Work

Despite the generally positive observations in the evaluation, the presented approach has its limitations. Probably the biggest is the focus on class level. While the computation of the graph on class level leads to a well structured and for humans easily readable graph, architectural misconceptions can not be tackled down entirely. To make an example: Two classes are in the same cluster because one class uses only one specific function of the other class. In this case, the better solution (if applicable in respect to the microservice principles) would be to migrate only the function and assign the class to a cluster where it might fit better. Defining class-methods as the smallest unit of measurement would bring more granularity - for sure this would be an endorsable feature but it might also decrease the human understanding of the result and increase the potential of wrong coupling.

Described microservice architectures by this tool are very sophisticated and domain-oriented respecting the simplicity of the approach but can only be seen as recommendations and not as finished and fixed architecture. This is due to the fact, that a microservice in practice is more than a set of classes [15]. To increase the precision, granularity and to cope with domain-specific restrains such as design patterns, domain requirements, security aspects and 3rd party connections, the described approach has to be altered. This can be done by adding i.e. a supervised learning algorithm. In this case a domain expert could define some ideal structures and the system would try to build microservices as similar as possible to the defined ones.

A big strength of the described approach is that it can be used in different domains as long as the provided input data is correct. The plan is to further test the approach in cooperation with other companies and to elaborate a more sophisticated and company-focused way of extracting by allowing the user to input company-specific constrains.

7 Appendix

7.1 I-MULT

Name	Afferent	Efferent	Instability Index	Page Rank
Cluster 0	6	49	0.890909	0.275644
Cluster 1	1	38	0.974359	0.172882
Cluster 2	9	24	0.727273	0.042600
Cluster 3	7	4	0.363636	0.032560
Cluster 4	2	5	0.714286	0.034228
Cluster 5	1	0	0.000000	0.004123
Cluster 6	11	9	0.450000	0.010577
Cluster 7	4	12	0.750000	0.014129
Cluster 8	0	2	1.000000	0.056456
Cluster 9	2	3	0.600000	0.031923
Cluster 10	7	1	0.125000	0.004442
Cluster 11	13	1	0.071429	0.005769
Cluster 12	0	11	1.000000	0.014622
Cluster 13	0	12	1.000000	0.015955
Cluster 14	2	10	0.833333	0.013439
Cluster 15	7	4	0.363636	0.009226
Cluster 16	4	6	0.600000	0.007852
Cluster 17	5	1	0.166667	0.004393
Cluster 18	8	0	0.000000	0.004123
Cluster 19	4	0	0.000000	0.004123
Cluster 20	6	2	0.250000	0.004743
Cluster 21	4	3	0.428571	0.010050
Cluster 22	6	0	0.000000	0.004123
Cluster 23	8	0	0.000000	0.004123
Cluster 24	2	1	0.333333	0.004393

Name	Afferent	Efferent	Instability Index	Page Rank
Cluster 25	4	0	0.000000	0.004123
Cluster 26	8	2	0.200000	0.005167
Cluster 27	3	1	0.250000	0.004368
Cluster 28	2	5	0.714286	0.007476
Cluster 29	1	2	0.666667	0.004911
Cluster 30	4	0	0.000000	0.004123
Cluster 31	0	2	1.000000	0.006771
Cluster 32	2	0	0.000000	0.004123
Cluster 33	3	2	0.400000	0.005244
Cluster 34	11	0	0.000000	0.004123
Cluster 35	0	0	0.000000	0.000000
Cluster 36	0	2	1.000000	0.005420
Cluster 37	5	0	0.000000	0.004123
Cluster 38	5	0	0.000000	0.004123
Cluster 39	4	0	0.000000	0.004123
Cluster 40	2	0	0.000000	0.004123
Cluster 41	0	0	0.000000	0.000000
Cluster 42	0	1	1.000000	0.030155
Cluster 43	2	1	0.333333	0.005291
Cluster 44	2	0	0.000000	0.004123
Cluster 45	3	0	0.000000	0.004123
Cluster 46	2	0	0.000000	0.004123
Cluster 47	1	0	0.000000	0.004123
Cluster 48	2	0	0.000000	0.004123
Cluster 49	3	0	0.000000	0.004123
Cluster 50	1	0	0.000000	0.004123
Cluster 51	0	2	1.000000	0.005175
Cluster 52	3	0	0.000000	0.004123
Cluster 53	1	0	0.000000	0.004123
Cluster 54	0	2	1.000000	0.005003
Cluster 55	3	0	0.000000	0.004123
Cluster 56	2	0	0.000000	0.004123
Cluster 57	2	0	0.000000	0.004123
Cluster 58	3	0	0.000000	0.004123
Cluster 59	0	0	0.000000	0.000000
Cluster 60	5	0	0.000000	0.004123

Name	Afferent	Efferent	Instability Index	Page Rank
Cluster 61	3	0	0.000000	0.004123
Cluster 62	2	0	0.000000	0.004123
Cluster 63	0	1	1.000000	0.004393
Cluster 64	2	0	0.000000	0.004123
Cluster 65	3	0	0.000000	0.004123
Cluster 66	0	1	1.000000	0.004561
Cluster 67	1	0	0.000000	0.004123
Cluster 68	0	0	0.000000	0.000000
Cluster 69	2	0	0.000000	0.004123
Cluster 70	1	0	0.000000	0.004123

Table 7.1: Complete results for I-MULT

7.2 I-DIV

Name	Afferent	Efferent	Instability Index	Page Rank
Cluster 0	6	50	0.892857	0.276160
Cluster 1	1	38	0.974359	0.171123
Cluster 2	11	23	0.676471	0.043168
Cluster 3	7	4	0.363636	0.032504
Cluster 4	2	5	0.714286	0.034276
Cluster 5	10	10	0.500000	0.011356
Cluster 6	1	0	0.000000	0.004072
Cluster 7	4	12	0.750000	0.013932
Cluster 8	2	3	0.600000	0.031709
Cluster 9	0	2	1.000000	0.056584
Cluster 10	7	1	0.125000	0.004387
Cluster 11	13	1	0.071429	0.005601
Cluster 12	0	12	1.000000	0.015361
Cluster 13	2	10	0.833333	0.012471
Cluster 14	5	5	0.500000	0.007196
Cluster 15	7	4	0.363636	0.008924
Cluster 16	4	0	0.000000	0.004072
Cluster 17	8	0	0.000000	0.004072
Cluster 18	5	1	0.166667	0.004419

Name	Afferent	Efferent	Instability Index	Page Rank
Cluster 19	4	3	0.428571	0.008111
Cluster 20	6	2	0.250000	0.004765
Cluster 21	6	0	0.000000	0.004072
Cluster 22	2	1	0.333333	0.004419
Cluster 23	4	0	0.000000	0.004072
Cluster 24	8	2	0.200000	0.005104
Cluster 25	3	1	0.250000	0.004310
Cluster 26	2	5	0.714286	0.007482
Cluster 27	4	0	0.000000	0.004072
Cluster 28	2	0	0.000000	0.004072
Cluster 29	2	1	0.333333	0.004419
Cluster 30	0	11	1.000000	0.014360
Cluster 31	0	0	0.000000	0.000000
Cluster 32	3	2	0.400000	0.005176
Cluster 33	0	2	1.000000	0.005430
Cluster 34	5	0	0.000000	0.004072
Cluster 35	5	0	0.000000	0.004072
Cluster 36	4	0	0.000000	0.004072
Cluster 37	0	2	1.000000	0.006703
Cluster 38	2	0	0.000000	0.004072
Cluster 39	8	0	0.000000	0.004072
Cluster 40	0	1	1.000000	0.030155
Cluster 41	2	0	0.000000	0.004072
Cluster 42	2	1	0.333333	0.005601
Cluster 43	3	0	0.000000	0.004072
Cluster 44	2	0	0.000000	0.004072
Cluster 45	2	0	0.000000	0.004072
Cluster 46	1	0	0.000000	0.004072
Cluster 47	3	0	0.000000	0.004072
Cluster 48	1	0	0.000000	0.004072
Cluster 49	0	2	1.000000	0.005192
Cluster 50	0	0	0.000000	0.000000
Cluster 51	3	0	0.000000	0.004072
Cluster 52	3	0	0.000000	0.004072
Cluster 53	5	0	0.000000	0.004072
Cluster 54	1	0	0.000000	0.004072

Name	Afferent	Efferent	Instability Index	Page Rank
Cluster 55	2	0	0.000000	0.004072
Cluster 56	2	0	0.000000	0.004072
Cluster 57	0	2	1.000000	0.005031
Cluster 58	1	1	0.500000	0.005601
Cluster 59	2	0	0.000000	0.004072
Cluster 60	2	0	0.000000	0.004072
Cluster 61	3	0	0.000000	0.004072
Cluster 62	0	0	0.000000	0.000000
Cluster 63	3	0	0.000000	0.004072
Cluster 64	2	0	0.000000	0.004072
Cluster 65	0	1	1.000000	0.004505
Cluster 66	0	0	0.000000	0.000000
Cluster 67	0	3	1.000000	0.006005
Cluster 68	2	0	0.000000	0.004072
Cluster 69	1	0	0.000000	0.004072
Cluster 70	10	0	0.000000	0.004072
Cluster 71	1	0	0.000000	0.004072

Table 7.2: Complete results for I-DIV

7.3 L-MULT

Name	Afferent	Efferent	Instability Index	Page Rank
Cluster 0	8	20	0.714286	0.259608
Cluster 1	10	3	0.230769	0.014339
Cluster 2	4	18	0.818182	0.178700
Cluster 3	7	6	0.461538	0.041880
Cluster 4	5	6	0.545455	0.053855
Cluster 5	9	5	0.357143	0.021240
Cluster 6	7	9	0.562500	0.061537
Cluster 7	11	16	0.592593	0.133134
Cluster 8	0	0	0.000000	0.000000
Cluster 9	6	9	0.600000	0.049198
Cluster 10	1	0	0.000000	0.006818
Cluster 11	5	4	0.444444	0.022895

Name	Afferent	Efferent	Instability Index	Page Rank
Cluster 12	0	0	0.000000	0.000000
Cluster 13	6	6	0.500000	0.017572
Cluster 14	7	6	0.461538	0.030252
Cluster 15	7	1	0.125000	0.009390
Cluster 16	5	3	0.375000	0.026516
Cluster 17	3	0	0.000000	0.006818
Cluster 18	4	1	0.200000	0.028517
Cluster 19	5	2	0.285714	0.010458
Cluster 20	2	0	0.000000	0.006818
Cluster 21	1	0	0.000000	0.006818
Cluster 22	1	0	0.000000	0.006818
Cluster 23	1	0	0.000000	0.006818
Cluster 24	0	0	0.000000	0.000000
Cluster 25	0	0	0.000000	0.000000

Table 7.3: Complete results for L-MULT

7.4 L-DIV

Name	Afferent	Efferent	Instability Index	Page Rank
Cluster 0	9	17	0.653846	0.259117
Cluster 1	8	3	0.272727	0.020574
Cluster 2	6	15	0.714286	0.180373
Cluster 3	6	5	0.454545	0.049080
Cluster 4	4	8	0.666667	0.082238
Cluster 5	9	5	0.357143	0.030702
Cluster 6	8	8	0.500000	0.037603
Cluster 7	9	13	0.590909	0.159383
Cluster 8	2	2	0.500000	0.018924
Cluster 9	0	0	0.000000	0.000000
Cluster 10	1	0	0.000000	0.007895
Cluster 11	5	4	0.444444	0.027684
Cluster 12	0	0	0.000000	0.000000
Cluster 13	7	6	0.461538	0.028619
Cluster 14	5	4	0.444444	0.018546

Name	Afferent	Efferent	Instability Index	Page Rank
Cluster 15	4	1	0.200000	0.023226
Cluster 16	3	2	0.400000	0.019208
Cluster 17	5	2	0.285714	0.013144
Cluster 18	2	0	0.000000	0.007895
Cluster 19	1	0	0.000000	0.007895
Cluster 20	1	0	0.000000	0.007895
Cluster 21	0	0	0.000000	0.000000
Cluster 22	0	0	0.000000	0.000000

Table 7.4: Complete results for L-DIV

Bibliography

[1] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, oct 2008.

[2] L. Bohlin, D. Edler, A. Lancichinetti, and M. Rosvall. *Community Detection and Visualization of Networks with the Map Equation Framework*, pages 3–34. 09 2014.

[3] A. Carrasco, B. v. Bladel, and S. Demeyer. Migrating towards microservices: Migration and architecture smells. In *Proceedings of the 2nd International Workshop on Refactoring*, IWoR 2018, page 1–6, New York, NY, USA, 2018. Association for Computing Machinery.

[4] R. Chen, S. Li, and Z. E. Li. From monolith to microservices: A dataflow-driven approach. pages 466–475, 12 2017.

[5] A. De Alwis, A. Barros, A. Polyvyanyy, and C. Fidge. Function-splitting heuristics for discovery of microservices in enterprise systems. pages 37–53, 11 2018.

[6] J. Evans. *Mastering Chaos - A Netflix Guide to Microservices*, accessed December 19, 2020. https://www.infoq.com/presentations/netflix-chaos-microservices/.

[7] M. Fowler. *Monolith First*, 2015 (accessed July 23, 2020). https://martinfowler.com/bliki/MonolithFirst.html.

[8] M. Gysel, L. Kölbener, W. Giersche, and O. Zimmermann. Service cutter: A systematic approach to service decomposition. pages 185–200, 09 2016.

[9] N. M. Josuttis. *SOA in practice: the art of distributed system design.* " O'Reilly Media, Inc.", 2007.

[10] G. Kohring. Complex dependencies in large software systems. *Advances in Complex Systems*, 12(06):565–581, 2009.

[11] A. Krause, C. Zirkelbach, W. Hasselbring, S. Lenga, and D. Kröger. Microservice decomposition via static and dynamic analysis of the monolith, 2020.

[12] J. Lewis and M. Fowler. *Microservices - a definition of this new architectural term*, 2014 (accessed July 23, 2020). https://martinfowler.com/articles/microservices.html.

[13] R. C. Martin. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall PTR, USA, 2003.

[14] 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, 2017.

[15] S. Newman. *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, 1st edition, February 2015.

[16] N. Niknejad, W. Ismail, I. Ghani, B. Nazari, M. Bahari, and A. R. B. C. Hussin. Understanding service-oriented architecture (soa): A systematic literature review and directions for further investigation. *Information Systems*, 91:101491, 2020.

[17] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

[18] F. Ponce, G. Márquez, and H. Astudillo. Migrating from monolithic architecture to microservices: A rapid review. In *2019 38th International Conference of the Chilean Computer Science Society (SCCC)*, pages 1–7, 2019.

[19] C. Richardson. *Pattern: Monolithic architecture*, accessed July 23, 2020. https://microservices.io/patterns/monolithic.html.

[20] M. Rosvall, D. Axelsson, and C. T. Bergstrom. The map equation. *The European Physical Journal Special Topics*, 178(1):13–23, Nov 2009.

[21] D. Taibi, V. Lenarduzzi, and C. Pahl. Processes, motivations and issues for migrating to microservices architectures: An empirical investigation. *IEEE Cloud Computing*, 4, 10 2017.

[22] D. Taibi and K. Systä. From monolithic systems to microservices: A decomposition framework based on process mining. In *CLOSER*, 2019.

[23] O. Zimmermann. Microservices tenets: Agile approach to service development and deployment. *Computer Science - Research and Development*, 11 2016.