Pontifícia Universidade Católica
do Rio de Janeiro

Luiz Matheus de Alencar Carvalho

Identifying Microservices Candidates in Legacy Code

Dissertação de Mestrado

Dissertation presented to the Programa de Pós–graduação in Informática of the PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática.

Advisor: Prof. Alessandro Fabricio Garcia

Rio de Janeiro
April 2020

Pontifícia Universidade Católica
do Rio de Janeiro

Luiz Matheus de Alencar Carvalho

Identifying Microservices Candidates in Legacy Code

Dissertation presented to the Programa de Pós–graduação in Informática of the PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática. Approved by the Examination Committee.

Prof. Alessandro Fabricio Garcia
Advisor
Departamento de Informática – PUC-Rio

Prof. Marcos Kalinowski
Pontifícia Universidade Catolica do Rio de Janeiro – PUC-Rio

Renato Fontoura De Gusmão Cerqueira
IBM Research Brasil – IBM

Prof. Rafael Maiani de Mello
Centro Federal de Educação Tecnológica Celso Suckow da Fonseca – CEFET-RJ

Rio de Janeiro, April the 28th, 2020

Luiz Matheus de Alencar Carvalho

I am a Master's student of Informatics at Pontifical Catholic University of Rio de Janeiro (PUC-Rio) and a member of OPUS Research Group at PUC-Rio. My research mainly focuses on the process of migrating to a microservice architecture. In particular, I am interested in improving techniques for supporting developers along the identification and extraction of microservices. I conducted empirical studies to understand what criteria are considered relevant by specialists involved in the process of migrating to a microservice architecture. Besides, I am graduated in Computer Science at the Federal University of Alagoas (UFAL), where I conducted research on refactoring, software product line, and software testing.

Bibliographic data

Carvalho, Luiz Matheus de Alencar

Identifying Microservices Candidates in Legacy Code / Luiz Matheus de Alencar Carvalho; advisor: Alessandro Fabricio Garcia. – Rio de janeiro: PUC-Rio, Departamento de Informática, 2020.

v., 120 f: il. color. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui bibliografia

1. Arquitetura de Microsserviços;. 2. Extração de Microsserviços;. 3. Arquitetura de Software;. 4. Evolução de Software;. 5. Variabilidade de Software;. I. Garcia, Alessandro Fabricio. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

I would like to dedicate this thesis to my family, girlfriend, friends, and professors. Without these loved and kind people, this thesis would not be feasible.

Acknowledgments

First, thanks to all professors who provided intellectual support and incentive to make this work possible. Furthermore, for the support so that the love for learning remained puissant. To my family, for the formation and support provided throughout my life through the most diverse ways and instruments. I would also want to express gratitude to my girlfriend and friend Silvia Costa for the endless conversations, understanding, and patience during my formation. Moreover, to my friends for the moments of group study, encouragement, esteem, or casualness. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Abstract

Carvalho, Luiz Matheus de Alencar; Garcia, Alessandro Fabricio (Advisor). **Identifying Microservices Candidates in Legacy Code**. Rio de Janeiro, 2020. 120p. Dissertação de mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Microservices is an industrial technique to promote better scalability and maintainability of small and autonomous services. Previous studies suggested that microservice architectures have been widely used to reduce limitations found in legacy monolithic systems such as the inclusion of innovation, use of a different stack of technologies, among others. The process of migrating to a microservices architecture is far from trivial. This is particularly the case for the task of identifying candidate microservices and the source code associated with each candidate, which is recognizably time-consuming and error-prone. Thus, automated approaches have been proposed to reduce the effort related to that task. These approaches commonly adopt one or two criteria to support the identification of microservices from a legacy monolithic system. However, there is a lack of understanding on the usefulness of these criteria in practical settings. Moreover, there is limited knowledge on what are the criteria that practitioners consider relevant. Taking into account these existing limitations, we conducted a survey and interviews to better understand the usefulness of criteria reported in empirical studies (e.g, case studies and reports) from the point of view of practitioners. The results of the survey and interviews revealed that existing automated approaches and tools are far from being aligned with practical needs. To fulfill these needs, this work defines a automated approach named `toMicroservices`. The approach relies on a combination of static and dynamic analysis of the legacy code. The approach aims at indicating the microservice candidates and the corresponding source extracted from the legacy system. `toMicroservices` makes use of search-based software engineering (SBSE) to optimize and balance the five criteria commonly adopted by practitioners, namely feature modularization, network overhead reduction, reuse, coupling and cohesion. Additionally, an industrial case study and a focus group were conducted *a posteriori* to support the evaluation and improvements of `toMicroservices`.

Keywords

Microservices Architecture; Microservices Extraction; Software Architecture; Software evolution; Software Variability;

Resumo

Carvalho, Luiz Matheus de Alencar; Garcia, Alessandro Fabricio. **Identificando Candidatos a Microsserviços em Código Legado**. Rio de Janeiro, 2020. 120p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Microsserviços é uma técnica industrial para promover melhor escalabilidade e manutenibilidade de pequenos e autônomos serviços. Estudos prévios sugerem que a arquitetura de microsserviços vem sendo amplamente usada para reduzir limitações encontradas em sistemas monolíticos legados tais como melhoria de inovação, uso de diferentes tecnologias, entre outras. O processo de migração para a arquitetura de microsserviços não é trivial. Este é particularmente o caso da tarefa de identificar candidatos a microsserviço e o código fonte associado com cada candidato que é dispendiosa e propensa a erro. Consequentemente, abordagens automatizadas têm sido propostas para reduzir o esforço relacionado a essa atividade. Essas abordagens comumente adotam um ou dois critérios para suportar a identificação de microsserviços com base no sistema monolítico legado. Contudo, existe uma falta de compreensão da utilidade desses critérios adotados na prática. Além disso, há limitado conhecimento em quais são os critérios que profissionais consideram relevantes. Levando em consideração esses limitantes existentes, nós conduzimos um survey e entrevista para melhor compreender a utilidade de critérios relatados em estudos empíricos (e.g, estudos de caso e relatos) do ponto de vista dos profissionais. Os resultados do survey e da entrevista mostram que as abordagens automatizadas e ferramentas existentes não são totalmente alinhadas com necessidades práticas. Para atender às necessidades deles, este trabalho define uma abordagem automatizada chamada `toMicroservices`. A abordagem baseia-se em uma combinação de análise estática e dinâmica do código legado. A abordagem visa indicar os candidatos a microsserviço e a fonte correspondente extraído do sistema legado. `toMicroservices` faz uso da engenharia de software baseada em busca para otimizar e balancear os cinco critérios comumente adotados por profisionais, nomeados de modularização de funcionalidade, redução de sobrecarga de rede, reúso, acoplamento e coesão. Além disso, um estudo de caso e grupo focal foram conduzidos *a posteriori* para avaliar e melhorar `toMicroservices`.

Palavras-chave

Arquitetura de Microsserviços; Extração de Microsserviços; Arquitetura de Software; Evolução de Software; Variabilidade de Software;

Table of contents

List of figures

List of tables

A inspiração mais profunda da ciência não é um privilégio dos cientistas, porque a exigência da ordem se encontra presente mesmo nos níveis mais primitivos da vida.

Rubem Alves, *Filosofia da Ciência: introdução ao jogo e suas regras.*

1
Introduction

Microservices are small and autonomous services that work together by using lightweight communication protocols (1). The notion of small refers to the need for producing fine-grained microservices with each modularizing a single *feature* (2). The notion of autonomous implies that each microservice should be highly decoupled (1, 2). In the same vein, lightweight protocols are adopted to avoid overheads and further improving decoupling. These microservice characteristics (1, 2) facilitate the incorporation of variability in a software's system. In fact, a well-designed *microservice architecture* promotes separation of features as tiny services, which favors the design of configurable systems (3).

Many companies have been adopting microservice architectures to modernize legacy systems (3, 4, 5, 6, 7, 8). Legacy systems are commonly found in industry and represent a long term massive investment. Despite their business importance, legacy systems are difficult to extend, include innovation, and expensive to maintain (9, 10). These systems usually have a monolithic architecture, with components realizing tangled features and being strongly connected with each other (6, 11).

There are many factors driving the process of migrating a system to a microservice architecture. Previous studies indicate that practitioners involved in these migrations are mainly motivated to improve the maintainability of existing systems (6, 11). Moreover, the benefits perceived by practitioners and reported in the microservices adoption are: reduced effort for maintenance and evolution, increased availability of services, easiness of innovation, continuous delivery, easiness of DevOps incorporation, facilitated scalability of components with more demand, and the like (6, 11).

However, there are various challenges along the process of migrating legacy systems to a microservice architecture. Separating features of a legacy system into small units to create the microservices is reported as one of the main challenges by practitioners (1, 6, 11). To perform this task, practitioners should reflect upon the source code structure of the legacy system (12).

Automated approaches have been proposed to analyze legacy systems with the goal of identifying microservice candidates (13, 14, 15, 16). These

approaches also determine which legacy code elements may be used to support the implementation of each microservice candidate. There are certain criteria, some of them adopted by existing automated approaches, to support decisions on deriving a microservice architecture from a legacy system.

In this work, a criterion is an aspect used to support the decision making process to transform (either partially or fully) the legacy system in a microservice architecture. Each criterion requires the extraction of information from either the structure or execution of the legacy system. Automated approaches usually adopt coupling and cohesion in the target system as the key criteria (13, 14, 15, 16). However, there is a lack of understanding about the importance of these criteria and other ones for supporting the adequate identification of microservice candidates.

In this research, we address three research problems. We describe them below with their corresponding solutions proposed in this dissertation. Section 1.1 introduces our empirical studies aimed at assessing the usefulness of criteria observed in legacy systems by practitioners. Developers may also take into consideration variabilities already defined in the legacy code when "microservifying" their legacy systems. The relationships between variability and microservice architectures are presented in Section 1.2. In particular, we discuss how variability is implemented before and after the process of migrating to microservice architecture and the usefulness of variability to identify microservice candidates.

Section 1.3 introduces a new search-based approach that relies on various criteria to identify microservice candidates. The selected criteria were those considered useful or moderately useful in practice through our empirical studies introduced in Section 1.1. Five criteria are adopted. Moreover, we performed two empirical studies to evaluate and improve our automated approach. These studies are based on an industrial legacy system, which was undergoing a migration to a microservice architecture.

1.1 On the Usefulness of Criteria Observed in Legacy Systems

This section focuses on the problem of revealing which criteria practitioners consider useful when identifying microservice candidates in their systems.

1.1.1 Problem Statement

The source code of legacy systems contains valuable information for enabling the identification of microservices. In fact, practitioners make use of

legacy code along the process of migrating to a microservice architecture (12). Therefore, the use of proper decision-making criteria in this process is of paramount importance. The mistake of not observing adequate criteria when identifying microservices could lead to several problems, including the wrong choices on selecting wrong feature boundaries or even the complete failure of the process (17).

For example, network communication overhead may eventually be disregarded as a key criterion in microservice extraction. However, this negligence may cause afterwards a high network overhead and harming response time as the communication between microservices is made through the network. As far as modularity is concerned, cohesion and coupling are common indicators of which parts of the legacy code could be modularized as microservices (18). In fact, one should not miss to observe cohesion and coupling in the legacy system as they are essential for adequate modularity of the microservice candidates (18).

Several studies have been conducted with experienced practitioners concerning the process of migrating to a microservice architecture (6, 11, 12). They reported that: (i) the processes of identifying and extracting microservices remain as the key challenge (6, 11, 12, 17), and (ii) it is important to observe appropriate criteria for identifying microservices (6, 11, 12, 17). However, there is a lack of understanding on how to identify microservices based on legacy systems in terms of: (i) a list of adequate decision-making criteria, and (ii) how useful are these criteria under the point of view of experienced practitioners.

These prevailing challenges may be motivated by the fact that microservice architectures emerged entirely in the industry and not in the academia, differently from many other architectural styles, such as the blackboard architectures and the N-tier architectures (19, 20). This initial industrial emphasis, without direct involvement of the academia, possibly slowed the process for researchers to fully understand the peculiarities of the intricacies of identifying microservices from legacy systems.

> ***1st problem:*** *Lack of understanding about the criteria for identifying microservices of legacy systems and their usefulness.*

1.1.2 Proposed Solution

To address this problem, we conducted a search by empirical studies relating the process of migrating to microservice architecture. The goal was

to reveal a set of criteria cited in the process. The set of criteria were used to conduct an online survey that investigate the criteria perceived as useful by practitioners for identifying microservices of legacy systems. Practitioners were inquired about the usefulness of the criteria identified in technical reports, case studies, and other empirical studies. Moreover, we inquired information on how these criteria were measured and analyzed. The tools adopted to measure and analyze each criterion was also questioned.

As a result, we found that four criteria are considered useful while the others are moderately useful. The useful criteria are coupling, cohesion, reuse, requirements. The moderately useful criteria are database schema, visual representation, and network communication overhead. Our results point out that the criteria set in an overall perspective are adequate to identify microservices based on legacy code. In addition, we questioned the survey participants regarding other useful criteria, and the responses did not suggest additional criteria. These results support the completeness of the criteria set found during the review in mapping studies.

Moreover, the survey results indicate that academic solutions and tools do not satisfy the practitioner's needs to properly observe the relevant criteria. The used tools are insufficient to support criteria analysis which makes it hard for practitioners to analyze and reveal trade-offs. In this way, practitioners often consider simultaneously at least four useful criteria in the decision-making process to identifying microservices. However, academic solutions tend to only support the analysis one or two criteria. In summary, our survey provide valuable findings to produce realistic approaches. These approaches should better support the decision making process for identifying microservices. Chapter 2 presents the online survey.

1.2 Variability as a Criterion

Among the criteria, variability is a primordial and fundamental concept in software engineering (21, 22). Variability is the ability to derive different products from a common set of artifacts, usually incorporated in systems with flexible configuration demands (21).

1.2.1 Problem Statement

Configurable systems contain a common set of artifacts to generate software products (21, 22). These systems can be configured in several ways. For example, the use of preprocessor directives in the source code can be

used to enable and disable features (23, 24) to generate different system configurations. Alternatively, the use of feature models (21) can also be used to (de)select features in model-driven product lines. Variability has been supported in many well-known systems, including operating systems (e.g., Linux) and cryptography libraries (e.g., OpenSSL). In addition, there is software in enterprise environments that also contain variability (25, 26).

However, there is still little understanding of how variability is a criterion influential in the design of microservice architectures (3, 11, 12, 27). One should also try to understand how variability already present in the legacy system influences the selection of microservice candidates. Among the few studies that report this relationship, Tizzei *et al.* (3) reported a case involving the migration of a legacy system to a microservice-based product line. Their goal was to reduce the effort to fix bugs, facilitate the incorporation of new features to satisfy customer requirements as well as to improve scalability and configurability of computational resources. Tizzei *et al.* (3) concludes the resulting system achieved better reuse, configurability, and reduction of bug failures as compared to the original legacy system.

> ***2nd problem:*** *Lack of understanding about the influence of variability along the migration to a microservice architecture.*

1.2.2 Proposed Solution

Seven criteria were analyzed in the first sampling of the online survey presented in Chapter 2. We could not analyzed the variability criterion in depth. We had a small sample of practitioners with previous experience in taking part of processes involving the migration to microservice architectures with variability in mind.

Thus, we have increased the sample of the survey to allow the analysis of the variability criterion. Our goal was to understand to what extent this criterion is important in identifying microservices. Moreover, we invited the survey participants for an interview. In this interview, we inquired about the information prior and after the process of migrating to the microservice architecture.

In the new survey sample, the reasoning about variability along the migration process was considered useful by the experienced participants. Moreover, we found that half the participants already had some variability in some of their legacy systems, which were migrated to a microservice

architecture. Besides that, the mechanisms existing in the legacy system to implement variability more frequently reported by the survey participants are less dependent on the programming language extensions.

Regarding the interview results, three patterns used to implement variability in microservice architecture were cataloged. Furthermore, the evidence points out that microservice extraction may increase software customization after the process of migrating to a microservice architecture in customization-free legacy systems. Chapter 3 presents the variability criterion analyses based on the survey and interview studies.

1.3 A Microservice Extraction Approach with Many-objective Optimization

1.3.1 Problem Statement

As previously mentioned, developers commonly consider at least four criteria simultaneously while identifying and extracting the microservice candidates of legacy systems. These criteria widely vary from coupling and cohesion to requirements and network overhead. For instance, the consideration of network overhead before the microservice extraction may be more productive. Network overhead analysis consists of measuring the network overhead estimates given the separation of legacy code elements into two microservices, which communicate with each other. If network overhead is measured only after the refactoring into microservices is performed, the effort of changing already implemented decisions is high.

In the same way, coupling and cohesion are commonly adopted criteria by automated approaches to indicate microservice candidates (13, 14, 15, 16). The automated approaches maximize cohesion and coupling since a common goal of the process of migrating to a microservice architecture is improve modularity and maintainability, usually missing on legacy systems (6, 9, 10, 11). Another goal of these approaches also reduces the need for redesign of the extracted microservices in a short, medium, or long term.

Given the use of multiple criteria for identifying microservice candidates, practitioners have to deal with emerging trade-offs. For example, a software engineer may desire that extracted microservices have a low or limited network overhead. The same software engineer may consider cohesion as a very useful criterion; that is, they intend to maximize the cohesion of extracted microservices. However, maximizing cohesion can eventually increase the network overhead caused by the communication of two or more extracted microservices

to risky levels.

In addition, all the criteria, considered by a maintainer, require processing and measuring data from different sources of information. The cohesion criterion is often measured based on structural properties extracted from the code structure. In fact, static analysis is often applied to measure cohesion (16). Moreover, the size of the trafficked data between two candidate microservices may be used to quantify the potential impact on network overhead. Properties extracted from executions of the legacy system through dynamic analysis (29) are more adequate to support this quantification.

> ***3rd problem:*** *Simultaneously satisfy multiple interacting and possibly conflicting criteria is challenging along microservices extraction.*

1.3.2 Proposed Solution

In order to address the third problem, this dissertation research proposes an approach, named `toMicroservices`, to identify microservices. The proposed approach supports the use of five criteria observed in our empirical studies (Chapter 2) as moderately useful or useful for the process of migrating to a microservice architecture. The evaluation of these criteria was made by practitioners with experience in extracting microservices. Therefore, `toMicroservices` intends to be a more realistic approach because the adopted criteria are more aligned with criteria perceived as useful in practice. Moreover, our approach does not restrict the decisions to the optimization of a single or two conventional criteria, such as coupling and cohesion, which are not sufficient to reflect particularities of a microservice architecture.

`toMicroservices` makes use of static and dynamic analyses. That is, these analyses are made on legacy systems to enable to extract relevant information from the source code and the program executions (29). These different sources of information are adopted to measure the criteria, exploring the potential of each one. Finally, `toMicroservices` also deals with trade-offs like the one aforementioned between network overhead and cohesion by adopting search-based software engineering (SBSE) techniques. Basically, SBSE is the use of search-based optimization algorithms to address difficult problems of balancing competing constraints in software engineering (30, 31). The aforementioned survey results also indicate that experienced practitioners commonly consider at least four criteria as really need to be balanced along migrations to microservices. Thus, we adopted a many-objective optimization

algorithm suited to address the challenges of dealing with more than three objectives that represent conflicting criteria (32) in the `toMicroservices` approach.

A case study using a industrial legacy system was conducted to evaluate and improve `toMicroservices`. A subset of features in the legacy system was analyzed by `toMicroservices`. In a quantitative study, `toMicroservices` was compared with a baseline approach. The baseline approach used only two criteria found in automated approaches to identify microservice candidates (13, 14, 15, 16). The criteria are coupling and cohesion insofar as the most adopted ones in those existing automated approaches. Our results point out that `toMicroservices`, which supports five criteria – coupling, cohesion, reuse, overhead network, and feature modularization, generated better solutions than the baseline approach. The fact of optimizing coupling and cohesion does not imply that adequate solutions are automatically found also in terms of network overhead and feature modularization.

In our case study, microservice candidates generated by `toMicroservices` were individually presented to developers, and we questioned about their adoptability. In addition, we also observed to what extent developers were able to recognize features during the analysis of each microservice candidate. Several features and subfeatures were recognized by the developers, and half of the microservices would be adopted by them. In addition, lessons learned were obtained, such as the importance of visual representations and the user interaction with `toMicroservices` during the optimization to better explore the profile and knowledge of the user.

Furthermore, we conducted a focus group to evaluate a complete solution (i.e., a entire microservice architecture) generated by `toMicroservices`. Lessons learned from the case study were added in `toMicroservices` to generate new microservice architectures. After this improvement, we inquired two groups of developers about the architecture generated by `toMicroservices`. The results reveal a high level of adoptability of the architectural solution after merge operations were applied to microservices present in the recommended architecture.

This dissertation is a collection of four papers either accepted or under submission. In addition to Chapters 2 and 3 in which the criteria are investigated, Chapter 4 presents `toMicroservices` and the proposed measurements and optimizations to identify microservices. Chapter 5 shows the evaluation of the proposed approach in an industrial legacy system. The evaluation includes (i) an objective comparison between a baseline approach and `toMicroservices`, and (ii) a qualitative study in which developers evaluate

the microservice candidates. Moreover, Chapter 6 introduced an evaluation of the microservice architectures generated by `toMicroservices` with two groups of developers. The solutions generated by `toMicroservices` in this study also incorporate several improvements needed based on the lessons learned in the previous case study. Finally, Chapter 7 presents the conclusions and ideas for future work.

2
Analysis of the Criteria Adopted in Industry to Extract Microservices

Previous automated approaches to identify microservice candidates in legacy code are usually based on a small number of criteria, often one or two (13, 15, 14, 16, 33). The criteria most adopted are coupling and cohesion insofar as they are criteria very consolidated in software architecture to improve modularity. However, there is a lack of understanding of their adequacy, or the need to adopt other criteria by automated approaches to identify microservice candidates.

Thus, we conducted a search for empirical studies reporting the process of migrating to microservices architecture. In this task, we searched empirical studies elicited from two mapping studies (34, 35) about microservices architecture. After that, we have collected a set of eight criteria mentioned in these empirical studies to identify microservices based on legacy code. Moreover, we conducted an online survey with experienced practitioners in the identification and extraction of microservices. The sampling was made by inviting authors and subjects from the empirical studies found to answer the survey. We inquired about the usefulness of criteria, measurement ways, and tools used.

This chapter contains the paper: *"Analysis of the Criteria Adopted in Industry to Extract Microservices"* (67). This paper was published at *Joint 7th International Workshop on Conducting Empirical Studies in Industry (CESI) and 6th International Workshop on Software Engineering Research and Industrial Practice (SER&IP)*, a co-located workshop in the International Conference on Software Engineering (ICSE), hosted in Montrèal, Canada, in 2019. The paper presents seven criteria, including their perceived usefulness. The additional criterion (variability) and its results are addressed in Chapter 3. The results conclude that experienced practitioners usually adopt at least four criteria in the identification of microservices, suggesting that automated approaches to identify microservices based on legacy code (13, 14, 15, 16, 33) are oversimplifying the problem. Moreover, the survey participants indicated a limited set of ways and tools used to measure criteria.

Analysis of the Criteria Adopted in Industry to Extract Microservices

Luiz Carvalho*, Alessandro Garcia*, Wesley K. G. Assunção†, Rafael de Mello*, Maria Julia de Lima‡
*Pontifical Catholic University of Rio de Janeiro (PUC-Rio). Rio de Janeiro, Brazil
{lmcarvalho, afgarcia, rmaiani}@inf.puc-rio.br
†Federal University of Technology - Paraná (UTFPR). Toledo, Brazil
wesleyk@utfpr.edu.br
‡ Tecgraf Institute, Pontifical Catholic University of Rio de Janeiro (PUC-Rio). Rio de Janeiro, Brazil
mjulia@tecgraf.puc-rio.br

2.1 Introduction

Microservices have been successfully adopted for developing software systems in successful companies, such as Netflix (5) and Uber (4). Microservices are small and autonomous services that work together (1). A microservice is expected to have fine granularity (61). It is also expected to be autonomous: (i) it should consist of a service highly independent from others, and (ii) it should enable an independent choice of technologies, such as the programming language, the database, the communication protocol, and the like. A number of benefits are typically associated with a microservice-based architecture, such as reduced maintenance effort, increased availability, improved innovation, continuous delivery, DevOps enabler, scalability, and reduced time to market (6, 11).

Not rarely, microservices are not developed from scratch, but they result from the migration of existing systems (4, 5, 6, 7, 37). Given the claimed benefits of microservices, there is a growing interest of both industry and academia on streamlining the migration to a microservice architecture. However, the migration from a monolithic architecture to microservices is perceived as challenging by developers who have experienced it (1, 6, 11). In particular, the success of this migration is largely dependent on the use of appropriate criteria for extracting microservices from a code base.

In this work, microservice extraction comprehends the task of deciding whether and which parts of existing system functionality(ies) will be migrated to a microservice, by the selection, decomposition, and reuse of existing system parts. Academic approaches for supporting the microservice extraction have been proposed (1, 13, 14, 15). Most often, they tend to support the extraction of microservices with either one or two conventional criteria, namely coupling and cohesion (13, 15). A few others tend to solely use the information on database schema (1, 14) to support extraction decisions.

There is limited knowledge on whether academic techniques (e.g., (1, 13, 14, 15)) are aligned with how practitioners perform microservice extraction.

Existing studies in the industry often focus on reporting benefits and challenges on the migration to a microservice architecture (e.g., (3, 4, 5, 6, 7, 11, 37)). For instance, they point the selection and decomposition of microservices are often the most challenging activities (3, 6). However, it remains unclear what are the extraction criteria perceived as useful by practitioners along these tasks. Differently from many software engineering techniques, microservices emerged in the industry and not in the academia*. This fact possibly slowed the process for researchers to fully understand the peculiarities of this architecture style.

To address this gap, this paper presents an investigation of the criteria perceived as useful by practitioners for extracting microservices. To achieve this aim, we perform an exploratory study based on an online survey. The basis for our survey is a set of seven criteria mentioned in articles written by either researchers or practitioners. We want to understand the degree of usefulness (if any) of these criteria. We also questioned which tools are actually used by practitioners while analyzing the criteria for microservice extraction. The survey was distributed to industry specialists, which included developers, architects, project managers, and industry researchers. Our survey relies on the participation of 15 specialists from different countries located in North America, South America and Europe. They all have considerable practical experience on migrating existing systems to microservices architecture. These participants also have extensive experience in software development.

Overall, the survey results suggest academic solutions do not satisfy the industrial needs. For instance, practitioners often consider simultaneously at least four dominant criteria as well as their trade-offs to support their decisions. Academic solutions tend to only support the analysis a few criteria, which makes it hard for developers to reveal and analyze trade-offs with unsupported criteria. This is one of the reasons why practitioners reported they consider existing tooling support insufficient or even irrelevant for enabling well-informed decisions on microservice extraction. In summary, our survey provides a step further on a more realistic understanding of the relative usefulness of microservice extraction criteria in practice, from the point of view of industry specialists. Thus, we expect our survey results encourage researchers and practitioners to work more closely to design appropriate techniques and tooling support for microservice extraction.

The remainder of this paper is organized as follows. We present an illustrative example in Section 2.2, the set of investigated criteria in Section 2.3, our survey design in Section 2.4, results and discussions in Section 2.5, threats to validity in Section 2.6, limitations of related work in Section 2.7, and

*https://martinfowler.com/articles/microservices.html

conclusions in Section 2.8.

2.2 Microservice Extraction: An Illustrative Example

In spite of the potential benefits, the decomposition of an existing system into microservices is complex. We present a real case to illustrate how a particular criterion can affect the (mis)decision of extracting microservices from an existing system. This example comes from a system (from herein called "monolithic system") with limited modularization and no documentation. Maintainers of this system reported its notable limitations to incorporate new technologies. Some of the characteristics (and other ones) led to the partial migration of a monolithic structure to a microservice architecture.

The monolithic system was initially developed assuming a single process execution with communication made by local function calls. For the partial migration to a microservice architecture, engineers and developers decided to identify and extract some functionalities to microservices, thereby introducing network communications among them. Even though the extracted microservices fostered the decoupling of functionalities, a drawback was observed afterwards in the response time of some requests to the REST API. The analysis of the code extracted from the original monolithic system revealed some inner loops producing remote communication overhead. Developers decided to add a cache layer in the calls of this API, reducing roundtrips to dependencies and achieving the expected time response by the users.

This example illustrates how certain criteria affect the outcome of the extraction process. The decoupling among system functionalities was the dominant (maybe single) criterion considered along extraction. However, the result decomposition indirectly affected communication overhead, deteriorating system performance. The lack of reasoning upfront about other important criteria (communication overhead, in this case) and their trade-offs led to the microservices' unsuccess. A better understanding of these criteria is critical to support decision-making process during the microservice extractions.

2.3 On the Criteria for Microservice Extraction

To define a representative set of criteria, we searched in the technical literature the criteria explicitly reported for migrating existing systems to a microservice architecture. This review involved reading and analyzing papers cited by two mapping studies on microservices (34, 35). As a result, we compiled an initial set of seven criteria, including (i) the ones used in academic

techniques, and (ii) the ones mentioned in technical literature, albeit not necessarily supported by existing academic automated techniques. Our goal was to understand their usefulness from the perspective of practitioners with experience on microservices migration processes. These criteria, included in our survey with specialists (Section 2.4), are described below.

Coupling. Coupling comprehends the manner and degree of interdependence between software modules or functionalities (36). Newman mentions that microservices should be decoupled from each other as much as possible (1). Different coupling metrics are used by existing techniques (e.g., (13, 15)) for microservice extraction. Coupling is the most cited criterion in the literature (35, 34). In addition, case studies select coupling as the dominant (sometimes, single) criterion along the microservices migration process (e.g., (8)).

Cohesion. Cohesion is the manner and degree in which inner elements – data and behaviors – of a single module (or functionality) are related to each other (36). A microservice is expected to modularize a functionality (e.g., a domain entity) with highly cohesive inner elements. Therefore, cohesion is an important aspect for microservice architectures. Existing academic techniques indeed consider cohesion to recommend the extraction of microservices from existing systems (15). We also found case studies of microservice-driven migrations reporting cohesion as a relevant criterion (8). However, cohesion is used as a secondary criterion in some academic solutions; coupling is considered the main criterion.

Overhead in the communication of extracted microservices. Overhead is related to the amount of time a system would spend on performing actions not directly addressing the user needs (36). Communication overhead concerns the negative impact of extracting microservices on time spent along future microservice communication, which was originally performed locally (e.g., via function calls) in the "monolithic system". The derived microservices will need to keep communicating by protocols such as HTTP and AMQP, therefore, this communication may result in some penalties, possibly prohibitive ones, to the system performance (1, 38, 91). That is the reason why communication overhead might need to be considered beforehand, through microservice selection activities. It should be noted that coupling and overhead in the communication of extracted microservices are different criteria. Coupling is the degree between different elements while overhead is time consumed in network communication.

Potential of reuse. Reuse is the use of already developed assets in the solution of different problems (36). For example, microservices, software

product lines, frameworks, and libraries aim to support different levels of reuse. Each extracted microservice could be reused by two or more of its callers. The cost of microservice extraction can be justified by actual reuse in the short or long term. Higher the potential of microservice reuse, the better. That is why practitioners may consider this criterion when selecting and decomposing microservices. There is another way of considering reuse along the migration process: only built microservices that can reuse parts of the existing code base. Previous studies carried out migrations to microservices in order to maximize code reuse (6, 37).

Data dependencies in the database schema. Data entities have dependencies in a database schema (39). Moreover, the database structure could be considered the greatest source of dependencies for a wide range of systems. Migrating to the microservices architecture may lead to splitting the database into smaller databases. Database decomposition can achieve data coupling reduction and increase microservices' independence. Such an independence make it possible the selection of different database technology for each microservice. In addition, within the database schema it is possible to find valuable information about the domain entities (e.g, their relations, constraints, and the like). The analysis of the database schema is mentioned as a way of possibly starting the microservice extraction process (1, 14).

Impact on software requirements. Requirement is the software capability needed by a user to solve a problem or achieve an objective (36). In microservices migration process, practitioners may also consider the possible impact of candidate microservices on certain non-functional (or functional) requirements. For example, non-functional requirements as security may influence which parts of existing systems could be extracted. Functional requirements may provide a better understanding of the domain concepts.

High-level criteria from visual models. Some decomposition criteria can be more easily analyzed from visual models. The migration of an existing system to a microservice architecture can be seen as a combination of reverse engineering and re-engineering. Reverse engineering (40) may use graphical resources to build representations of the existing systems in higher levels of abstraction. Thus, visual representations, such as UML, could be useful to derive architectural on detailed designs of the system. These representations may be used to support analyses of high-level criteria that impact on the selection and/or decomposition of microservices. For instance, two functionalities with a strong architectural dependency may be a criterion used to give up on their modularization as two microservices. Moreover, behavioral models of the system may be used to decide on the possible boundaries of a

candidate microservice. Our literature review revealed a previous study that consider use cases to define some criteria for microservices extraction (91).

2.4 Survey Design

This section presents the design of our survey based on the potentially useful criteria used along microservice extraction.

2.4.1 Goal, Population and Sample

We conducted an exploratory survey (41) with the goal of understanding what criteria are considered useful. Specialists on microservice migration participated in the survey. To achieve ours goal, we asked the subjects to evaluate the perceived relevance of the criteria presented in Section 2.3. Moreover, we inquired the subjects which techniques and tools they use when applying each criterion. The target of our survey is composed of specialists with a background in migrating existing systems to microservices. We developed a search plan (42), aiming at identifying a representative sample of this target. We selected a source of sampling composed of two mapping studies on microservices (34, 35). Then, we searched in this source by works addressing the migration of existing systems to microservice architecture. In the following step, we performed a snowballing search(43) in the pieces of work that cited the referred mapping studies. For performing this search, we used Google Scholar[†].

The survey was sent to authors of scientific papers. Some of these authors actively participated in the migration process to microservices architecture. They have played key roles, such as developers or architectures. In this way, they represented the target to our survey. Moreover, we are also invited subjects (i.e., participants) of the empirical studies (reported in the analyzed papers) to take part of the survey when the paper authors agreed is distributing the survey to the corresponding study's subjects. After execution, our search plan resulted in the recruitment of 70 subjects. The survey was executed during January 2019. From the 70 subjects recruited, 15 participants responded, resulting in a participation rate of 21.23%.

2.4.2 Instrumentation

In this study, we applied an online questionnaire for gathering subjects' data. The questionnaire items are divided into three groups. The first group is

[†]scholar.google.com

composed of questions for characterizing the survey participants. This group of questions is presented in Table 2.1. Among others, we asked the subjects' academic background, development experience, and position in the current job. More specifically, we asked about their background in migrating existing systems to microservices.

Table 2.1: Survey questions about characterization of respondents

Question	**Type**
Do you want to receive future information about the survey results?	Choice: Yes, No
Name	Open Question
Email	Open Question
What is your academic background?	Choice: HS, Grad, Master, PhD, Other
How long have you been developing software? (years)	Positive Number
What is your position in your current job?	Open Question
How many migration processes to a microservice architecture did you only participate in the past (e.g., in architecture decisions, programming tasks, and others)?	Positive Number
How many migration processes to a microservice architecture are you currently participating (e.g., in architecture decisions, programming tasks, and others)?	Positive Number
How many migration processes to a microservice architecture did you only observe in the past (e.g., examination, analysis, review, consulting, and others)?	Positive Number
How many migration processes to a microservice architecture are you currently observing (e.g., examination, analysis, review, consulting, and others)?	Positive Number
What is (are) the domain(s) of the systems that underwent migration(s) to a microservice architecture with your involvement?	Open Question

The second group of questions (Table 2.2) aims at gathering the perceived utility of criteria presented in Section 2.3. We used a five-point Likert scale associated with the following levels of usefulness, from the lowest to the highest: 1) Not useful - it was not useful at all, 2) Slightly useful - it was only useful to a small degree, 3) Moderately useful - it was not considerably usefulness, 4) Useful - it was considerably usefulness, 5) Very useful - it was indispensable.

To avoid misinterpretations, we provided in the questionnaire a formal

Table 2.2: Specific questions to all the criteria of second group

Criterion	**Question**	**Type**	**Answer**
Coupling	How did you measure coupling?	Checkbox	1. The structure of the source code 2. Program execution 3. Others
Cohesion	How did you measure cohesion?	Checkbox	1. The structure of the source code 2. Program execution 3. Others
Communication Overhead	How did you measure overhead in future network communication between extracted microservices?	Checkbox	1. The structure of the source code 2. Program execution 3. Others
Reuse Potential	How did you measure reuse?	Checkbox	1. The syntax of the code 2. Code duplication 3. Others
Requirements Impact	What are the requirements considered (if any)?	Open, Likert Scale	Textual
Visual Models	What was (were) visual representation(s) used?	Open	Textual

definition of each criterion evaluated (see Section 2.3). We also asked the participants to justify their answers. We also want to understand what is used to analyze or measure each criterion. For this purpose, we asked the tools and techniques they use in each criterion evaluated. We also asked whether developers consider these tools sufficient to support the migration process, as shown in Table 2.3.

Table 2.3: Common questions to all the criteria of second group

Question	**Type**
Feel free to justify the usefulness (or not) of the criteria in the decision-making process	Open Question
What tool(s) was (were) used for measuring or analyzing the criteria?	Open Question

For each criterion investigated, we also applied particular questions, based on its characteristics. Coupling, cohesion and overhead are usually measured through static and dynamic analysis (44). In this way, we asked whether subjects use one of these approaches or both. For reuse, we asked whether this criterion is measured through the syntax of the source code and by the incidence of duplicated code. Regarding software requirements, we questioned whether non-functional and functional requirements are used. Regarding visual models, we inquired about the type of artefacts commonly used to represent the system in higher levels of abstraction.

The third group of questions are about the sufficiency of the tools to support the extraction of microservices. We also asked about cases where the extraction may have failed. These questions are all open answer and are presented in Table 2.4.

Table 2.4: Questions in the third group

Question
Do you believe that existing tools are sufficient to support the migration process?
Was there any case that the extraction of the microservice was not successful (led to a high overhead communication between microservices, data inconsistency, or some other problems)?

2.5 Results and Discussions

Next we present the results of the application of our survey to a group of 15 specialists. The experience of the respondents can be evidenced by the

Table 2.5: On the Criteria Usefulness: Participant Responses

Criterion	Responses															Median
	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15	
Coupling	3	3	3	4	4	4	4	5	4	4	5	5	4	5	4	4
Cohesion	5	4	5	4	4	5	4	5	2	4	5	5	5	3	3	4
Communication Overhead	2	2	1	4	3	4	2	3	4	3	5	1	4	3	2	3
Reuse Potential	4	4	4	3	5	2	4	5	1	2	1	4	2	2	5	4
Database Schema	4	4	1	3	2	4	5	3	2	1	5	5	1	4	3	3
Requirements Impact	4	4	4	3	3	5	5	3	4	1	4	5	5	4	4	4
Visual Models	3	2	5	3	2	2	4	2	1	5	3	5	3	5	3	3

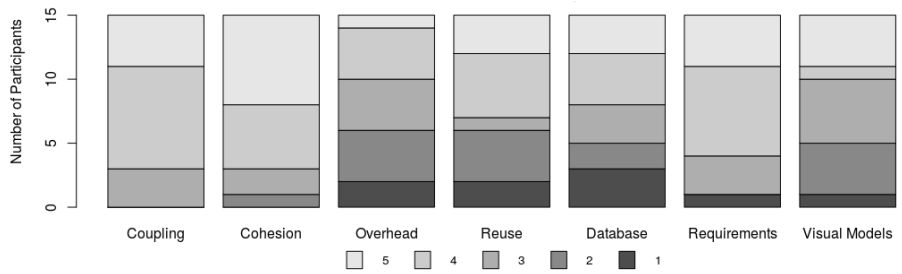


Figure 2.1: Criterion Usefulness: Distribution of Responses

fact they all have both previous and ongoing experience with microservices migration processes. For instance, the vast majority of the respondents had already concluded their participation in at least two migration processes (a mean of 2.7 processes). Moreover, most of them are actively participating in at least one project (a mean of 1.6) where the microservices extraction is currently underway.

The respondents have extensive experience in software development. The time in years the participants have been developing software is considerably high: (i) the mean is 17.4 years and median is 18 years, and (ii) with a maximum of 35 years and a minimum of 3 years. Our sample of respondents is also well diversified in terms of the roles they play in their projects: developers (33.3%), architects or engineers (20%), team leaders (20%), and industry researchers (26.7%).

Table 2.5 and Fig. 4.1 presents the answers of the participants with respect to the usefulness of each extraction criterion. As explained in Section IV.B, we used a five-point Likert scale to enable the respondents to classify each criterion from not useful at all (1) to very useful (5). The analysis of the medians (last column) of each criterion (rows) in Table 2.5 reveals which criteria are useful (median equal to 4) and moderately useful (median equal to 3). The columns show the answers of each participant with their IDs varying from #1 to #15. Fig. 4.1 provides a different perspective on the responses by showing which criteria concentrates more positive, neutral, or negative scores. In the following, we discuss first the usefulness of each criterion. We start discussing those criteria considered as the most useful by the practitioners.

After that, we explore other dimensions in the obtained responses.

Cohesion as the most useful criterion, but proper tool support is missing. Cohesion was considered the most useful criterion: it has the highest concentration of "very useful" answers, represented by the light grey color in Fig. 4.1. This criterion presented the median equal to 4 (Table 2.5). This was a surprising result as we expected that coupling would outperform cohesion. Coupling is usually prioritized in the conventional decomposition of software modules, while cohesion tends to be used to confirm the incidence of modularity flaws. One participant justified that *"A highly cohesive functionality should make a natural unit that is very easy to spot and derive the microservice boundaries".* Cohesion was described by the majority of the participants as more useful than coupling to support decisions on microservice extraction. Multiple respondents mentioned cohesion is the key criterion to identify domain entities by domain driven design, a typical method used in industry to identify microservice candidates.

Despite the relevance of cohesion, surprisingly, only 33.3% respondents reported they use some tool, such as IntelliJ[‡], to apply this criterion. These results suggest that cohesion is predominantly analyzed manually. One of the reasons it that some practitioners mentioned they need to understand the cohesion from a dynamic analysis perspective. Actually, more than 30% of the participants reported using the system execution to analyze cohesion. One of the respondents even mentioned he tries to infer cohesion while debugging the code with IntelliJ. Some participants use static cohesion measurement, but most of them do it by convenience as most of the existing tools support only static cohesion measurement.

Coupling is also an useful criterion, but again dynamic analysis support is missing. Coupling also achieved a median of 4 (Table 2.5). It was the second criterion in terms of concentrating "very useful" and "useful" answers (Fig. 4.1). Few participants (26.6%) again reported using some tool to support coupling analysis, such as IntelliJ and SonarQube[§]. Even though many participants reported to use the program structure for coupling analysis, almost 40% of the respondents mentioned other strategies. In particular, 33.3% mentioned they need to infer actual coupling of functionalities from a dynamic analysis perspective while considering a functionality to be extracted to a microservice. They need to understand which of the static dependencies can have indeed an impact on possible performance bottlenecks if a functionality is modularized as a microservice. However, the use of dynamic coupling

[‡]https://www.jetbrains.com/idea/
[§]https://www.sonarqube.org/

analysis is always challenging as it can generate a large output, and clearly requires proper tool support (89). One participant also mentioned he usually analyzes the change history of functionalities to reveal their co-changes that imply logical coupling (not capture by static dependencies).

Impact on requirements: functional and non-functional requirements are equally important. Impact on requirements was the third most relevant criterion. The distribution of answers for this criterion in the Likert scale was quite similar to the one observed for coupling. Most of the participants perceived this criterion as "useful" or "very useful", but there was one participant that gave the lowest score for this criterion. He mentioned the requirements specification of the existing system did not predict at all relevant requirements (either functional or non-functional) for the microservices being considered in the migration process.

We expected that non-functional requirements would appear much more frequently than functional requirements. Functional requirements were by far the most quoted type of requirement considered along the microservice extraction: 77% of the valid responses considered them as "very useful". Functional requirements are used to infer the domain concepts which can be isolated in microservices. Performance was perceived as the most useful (median of 4) non-functional requirement. Respondents were clearly concerned with performance deterioration. A total of 86% of participants rely on explicit description of requirements in the migration process.

Reuse opportunities are considered on microservice extraction, albeit not unanimously. Reuse also achieved a median of 4 in the responses. One participant mentioned the practice of reuse *"is indeed a key driving factor for the migration to microservices architecture; it is important to promote the reuse of the extracted microservice by other systems"*. However, the relevance of reuse was not unanimous. There was a distribution of different answers across the five points in the Likert scale. While 8 participants gave the highest scores (either 5 or 4), 6 partipants gave the lowest scores (either 1 or 2) and 1 participant gave a borderline score (3).

Some participants argue there are other more effective ways to achieve reuse. One participant quoted that: *"Though arguably a good reason to create new microservices, I haven't witnessed any project in which its main motivation was reuse. The most common solution for achieving reuse was the use of software libraries instead."*. Others argue that the existing monolithic code was so intermingled that opportunity for reuse could not even be considered. Among the participants, 40% reported they use tools, such as IDEs and its plugins, for supporting reuse analysis. Code duplication (26.6%), code syntax

(46.6%), and other factors were adopted in the decision-making process to find reuse opportunities while deciding on extracting microservices. However, 60% of the respondents stated that reuse analysis is predominately manual.

Communication overhead is a secondary criterion as participants often postpone its analysis. Microservices may need to communicate with other microservices or subsystems. Then, the overhead on microservice communication was expected to be a primary criterion while deciding for microservice extraction. However, it was overall perceived as moderately useful with a median equal to 3. Fig. 4.1 shows it has the highest distribution of answers in the Likert scale. Some comments state that the other benefits of extracting microservices outweight the possible drawback of increased communication overhead.

Only five of the respondents considered communication overhead is "very useful" or "useful". These respondents were the same who classified performance as an important non-functional requirement. We observed that many participants postpone the analysis of communication overhead, *i.e.*, they tend to postpone it for after the extraction of the microservice(s). However, some participants mentioned they try to predict if the communication overhead will be prohibitive, and/or which kind of microservice decomposition will minimize the overhead. Some participants reported the use of different tools to support overhead analysis, such as Intellij IDEA, JMeter, and debugging tools. Runtime analysis via logs and microservices monitoring tools were also mentioned.

Surprisingly, dependencies on database schemas is frequently a neglected criterion. Opinions on the usefulness of this criterion also varied as much as communication overhead. This was a surprising result. Proponents of microservices often state the each microservice should have its own tiny database. Therefore, we expected that data dependencies in database schema would be often used in the extraction of microservices. However, our respondents seem to decompose microservices using different strategies with respect to the database.

Half of the respondents seem to follow recommendations of proponents of microservices architectures. In this case, most of them stated they use tools associated with relational database management systems to reason about the database schema decomposition. These participants also mentioned database schema is useful for identifying independent domain concepts.

High-level criteria from visual representations used for determining microservice boundaries. Only five (one third) of the respondents considered high-level criteria, derived from visual representations, as "very use-

ful" or "useful". One third considered their usefulness as moderate, while the other one third qualified them as "not useful" or "slightly useful". The most common representations are class diagram, dataflow models, and use cases. Regardless the type of representation, the most common use of it was to support the definition and analysis of criteria to determine the boundaries of the microservices being considered for extraction.

All practitioners often use at least four dominant criteria simultaneously. Most of the academic solutions investigated in our literature review (Section III) suggest some dominant criteria for extracting microservices. Most of them: (i) support the analysis of one or two criteria only (1, 13, 14, 15), and (ii) suggest to use coupling and cohesion as the dominant criteria. In fact, the vast majority of the proposed techniques for supporting microservice extracting rely on coupling or on a combination of coupling and cohesion only (13, 15). Moreover, they only support the analysis of coupling and cohesion of functionalities already structured as separated modules. However, our survey revealed that existing systems are often legacy systems. Thus, some functionalities being considered to be extracted as microservices tend to be scattered and tangled with each other in the implementations of existing modules. They are not modularized in the existing system, which hampers the use of academic techniques for supporting microservice extraction.

To make the matters worse: most of the respondents (9 out of 15) reported at least four criteria were simultaneously used to support decisions on the extraction of each microservice. This result can be observed in Table 2.5: these respondents are the ones who pointed out at least four criteria "very useful" or "useful". Table 2.5 shows that four of these respondents qualified more than four criteria "very useful" or "useful". The 6 remaining respondents used 3 dominant criteria. However, the results of our suggest that academic solutions are currently oversimplifying the problem of microservice extraction. Practitioners need to consider to maximize the satisfaction of these multiple criteria and deal with various emerging trade-offs.

Most specialists believe existing tools are not sufficient. We questioned about whether existing tools are sufficient to support the decision-making process of extracting microservices. A tally of 53,3% of the participants answered no. Only 26,7% said that tools are sufficient, while 20% stated that do not believe in tools to support the migration process. This was the quote of one of the participants: *"The tools themselves cannot provide a path clear enough to allow easy decisions towards extraction or not."* In the opposite direction, other participant said: *"Tools could help to provide more confidences on the effectiveness of these decisions"*. We also inquired whether any extraction of

certain microservices was unsuccessful due to some criterion neglected in the process. Surprisingly, 40% of the respondents answered yes. They stated that the lack of synthesized information about the relevant criteria was one of the reasons to overlook a certain criterion in spite of their ultimate importance.

2.6 Threats to Validity

The survey questionnaire is composed of a large number of questions, which may discourage the subjects' participation. In this sense, before running the survey, we first invited other researchers to review the questionnaire. Based on their feedback, we then conducted a pilot with real subjects. In this pilot, we observed an acceptable participation rate for the context of surveys in the field.

The small sample of respondents in the survey is a threat to validity. However, most of survey participants declared significant experience on migrating systems to microservice. This achievement was possible due to the fact we followed a formal recruitment strategy for identifying highly qualified subjects to participate in the survey (see Section 2.4). This sampling process resulted in the recruitment of 70 individuals involved with the research and practice of implementing microservices. A participation rate of 21.5% is quite high for online surveys of this kind, which usually ranges from 3% to 10%.

2.7 Related Work

Along the migration of an existing system to a microservice architecture, some techniques have been proposed to extract source code information and recommend microservices. Mazlami *et al.* (13) present a strategy to decompose systems into microservices considering three coupling metrics. These metrics are used to weight a graph where nodes represent system classes. Edges are associated with weights provided by the coupling metrics. The components provided by a clustering algorithm are used to recommend microservices to the developer.

Other previous study considers the production and use of diagrams to understand the legacy systems for suggesting microservices. The idea is to separate and group Enterprise Java Beans (EJB) according to the type of data it handles (14). In other words, this is a strategy centered on a model-based criterion. Moreover, Newman (1) presents recommendations of microservice extractions based on certain criteria associated with the database schema. Jin *et al.* (15) propose a functionality-oriented microservice extraction method by

monitoring system execution traces and clustering them. In spite of promoting the use of dynamic execution of the system, it is still limited because it only considered coupling and cohesion. As observed in our survey, practitioners need to explore various combinations of criteria in order to make successful microservice extractions in industry cases. Unfortunately, these combinations are not either supported or used in existing techniques and methodologies found in the literature review.

Francesco *et al.* (12) interviewed and applied a questionnaire to developers. Their goal was to understand the performed activities, and the challenges faced during the migration. They reported what are the existing system artifacts (e.g., source code and documents) the respondents used to support the migration. The main reported challenges were: (i) the high level of coupling, (ii) the difficulty of identifying the service boundaries, and (iii) the microservices decomposition. However, they did not specifically analyzed the usefulness of the extraction criteria addressed in our survey.

Taibi *et al.* (11) also conducted a survey with the objective of elucidating motivations that led to an microservice migration process and what were the expected returns. The main motivations were the improvement of maintainability, scalability, and delegation of team responsibilities. In addition, difficulties were cited in this process, such as decoupling from the monolithic system, followed by migration, and splitting of data in legacy databases.

2.8 Conclusions

In this paper, we reported a survey performed with specialists to assess what criteria are useful to extract microservices during the migration to a microservice architecture. We questioned how the participants measure and analyze (with or without tools) each criterion. We also inquired them if: (i) existing tooling support is sufficient, and (ii) unsuccessful microservice extractions occurred. To identify potential respondents, we searched for studies in two mapping studies about microservices migration. We also made a snowballing search starting from the mappings.

Even though there were some variations across participants' answers, the results revealed that participants consider criteria related to modularity – *i.e.*, coupling, cohesion, and reuse – and requirements impact as relevant. The other three criteria are commonly seen as moderate, albeit considered "useful" or "very useful" by certain respondents. In other words, there is no criterion that can be claimed to be not useful. Their degree of usefulness clearly varies from a context to another. Moreover, effective decisions on microservices extraction

are clearly far from being simplistic. Our results suggest four criteria are usually considered simultaneously to support decisions. This finding questions the practicality of academic solutions, which generally consider only one or two criteria.

In fact, existing techniques and tools were seen as insufficient. Mistaken decisions on microservice extractions are mentioned to be often related to the lack of synthesized information about the relevant criteria and their trade-offs. All these findings of our survey suggest researchers and practitioners to work more closely. Otherwise, it is unlikely we will able to design appropriate techniques and tooling support for microservice extractions. There are many unaddressed questions: how to automatically identify possible trade-offs among the seven criteria based on the legacy code? how to better combine static and dynamic analysis to reveal such trade-offs? how to anticipate information on potential communication overheads before the microservice extractions or in a step-wise manner?

As future work, we plan to perform interviews and additional analysis, such as the identification of existing patterns, in the answers. For example: how the degree of developers' experience related to their criteria prioritization? what are other new criteria (beyond the seven ones) spontaneously mentioned in their responses? Moreover, so far we have 15 answers. However, the survey is still open. We are expecting to receive more answers. These additional answers may help us to further confirm or reveal new insights.

3
Extraction of Configurable and Reusable Microservices from Legacy Systems: an Exploratory Study

Software engineering pioneers (45), including Parnas (22) and Brooks (47), claim the need for the management of software families in the least 50 years. Since then, variability has become a key concept in software engineering (21). However, the relationship between variability and a microservice architectural style is little understood or explored from a practical perspective, and insofar, to the best of our knowledge, there is a only a concrete case reported in the literature that systematically explore this relationship (3).

Chapter 2 evaluated the usefulness of seven criteria, excluding variability. This happened because few practitioners with previous experience in identify microservice candidates taking variability into account answered the survey. Once the microservice architecture is an emergent topic on software engineering, we conducted further searches by empirical studies relating the process of migrating to a microservice architecture. In this work, we adopted a more recent and exhaustive mapping study (27) regarding microservices to perform this expansion. In this order, we obtained 26 participants and half of them are experienced with variability. Moreover, we invited survey participants to an interview. The interview allows a deeper understanding of the process of migrating a legacy system to a microservice architecture. We focus on topics not inquired in the survey of the previous chapter, such as adopted technologies and patterns to implement variability.

This chapter contains the paper: *"Extraction of configurable and reusable microservices from legacy systems: an exploratory study"*. This paper was published at *the 23rd International Systems and Software Product Line Conference (SPLC)*, hosted in Paris, France, 2019. The main finding of the paper is that variability is a key criterion to identify microservice candidates, when present in the legacy code. In addition, half of the survey's participants indicated they had previous experience to identify microservice candidates in the presence of variability. *Copy and paste* through the use of a version control system is the most adopted mechanism to implement variability in the source code before the process of migrating to a microservice architecture. Finally, three patterns

to implement variability in microservice architecture were identified during the interviews. In case the reader has read Chapter 2, you may consider to skip Section 3.2.2, which provides a succinct description of microservice architecture's concepts.

Extraction of Configurable and Reusable Microservices from Legacy Systems: An Exploratory Study

Luiz Carvalho
Pontifical Catholic University of Rio de Janeiro
Rio de Janeiro, Rio de Janeiro, Brazil
lmcarvalho@inf.puc-rio.br

Alessandro Garcia
Pontifical Catholic University of Rio de Janeiro
Rio de Janeiro, Rio de Janeiro, Brazil
afgarcia@inf.puc-rio.br

Wesley K. G. Assunção
Federal University of Technology - Paraná
Toledo, Paraná, Brazil
wesleyk@utfpr.edu.br

Rodrigo Bonifácio
University of Brasília
Brasília, DF, Brazil
rbonifacio@unb.br

Leonardo P. Tizzei
IBM Research
São Paulo, São Paulo, Brazil
ltizzei@br.ibm.com

Thelma Elita Colanzi
State University of Maringá
Maringá, Paraná, Brazil
thelma@din.uem.br

3.1 Introduction

Legacy systems, commonly found in industry, represents a long term massive investment. Despite of their business importance, legacy systems are difficult to extend and include innovation (9). In addition, these systems usually have a monolithic architecture, with components tangled in a single unit, strongly connected and interdependent (6, 11). Currently, many companies have been adopting microservice architectures to modernize monolithic legacy systems (3, 4, 5, 6, 7, 8). Microservices are small and autonomous services that work together (1). The benefits of adopting microservices are: reduced effort for maintenance and evolution, increased availability of services, ease of innovation, continuous delivery, ease of DevOps incorporation, facilitated scalability of parts with more demand, etc (6, 11).

Service-based architectures must meet some attributes to satisfactorily fulfill their purpose (48): (i) scalability, enabling optimization in the use of hardware resources to meet high demand services; (ii) multi-tenant efficiency, offering transparency in the use of shared services, since the service should optimize the sharing of resources and also isolate the behavior of different tenant; and (iii) variability, where a single code base provides common and variable functionalities to all tenants. Microservices are known to work well with the first two attributes (3), however, the literature is scarce in relation to the use of variability. A possible explanation for the lack of discussion may be the fact that this technology was initially conceived in the industry and only in the last years have had attention of academia*.

Configurable systems are widely developed in the industry to meet demands related to different types of hardware, different platforms, serving diverse customers or market segments, etc (49). In this context, we should understand how functionalities should be modularized and customized as microservices to fulfill customer needs.

*https://martinfowler.com/articles/microservices.html

The goal of this paper is to investigate the importance of variability related to the process of extracting microservices from monolithic legacy systems in industry. To this end, we analyzed 26 survey responses and seven interviews with practitioners. Among the 26 participants, 13 dealt with variability during the extraction and they stated that variability was a key criterion for structuring the microservices. Moreover, we observed an increase of requests for customization after microservices extraction from legacy systems that initially were not configurable. In this way, initial evidence points out that the microservices extraction can increase software customization.

This paper is organized as follows. Section 3.2 presents the background. Section 3.3 details our study. Results and discussion are in Section 3.4. Sections 3.5 and 3.6 address threats to validity and related work, respectively. Section 3.7 concludes the paper.

3.2 Background

This section presents the main concepts involved in this paper, such as customization, variability, and microservices.

3.2.1 Customization and Variability

Service-oriented architectures must allow tenant-specific configuration and customization. The tenant-specific adaptations may affect all layers of the application, from functional requirements to database schema. Furthermore, tenants do not only have different requirements regarding functional properties, they can also require different non-functional requirements of service properties (50).

To achieve such customization, industry must deal with software variability, which is the ability of a system or an asset to be adapted for using in a particular context (51). Variability can be viewed as consisting of two dimensions (52). The space dimension is concerned with the use of software in multiple contexts. The time dimension is concerned with the ability of software to support evolution and changing requirements in its various contexts.

3.2.2 Microservices

A microservice is expected to have fine granularity (61) and be autonomous: (i) it should consist of a service highly independent from others,

and (ii) it should enable an independent choice of technologies. For communication, microservices architecture commonly adopted lightweight protocols.

Not rarely, microservices are not developed from scratch, but they result from the migration of existing systems (1, 4, 6). However, the migration from an existing system to microservices is perceived as challenging by developers who have experienced it (6, 11). Studies indicate that developers use the source code of the existing system in migration to microservice architecture (12). In this way, approaches were proposed using criteria observed in source code (13, 15), generally using coupling and cohesion criteria. Others approaches recommend observing the database schema (1, 14). However, there is a lack of the comprehension of variability usefulness in the migration process when the existing system is a configurable system or software product line.

3.3 Exploratory Study Design

The goal of our exploratory study is to better understand the migration process to microservice architecture and how software variability is useful in the decision-making process. In what follows, we show in Subsection 3.3.1 the research questions and their motivations. Subsection 3.3.2 presents the two phases (survey and interview), population, and sampling. In addition, Subsection 3.3.3 shows the instruments used in the survey and interview.

3.3.1 Research Questions

Based on the aforementioned goal, we intend to answer the following research questions:

RQ1 How important is variability in the migration process to the microservice architecture?

- **RQ1.1** Did the original systems contain some sort of customization, prior to the migration?
- **RQ1.2** What mechanisms were used to implement variability prior to the migration process to microservice architecture?
- **RQ1.3** Do developers consider variability as useful criteria in migration process to microservice architecture?

RQ2 How is variability present after migration to the microservice architecture?

- **RQ2.1** Does microservices extraction increase software customization?

– **RQ2.2** What mechanisms are used to software customization in the microservice architecture?

In summary, for RQ1 we want to investigate the relevance of variability and its mechanisms to support and guide the migration of configurable systems to a microservice architecture. RQ2 aims to understand how variability is present after migration to microservice architecture, since there is a lack of understanding about how configurable systems coexist with the microservice architecture.

3.3.2 Study Phases, Population and Sample

Our exploratory study is composed of two phases: (i) a survey with specialists, and (ii) interviews with specialists who answered the survey. The specialists target of our survey are practitioners and industrial researchers with a background in migrating existing systems to microservices. We used the results of three mapping studies (27, 34, 35) to identify an initial target population. In addition, we performed a snowballing search (43) in studies that cited the referred mapping studies. For performing this search, we used Google Scholar[†].

Our strategy to contact specialists was to invite the authors of the papers found in the three mapping studies to participate in the survey. We requested that they submit the survey to the subjects of the empirical studies. After execution, our search plan resulted in the recruitment of 90 subjects. The survey was executed from January to March 2019. From the subjects recruited, we collected answers from 26 participants, resulting in a participation rate of 29%. Of which 13 participants also answered our questions related to software variability. To carry out deep analysis, we invited the survey participants to an interview. We have conducted seven interviews.

3.3.3 Instrumentation

Survey. We organized the survey questionnaire into two groups. The first group is composed of questions for characterizing the participants. We asked the academic background, development experience, and position in the current job. We also inquired about their background in migrating existing systems to microservices.

[†]https://scholar.google.com/

The second group of questions has the objective of perceiving the usefulness of variability, in the extraction of microservices. For this group of questions, we used a five-point Likert scale associated with the levels of usefulness for variability, from the least (1) to the most (5). We also asked the participants to justify their answers. To avoid misinterpretations, we provided in the questionnaire a definition of variability, which is the ability to derive different products from a common set of artifacts (21).

Interview. The goal of this phase was to understand the post characteristics of the migration process. The interviews were conducted using video conference tools. Each interview was conducted by an interviewer and a scribe that took notes. For the execution of the interview, we chose a semi-structured approach. That is, questions that answers can be quantified (structured) and also other questions that suggest the theme (unstructured) (92).

Regarding the questions, the participants were first questioned with open and general questions about more high-level themes, in an unstructured way. For example, explaining about the existing system that was/is being migrated to microservice architecture. After that, we inquired them with quantitative questions. For example, the number of microservices extracted. This approach was chosen by observing that survey participants had an experience mean of 15.77 years in software development. Moreover, participants have been involved in the migration process to the microservice architecture. In the questions we investigated the themes: (i) existing systems, (ii) migration process, (iii) variability, and (iv) tools.

3.4 Results and Analysis

In this section, we present the results and their analysis. The first phase of our study (survey) brings a better comprehension about usefulness of variability. In a previous study, we observed how relevant are seven criteria during migration to microservices architecture. However, the previous study did not perform an analysis of variability (67). This happened because of the low number of participants with expertise on migrating some existing system with variability to a microservice architecture. In this present work, the greater number of survey answers allowed us made analyses about variability in the first phase. Moreover, the second phase of study (interview) provided initial results about the post-migration process to microservice architecture. Among them, the request for increased customization of the microservices in order to deal with different customers or groups, in which, before the migration process was not made.

3.4.1 Participants Characterization

Survey respondents experience may be evidenced by previous and ongoing activities with the migration process to microservice architecture. The vast majority of the respondents had already concluded their participation in at least two migration process (a mean of 2.5 and median of 2) and most of them are actively participating in at least one project (a mean of 1.3 and median of 1) where the microservices extraction is currently underway. Besides that, participants observed at least one process (a mean of 1.5 and median of 1) and they are observing at least one migration (a mean of 1.5 and median of 1). The respondents have extensive experience in software development. The time in years the participants have been developing software is considerably high: (i) the mean is 15.8 years and median is 15 years, and (ii) with a maximum of 35 years and a minimum of 3 years. Our sample of respondents is diversified in terms of the roles that they play in their employment: developers (42%), architects or engineers (23%), team leaders (19%), and industry researchers (19%).

3.4.2 Variability

From the 26 participants, 50% have never considered variability. For these respondents, the domain they work does not require the ability of deriving different products from a common set of artifacts (21). Their intention was to specifically migrate a single software system to a microservice architecture. However, the other half of the participants have answered that their existing systems, which were the target of microservices extraction, had variability. Thus, these respondents were able to answer about the usefulness of variability along the migration to a microservice architecture.

Among the participants with previous experience to answer on the usefulness of this criterion (see Figure 3.1): (i) 31% considered it as not relevant by assigning values of 1 or 2 in the Likert scale, and (ii) 69% of the participants considered it as useful or very useful. There is no response with moderate usefulness. Among those that considered variability important, one participant said: *"It was useful to identify the variabilities and features of each product"*. Other participant said that *"There are 3 systems that will be affected by the migration and we need to ensure the differences between them and how these differences can be handled by the migration process"*. In this way, most participants consider variability useful or very useful in the migration process to the microservices architecture. However, previous approaches to

microservices extraction do not make use of variability criterion.

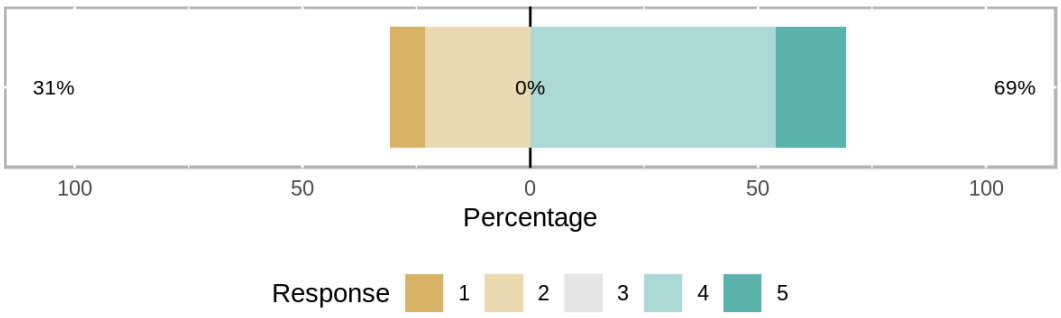


Figure 3.1: The usefulness of variability for the migration process to microservices architecture

Regarding the variability implementation approaches, the most common ones are those shown in Table 3.1. In the last column of the table, we present the proportion in which they were cited.

Table 3.1: Variability Implementation Approaches

Approach	**Percentage**
Version control	77%
Design patterns	62%
Framework	62%
Components	54%
Parameters	46%
Build systems	38%
Aspect-oriented programming	31%
Feature-oriented programming	15%
Pre-processor	8%

Version control was the most common variability implementation approach used in existing systems, that is, in systems before the migration to microservice architecture. One might consider this is the most simplistic approach as it leads to the management of copy and paste of different products. For example, without the need of extending the programming language being adopted as in feature oriented programming. Approaches more sophisticated are less common ones, like the use of aspect-oriented programming and feature-oriented programming. The mean of variability implementation approaches used is 4.5. All the respondents mentioned more than one approach to implement variability.

The survey results enable us to answer **RQ1** as follows:

RQ1.1: *50% of participants answered that at least one of their existing systems (from which microservices were extracted) had variability.*
RQ1.2: *Version control is the most used mechanism (77%) to implement variability in those existing systems.*
RQ1.3: *69% of participants considered variability during the migration process as useful or very useful*

3.4.3 Microservice Customization

During the interview phase, we inquired the participants whether the extraction allowed that some microservices could be used in different contexts. To our surprise, four (57%) of the interviewed participants said that some customization was required after this migration process to microservice architecture. The customization was needed to attend requests of different groups inside the same enterprise or customers of the software system. The four cases previously answered in the survey that the systems prior to migration to microservices had no variability. This seems to be an indication that the process of migration to microservices leverages the customization of the single system by increasing modularity and more interfaces available to its customers. Common ways to manage these new requirements for dealing with customization of the software systems are reported in the following paragraphs.

Regarding the other three participants (43%) of the interview, two of them reported in both the survey and the interview that: (i) the system has variability before the migration to microservice architecture, (ii) and the system continued to have variability after the migration. In one of these cases, the variability in the migrated system relied on design patterns and parameters in the interface to identify a tenant (50). In the other case, the migration process is still underway. However, it is expected that some microservices will not exist or have their behavior drastically modified in different usage contexts. In the third case which there was no variability before or after the migration, the software system was completely developed and after migrated for a single customer and it will not be used by different groups of the enterprise.

RQ2.1: *Initial evidence points out that the migration process to a microservices architecture increase the customizations of the system.*

During the interviews, four participants reported that the migration to microservices was not made with the goal of turning the system more easily

customizable. However, after the migration, customization requests emerged by different clients or groups within the same enterprise.

Among the four cases that reported this growth for customization post-migration, three of them described which approaches were adopted to implement variability in microservice architecture. The other participant report that the need for customization emerged after the microservices extraction; however, he did not observe or participate in the process of implementing or managing the customizations.

We describe below the three implementation approaches employed to deal with the customization required after the microservices extraction: *Copy and Paste*, *Big Interface*, and *Filtering to the Different Platforms*.

Copy and Paste was adopted after the migration process to microservice architecture by one of the participants. That happened when customizations in the interface of extracted microservices began to be requested by different groups within the same enterprise. In this case, it was decided to copy the microservices implementation and perform the customization. Thus, the copied microservice coexists with the original (inclusive at runtime). Changes are performed in both microservices by the original developers when maintenance is required in mandatory features. It was also reported that code of test cases is submitted to the same copy and paste process and it is managed in the same way. This approach is often used in the industry as it promptly promotes reuse of verified functionalities, without requiring high upfront investment and achieving short-term benefits, as reported by Dubinsky et al. (53).

Big Interface was also an adopted solution. In this reported case, certain consumers of the microservice APIs needed additional information, or the same information in different formats. To fulfill this demand, the developers just included all required information in already existing APIs. However, it should be noted that this solution raises well-known issues related to the big interface problem (54).

Filtering to the Different Platforms was an approach that uses an intermediary microservice between other microservices and consumers, similar to a gateway microservice. This intermediary microservice has the sole purpose of filtering the outputs for specific platforms. The case described by the participant had three different platforms, namely mobile, web, and desktop, that need customized information from the same microservice. For example, when a consumer uses a mobile platform, the user interface returns more targeted and lean responses to these devices, that is, it was a filter of the original outputs from target microservice interface with its customization for the different platforms. This pattern seems to be a form of resolution to

mitigate problems from the previously presented approach.

In addition to the approaches described to deal with the customization that emerged after the migration process to the microservice architecture, the interviewed participants were also inquired about integration between extracted microservices and the legacy system. It was possible to observe that, in five cases, where the migration process was completed by the interview date, four of them extracted the microservices in an incremental process, that is, some microservices were extracted and integrated with the legacy system and their behavior was observed. In this integration, it was common to use feature toggles or similar mechanisms to switch between the legacy system and the integration between extracted microservices. A feature toggle is an age-old and simple concept (55, 56). It basically is a variable used in a conditional statement to guard code blocks, with the aim of either enabling or disabling the feature code in those blocks for testing or release. (56). This was used in a production environment for the problems reported or observed by the team monitoring this transition. This approach allowed a fast return to a stable version (legacy system). In general, some microservice with database access (usually a key-value database) was used as a microservice filter to choose the switch between using only legacy system and the extracted microservices integrated with the legacy system. One participant of the interview reported using this switch strategy between both versions for a small portion of their users. In survey participants, 43% reported steps related to the integration between the legacy system and the extracted microservices. Besides that, the effort related in most cases were medium or high.

> ***RQ2.2:*** *Three approaches employed to implement the customization required after the microservices extraction are (i) Copy and Paste, (ii) Big Interface and (iii) Filtering to the Different Platforms. Microservices are usually extracted in an incremental process. Moreover, feature toggles (or similar mechanisms) are used to switch between the legacy system and the extracted microservices integrated with the legacy system.*

3.5 Threats to Validity

The first threat is related to the number of questions in the survey, that might discourage the subjects' participation. To alleviate this threat we made a subject matter expert reviews (41) with two experts. After considering their

feedback, we conducted a pilot study with four real subjects. In this pilot, we observed an acceptable participation rate.

Another threat concerns the number of valid respondents in the survey. This sampling process resulted in the recruitment of 90 individuals. A participation rate of 29% is considered good for online surveys of this kind, which usually ranges from 3% to 10%. In addition, most of the survey participants declared to have significant experience in migrating systems to microservice architecture (see Section 3.4.1). This quality of subjects was reached due to the fact we followed a formal recruitment strategy (see Section 3.3.2).

About the interview, a threat is the process of collecting data during the interviews. To deal with this threat we asked the participants permission to record the interviews for future analysis and transcriptions. In addition, all interviews were performed by an interviewer, which made the questions, and the scribe, responsible for taking notes, analysis of the respondent behavior, and ask additional questions.

3.6 Related Work

In the migrating of existing system to a microservice architecture, some techniques extract source code information and properties to recommend microservices. Mazlami et al (13) present a strategy to decompose systems into microservices considering three coupling metrics. These metrics are used to weight a graph where nodes represent system classes. Edges are associated with weights provided by the coupling metrics. The components provided by a clustering algorithm are used to recommend microservices to the developer.

Other previous study considers the production and use of diagrams to understand the legacy systems for suggesting microservices. The idea is to separate and group Enterprise Java Beans (EJB) according to the type of data it handles (14). In other words, this is a strategy centered on a model-based criterion. Moreover, Newman (1) presents recommendations of microservice extractions based on certain criteria associated with the database schema. Jin et al (15) propose a functionality-oriented microservice extraction method by monitoring system execution traces and clustering them. In spite of promoting the use of dynamic execution of the system, it is still limited because it only considered coupling and cohesion. As observed in our survey, practitioners need to explore various combinations of criteria in order to make successful microservice extractions in industry cases. Unfortunately, these combinations are not either supported or used in existing techniques and methodologies found in the literature review.

Francesco et al. (12) interviewed and applied a questionnaire to developers. Their goal was to understand the performed activities, and the challenges faced during the migration. They reported what are the existing system artifacts (e.g., source code and documents) the respondents used to support the migration. The main reported challenges were: (i) the high level of coupling, (ii) the difficulty of identifying the service boundaries, and (iii) the microservices decomposition. However, they did not specifically analyzed the usefulness of the extraction criteria addressed in our survey.

Taibi et al. (11) also conducted a survey with the objective of elucidating motivations that led to the microservices migration process and what were the expected returns. The main motivations were the improvement of maintainability, scalability, and delegation of team responsibilities. In addition, difficulties were cited in this process, such as decoupling from the monolithic system, followed by migration, and splitting of data in legacy databases.

3.7 Conclusions

This paper presented an exploratory study composed of two phases. In the first phase, a survey was applied to specialists experienced in the migration process to microservice architecture. In this survey, we also inquired the utility of variability and mechanisms used to implement them. In the second phase, we performed an interview with survey participants.

We could ask about the requests for customization after the microservice extraction, and the mechanisms used by the participants to deal with post-migration customization. We also observed initial evidence that microservice extraction can increase software customization, mainly because some users made demand for customization after microservices were extracted. The more common approaches to implement customization in extracted microservices are copy and paste, big interface, and filtering to the different platforms. Moreover, feature toggles (or similar mechanisms) are commonly used to switch/integrated the legacy system and the extracted microservices.

Our study is still ongoing, so for future work we expect to have more responses to our survey and more participant interviews. In this way, we will be able to draw more analysis about useful criteria for extracting configurable microservices from monolithic legacy systems.

4
The `toMicroservices` Approach

The process of migrating to a microservice architecture is the examination and alteration of a legacy system to be (fully or partially) adherent to a microservice architecture. That is, the process of migrating are structured into two phases: (i) microservice identification, i.e., the selection of microservices for which possibly reusable code elements are spotted in the legacy system, and (ii) microservice extraction wherein the actual creation of each microservice is performed using partially or fully the code elements spotted in the previous phase.

Survey results indicated that practitioners commonly consider four criteria as useful or very useful in the decision making to identify microservices of the legacy systems. However, automated approaches to identify microservices (13, 14, 15, 16) commonly make use of one or two criteria, usually including the coupling criterion. Moreover, the practitioners often reported the available tools are limited to identify microservice candidates (Chapters 2 and 3). Nevertheless, survey results suggest that automated approaches simplify the process of migrating to a microservice architecture.

This chapter proposes `toMicroservices`, an automated approach to identify microservice candidates by relying on the existing code of a legacy system. `toMicroservices` is built based on: (i) the outcomes of the empirical study already performed in this dissertation (Chapter 2), and (ii) a comparative analysis of existing approaches (Section 4.1), which enable us to figure out what are their limitations to be addressed by our approach. The goal is to create a practical approach to identify microservice candidates from a legacy system, in which the approach is more aligned with the developers' needs.

`toMicroservices` is presented in detail, including artifacts generated to and required from the user. Moreover, the processes of generating these artifacts are introduced. The processes includes a many-objective evolutionary algorithm to optimize five criteria measured based on static and dynamic properties extracted from the legacy system. In addition, a domain-specific language is proposed to indicate boundaries of features, which are, in principle, considered as possible candidates to be fully or partially modularized as microservices.

4.1 Existing Approaches for Microservice Identification

Before presenting our approach, we present and compare the automated approaches to identify microservices, which were found in three mapping studies (27, 34, 35). We also followed a backward snowballing through the found papers (describing the automated approaches) to identify other approaches not discussed in those mapping studies. Finally, we updated it through a rapid review on approaches to the process of migrating to a microservice architecture (57). Several manual and automated approaches have been proposed to identify microservice candidates (27, 34, 35, 57). Such an identification consists of establishing parts of the source code of the legacy system that are related to each microservice candidate. The approaches adopt different criteria along the decision making of microservice identification.

Coupling is the most commonly adopted criterion by the automated approaches. Coupling is the manner and degree of interdependence between software modules (36). Similarly to coupling, cohesion is also commonly used by the automated approaches to quantify the desirable interdependence between internal member of a module. This criterion is often defined as the manner and degree to which the tasks performed by a single software module are related to one another (36).

Moreover, database schema is another criterion being considered. A schema provides the (description of the data types and their relationships in a database (46)). Database schema is a criterion considered in a manual approach (57) and in one of the existing automated approaches (14). Besides, the criterion of functional requirement, which specifies a functionality that a system shall perform (36), is often used to derive cases of use of the legacy system (15, 16). Other criteria found in empirical studies are presented in Chapter 2. The existing approaches for microservice identification are introduced, detailed and compared in what follows.

Newman (1) suggests a manual approach adopting database schema as especially important and useful criterion for extracting microservices. The adoption of the database schema criterion is justified in the sense that legacy systems are too much coupled to a particular database and its schema with databases. Schema analysis indicated by Newman (1) is predominantly manual, that is, there is not an algorithm or automated tool to identify microservice candidates, and the schema analysis is focused on the relationship between database entities. Several other manual approaches have been proposed after the appearance of Newman's approach (1). Their description can be found in the rapid review performed by Mella *et al.*(57). However, the fully manual

identification of microservice candidate in legacy systems is time-consuming and risky (6, 12, 28).

In this sense, automated approaches have been proposed to reduce the effort related to identify microservice candidate and increase the quality of these identifications. In the context of using the database schema to identify microservices, Escobar *et al.*(14) created an automated approach and evaluated it in a case study with a legacy system that makes use of EJB (Enterprise JavaBeans). In summary, the proposed approach relates classes representing entities in the database schema (entity beans in EJB) with other classes responsible for the business rules (session beans in EJB). In summary, the approach relates each class representing an entity in the database (entity beans in EJB and called class-entity in this work) with other classes responsible for the business rules. Each relationship that started with an entity-class (class representing a database entity) is called by a cluster in Escobar *et al.*(14) work. After that, the intersection between the same classes in different clusters is used to measure coupling. Consequently, a clustering algorithm minimizes the measured coupling between the microservice candidates identified. The coupling between classes is examined through the source code of the legacy system.

Regarding the coupling usage in automated approaches, Mazlami *et al.*(13) also adopted a clustering algorithm based on coupling criteria to indicate microservices using the source code or change history from the legacy system. The approach uses coupling measurement to weight edges that indicate relationships between vertices (representing classes). After that, the Kruskal's algorithm is used to cluster the graph vertices according to the number of desired microservices given by the user of the approach. Among the ways to measure coupling, three different forms were defined: (i) contributor coupling, (ii) logical coupling, and (iii) semantic coupling. Contributor coupling strategy defined the weight with the cardinality of the intersection of the developer's sets that contributed to the classes related. Regarding the logical coupling strategy, it is calculated based on the number of times that classes were changed together in the same commit. Both contributor coupling and logical coupling take into account change history to measure coupling. However, Mazlami *et al.*(13) did not observe how the change history is affected by the quality of the practices employed along the changes (e.g, the number of commits or pattern adopted in merge requests). In order to measure the semantic coupling, Mazlami *et al.*(13) uses the frequency of a term in classes to measure coupling.

Eski *et al.*(33) identify microservice candidates combining two ways to measure coupling. The first way considers the coupling between classes. The

second way of computing coupling take into account that change frequently based on software repository (e.g, git). This approach is similar to Mazlami *et al.*(13) that also measure coupling by static and change history analysis. However, Eski *et al.*(33) made use of the Fast Community Algorithm (58) to identify microservice candidates in the graph that represents the legacy system. In this graph, each edge is weighted by the two mentioned coupling measures, and the Fast Community Algorithm minimizes both.

Different to the aforementioned studies, Jin *et al.*(15) made an approach based on the dynamic analysis of the legacy system, that is, properties are extracted from the system execution (29). Functional requirements are previously identified and cataloged by users of the approach to execute the legacy system. Thereafter, the trace executions are analyzed to remove traces the are covered by others. The resulting traces generate clusters. After that, clusters similarity is used to indicate microservices considering the coupling criterion. Furthermore, Jin *et al.*(15) proposed a mutation operator that it is analogous to the move method refactoring proposed by Fowler (71). In other words, the operator moves different classes inside a microservice candidate to another microservice candidate. Thus, the genetic algorithm adopts this mutation operator to generate offspring.

The approach proposed in (15) was extended by Jin *et al.*(16) to use the intra-connectivity of classes in a microservice candidate to identify the internal relationship in a cohesion sense, and the inter-connectivity evaluates the relationship between different traces of the system execution, which provides indicators about the coupling criterion. In this sense, Jin *et al.*(16) also exercise features implemented in the legacy system. However, feature modularization is not directly measured and optimized. In addition, textual terms also weight the relationship between classes. Thus, cohesion and coupling are measured each one by to different measurements from a dynamic and static perspective. Even as Mazlami *et al.* (13) which measure semantic coupling, there is little evidence of how these textual terms are suitable for legacy systems since the systems analyzed in their comparisons are not legacy systems. Finally, a genetic algorithm is used to optimize the measured coupling and cohesion. Furthermore, the genetic algorithm also adopts the mutation operator aforementioned of the Jin *et al.*(15) work.

The characteristics of automated approaches are summarized in Table 4.1. The most common criterion is coupling. Table 4.1 also shows the types of analyses that are adopted. That is, from where the information to measure the criterion mentioned are extracted, ranging from static and dynamic analysis to change history analysis. Besides, granularity represents the unit(s) of the

Table 4.1: Characteristics of the automated approaches for microservice identification

Paper	Criteria	Analysis Type	Granularity	Algorithm
Escobar *et al.* (14)	Coupling	Static	Class, Class-Entity	Not informed
Mazlami *et al.* (13)	Coupling	Static, Change History	Class	Kruskal
Eski *et al.* (33)	Coupling	Static, Change History	Class	Fast Community (58)
Jin *et al.* (15)	Coupling	Dynamic	Class	Genetic Algorithm
Jin *et al.* (16)	Coupling, Cohesion	Static, Dynamic	Class, Method	Genetic Algorithm (NSGA-II (77))

program structure used in the analyses of the legacy system (i.e., class-entity, class and method). Finally, Table 4.1 also presents the algorithm used by each approach to optimize the measured criteria.

As far as evaluations of the automated approaches are concerned, Jin *et al.*(16) performed a study of their approach that represents the widest evaluation as compared to reported assessments of other existing approaches. Jin *et al.*(16) compare their approach with two other automated approaches to identify microservice candidates (13, 15) and one automated approach commonly used to modularize legacy systems (59). In order to perform the evaluation, a quality model is confronted against the results, that is, the microservice candidates resulted from the four compared approaches. This quality model adopts metrics of independence of functionality, modularity, and evolvability. In this context, Jin *et al.*(16) suggest that their proposed approach achieves better results than the three compared approaches.

Case studies are powerful instruments for refining approaches (60) as they make it possible an in-depth, contextual, long-term assessment of the approach's elements. In this way, Eski *et al.*(33) present a case study wherein the practitioners identified microservice candidates. These microservice candidates were compared with the ones generated by the automated approach proposed by them (33). The method for comparing both sets of microservice candidates is based on the similarity of these sets. The minimum number of operations to transform a set of microservice candidates into other set is computed and adopted in the case study. Lastly, the similarity of the microservice candidate proposed by the automated approach is compared with the microservice

candidate identified by the practitioners. The results present a high similarity about the set of microservices identified by the automated approach and practitioners. However, there is a lack of understanding on the limitations of the automated approach, such as the neglected optimization of other key criteria, and how the metrics can be more adherent to the practitioners' specific purposes.

Table 4.1 summarizes the automated approaches and shows that a very few criteria are adopted by them. In addition, the studies conducted with automated approaches do not consider the state of practice by ignoring other useful criteria for the process of identifying microservices. These and other limitations of the existing approaches described here also motivated a number of decisions made on the design our approach, to be described in the next sections. We present first an overview of our approach in the next section.

4.2 An Overview of `toMicroservices`

Our automated approach `toMicroservices` uses five criteria, namely coupling, cohesion, reuse, feature modularization, and network overhead. These criteria are adopted insofar as they were often mentioned in our empirical studies (Chapter 2). In fact, the five criteria were classified by practitioners as useful or moderately useful.

`toMicroservices` uses different data sources to measure the five criteria. Static and dynamic analyses are adopted to quantify these criteria. Also, `toMicroservices` uses code elements as entry points to associate features with execution traces. Each entry point defines one of the possible entries to the feature boundaries. Each entry point is a method that is part of the feature boundaries. That is, the boundaries of a feature delimit the feature scope; i.e, they consist of the set of program methods that directly interact with methods of other features. The seeds of each feature are regular expressions and are used to labeled methods with the features.

Figure 4.2 shows an overview of `toMicroservices`, including required inputs and generated outputs under the perspective of the `toMicroservices` user. To execute `toMicroservice` in a legacy system, the user must provide the following inputs: (i) the legacy source code, including indicators of code elements that will not be parsed, (ii) program inputs and the corresponding executions of the legacy systems, (iii) regular expressions indicating entry points to each feature, and (iv) the number of desired microservice candidates. The inputs are used to automatically measure and optimize the criteria aforementioned. Finally, the `toMicroservices` user receives a Pareto set, where each

solution contains microservice candidates (including the measurements of each criterion), and the code in method level related to each microservice candidate.

Figure 4.1: `toMicroservices` overview under user perspective

Our survey shows that functional requirements was the most cited category of useful requirements (more than non-functional requirements) by the survey participants. Thus, `toMicroservices` adopts a measurement strategy for a criterion called "feature modularization". In summary, `toMicroservices` associates each feature and its corresponding source code from the legacy system as provided by the user. Furthermore, our approach tries to minimize the amount of features per each microservice candidate, thereby avoiding feature tangling in the resulting microservice implementation.

The second most cited and useful criterion was network overhead. Thus, `toMicroservices` deals with this performance issue by estimating the network overhead. Performance may become a scalability problem if network overhead is not reduced in microservice candidates. `toMicroservices` also address other non-functional requirements: (i) reusability by quantifying reuse in terms of how many times a microservices is (re)used by other microservice candidates, and (ii) maintainability by promoting developing of features (see above) into cohesive microservices.

Another non-functional requirements can be included in a future version of `toMicroservices` as a criterion. For example, adding security constraints to `toMicroservices` to avoid extracting methods or classes in different microservices, which would otherwise facilitating attacks through network communication. In this way, addressing the security non-functional requirement.

`toMicroservices` still does not use explicitly visual models and variability because they can be present in legacy systems in many diverse ways. Regarding variability, participants reported a wide range of implementation mechanisms adopted in the legacy system. The most mentioned mechanisms reported by the participants include version control, design patterns, and frame-

work. Thus, the analysis of legacy systems with variability would need to contemplate the detection of those mechanisms and their different forms. In the same sense, the visual models' answers in the survey (Section 2) cited several types of UML diagrams. Thus, variability and visual models were kept beyond the current scope of `toMicroservices`.

`toMicroservices` measures and optimizes the five aforementioned criteria by default: coupling, cohesion, reuse, feature modularization, and network overhead. To perform this optimization, `toMicroservices` is a search-based approach, in which a genetic algorithm is adopted. The genetic algorithm uses search operators that will manipulate the graph representation of the legacy systems. During this manipulation, the measured criteria are adopted as fitness functions to the comparison between the generated solutions. Finally, `toMicroservices` presents the best alternative solutions (Figure 4.2) found to the user of the approach in a Pareto set.

`toMicroservices` approach is presented in Figure 4.2 with its complete process. The first step (automated graph generator) requires: (i) the legacy code, (ii) its executions, and (iii) regular expressions associated with features as shown in Section 4.5; each regular expression describes the entry points of a feature. The aforementioned inputs are provided by the `toMicroservices` user as aforementioned. The artifact generated by the first step is the graph representation of the legacy system; this representation is introduced in Section 4.4.

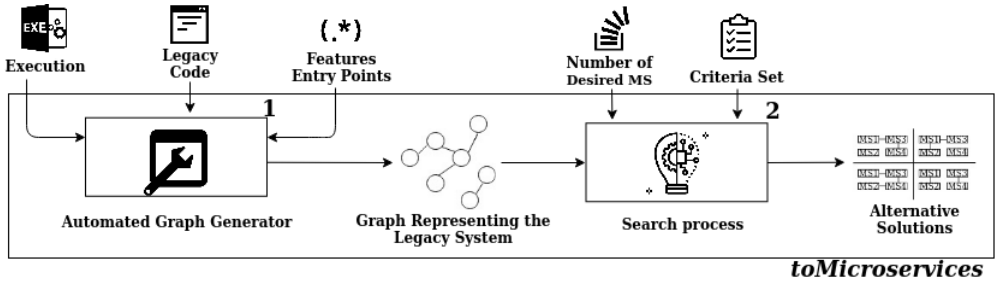


Figure 4.2: `toMicroservices` process

The second step is the search process (Section 4.3 introduces search-based software engineering), i.e., a set of steps to search for alternative solutions with microservice candidates. In this process, a many-objective genetic algorithm, NSGA-III, was adopted to optimize the criteria (Section 4.6). The input to this process are the: (i) graph representation of the legacy code, (ii) number of desired microservices provided by the user, and (iii) criteria set presented in Section 4.7. The result is a Pareto set, in other words, a set of microservice architectures. Each alternative microservice architecture is also structured as a graph (Section 4.4).

4.3 Search-Based Software Engineering

The term Search-Based Software Engineering (SBSE) was coined by Harman and Jones (30) to describe an approach to software engineering in which search-based optimization algorithms are used to address problems in the field (31). Such search-based techniques could provide solutions to the difficult problems of balancing competing constraints (30). Therefore, the optimization algorithms may suggest ways of finding acceptable optimal or near-optimal solutions. Search-based techniques are usually applied in situations where perfect solutions are either theoretically impossible or practically infeasible (30). Successful application of SBSE could be found in the resolution of several problems. Among them, test case generation (62, 88), software product lines architecture recommendations (63), test order selection (64), and microservice identification (16).

Harman and Jones (30) also argue that a software engineering problem needs to be reformulated as a search-based problem. In this way, it is necessary to define:

- the representation of the problem which is amenable to symbolic manipulation.
- fitness functions.
- manipulation operators.

The representation is how the solution to the software engineering problem is stored or theoretically described. The representation is of fundamental importance insofar as it accurately describes the desired solution and allows manipulation of these solutions by operators. The quality of the proposed solution is evaluated by a fitness function. Different solutions are generated by search operators that either modify previous solutions or compare them in order to obtain better solutions, while the fitness function verifies how good the solutions are.

Several problems in software engineering are many-objective or multi-objective (31), that is, the problems contain more than one fitness function to be optimized. In the many-objective, more than three fitness function are adopted. These problems usually have not only one solution. Because the objectives to be optimized can be in conflict leading to a compromise among the objectives. In such a situation, the result is a set of optimal solutions. This set is known as Pareto-optimal solutions or Pareto set (31). Among the Pareto set elements, each element cannot be claimed better than others directly by the SBSE approach. Usually, multi-objective search-based algorithms return

the Pareto set to the user of the approach. Thus, the user needs to select the best possible solution from the Pareto set according to her priority.

A common metaheuristic in SBSE is the genetic algorithms among the search-based optimization algorithms, including the handling of multi-objective problems (31). The genetic algorithm uses concepts of population and recombination (65). In general, the genetic algorithms start with a randomly chosen population. In each generation, members of the population (a representation of the problem solution in software engineering) are manipulated by recombination or crossing over their elements (chromosomes). These elements are joined to the population. After that, part of the offspring and the original population is mutated. At the end of each generation, the selection process is used to generate a new population. The selection is guided by the fitness functions. Finally, a new generation is started or the search process for the best individuals is completed.

4.4 Graph Representation

The extracted information from the legacy system is represented in a directed graph where each vertex represents a legacy system method. Each edge contains information about method calls from either a static or dynamic perspective. In summary, the graph that represents the legacy system is G = (V, E) in which:

$$\begin{aligned} V \text{ is a set of vertices} \\ E \subset \{v_i \wedge v_j \mid (v_i, v_j) \in V^2 \wedge v_i \neq v_j\} \text{ a set of edges} \end{aligned} \tag{4-1}$$

$$sc, dc, dt \colon E \to \mathbb{N} \tag{4-2}$$

Each edge represents a relationship between methods. Moreover, contains information about data traffic, syntactic calls, and dynamic calls occurrence. These pieces of information are respectively represented by dt, sc, and dc functions, defined in Equation 4-2. Thus, from each edge, there is a function for each of the three aforementioned information.

If $e = (v_i, v_j)$ where v_i and v_j represent methods from the legacy system, and $v_i \in V$ and $v_j \in V$, then $sc(e)$ provides the amount of syntactic calls to another v_j method present in the body of the v_i method. In the same way, $dc(e)$ provides the amount of calls from method v_i to method v_j at runtime. Moreover, $dt(e)$ is the data traffic between the method v_i and the method v_j. The heuristic to measure data traffic is discussed in network overhead

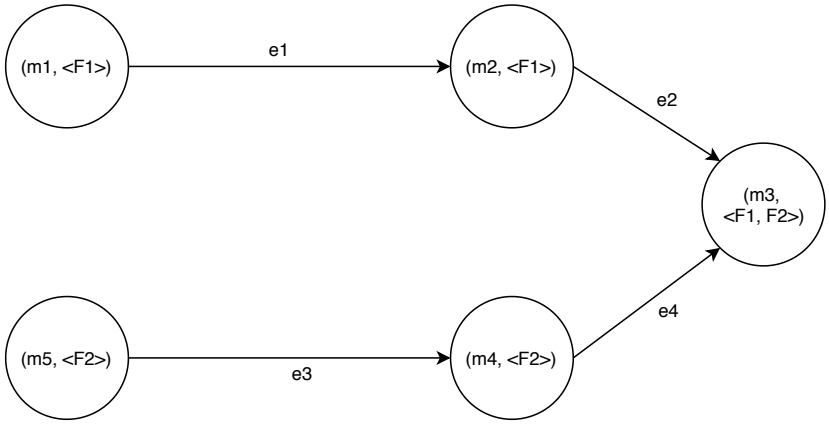


Figure 4.3: Graph representation of a legacy system

measurement (Section 4.7). In addition, the edge e exists only when there is at least a syntactic call in the v_i method body to the method v_j.

$$fc\colon V \rightarrow F \tag{4-3}$$

The information captured about features is used to label vertices (methods in a legacy system) with features responsible for implementing them. Thus, each vertex is associated with features, as defined in Equation 4-3, where F is a set of features.

$$md\colon V \rightarrow D \tag{4-4}$$

Besides that, the metadata contains data about the vertices name and additional information as the class name where a method is contained in the legacy system. The function in Equation 4-4 associates each vertex with a D set of metadata.

An example of the representation adopted by `toMicroservices`, based on the definition of the Equation 4-1, is shown in Figure 5.2. The graph presents five vertices, each representing a method in the legacy system under analysis. The vertices m1, m2, and m3 were labeled with the feature F1 in a first execution. Afterward, the next executions labeled a feature F2 in the methods m3, m4, and m5.

A possible output of `toMicroservices` is shown in Figure 5.3, where (i) m1, m2, and m3 are part of the MS_1 microservice, and (ii) m4 and m5 are included in MS_2 microservice. In this example case, the communication between m3 and m4 in the solution representation realizes the communication between microservices MS_1 to MS_2. The values of md, sc, dc, and dt functions were omitted in the possible cases of inputs and outputs shown in Figure 5.2 and Figure 5.3.

`toMicroservices` provides to the user of approach: (i) a Pareto set

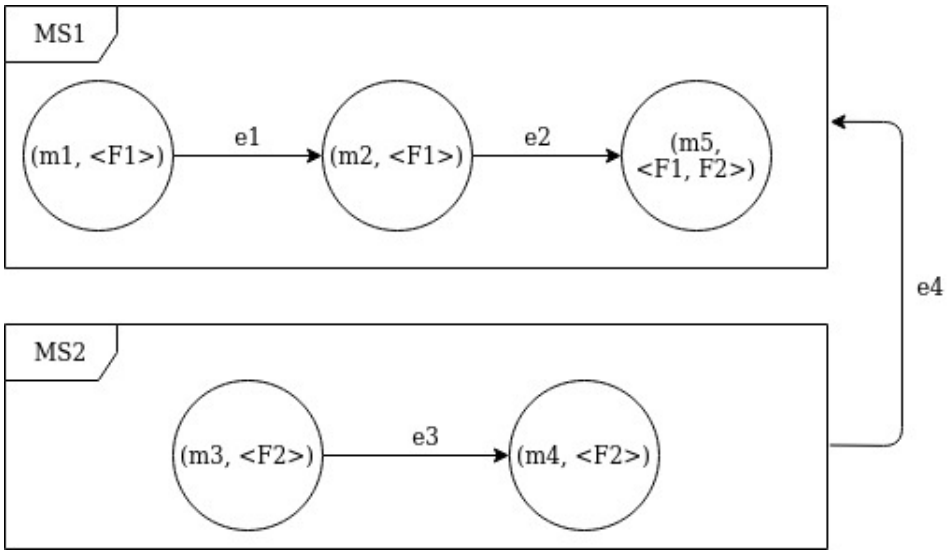


Figure 4.4: A simplified example of a solution generated by `toMicroservices`

where each element is a graph of microservice candidates, and (ii) the source code in the legacy system associated with each candidate. The microservice candidates generated by `toMicroservices` is a clustering of the graph defined in Equation 4-1. The result is another graph where the vertices are microservice candidates and the edges are the communication between methods by the particular choices of the microservice candidates.

The graph of the microservice candidates (vertices) and their communications (edges) generated by `toMicroservices` is from now on called MSA (i.e., microservice architecture). In addition, the cardinality of MSA is computed by the number of microservice candidates and indicated by |MSA|. In this work, a c-microservice or c microservice is some vertex in the MSA, in other words, a microservice candidate.

4.5 A Domain-Specific Language for Describing Feature-to-Code Mapping

`toMicroservices` adopts a dynamic approach to reveal the traces between each feature and the code element that contribute to its implementation. In addition, `toMicroservices` inquire users to indicate the program entry points to each feature in the execution trace. The decision of exploring the synergy between execution trace and seeds (entry points provided by users) was made to provide a better accuracy in feature location (90). Feature location is a well-known software engineering problem, which consists of establishing which program element implement a feature (90).

Basically, an entry point is a relationship between a regular expression and a feature. Each regular expression is compared with patterns in the

packages, classes, or methods names in the execution trace. That is, the regular expression can provide a match with some method of the execution trace by some name patterns. In the match, the method in the execution trace is labeled in graph vertex that represents the legacy system under analysis with the related feature.

Moreover, the dynamic analysis in the legacy system captures the depth of each method call and store in the execution trace. The depth information is used to label each method in the execution trace that is not an entry point with the feature related to the last entry point detected with a lower depth number than the current.

In order to facilitate this relationship between a feature and the regular expression, a domain-specific language (DSL) was defined in Backus-Naur Form (BNF) to facilitate this interactions to `toMicroservices` users. Moreover, reducing the effort in feature location task. Grammar 4.1 presents the BNF to describe the relationship between a feature and the regular expression. The character set (<Character> in the BNF) is an 8-bit Unicode, and ϵ represents an empty transition (also called ϵ-transition). An execution trace example is provided in Listing 4.1, while Listing 4.2 provides the description of two features and their regular expressions.

Grammar 4.1: Our DSL to describe the relationship between regular expression and features

$\langle Digit \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

$\langle Feature \rangle ::= \langle Digit \rangle \mid \langle Character \rangle$

$\langle Regular\_Expression \rangle ::= \langle Digit \rangle \mid \langle Character \rangle$

$\langle Row \rangle ::= \langle Feature \rangle$ '$\langle$' $<Composed\_Regular\_Expression\rangle$ '$>$'

$\langle Composed\_Regular\_Expression \rangle := \langle Regular\_Expression \rangle$
$\mid \langle Regular\_Expression \rangle$ ',' $\langle Composed\_Regular\_Expression \rangle$
$\mid \epsilon$

The analysis of an execution trace and the relationships of features and regular expressions generates a labeled graph that represents the legacy system. The labeled graph of the examples Listing 4.1 and Listing 4.2 is presented in Figure 4.5. The examples are simplified cases from the industrial legacy system presented in the Chapters 5 and 6.

Listing 4.1: Execution trace example

```
Name: User.getAdminIds#Depth:12
```

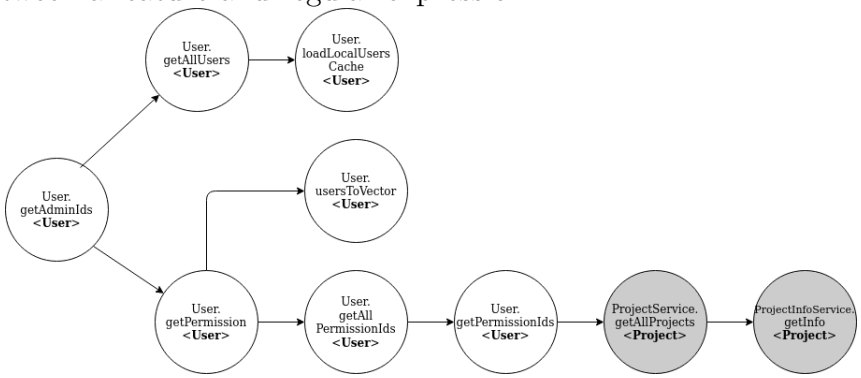
```
Name: User.getAllUsers#Depth:13
Name: User.loadLocalUsersCache#Depth:14
Name: User.getPermission#Depth:13
Name: User.getAllPermissionIds#Depth:14
Name: User.getPermissionIds#Depth:15
Name: ProjectService.getAllProjects#Depth:16
Name: ProjectInfoService.getInfo#Depth:17
Name: User.usersToVector#Depth:14
```

Listing 4.2: DSL example

```
User<User.getAdminIds>
Project<ProjectService.*>
```

The method *User.getAdminIds* (in the absolute name that includes a class and package name when existing) is matched with the regular expression associated with the feature User. In this case, the method *User.getAdminIds* is associated with the regular expression of the feature User, then the method is an entry point and labeled with this feature. The next methods with higher depth are also labeled with User feature until the method *ProjectService.getAllProjects* that match with the regular expression of the Project feature. Thus, *ProjectService.getAllProjects* is labeled in the graph with Project feature and the method *ProjectInfoService.getInfo* also is labeled since it has higher depth. Finally, the method *User.usersToVector* is labeled with the User feature because the lower depth as an entry point is User and not Project.

Figure 4.5: Graph labeled from the simplified execution trace and the relationship between a feature and regular expression



4.6 Search-Based Approach

There are various criteria considered useful or moderately useful by experienced practitioners along decision making process to identify microservice candidate. Many of them may conflict with each other. For example, the modularity-related criteria, such as maximizing cohesion, may lead to additional undesired network overhead like aforementioned in Chapter 1. The two facts, that is, the several criteria and conflicting relation between them motivates a many-objective solution to indicate microservice candidate.

Thus, we adopted the evolutionary algorithms NSGA-III (32) with the aim of finding a Pareto set to derive microservice candidates. Basically, NSGA-III was chosen among genetic algorithms since it is able to optimize three or more objectives (32). Our proposed metrics for all the criteria (Section 4.7) are used as fitness functions. The values functions presented in Equations 4-9, 4-11, and 4-14 are maximization functions, while the ones in Equations 4-6, and 4-17 are minimization functions.

Previous studies (30, 66) point that the Pareto set resulted from the genetic algorithm is strongly related to the manipulation operators, that is, the way on how a new population is generated in each cycle of the genetic algorithm. Thus, we adopt a mutation operator defined in previous works with positive results (15, 16). However, those previous works adopted a lower number of criteria to indicate microservice candidates. The mutation and crossover operators are introduced in what follows.

Mutation Operator. The operator presented by previous studies that adopted genetic algorithms (15, 16) consists of moving a single method from one microservice candidate in a current individual of the population to another microservice candidate. In other words, be a vertex $v_i \in MS_k$, then v_i is moved from the MS_k to another microservice MS_z, where $MS_k \in MSA$, $MS_z \in MSA$, and $k \neq z$. In a simplified form, this mutation operator can be seen as an analogy of the move method refactoring presented by Fowler (71).

Crossover Operator. We also intend to adopt the crossover operator to generate more diversity of microservice candidates. This operator swaps the half of methods (represented by m_k) in a MS_k to other a MS_{k+1}. Afterwards, the half of methods in the MS_{k+1} (m_{k+1}) are moved to MS_k, where each element of m_k is different of the elements in m_{k+1}. In summary, the crossover operator exchange half of the methods in a c-microservice to other c-microservice. The methods (vertices in our graph representation) are chosen randomly. Finally, the parent solution and the applied swaps between two microservice candidates are used to generate a offspring solution.

4.7 Objective Functions Computation

Our approach uses five criteria found in empirical studies that evaluate the process of migrating to microservice architecture. They are formalized in what follows.

Coupling - To compute coupling between candidate microservices, we rely on coupling using static information. The Equation 4-5 presents the coupling computed for each microservice candidates (MS_c), which is represented as a vertex of MSA, where MS_c is a vertex of MSA. In addition, sc is the number of calls present in the body of v_i method and particularly made to the v_j method, where $(v_i, v_j) \in E$ (edges set in the graph that represent the legacy system).

$$\delta(MS_c) \ = \sum^{v_i \ \in \ MS_c \ \wedge \ v_j \ \notin \ MS_c} sc(v_i, v_j) \tag{4-5}$$

In summary, δ function showed in Equation 4-5 is the number of static calls from methods within a MS_c to another microservice candidates in the same MSA (microservice architecture).

Equation 4-6 describes how to compute the overall coupling of all MS_c in the MSA. Basically, it is the sum of the couplings associated with all microservice candidates. Thus, the `toMicroservices` approach minimizes this value.

$$Coupling \ = \sum^{\forall MS_c \in MSA} \delta(MS_c) \tag{4-6}$$

Cohesion. The cohesion of a microservice candidates is computed by dividing the number of the static calls between methods within the microservice boundary (i.e., the set of methods assigned to the candidate) by all possible existing static calls. This declared way of measuring cohesion indicates how strongly related the methods are within a microservice candidate.

$$ce(v_i, v_j) = \begin{cases} 1, if sc(vi, vj) > 0 \\ 0, \ otherwise \end{cases} \tag{4-7}$$

In order to compute it, the ce function is defined in Equation 4-7 as a boolean function indicating the existence of at least a static call.

$$C(MS_c) \ = \ \frac{\sum^{\forall v_i \in MS_c \wedge v_j \in MS_c} ce(vi, vj)}{\frac{|MS_c|(|MS_c| - 1)}{2}} \tag{4-8}$$

The cohesion of a microservice candidates is presented in Equation 4-8, where $|MS_k|$ is the cardinality of a MS_c. Basically, Equation 4-8 divides

the number of static calls by the number of all possible dependencies between methods of a candidate microservice. In this sense, the denominator of Equation 4-8 is the combination two-by-two of all methods within a MS_c. The desired value in Equation 4-8 is the closest possible to 1.

Lastly, Equation 4-9 defines the microservice architecture cohesion as the sum of cohesion of all MS_c in the MSA. Thus, the genetic algorithm will maximize the computed cohesion of the microservice architecture indicated.

$$Cohesion \ = \ \sum^{\forall c \in MSA} C(c) \tag{4-9}$$

Reuse. We computed microservice candidates reuse considering the relationships between the microservice candidates and the user of the legacy system (e.g, calling the API or user interface). In this order, we propose to combine static and dynamic analysis to observe the level of reuse of a microservice within the microservice architecture. In the dynamic analysis, each microservice candidate is reusable when it is directly called by a user. This concept is captured by the *mdu* function (**m**icroservice **d**irectly called by the **u**ser). *mdu* function considers the system executions that allow identifying dynamic calls between vertices, including start points by the user.

$$r(M) \ = \ \begin{cases} 1, if \ \sum^{v_i \in M \wedge v_j \notin M} sc(v_j, v_i) + mdu(M) > 1 \\ 0, \ otherwise \end{cases} \tag{4-10}$$

In order to measure the reuse associated with each microservice, we defined the Equation 4-10. The goal of the equation captures the expectation that each microservice is useful for other microservices in the architecture or directly by the user. Whenever a microservice candidates is reused at least twice, the microservice candidates indicates an adequate reuse level.

The reuse of a microservice architecture is defined in Equation 4-11 when $|MSA|$ is the number of microservices. In addition, the property $0 < Reuse < 1$ is valid. The Equation 4-11 assume the value 1 when all microservices are used at least twice by other microservice or the user.

$$Reuse \ = \ \frac{\sum^{\forall M \in MSA} r(M)}{|MSA|} \tag{4-11}$$

Finally, the genetic algorithm tries to maximizes the reuse defined in Equation 4-11 for each of the candidate microservice architecture obtained.

Feature Modularization. We propose an strategy to indicate the responsibility of microservice candidates based on the features associated with executions of the target system. This information is provided by the user

of `toMicroservices` approach. Basically, the user provides a list of features names accessible via an interface (e.g, Rest API) of the legacy system under analysis. In addition, each feature label is associated with a part of an execution case (e.g, an interaction case) indicated by the user.

Consequently, `toMicroservices` performs the traceability between features labels and vertices (i.e, methods). This traceability is made during the execution of the legacy system responsible for implementing them. We used the vertices labeled to recommend feature modularization in the microservice candidates with fine granularity and limited responsibility.

In this order, the notion of predominant feature was created to indicate the occurrence of the feature that most occurs in the vertices (methods) associated with a microservice candidate. This notion of predominant feature is used to minimize the amount of features per microservices. Equation 4-12 defines the predominant feature (pf function) of a MS_c, where F_{M_c} is a set of occurrence by features in a MS_c. F_c set is computed using fc defined in Equation 4-3 that relates each vertex to a set of features indicated in the execution of the legacy system.

$$pf(MS_c) \ = \ max \ _{\forall k \in F_{MS_c}} \{k\} \tag{4-12}$$

$$f(MS_c) \ = \ \frac{pf(MS_c)}{\sum^{\forall k \in F_{MS_c}} k} \tag{4-13}$$

Thus, the feature modularized measured of a c-microservice was defined in Equation 4-13, that is, a measure of the number of occurrences of the most common feature divided by the sum of all functionalities occurrences within a c-microservice. Maximizing the values of the Equation 4-13 leads to the presence of a single feature or a limited set of features in a c-microservice.

Regarding the feature modularization in the proposed microservice architecture, Equation 4-13 introduced the feature per microservices. This equation avoids the fact that each microservice candidates has largely different features.

Equation 4-14 shows the measurement of functional requirement where $FMSA$ is the set of different predominant functionalities in the MSA. For example, in the case of Figure 5.3, $|FMSA|$ is two because microservice MS1 has F1 and MS2 has F2 as predominant functionality. Whether the predominant functionality of MS2 was F1, then $|FMSA|$ would be one.

The division in Equation 4-14 of FMSA cardinality and MSA cardinality is to avoid a separation of the same functionality by different microservice candidates. In addition, the occurrences of the predominant functionality in each c-microservice are summed.

$$F = \sum_{\forall c \in MSA} f(c) + \frac{|FMSA|}{|MSA|} \tag{4-14}$$

Regarding the optimization step, the genetic algorithm aims at maximizing the value obtained by Equation 4-14.

Network Overhead. A potential problem is the generated network overhead from the extracted microservices, possibly affecting negatively non-functional requirements as performance. In order to minimize that problem, we created a heuristic to predict the network overhead. The heuristic uses the size of the objects and primitive types assigned as parameters between methods during the execution of the legacy system.

The network overhead measurement is showed in Equation 4-15 where the function $P(v_j)$ return the set of arguments used in a execution of the method v_j. The function sizeOf(p, m) is the size of the p-th parameter in the m-th call from v_i to v_j.

$$overhead(v_i, v_j, m) = \sum^{\forall p \in P(v_j)} sizeOf(p, m) \tag{4-15}$$

$$dt((v_i, v_j)) = max_{m=1}^{m=dc(v_i,v_j)}(overhead(v_i, v_j, m)) \tag{4-16}$$

Thus, data traffic function (dt) is computed as shown in Equation 4-16, where dc function is the total of calls from method v_i to method v_j in execution time. The network overhead in a proposed MSA set is defined in Equation 4-17. In summary, network overhead is the sum of data traffic to each MS_c. The values computed by Equation 4-17 is minimized by the genetic algorithm.

$$\begin{aligned} O(MS_k) &= \sum^{\forall v_i \in MS_k \wedge \forall v_j \notin MS_k} dt((v_i, v_j)) \\ Overhead &= \sum^{\forall MS_k \in MSA} O(MS_k) \end{aligned} \tag{4-17}$$

5
Search-Based Many-Criteria Identification of Microservices from Legacy Systems

The cost of manual analyses for identifying microservices and their associated legacy code is high (3, 6, 28). Moreover, potential risks as network overhead and poor modularization can lead to an inadequate or complete failure of microservice extraction (17). In this way, automated approaches have been proposed to reduce cost and risk. Existing automated approaches for microservice identification adopt only a few criteria (13, 14, 15, 16, 33). Our previous results indicate that practitioners usually consider useful at least four criteria. Such an empirical evidence indicates that the current automated approaches are oversimplifying the microservice identification, and possibly yielding microservice recommendations that developers would not adopt in practice.

This chapter contains the paper: *"Search-Based Many-Criteria Identification of Microservices from Legacy Systems"*. This paper was accepted as a poster at *Genetic and Evolutionary Computation Conference (GECCO)*, to be hosted in Cancun, Mexico, 2020. The full version is being submitted to the International Conference on Software Maintenance and Evolution (ICSME). The paper presents how `toMicroservices` optimizes a selected set of five criteria with a many-objective algorithm (NSGA-III (32)). Our automated approach is applied to an industrial legacy system to support the evaluation and improvement of `toMicroservices`. In this case study, `toMicroservices` was compared to a baseline solution that optimizes two criteria: coupling and cohesion.

The adoption of coupling and cohesion was motivated insofar as they are the most common criteria in current automated approaches. We observed in our case study that the adoption of that two criteria is not sufficient to optimize feature modularization, neither reduction of network overhead as performed by `toMicroservices`. In addition, developers classified the microservice candidates as adoptable in 50% of the cases and were capable of recognizing between 4 to 7 features or subfeatures in the microservice candidates. They recognized (sub)features, which over afterwards considered as potential microservice candidates. We also report our lessons learned with respect to the use of search-based software engineering along our case study.

The next sections are part of the full paper mentioned above. In case the reader has read Chapter 4, you may consider to skip Section 5.3, which provides a more succinct description of our approach.

5.1 Introduction

Legacy systems are commonly found in industry and represent a long-term massive investment (9, 10). Despite their business importance, these systems often depend on obsolete technologies (3, 6). Moreover, their maintenance is inherently expensive as many features suffer from poor modularization. Their features are highly tangled to each other within various system's modules (9, 10). Many industries are increasingly modernizing legacy systems by extracting their features into *microservices* (3, 4, 5, 6, 7, 8). Nowadays this is a key strategy to increase the longevity of legacy systems, while offering new business opportunities to organizations (6, 11).

The identification of each single microservice from legacy code is a complex, time-consuming, and error-prone task (6, 17, 12, 28, 67). To reap the benefits of microservices, developers must reason about multiple criteria while identifying legacy's features as microservice candidates. A recent study with industry experts revealed that developers must simultaneously satisfy at least four criteria (67). These criteria are intrinsically associated with the definition of what is a microservice. Microservices are *small* and *autonomous* services that work together by using lightweight network protocols (1). The notion of *small* refers to the need of producing cohesive, fine-grained microservices (61) with each modularizing a single feature (2). The notion of *autonomous* implies that each microservice should be highly decoupled, reusable, whereas reducing the risk of unacceptable network overhead (1, 2, 11, 17).

The problem of microservice identification in legacy code is commonly seen as a software remodularization task, which is known to be an NP-hard problem. There is a huge number of possible combinations of source-code elements and its multi-criteria nature (68). Some studies have proposed search-based approaches for identifying microservices in existing systems, but they are of limited applicability (13, 15, 16, 33). First, most of such approaches solely rely on the use of coupling as a criterion to identify microservice candidates (13, 15, 16, 33). The most recent approach explores the use of coupling and cohesion (16).

However, none of them consider the multiple basic criteria that must be addressed by each microservice. Second and even worse, the performance of such approaches is merely assessed quantitatively and in the context of: (i) simple, academic, already well-modularized systems (33), or (ii) an open source project where they do not have any access to the actual developers (13, 15). The performance of existing search-based approaches on confirming the potential adoption of the microservices in practice is unknown. Third, such approaches

cannot be applied (or even easily adapted) to the context of real legacy systems. They are too restrictive as they perform a coarse-grained search at the level of modules (or files) and not a fine-grained search at the level of methods (or functions). However, legacy systems often have very large files (or modules) where many features are tangled to each other. In other words, methods in a legacy module are often implementing various intertwined features, thereby complicating the identification of microservice candidates.

Thus, we propose a many-criteria search-based approach, called `toMicroservices`, for supporting the identification of microservices in legacy code. Our approach simultaneously optimizes five criteria intrinsically associated with microservices, ranging from cohesion and coupling to feature modularization, reuse and network overhead (Section 5.3). Differently from existing approaches, `toMicroservices` is designed to identifying microservices scattered in methods, rather than simply modules (files), in legacy code.

Our quantitative and qualitative evaluation of `toMicroservices` (Section 5.4) is the first to be performed in the context of an industrial *in-situ* case study (Section 5.2). The evaluation involved observations along two years as well as interactions with the actual developers along the process of migrating part of the legacy system to microservices. They relied on the use of `toMicroservices` to identify microservice candidates.

Our main findings about our approach are (Section 5.5): (i) it outperforms an adaptation of a representative, state-of-the-art solution (our baseline), (ii) it is able to find various microservices that simultaneously satisfy well all the five criteria; many of these microservices were considered as highly adoptable by the legacy developers, and (iii) it is able to successfully identify microservices that modularize features otherwise tangled and scattered through many modules of the legacy code. Finally, we discuss threats to validity, related work and conclude the paper with suggestions of future work (Section 5.8).

5.2 Industrial Case Study

Target Legacy System. Our case study relies on a Java legacy system maintained for more than 15 years in the context of oil and gas industry. The developers of this system reported that its maintenance is very complex and time-consuming even to add a simple new feature. For example, as the system does not use a database, the addition of a search for a keyword in files was more time-consuming than expected. Moreover, the system exhibits all the legacy problems mentioned in Section 5.1.

Microservices Identification Context. Aiming at mitigating the difficulties to maintain the legacy system, five developers involved in its maintenance started a process of migrating the legacy system to a microservice architecture. Initially, the developers tried to perform a manual analysis of the source code to identify possible features to be extracted into microservices by considering four main criteria: coupling, cohesion, reuse, and feature modularization. However, this manual analysis required a high effort from developers due to the system complexity and size. To reduce effort, the developers used visual tools that represent classes and their relations of the legacy system as a graph. The use of such tools was not helpful due to poor, complex modularization of the software features.

Developers also considered to apply state-of-the-art search-based approaches (Section 5.1) to derive microservice candidates. However, they struggled to use them for the reasons mentioned in the previous section. In fact, the developers considered such solutions unsuitable in practice. Moreover, the quality of the approaches' results were considered largely unsatisfactory. Thus, we presented `toMicroservices` (Section 5.3) to our industry partner as a possible alternative to automate the identification of microservice candidates for the legacy system. The developers of the legacy system agreed to analyze the microservice candidates generated by `toMicroservices` along their migration to a microservice-oriented architecture.

"Microservification" of certain legacy's features. Despite the legacy's several features, the developers decided for considering the "microservification" of three large, important features (or eventually their various subfeatures). These three features are described as follows:

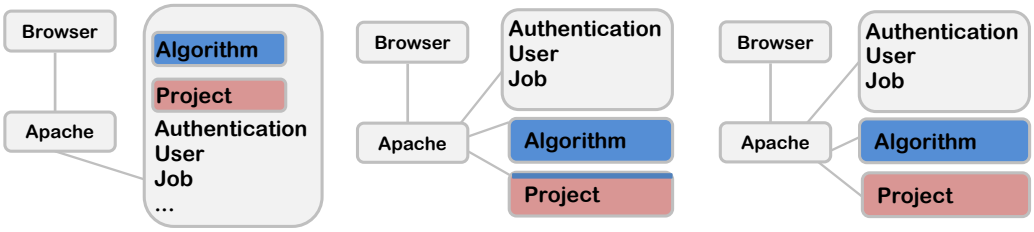
- Algorithm: responsible to store and provide algorithms information by a REST API, including parameters, binary, documents, and connection points with other algorithms. In addition, this information can be stored in different versions.

- Authentication: responsible for verifying the identity of system's users. This includes the creation of tokens and their validation, verification of login and password, update of password, and related simple information about the system's users. Source codes related to this core functionality are used extensively by almost the entire system for checks and information retrieval.

- Project: responsible for the concept of a collaborative environment between system's users. This collaborative environment includes share projects and their metadata between different users or types of users. In addition, the projects also store shared files.

Example of Many-Criteria Analysis. In the context of our case

study, Figure 5.1(a) depicts the current monolithic architecture, which is the source of information. Figures 5.1(b) and 5.1(c) present illustrative alternative microservice architectures, each one with two microservice candidates and the residual legacy system. Source code elements responsible for the feature *Algorithm* are highlighted in blue whereas the code elements highlighted in red are related to the feature *Projects*.



5.1(a): Monolith 5.1(b): Alternative 1 5.1(c): Alternative 2

Figure 5.1: Alternative Architectures for the Legacy System

Let us consider the Alternative 1 in the Figure 5.1(b). This architecture has one microservice with implementation exclusively of the feature *Algorithm*. Still, it has another microservice with implementation predominantly related to *Project* but also with some methods related to *Algorithm*, which are called in the logic of *Project*. We can infer that this solution is good in accordance to coupling and cohesion, which are low and high, respectively. However, the features are not well-modularized, since the implementation of *Algorithm* is scattered in two microservices, and *Algorithm* is tangled with *Project* in the second microservice.

Also, this alternative does not take into account that a method allocated in the first microservice candidate could massively call a method allocated within the second microservice, that would lead to a prohibitive network overhead. On the other hand, Alternative 2 (Figure 5.1(c)) is better than Alternative 1 in terms of feature modularization and would avoid network overhead. In this case, all implementation of the feature *Algorithm* is within a single microservice, similar to the implementation of *Project*. Because of this good modularization, even existing massive calls between methods related to *Algorithms*, it would not be problem since they are in the same microservice.

Both alternatives illustrate that more than two criteria are needed to achieve satisfactory microservice identification. For industrial legacy systems, as our case study, approaches should consider several criteria and optimize them to obtain a suitable microservice architecture. Moreover, existing approaches make simplistic assumptions about real systems from which microservices will be identified and extracted – e.g., features of the existing system

usually has well-modularized features in files, i.e., not tangled and scattered through several methods of those files.

5.3 Proposed Approach

In this paper, we introduce `toMicroservices` - an automated approach to identify microservice candidates on legacy systems to aid developers in the migration process. Due to the complexity of the problem, the approach relies on a many-objective optimization. `toMicroservices` uses five objective functions related to criteria classified by developers as useful or moderately useful (67). The criteria are Coupling, Cohesion, Feature Modularization, Network Overhead, and Reuse, which are described in details in Section 5.3.2. The approach's details are presented in what follows.

5.3.1 Input, Representation, and Output

`toMicroservices` requires three pieces of information as input: (i) the legacy source code, including code elements indicators that will not be parsed, (ii) a list of features related to each execution of the legacy system and (iii) the number of microservices to be identified. Our approach analyzes the input at method level to achieve fine-grained microservices.

The proposed approach uses a graph-based representation. Each vertex represents a method of the legacy system assigned to its respective feature. Each edge contains information on static and dynamic perspectives. Besides, it represents a relationship between methods describing data communication, syntactic calls between them, and dynamic calls occurrence. Figure 5.2 depicts an excerpt of the adopted representation for the legacy system. The graph presents five vertices, each representing a method in the legacy system under analysis. The vertices $m1$, $m2$ and $m3$ are responsible for the feature *Algorithm*, whereas the vertices $m4$ and $m5$ realize the feature *Project*.

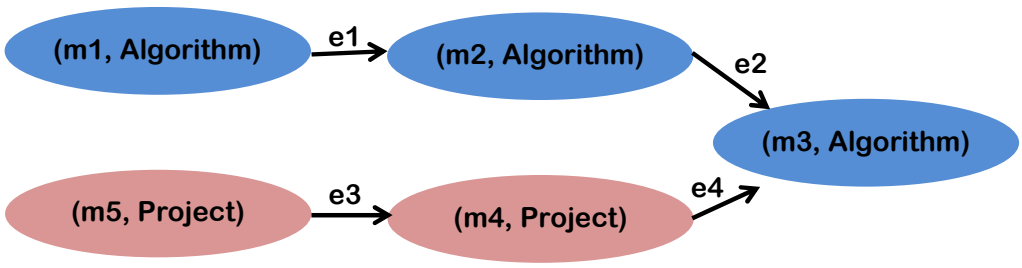


Figure 5.2: Excerpt of the legacy's representation

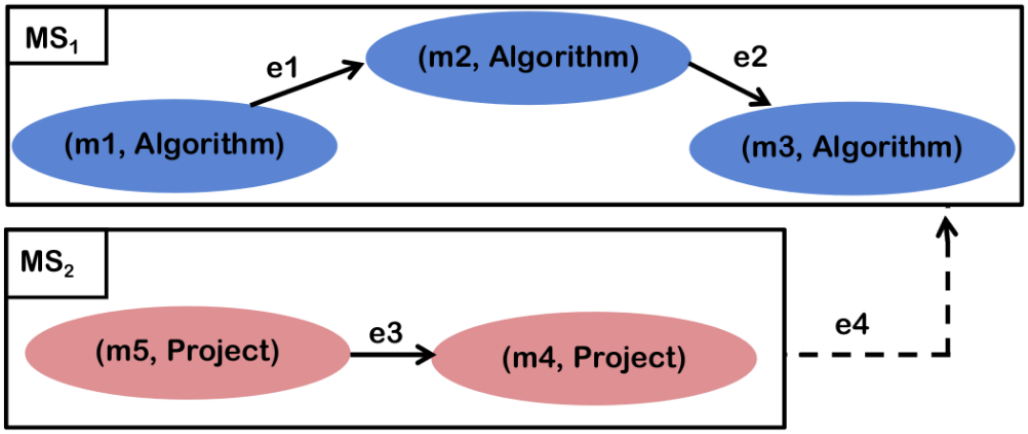


Figure 5.3: Microservice candidates for the excerpt of Figure 5.2

The output artifacts generated by `toMicroservices` are: (i) a set of candidate solutions, named Pareto Front (PF), where each element is a graph of microservice candidates, and (ii) the source code in the legacy system associated with each candidate. Each solution is another graph where the vertices are microservice candidates and the edges are the communication between methods and database entities by the particular choices of the microservice candidates. A possible output of `toMicroservices` is depicted in Figure 5.3, where: (i) $m1$, $m2$, and $m3$ are part of the MS_1 microservice, and (ii) $m4$ and $m5$ are included in MS_2 microservice. In this case, the communication between $m3$ and $m4$ in the solution representation realizes the communication between microservices MS_1 and MS_2. The values for data communication, syntactic calls between them, and dynamic calls occurrence related to each edge were omitted in the examples shown in Figures 5.2 and 5.3.

5.3.2 Objective Functions

Next, we describe each criterion adopted as objective function. The criteria of Coupling and Cohesion were based on related work, but adapted in our approach to the fine-grained (methods) level. The criteria of Feature Modularization and Reuse have been used in the context of traditional architectures, which inspired us on proposing their application in the context of microservices. Network Overhead is a novel designed criterion, which is introduced in this paper.

These criteria are described next. From now on, MSA (MicroService Architecture) refers to the graph of the microservice candidates (vertices) and their communications (edges) generated by `toMicroservices` as well as MS_C (MicroService Candidate) refers to some vertex in the MSA.

1. Coupling: this criterion is computed by using static information similarly to (69). The individual coupling for each microservice candidate MS_c is computed by the sum of the number of static calls from the methods in v_i (that belongs to MS_c) to methods in v_j (that does not belongs to MS_c). A static call consists of a syntactic call to another v_j method present in the body of the v_i method. The total coupling of a solution, i.e., an individual, is the sum of the values of coupling associated with every MS_c in a MSA. The lower the coupling, the better result.

2. Cohesion: cohesion is defined by dividing the number of the static calls between methods within the microservice boundary (i.e., the set of methods assigned to the candidate) by all possible existing static calls (similarly to (69)). Hence, this measure indicates how strongly related the methods are within a microservice candidate. The total cohesion of a solution is the sum of the cohesion associated with every MS_c in a MSA. The higher the cohesion better.

3. Feature Modularization: in a pre-processing traceability step of our approach, we use the legacy source code and the list of features related to each execution of the system to assign to each vertex (i.e., methods) in the MSA a label corresponding to the features it belongs to. Based on that, a MSA can have MS_c composed of methods belonging to several features. We used the vertices labeled to recommend feature modularization in the microservice candidates with fine granularity and limited responsibility. The predominant features number for a MS_c is the number of occurrences of the most common feature divided by the sum of all features occurrences within MS_c. The feature modularization of a solution MSA is the sum of the predominant features number in every MS_c added to the division of the number of distinct predominant features in the MSA by the number of microservice candidates. It is desired a degree of feature modularization as higher as possible.

4. Network Overhead: Some non-functional requirements may be affected by the network overhead of the identified microservices. To minimize the overhead, we created a heuristic that uses dynamic information to predict the network overhead. The heuristic uses the size of the objects and primitive types passed as parameters between methods during the execution of the legacy system. In addition, the heuristic considers the network overhead caused by the adopted protocol to communicate with the future extracted microservices. For example, the HTTP protocol adds a header to each call and, therefore, the size of this header is considered in our estimation of network overhead. The network overhead in a proposed MSA is defined as the sum of the sizes of the network traffic data to each MS_c.

5. Reuse: In `toMicroservices`, a microservice candidate is considered reusable when other microservices or the user call it. In this order, we combine static and dynamic analysis to measure the level of reuse of a MS_c within MSA. In the dynamic analysis, each microservice candidate is reusable when it is directly called by the user (e.g, calling the API or user interface). The static analysis computes the relationships between the microservice candidates. Ideally, a microservice should be reused as much as possible or at least twice (70). The reuse of a MSA assumes the value 1 when all microservices are used at least twice by other microservice or the user. Its value ranges from 0 to 1. The goal is maximizing the reuse of the microservices encompassed by a solution.

5.3.3 Genetic Operators

The *mutation operator* of `toMicroservices` is based on the operator used by previous studies that adopted genetic algorithms (15, 16). It consists of moving a single method from one microservice candidate in an individual of the population (a set or graph of the microservice candidate) to another microservice candidate. In a simplified form, this mutation operator can be seen as an analogy of the move method refactoring (71).

In our approach the *crossover operator* exchanges a fraction of the methods in a microservice candidate to another microservice candidate. The value of such a fraction is configurable. The methods (vertices in our graph representation) and microservice candidates are randomly selected. The main goal of this operator is generating more diversity of microservice candidates.

5.3.4 Implementation Aspects

One might argue the use of NSGA-II, which is the most common evolutionary algorithms to deal with multi-criteria problems. But, different from NSGA-III, NSGA-II faces some challenges and difficulties for problems with more than three objectives (73). To implement NSGA-III, we use the jMetal, which is an object-oriented Java-based framework that includes modern state-of-the-art optimizers (72). We also adopted jMetal to implement a baseline approach, which uses NSGA-II (77) with two objectives (see Section 5.4).

During the implementation, we decided to treat the microservice identification as a minimization problem. Hence, the objective functions related to Cohesion, Feature Modularization and Reuse have their values inverted during the evolutionary process. A constraint related to the minimum and maximum numbers of methods by microservice was established in order to balance the

granularity and preserve the reasonability of each microservice. Solutions that violate this constraint are discarded.

The pre-processing traceability step performed before the optimization process in order to label each vertex with the feature that it implements was realized as follows. The feature mapping was configured by the manual analysis of the legacy system execution. During this analysis, a developer, who is an expert in the design of the legacy system, informed entry points in the trace execution of the three features, in principle, selected to become microservices and their subfeatures. Each entry point in the trace execution was used to label the vertices of the graph representation introduced in Section 5.3.1.

5.4 Empirical Evaluation Design

In this section we present details of the empirical study conducted to evaluate `toMicroservices` in the context of the legacy system presented in Section 5.2. Next we describe the research questions, the algorithms' parameters, and the quantitative and qualitative studies. For the quantitative study, we perform two experiments. The first one is the application of `toMicroservices` using NSGA-III. The second experiment (from herein called "Baseline") is an adaptation of an existing approach (16) to perform a fine-grained search at the level of method. Baseline optimizes the same criteria considered in (16): coupling and cohesion.

5.4.1 Research Questions

The empirical study conducted to evaluate `toMicroservices` has the goal of providing answers for the following research questions:

RQ1- *How is the performance of toMicroservices during the optimization process in comparison to the baseline approach?* This question intends to provide information about the behavior and advantages of using a many-objective approach, namely `toMicroservices` that uses NSGA-III, in comparison to the Baseline approach, which considers only two objectives and uses NSGA-II. For answering it, we rely on a quantitative study based on traditional quality indicators used in multi-criteria optimization.

RQ2- *Do the developers judge the solutions found by toMicroservices adoptable to be implemented in practice?* `toMicroservices` was conceived based on criteria considered useful by practitioners (67). In this way, we want to investigate practical usefulness of obtained solutions from the point of view

of developers. In order to obtain answer for RQ2, we conducted a quantitative study with developers with expertise in the target system.

RQ3- *What are the most influential criteria taken into account by developers during the solutions evaluation?* To support our analysis, we explore the expertise of developers asking about the criteria they really care for evaluating microservice candidates for the legacy system and, more importantly, we analyze which criteria they really take into account during the evaluation.

5.4.2 Algorithm and Parameters

In NSGA-III the population size is defined according to the number of reference points. In our case, this size was defined as 50 individuals. The crossover and mutation rates were defined as 0.8 and 0.4 respectively (as in (15)), and the maximum number of fitness evaluation was 500,000. This last parameter is also the stopping criterion.

In addition to these traditional parameters, there are some parameters related to our problem: the fraction of methods exchanged in the crossover operator, that was set to 0.5, the number of microservice candidates and the range of methods allocated in each microservice. The latter two parameters were configured in two scenarios. In *Scenario 1*: number of microservices was 5, whose size should have between 5% and 50% of the total number of methods of the input. In *Scenario 2*, the number of microservices was 10 with minimum of 3% and maximum of 16% of the input size in terms of methods number. Our approach optimized five criteria: coupling, cohesion, feature modularization, reuse and network overhead.

As briefly described at the beginning of this section, to evaluate `toMicroservices` we considered a Baseline approach from a related work (15). Baseline is a multi-objective approach that optimizes two traditional criteria for identifying microservices, namely coupling and cohesion. This approach uses the traditional NSGA-II algorithm, widely used in search-based software engineering studies (74). The two objectives adopted by Baseline were implemented according to the criteria of coupling and cohesion described in Section 5.3.2. Baseline was setup mostly with the same parameters of `toMicroservices`.

In this way, we can perform a fairly comparison between the two approaches. In this sense, the main difference between them is the objective functions optimized during the evolutionary process. We executed 10 independent runs for each approach due to the execution time which is impacted by the legacy system size and the number of objectives.

5.4.3 Quantitative Comparison Against Baseline

The quantitative study has the goal of providing pieces of evidence to compare the behavior of `toMicroservices` against Baseline, providing answer for RQ1.

Since we are dealing with multi-criteria optimization, our analysis is based on two traditional quality indicators, namely Hypervolume (HV) and Euclidean Distance to the Ideal Solution (ED). HV measures the area of the objective space from a reference point to a front of solutions (75). In this study we use the HV computed by a recursive and dimension-sweep algorithm (76). The reference point adopted to compute HV was the worst values of all objectives, considering solutions of both algorithms, and incrementing 10% in each objective values to make the point strictly dominated by all existing solutions. ED is used to find the closest solution to the best theoretical objectives, i.e. an ideal solution (85). Since both NSGA-III and NSGA-II were implemented to deal with all objectives as a minimization optimization, an ideal solution has a value equal to 0 for all objectives.

To analyze the statistical difference between `toMicroservices` and Baseline we used the Wilcoxon signed-rank test (78) as the data sets have non-normal distribution. Furthermore, we also compute the effect size with the Vargha-Delaney's $\hat{A}_{12}$ measure (79). Both tests are widely used to assess search-based algorithms in Software Engineering (74, 80).

5.4.4 Qualitative Evaluation with Developers

This section describes a qualitative study conducted to evaluate the practical usefulness of solutions obtained with `toMicroservices`, from the point of view of developers. This study has the goal of providing pieces of evidence to answer RQ2 and RQ3.

This qualitative study involved interviewing some developers of the legacy system with respect to their opinion about some solutions generated by `toMicroservices` in order to answer the posed questions. The following steps were performed:

1) We selected eight developers specialist in the legacy system.

2) We selected five solutions from the set of non-dominated ones found by `toMicroservices`. Each solution has the best value for one of the proposed criteria. One of them has also the best trade-off among the five objectives as it represents the best trade-off.

3) Developers answered a survey where firstly they indicated, using the Likert scale, how useful they find each proposed criterion.

4) In a second part of the survey we presented the three features we are considering in our evaluation and asked the developers to indicate how specialist, using the Likert scale, they are for each feature. The question asks: *Rate your knowledge about <feature>: Not knowledgeable (I do not know anything); Somewhat knowledgeable (I have a vague idea); Knowledgeable (I am familiar with it); Very knowledgeable (I know all/most classes and methods of it).*

5) During the interview, each developer assessed four microservices from two different solutions. The solutions were selected from the set of solutions created in Step 2. The first selected solution has the best value of the criteria pointed as the most useful by the developer. The second one has the best value of the criteria pointed as the lowest useful. Given the selected solutions, two microservices were picked from each solution, according to the expertise of the developer indicated in Step 4. For each microservice, we repeated the following process: we presented the solution and asked the developer: *Would you adopt this microservice? Why?*. The adoptability was collected in a five-point Likert scale and the motivation was summarized by the interviewer and interviewee in a textual field.

6) In the last step, all data were collected and analyzed by three of the authors.

5.5 Results and Analysis

In the next sections we present the results of the empirical study to answer the posed research questions.

5.5.1 RQ1 - Performance of toMicroservices

As aforementioned, the Baseline approach optimizes only two objectives, namely coupling and cohesion. For the comparison of the performance of our approach, which uses five objectives, we computed for the solutions of Baseline the additional values of reuse, feature modularization, and overhead. This was carried out after the evolutionary process of NSGA-II. This allows us to reason about the advantages of using a many-objective approach in comparison to a multi-objective one.

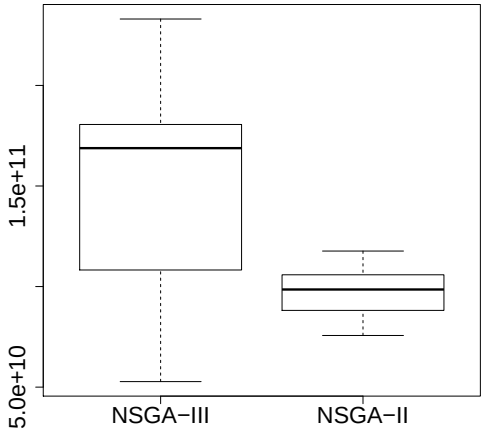
Table 5.1 presents results of the Wilcoxon statistical test and the $\hat{A}_{12}$ effect size for the indicators of HV and ED. One should recall that for HV

greater values are better and for ED lower values are better. In the table we can observe that NSGA-III reached a greater average value of HV than NSGA-II. However, the standard deviation is greater than the one of NSGA-II. To corroborate this analysis, we can analyze the boxplot in Figure 5.4(a). The values of HV for NSGA-III are in a great interval, differently from NSGA-II. The Wilconxon test pointed significant difference of HV values between `toMicroservices` and the Baseline approach. The efect size shows that NSGA-III of `toMicroservices` reaches better solutions than the NSGA-II of Baseline in 85% of the independent runs.

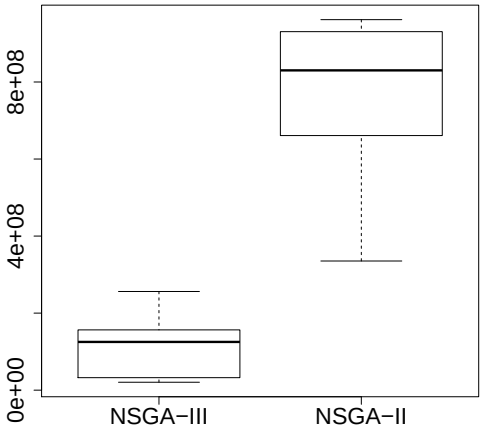
Regarding ED, there is a similar situation concerning the average and standard deviation values. However, here all the values of ED obtained by `toMicroservices` are better than the Baseline. This is clearly seen in the boxplot of Figure 5.4(b). This is also supported by the significant difference ($p\text{-}value < 0.05$) and the effect size, which pointed that NSGA-III is the best in all independent runs.

Table 5.1: Results of Hypervolume (HV) and Euclidean Distance from Ideal Solution (ED) of the 10 independent runs.

Indicator	**Average (Std dev.)**		**Wilcoxon**	**$\hat{A}_{12}$ Effect Size**	
	NSGA-III	**NSGA-II**	**p-value**	**NSGA-III**	**NSGA-II**
HV	1.50E+11 (5.36E+10)	9.69E+10 (1.23E+10)	6.84E-03	85%	15%
ED	1.13E+08 (7.67E+07)	7.59E+08 (2.25E+08)	8.25E+06	100%	0%



5.4(a): HV



5.4(b): ED

Figure 5.4: Boxplot of the Hypervolume (HV) and Euclidean Distance to the Ideal Solution (ED). For HV higher values are better, for ED lower values are better.

Figure 5.5 presents the values of coupling and cohesion on the search space for both approaches. The solutions of `toMicroservices` are close to

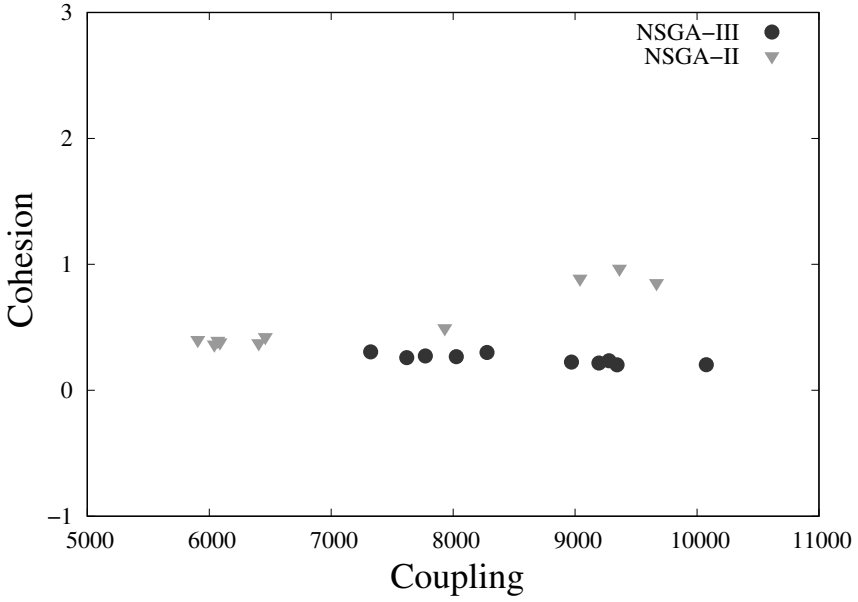


Figure 5.5: Solutions with best ED per run of toMicroservices (NSGA-III) and Baseline (NSGA-II) considering the traditional criteria of Coupling and Cohesion.

Baseline ones regarding coupling, with some better solutions for the latter. Taking into account cohesion, the values reached by NSGA-II are slightly better (see the scale on Y axis). Considering only the objectives of coupling and cohesion, we can observe that `toMicroservices` can reach acceptable solutions, when compared to the Baseline. A small advantage for the Baseline was expected, since the NSGA-II exclusively optimizes only these two objectives. On the other hand, our approach optimizes three additional objectives, leading to different benefits, as described in the following.

Here we investigate the relevance of using three additional objectives equally relevant to identify microservices. Reuse is a boolean criterion, in which all solutions of `toMicroservices` reached values equal to 1, indicating that the microservice candidates found by the algorithms are reused in other parts of the architecture. The impact of feature modularization and overhead on the results is presented in Figure 5.6. In this figure we can see the difference in the search space when `toMicroservices` considered additional criteria. All solutions found by NSGA-III in our approach are better than the solution of NSGA-II of the Baseline approach for the criterion of feature modularization. In addition, almost all solutions of NSGA-III are also better regarding overhead, since they are mostly grouped in the left area of the graph, which correspond to lower overhead.

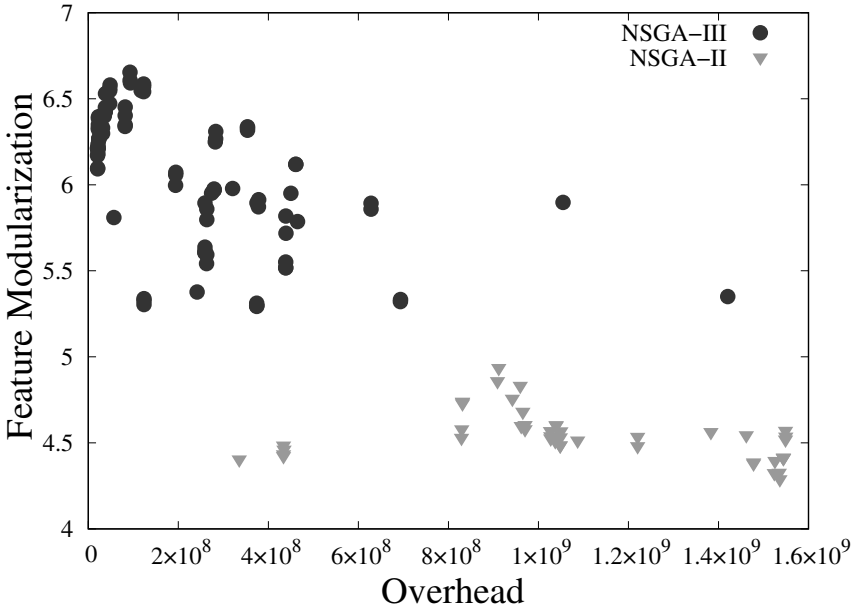


Figure 5.6: Solutions of toMicroservices (NSGA-III) and Baseline (NSGA-II) considering the aditional criteria of Feature Modularization and Overhead.

5.5.2 RQ2 - Analysis of solutions by developers

For the qualitative study, we inquired eight participants about the adoption of microservices generated by `toMicroservices`, as presented in Table 5.2. As seen in this table, we designed two scenarios for the qualitative study. In the first one, `toMicroservices` was setup to obtain solutions considering the identification of five microservices from the legacy system. In the second scenario, our approach was setup to identify 10 microservices. These two scenarios were based on the choices of the actual developers along the case study. Participants P1 to P4 were inquired about the adoptability of microservices identified in the first scenario, where as P5 to P8 did the same, but for the second scenario. Each participant was asked to analyze the adoptability of the four microservices of the their corresponding scenario.

Table 5.2: Results of the Qualitative Evaluation

Participant	Years of experience in the system	Recognizable features	New recognizable features	Preferred microservices
Scenario 1: Architectures with 5 microservices				
P1	0.5	5	2	3,2,4,5
P2	2	3	2	3,2,4,1
P3	2	4	1	2,4,2,4
P4	20	7	6	1,1,1,1
Scenario 2: Architectures with 10 microservices				
P5	13	6	4	5,3,2,1
P6	8	4	2	1,5,1,4
P7	1	5	3	3,3,2,4
P8	3	5	3	2,4,4,3

As far as the developers' experience is concerned, the participants of our

study are mostly experienced developers, with a median time experience in software development of 12.5 years. Regarding the time of experience with the target legacy system, we have both experienced developers, with time experience of 8, 13 and 20 years, and recent developers, with experience among 0.5 and 3 years. Seven participants are developers of the legacy systems and one is a team leader. This is an interesting set of participants as we can identify how both experienced and novice developers analyze the `toMicroservices` solutions for microservice identification.

Interestingly, regardless of their experience in the legacy system, all participants could identify the predominant features being modularized within the microservices. In addition, the participants also recognized new features (including subfeatures), which are not subfeatures previously informed as known by the participants, but identified during the adoptability analysis of the microservice candidates. However, participants with high experience in the target system were able to recognize more features and new features than moderately experienced participants. One participant of Scenario 2 stated the following with respect to one feature (algorithm) and two subfeatures (parser and writer): *"This microservice is a subset of a more general microservice of algorithm with at least parser and writer"*.

The participants who analyzed solutions of Scenario 1 indicated that some microservices would not be adopted in practice because of their large size. According to the participants, these microservices had too many methods as part of a coarse-grainded feature, and different features undesirably tangled in the same candidate microservice. A participant stated that he would not adopt this microservice because: *"Very large with too many methods, which realize different, unrelated features, i.e., project and authentication"*. The quantitative results indicated that 50% (8 microservices) of the proposed microservices are partially adoptable or adoptable. One participant argues: *"This microservice realizes management of project files, and it is highly reusable by different tenants."*. Other participants accepted another microservice: *"The microservice is highly cohesive. However, I would further decompose it into a smaller microservice, even tough given certain constraints in our architecture, I would adopt the proposed microservice as is"*.

For Scenario 2, the analysis of the solutions by the developers pointed that they would fully or partially adopt 63% (5 microservices) of the proposed microservices. The participants indicated three microservices as not interesting at all to be adopted in practice, mainly because the tiny features modularized by them are too small.

One participant said: *"This isolated microservice does not make sense. I*

Table 5.3: Criteria cited by participants during the microservices adoption analysis

Participant	# Useful Criteria	Most Influential Criteria
P1	4	Feature Modularization, Reuse, Cohesion
P2	5	Feature Modularization
P3	3	Feature Modularization
P4	5	Feature Modularization
P5	4	Feature Modularization, Cohesion
P6	5	Feature Modularization, Cohesion
P7	5	Feature Modularization
P8	5	Network Overhead, Cohesion, Feature Modularization

believe that it should be merged into the microservice in charge of mastering project files." However, the same microservice, which modularized the aforementioned subfeature above, was accepted by other participant. Similar contradicting cases occurred among other practitioners. These cases evidence a divergence among participants and their analysis on the adequate level of granularity for the microservice boundaries.

In general, the results regarding the adoption of microservices obtained by our approach were: three participants would adopt at least one microservice, four participants would adopt two microservices, and only one participant would not adopt any microservice.

5.5.3 RQ3 - Most Influential Criteria

During the analysis of the solutions we recorded the audio of the interviews and took notes about any comment made by the developers. Based on this information, we identified how many criteria the participants analyzed and considered as useful. The number of criteria identified for each participant is presented in the second column of Table 5.3. Five criteria were clearly considered by five participants. Two participants took into account four criteria during their analysis. Only one participant considered only three criteria during the analysis of solutions. These results are consonant with the literature reports, which state that in practice many criteria are useful during the identification of microservices from legacy systems (67, 81). Besides that, our results show that developers in fact use these criteria when to analyse a proposed microservice.

Based on the information collected during the interviews, we produced a ranking of the criteria that the participants most mentioned when considering the characteristics of the solutions. This is shown in the last column of Table 5.3. We can say that these are the most "preferable" criteria in the point of view of developers along our case study. We can observe in this column that feature modularization is the main criterion used in the analysis. This can

corroborate with the analysis in the previous subsection, where the participants many times reasoned about features and subfeatures to confirm or refute the adoption of a microservice candidate.

We can infer that developers search for microservices that contain a well-defined, cohesive and reusable feature (or subfeature). Cohesion is also often mentioned. Reuse and overhead were taken into account as the favorite criteria only by two developers. We believe that these criteria are more difficult to be analyzed, and the developers would need more time to reason about them. Interestingly, the traditional criterion of coupling, used by the majority of existing approaches, was not mentioned as the preferable criterion by any participant.

5.6 Lessons Learned

Scattered features need further attention. For instance, Authentication was not captured by a microservice as a main feature. The Authentication methods were scattered between several microservices in the execution setting with five and ten microservices generated. The scattered methods of Authentication can be due to two reasons: (i) bad modularization in the legacy system, (ii) it is a crosscutting feature. An intensive bad modularization should be difficult to convert to acceptable values of cohesion and coupling in a microservice. In addition, the crosscutting feature may be optimized with different fitness values since in microservice architecture is common to find methods regarding e.g. Authentication or Monitoring that are highly-coupled with other microservices.

After the interview, the developers were inquired about improvements that could help them in the analysis. Half of the participants pointed that a visual representation of the methods inside a microservice could make the analysis easier. In addition, at least one participant recommended: (i) to remove auxiliary functions providing a "clear" view of the microservices (ii) to remove dependencies that will not exist in the migrated system, and (iii) to highlight the predominant feature or subfeature in a microservice.

5.7 Threats and Literature Limitations

Threats to Validity. We discuss here the main threats to validity of our work. Regarding the qualitative study, three researchers analyzed the developers' textual answers to mitigate potential problems in the coding process. After a first analysis, the three researchers gathered to converge

and decide for the adopted code, which was analyzed by a fourth researcher. Another threat is related to the divergence of researchers' opinions about which criteria were taken into account by each participant. To mitigate this threat, three authors deeply analyzed and discussed each participant answer until achieving a consensus.

Another threat might be the number of independent runs executed in the quantitative study insofar as an evolutionary algorithm was used. We executed 10 runs because of the size of the legacy system and the number of objectives, which required a lot of runtime to conclude each run. Despite of being less than the recommended, our sample size was enough to execute statistical comparisons. The standard deviation of the HV and ED results were not abnormal, and there was no outlier in our sample.

Finally, the full microservice architectures (i.e., all possible microservice candidates and their inter-dependencies in each solution) generated by `toMicroservices` were not fully evaluated insofar as this task is time-consuming. To mitigate this threat we inquired each developer on the adoptability of a set of microservices selected by the same microservice architecture.

5.8 Conclusion

This paper introduced `toMicroservices`, an automated approach to identify microservices from legacy systems. `toMicroservices` deals with five criteria observed as relevant and useful in an industrial survey. `toMicroservices` was compared with a baseline approach in an industrial case study. The baseline considered only two traditional criteria, namely coupling and cohesion.

The quantitative results pointed out significant difference between the baseline and `toMicroservices`, reinforcing the needs to adopting more criteria than traditional ones. We observed that the criteria of Feature Modularization, Network Overhead, and Reuse introduced a new perspective in the optimization of the solutions since they are not subsumed by coupling and cohesion. The qualitative results indicated that developers would adopt microservices identified by `toMicroservices` during the process of migrating to microservice architecture. Moreover, the results of the case study show that developers usually take into account five criteria to assess the possible adoption of a microservice candidate. In addition to coupling and cohesion, the criteria of Feature Modularization, Network Overhead, and Reuse provided insightful information that were clearly used in the analysis by the developers. Feature modularization was the most influential criterion.

6
A Qualitative Evaluation of Recommended Microservice Architectures

The previous chapter presented a quantitative and two qualitative evaluations of `toMicroservices`. Those studies focused on comparing the performance of `toMicroservices` and a baseline approach. Moreover, practitioners of an industrial case were questioned about individual microservices generated by `toMicroservices`. The results suggest that `toMicroservices` generates better solutions then the baseline approach. Moreover, `toMicroservices` was evaluated as able to generate microservice candidates that are adoptable by practitioners.

Furthermore, the findings from the past study indicate the importance of adding a certain degree of interaction between users and the solutions generated by `toMicroservices`. In particular, our approach should be able to allow developers to perform certain modifications on candidate microservices, and ask for our approach to search for further improvements considering those modifications. Developers should also be able to accept adequate microservice candidates, partially modify adequate microservices and re-execute `toMicroservices` to discover new microservices given the constraints generated by the user. Thus, this chapter presents `toMicroservices` with these improvements.

Moreover, the previous studies in Chapter 5 did not investigate the complete adoptability of the microservice architectures generated by `toMicroservices`. Thus, this chapter presents a focus group study to expand the understanding of the adequacy of the solutions generated by `toMicroservices`. Groups of developers in our industrial case study were inquired about the entire microservice architectures generated by `toMicroservices`. For this purpose, architectural artifacts were presented to the developers using visual representations. In addition, the relative usefulness of the artifacts was questioned to each developer. Furthermore, developers were also inquired about the relative perception of the focus group method to better instruct the analyses of the candidate microservice architectures.

The content of this chapter presents an expansion of the paper presented in the previous chapter. Thus, we intend to submit these new results as original

contributions in a full paper to be submitted the journal *IEEE Transactions on Software Engineering*, the leading international journal in our field.

6.1 Introduction

A software architecture is composed of modular units, such as microservices and their relationships (19, 20, 82). Each architecture unit is traced to a set of source code elements (19, 20, 82). `toMicroservices` produces a set of microservice architectures and the traces of each microservice to its source code counterparts. Moreover, we evaluated whether developers would eventually adopt each microservice candidate. Our approach provides alternative microservice architectures, each composed of multiple microservice candidates. Thus, our previous studies have not assessed if those microservice architecture, as a whole, would help developers in making architectural decisions. We cover this gap in this chapter.

`toMicroservices` generates a Pareto set, where each individual (solution) in this set is a microservice architecture. Each microservice architecture is composed of microservice candidates, each realized by several methods in the source code. As presented in Chapter 4, each architecture is represented as a graph microservices, and each microservice, in turn, is represented as a graph of methods.

The study of this chapter consists of a focus group (83) conducted to evaluate the architectural solutions provided by `toMicroservices`. The choice of the focus group was made since architectural decisions are usually performed together within the system's organization. The focus group allows to assess `toMicroservices` in an industrial system, where multiple perceptions are discussed and decisions are made by a group of people.

Moreover, refinement operators were built to be used by an specialist in the system under analysis. The operators allow to better explore different developers' profiles and knowledge about the system under analysis. The operators are based on results of the qualitative study presented in Chapter 5, where the collected data supported the creation of two operators. Among them, the frozen operator allows `toMicroservices` users to select methods that will not be moved between microservice candidates in the future generations of offspring by the genetic algorithm. The additional operators allow moving a single method or a set of methods between microservice candidates.

The study introduced in Chapter 5 analyzed the adoptability of microservice candidates generated by `toMicroservices` in an industrial legacy system. Besides, the traceability to the associated code of the legacy system was also

investigated. Since our initial goal was to understand the preference of developers for the characteristics of isolated microservices, in that previous study only some parts of the whole architecture were analyzed.

To complement the aforementioned study, we conducted a focus group study to observe possible global architectural decisions within the context of the same industrial legacy system. A focus group study is an research technique that collects data through group interaction on a topic determined by the researcher (84). The focus group method is suited to obtain an initial evaluation of solutions, collect lessons learned, recommendations, practitioners' feedback on research questions, and identify potential problems (83).

Our results indicate that developers needed a few modifications to refine the microservice architecture discussed by each group of developers. The developers chosen to merge some microservice pairs and moved a few proportion of methods between microservice candidates. The result also suggests that the visual representation was considered very useful by developers than: (i) each of the actual relationships (dependencies) between microservice candidates, and (ii) the traceability of architectural elements to the source code counterparts.

Section 6.2 introduces the refinement operations to be executed during the optimization by `toMicroservices`. Section 6.3 presents the study design, while Section 6.4 shows the results and answers to the research questions. Finally, Section 6.5 introduces threats to validity and Section 6.6 presents the conclusions.

6.2 Refinement Operators

In the study of Chapter 5, we collected decisions that developers performed in the analysis of adoptability in microservice candidates generated by `toMicroservices`. These decisions include moving several methods or even removing a single method. For example, a participant stated: *"There are ten methods that should be in another microservice"*. This quote suggests that in the real case, the developer would move the methods to another microservice. Furthermore, developers responded that they would adopt 50% of the microservice candidate that were presented.

Based on the aforementioned qualitative analysis of common developers' decisions, we created two refinement operators that can be applied during `toMicroservices` re-execution. The refinement operators and their decisions are:

– **Move operator** changes block of methods to another microservice, where this block should contain at least one method.

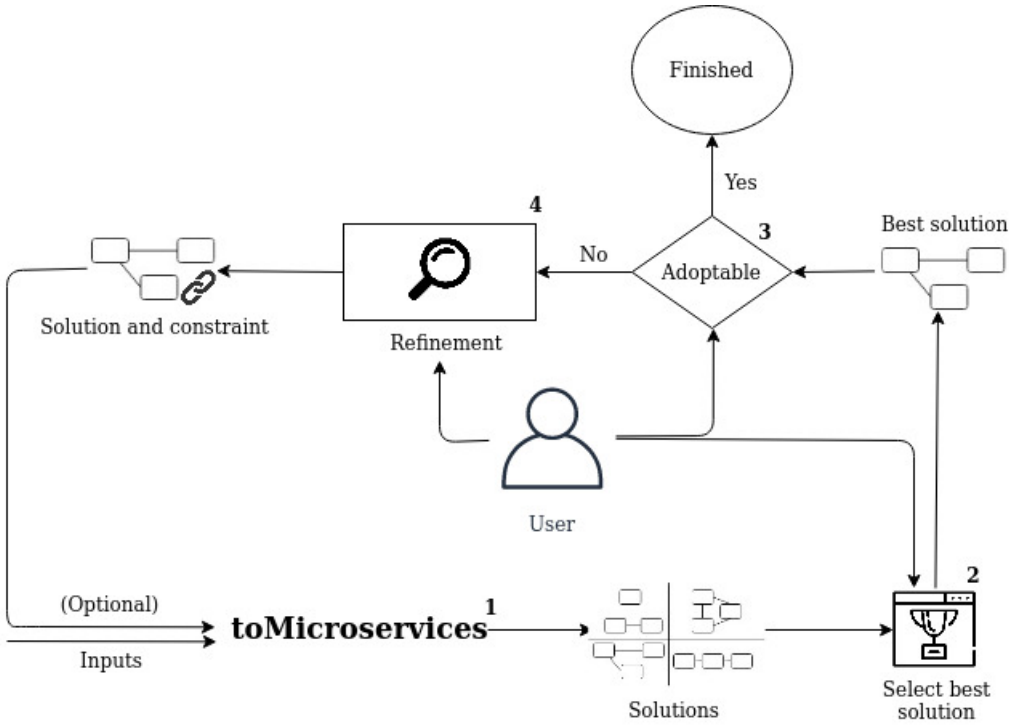


Figure 6.1: `toMicroservices` re-execution with refinement operators

– **Frozen operator** prevents that selected methods to be moved to the genetic operators during in the next offspring generation.

Figure 6.1 presents the re-execution of `toMicroservices` and the steps in which the user is required. First, `toMicroserives` is initiated by the same inputs presented in Section 4.2: the legacy source code, including indicators of code elements that will not be parsed, (ii) executions of the legacy systems, (iii) a list of features related to each execution of the legacy system, (iv) regular expressions to each feature, and (v) the number of desired microservice candidates. In Figure 6.1 this initiation is represented in the first step. After that, the `toMicroservices` user selects the best solution in the Pareto front as the second step. Several ranking strategies can be applied to select a solution in the Pareto front (64, 85, 86, 87). Among them, the selection can be guided by more valuable criteria or the best Euclidean distance from an ideal solution.

The third step consists of the user's assessment of the selected architecture (solution). The user can accept the solution and finish the re-executions; otherwise, the fourth step will request that the user to apply the refinement operators in the accepted solution. The operators allow better exploring the developer' profile and knowledge about the system under analysis. Among them, move operator only changes block of methods, while frozen operator imposes constraints on the next execution of `toMicroservices`. These constraints impose that mutation operations and the crossover operator are executed in the

next search process without being applied to frozen methods. After the fourth step, the modified solution and their constraints to the next search process is provided to the `toMicroservices` approach in addition to the same inputs previous given. Thus, a new execution is started with the constraints. Besides, the constraints imposed by the frozen operator can be modified in the next steps of refinement during the re-executions in the third step.

6.3 Study Design

This section introduces the study research question, subjects selection, and study execution that we designed to investigate the `toMicroservices` approach.

6.3.1 Research Questions

We intend to answer the following research questions to evaluate and improve future versions of `toMicroservices`.

RQ_1. *Do the practitioners judge the solutions found by `toMicroservices` adoptable to be implemented in practice?*
`toMicroservices` is an automated approach to identify microservices by relying on a legacy system and being built from an overall useful set of criteria based on empirical evidence. For a comprehensive evaluation of the use of such criteria, a post-build evaluation may guarantee the claim of practical application with deductive empirical evidence and even provide new indications to be added to the approach. Our previous study (Chapter 5) evaluated individual microservice candidates under the point of view of practitioners in a industrial legacy system. However, a complete solution (a microservice architecture) was not evaluated. Thus, **RQ1** intends to evaluate `toMicroservices` solutions in the view of its adoption in practical cases. We designed a focus group study that involved practitioners with experience in the system under analysis. This allowed us to gather results to understand the adoptability of a generated solution, and consequently how to potentially improve `toMicroservices` if needed.

RQ_2. *What is the effort to modify the microservice architectures generated by `toMicroservices`?*
Solutions generated by `toMicroservices` may naturally contain imperfections or even not satisfying a specific developer as previously investigated (Chapter 5). These imperfections, if not major, in the generated architectures may be fix without a considerable effort. However, the level of required effort is un-

known. Such a level of effort on fixing solutions generated by `toMicroservices` can be measured by the number of moved methods between microservice candidates, number of merges of two or more microservice candidates, or number of decompositions of one microservice in two or more. Therefore, the **RQ2** goal is to evaluate the effort to change a generated solution to better satisfy the developer's goal.

RQ_3. *Which are the most useful artifacts provided by `toMicroservices` to support developers on adopting a generated architecture?*
`toMicroservices` presents three architectural artifacts to the user: (i) a visual representation, (ii) a set of relationships between microservice candidates, and (iii) the set of traces from the source code to each microservice candidate (and vice-versa). These architectural artifacts are mentioned as useful by other authors (19, 20). The relative importance of each artifact is unknown in solutions generated by `toMicroservices` as complete solutions have not been evaluated yet. In addition, previous studies present gaps in providing an understanding of the useful artifacts to present the microservice architecture (27). This research question intends to evaluate the usefulness of each artifact for the microservice architecture generated by `toMicroservices` at a design stage.

RQ_4. *Does the focus group method support practitioners to make decisions related to the selected microservice architecture?*
As previously mentioned, focus group is a method of study commonly adopted in software engineering and social sciences, usually to explore the potential of groups and their reasoning. Architectural decisions influence further decisions made along the next phases of the development cycle (e.g, implementation and testing). In this context, the discussion of complementary or contradictory developers' viewpoints may contribute to formulate well-informed key decisions. Therefore, we intend to understand with the $\mathbf{RQ}_4$ whether focus groups was useful to assist in the process of evaluating an architecture generated by `toMicroservices` in practice.

6.3.2 Subject Selection

In total, seven developers participated in the study. The developers have years of experience with the industrial legacy system used in this study and they have participated in the qualitative study presented in Chapter 5. They were divided into two groups. The division choice was performed by their profile answers collected in the previous qualitative study (Chapter 5). We follow the guidelines of focus group studies that recommend clustering the groups by similarities (83).

Four developers presented a profile with a *fine-grained* microservice perspective since they positively evaluated microservice candidates that modularized single subfeatures with a very few amount of methods. The other three developers have a *coarse-grained* microservice profile, where several subfeatures or a complete feature in a microservice was indicated as adequate. The division by profile also helped to better configure `toMicroservices` to the particular needs or perspectives from those two groups. The *fine-grained* group received a solution where `toMicroservices` was configured to generate ten microservices. The *coarse-grained* group received a solution with five microservices. The choice of the microservices number was made based on the number of subfeatures previously recognized by the developers, and the grouping of subfeatures observed as adequate by the developers per microservice candidates. These data were collected in the interview of the previous study (Chapter 5), where generated microservices were individually analyzed.

In addition to the seven developers, an additional developer was selected to apply the refinement operators before the focus group. This developer has two years of experience in the maintenance activities of the system under analysis, which include the features analyzed. Thus, he was not included in the focus group study.

6.3.3 Study Execution

The additional developer used features and subfeatures recognized in the previous qualitative study (Chapter 5) to guide the refinement applications. He chose either to freeze a method related to a subfeature previously recognized or move methods that did not relate to the frozen subfeature.

The additional developer performed three complete refinement cycles as shown in Figure 6.1, where refinement operators were applied. In the total, three hours per group were spent on activities of: (i) a manual analysis of the generated solution, and (ii) a comparison of feature modularizations indicated as positive in the analysis of individual microservice candidates (Chapter 5). The solution with the best Euclidean distance was chosen in each cycle. The ideal solution was simulated with the best criteria found per each fitness function, that is, creating a simulated point with the best fitness function found in the Pareto set. The simulated point was used as a reference point to compute the Euclidean distance from the ideal solution.

The artifacts given to developers during the focus group execution were: (i) visual representation of the microservice architecture, (ii) the details of the relationships between methods pertaining to different microservice

candidates, where each method call is presented as a relationship, and (iii) the traceability to the legacy code at the level of methods. The visual representation contained the microservice candidates and their relationships. The relationship is directional, and the weight in each relationship is the coupling between the microservice candidates, in terms of the number of static calls between methods in a microservice candidate to methods in another microservice candidate.

During the execution of the focus group, the interview of each group was performed at a different moment. The developers were first introduced to the visual representation of the generated microservice architecture. The moderator of the focus group explained how the relationships presented in the visual representation between each candidate microservice were computed. After that, each microservice candidate has the main responsibilities and subfeature described. Moreover, the moderator stated that developers could request in any moment details of the relationships between microservice candidates, the list of methods in each microservice candidate, or the source code of each method. The relationships between microservice candidates include the methods that call or are called from other microservice candidates. That is, they determine the provided and required interfaces of each microservice candidate.

After the previous explanation, the moderator required developers to evaluate the architecture presented. During the discussions, the moderator required that each decision was justified. The justifications included the refactoring operations required to modify the architecture, information based on their experience with the legacy code, or reasons for accepting or rejecting candidate microservices and/or their relationships.

At the end of the focus group execution, the developers were questioned about the adoptability of the microservice architecture generated by `toMicroservices`. They were also questioned about the usefulness of (i) the visual representation, (ii) the relationships and interfaces, (iii) microservice-to-code traces, and (iv) the focus group method. Finally, developers were inquired about the adoptability of the modified microservice architectures. The questions aforementioned were responded on a five-point Likert scale, where the categories from the highest level to the lowest are: very useful, useful, moderately useful, little useful, useless. Regarding adoptability questions, the categories were: clearly adoptable, adoptable, moderately adoptable, little adoptability, not adoptable at all. Moreover, the focus group execution lasted for one hour.

Table 6.1: Questions conducted after the focus group execution, and their median

Inquired Questions	Fine-grained	Coarse-grained
Adoptability of initial microservice architecture	3	1
Usefulness of Visual Representation	4.5	3
Usefulness of Relations and interfaces	3	1
Usefulness of Focus Group	5	4
Adoptability of final microservice architecture	4	4

6.4 Results and Analysis

The developers were inquired whether the generated architecture by `toMicroservices` were adoptable in the industrial system. Table 6.1 shows the result in terms of the median of a five-point Likert-scale. The developers in the *fine-grained* group indicated a moderate adoptability in the initial solution. The developers in the *coarse-grained* group pointed that the initial architecture was not adoptable.

Developers were also inquired about the adoptability of the modified solution during the focus group execution. The median suggests the adoptability of the solution to both groups. To answer **RQ1**, the analysis of both presented questions suggested that the developers would not adopt the solution generated by `toMicroservices`. However, with modifications such as those performed in the focus group study, the solution would be adopted in the real system.

The moderator of the focus group collected the modifications suggested during the focus group execution. The goal was understanding which modifications were carried out to make the generated solutions adoptable in the industrial system. Regarding methods in each microservice candidate, the developers in the *fine-grained* group moved less than 1% of the methods to other microservice candidates. The methods moved are related to the authentication feature. As previously discussed in Chapter 5, modularizing crosscutting features, such as authentication, into microservices is a challenge. In the *coarse-grained* group, no method was moved. The additional modifications performed by both groups were limited to merge pairs of microservice candidates.

In response to **RQ2**, the modifications being predominantly the merge of microservices pairs leads to a low effort to modify the generated microservice architecture. In summary, this fact shows the suitability of the microservice candidates and their related code, motivated by the addition of interactivity with a specialist of the system under analysis. Regarding the ability to recognize features and subfeatures, all developers were able to recognize them through visual representation and the source code presented. In addition, other subfeatures were not indicated by developers during the analysis of

the architectures, or even, subfeatures as poorly modularized through the microservice candidates.

Table 6.1 also indicates that the visual representation of the solution was accepted as more useful than observing the relationships between method calls from different microservices. In both groups, developers spent more time discussing elements related to visual representation than all the other artifacts presented. The developers also stated that relationships were more useful than the source code during the focus group as also seen in Table 6.1. Regarding the **RQ3**, the visual representation of the microservice architecture was considered as the most useful artifact.

In the visual representations, the weights provided on the directed edges were the number of static calls between methods of different microservice candidates. The developers indicated that this number is difficult to interpret. That is, they cannot directly understand what this value in relation to the beneficial or harmful coupling involving the microservice candidates.

However, the weight presented was used in a comparative and useful way during the developers' arguments. The weights were compared relatively to each other. Both developers groups used the weights presented in the visual representation to make relative comparisons involving the candidates and to make decisions and justify them during the focus group. The comparison between the different weights was useful because the weights indicated elements with extremely strong coupling, which they considered unacceptable.

The *fine-grained* group argued then the weight less than 10 between microservice candidates were acceptable. In the microservice candidate that modularized the project file, the developers accepted a coupling of more than 100 since it was possibly indicating the satisfaction of other desirable criteria, such as reuse and cohesion. The *coarse-grained* group was much more rigorous than the *fine-grained* group since they tend to avoid strong (harmful) coupling between microservices. They argued that other forms of modularization could be adopted internally to each microservice when they discussed criteria like feature modularization.

The developers in the *fine-grained* group considered merging all microservice candidates related to project subfeatures in a single microservice. This was motivated by the high density of relations shown in the visual representation and confirmed by the relationship analysis. However, the high reuse and cohesion were decisive for the maintenance of the microservice that modularizes the project file subfeature. The other two microservices related to the project have been joined due to the high coupling between the pair of microservice candidates. Thus, they merged permission and metadata subfeatures in a sin-

gle microservice. In the *coarse-grained* group, the lack of weight between the resulting microservices was one of the arguments to maintain the modification of the proposed solution.

Finally, the developers in both groups were inquired about the focus group method and their usefulness to analyze and modify a proposed microservice architecture relied on a legacy system. The *fine-grained* group considered it a very useful method, while the *coarse-grained* group considered it useful. In response to **RQ4**, these responses suggest that focus groups are a pertinent method to be applied by practitioners in evaluations of architectural decisions.

6.5 Threats To Validity

The threats in the execution of the focus group were mitigated to a pilot execution with a group of two developers with less than 1 year in the maintenance on the system under analysis. The developers were inquired about a microservice architecture generated by `toMicroservices`. Moreover, a research specialist in empirical studies also reviews the artifacts. In both steps, artifacts problems were identified and corrected the: (i) refinement of visual representation to better the position the edges and their weight and (ii) the summary creation about the subfeatures in each microservice candidate.

The developers were questioned at a design stage in the process of migrating to a microservice architecture where implementation tasks on such microservice candidates are not yet being performed. That is, microservices are not actually being extracted during the focus group execution.

Focus group study guidelines in empirical software engineering suggest that groups should be selected relied on similarities (83). We dived the group relied on the granularity profile observed in a previous study. We divided the group to guide meetings more focused on the criteria and avoiding subjective discussions points to the developers' profile.

6.6 Concluding Remarks

We adopted a focus group method to evaluate `toMicroservices`. The focus group study was performed with an industrial legacy system under the process of migrating to microservice architecture and its experienced developers. Lessons learned were collected from our observations and results. In this study, we observed that `toMicroservices` is able to generate an adoptable solution after some merge operations.

The results also suggest the importance of visual artifacts that are generated by automated tools in conception phases. Otherwise, developers struggle to have a global picture of the future microservice architecture to be derived from the refactoring of the legacy system. In fact, developers indicated visual representations as more useful than other artifacts provided, contradicting our initial expectations. Developers indeed used more the visual representations to discuss and support their decisions during the focus group.

Furthermore, the focus group method itself has shown to be powerful method to discuss possible microservice architectures (and, thus, properly suppor the evalution of `toMicroservices`), extract complementary knowledge from the developers of the legacy system, and consolidate a single accepted architecture by the developers groups.

7
Conclusions

This dissertation focused on designing and implementing an automated approach to identify microservice candidates based on information extracted from the legacy code. Our approach was called `toMicroservices` and uses methods of search-based software engineering. `toMicroservices` adopts five criteria, and they are optimized by a genetic algorithm able to handle with many-objectives. This is quite different from the other existing automated approaches, which only use one or two criteria. The mere use of coupling and cohesion does not capture important influential criteria that are more specific to microservice architectures.

We conducted empirical studies to support the design and improvement of `toMicroservices`. We cataloged a set of criteria commonly found in empirical studies that report the process of migrating existing systems to a microservice architecture. Our evaluation of these criteria under the point of view of experienced practitioners allowed us to build `toMicroservices` avoiding the oversimplification of state-of-the-art automated approaches. Besides, the results also indicated a limitation in the used tools to measure each individual criterion. In a general sense, the experienced practitioners also indicated they consider existing tools limited to support migrations to microservice architectures.

Moreover, our empirical studies also allow a deeper understanding of the importance and presence of variability in legacy systems under the process of migrating. Half of the participants answered that they reasoning about variability as they progressed in their migration processes. Several mechanisms were used to implement variability in the legacy system before the migration, where the most common was the use of control version systems. After the migration process, the interview with developers suggests an increase in demands that lead to implementing variability in the migrated microservice architecture. Moreover, we cataloged in the interview three patterns used to implement variability in the migrated microservice architecture.

`toMicroservices` was evaluated and improved along a case if an industrial legacy system, which is undergoing a process of partially migrating it to a microservice architecture. In this legacy system, `toMicroservices` was able

to identify adoptable microservice candidates as evaluated by their developers. Regarding the criteria, feature modularization was the most used criterion to evaluate each microservice candidate. Moreover, the developers were able to recognize features modularized in microservice candidates. Finally, our study in the industrial legacy system, in which several developers were involved, indicated the importance of an interactive automated approach to identify microservice candidates in a step-wise fashion.

Therefore, a model for re-executing `toMicroservices` was created, favoring the interaction with the developers and allowing the automated application of refinements on the solutions. Thus, this interactive strategy allows a developer to incorporate his knowledge about the legacy system and to accommodate his preferences along the decomposition of the microservices. This also enable developers to follow either *fine-grained*, *coarse-grained* or hybrid decompositions of microservices. Our empirical evaluation suggests a low effort to modify solutions generated by `toMicroservices` after the addition of such an interactive process.

Our research outcomes lead to the at least two practical implications:

- The synergistic use of optimization, feature location, static analysis and dynamic analysis, as implemented by `toMicroservices`, has the potential of reducing effort of developers, supporting architectural reasoning, and making informed decisions along the identification of microservices from legacy code.
- There is room for the improvement of microservice identification approaches by making them more interactive. The creation of operators to support interactions with the developers relied on observation from our case study. We cannot generalize the completeness and usefulness of these set of operators to other projects. In any case, enabling those operations have shown to be promising in reducing effort and finding solutions that actually reflect architects' knowledge along the re-executions of the automated approach.

In future work, we plan to experiment `toMicroservices` in an additional industrial project and in an open-source software project, which are also undergoing partial or full migrations to a microservice architecture. These studies will enable us to improve the external generalization of our findings, as well as capture new opportunities for making `toMicroservices` more practical and robust. Moreover, we plan to improve the measurements of the five criteria adopted in `toMicroservices` as well as incorporate other criteria that may

capture challenging requirements, such as scalability and other non-functional requirements.

Finally, the empirical studies were reported in published papers or are (or being) submitted to conferences or journals. These papers are presented in Table 7.1 to facility future references. Table 7.2 also show papers that do not result from the core research of this dissertation, but are results of fruitful research collaborations during the Masters course.

Table 7.1: Papers and chapter book that resulted from this dissertation

Paper	**Chapter**	**Status**
Luiz Carvalho, Alessandro Garcia, Wesley K. G. Assunção, Rafael de Mello, Maria Julia de Lima. Analysis of the criteria adopted in industry to extract microservices. In Proceedings of the joint 7th International Workshop on Conducting Empirical Studies in Industry and 6th International Workshop on Software Engineering Research and Industrial Practice, CESSER-IP'19, ICSE 2019, p. 22–29, 2019.	2	Published
Luiz Carvalho, Alessandro Garcia, Wesley K. G. Assunção, Rodrigo Bonifácio, Leonardo P. Tizzei, Thelma Elita Colanzi. Extraction of configurable and reusable microservices from legacy systems: An exploratory study. In Proceedings of the 12rd International Systems and Software Product Line Conference - Volume A, SPLC'19, p. 26–31, 2019.	3	Published
Luiz Carvalho, Alessandro Garcia, Wesley K. G. Assunção, Thelma Elita Colanzi, Rodrigo Bonifácio, Leonardo P. Tizzei, Rafael de Mello, Renato Cerqueira, Márcio Ribeiro. Re-engineering Legacy Systems as Microservices: An industrial survey of Criteria for Identifying Microservices. In Handbook of Re-Engineering Software Intensive Systems into Software Product Lines, Springer.	2-3	Invited chapter
Luiz Carvalho, Alessandro Garcia, Thelma Elita Colanzi, Wesley K. G. Assunção, Maria Julia Lima, Baldoino Fonseca, Márcio Ribeiro, Carlos Lucena. Search-Based Many-Criteria Identification of Microservices from Legacy Systems. Poster, In the Genetic and Evolutionary Computation Conference, GECCO'20.	4-5	Accepted
Luiz Carvalho, Alessandro Garcia, Thelma Elita Colanzi, Wesley K. G. Assunção, Juliana Alves Pereira, Baldoino Fonseca, Márcio Ribeiro, Maria Julia Lima, Carlos Lucena. On the Performance and Adoption of Search-Based Microservice Identification with `toMicroservices`. In the Conference on Software Maintenance and Evolution, ICSME'20.	4-5	To submit in May 2020
Luiz Carvalho, Alessandro Garcia, Wesley K. G. Assunção, Thelma Elita Colanzi, Baldoino Fonseca, Márcio Ribeiro, Carlos Lucena. Many-objective Search-based Identification of Microservice Architectures: A Qualitative Study in the Industry. In IEEE Transactions on Software Engineering.	4-5-6	To submit

Table 7.2: Other papers resulting from the masters

Paper	**Status**
Towards a Catalog of Java Dependency Injection Anti-Patterns. Rodrigo Laigner, Marcos Kalinowski, **Luiz Carvalho**, Diogo Mendonça, Alessandro Garcia. In proceedings of the 32th Brazilian Symposium on Software Engineering, p. 104-113, 2019.	Published
On the density and diversity of degradation symptoms in refactored classes: A multi-case study. Willian Oizumi, Leonardo Sousa, Anderson Oliveira, **Luiz Carvalho**, Alessandro Garcia, Thelma Colanzi, Roberto Oliveira. In proceedings 30th International Symposium on Software Reliability Engineering, p. 346-357, 2019	Published

Bibliography

[1] NEWMAN, S.. **Building Microservices**. O'Reilly Media, 1st edition, 2015.

[2] LEWIS, J.; FOWLER, M.. **Microservices.**, 2014.

[3] TIZZEI, L. P.; NERY, M.; SEGURA, V. C. V. B. ; CERQUEIRA, R. F. G.. **Using microservices and software product line engineering to support reuse of evolving multi-tenant saas**. In: INTERNATIONAL SYSTEMS AND SOFTWARE PRODUCT LINE CONFERENCE, p. 205–214, New York, NY, USA, 2017. ACM.

[4] FOWLER, S.. **Production-Ready Microservices**. O'Reilly Media, 1st edition, 2016.

[5] WATSON, C.; EMMONS, S. ; GREGG, B.. **A microscope on microservices**, 2015.

[6] LUZ, W.; AGILAR, E.; DE OLIVEIRA, M. C.; DE MELO, C. E. R.; PINTO, G. ; BONIFÁCIO, R.. **An experience report on the adoption of microservices in three brazilian government institutions**. In: BRAZILIAN SYMPOSIUM ON SOFTWARE ENGINEERING, p. 32–41, New York, NY, USA, 2018. ACM.

[7] GOUIGOUX, J.; TAMZALIT, D.. **From monolith to microservices: Lessons learned on an industrial migration to a web oriented architecture**. In: INTERNATIONAL CONFERENCE ON SOFTWARE ARCHITECTURE WORKSHOPS, p. 62–65, 2017.

[8] BUCCHIARONE, A.; DRAGONI, N.; DUSTDAR, S.; LARSEN, S. T. ; MAZZARA, M.. **From monolithic to microservices: An experience report from the banking domain**. IEEE Software, 35(3):50–55, 2018.

[9] BISBAL, J.; LAWLESS, D.; WU, B. ; GRIMSON, J.. **Legacy information systems: Issues and directions**. IEEE Software, 16(5):103–111, Sept. 1999.

[10] RANSOM, J.; SOMERVILLE, I. ; WARREN, I.. **A method for assessing legacy systems for evolution**. In: EUROMICRO CONFERENCE ON

SOFTWARE MAINTENANCE AND REENGINEERING, p. 128–134, March 1998.

[11] TAIBI, D.; LENARDUZZI, V. ; PAHL, C.. **Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation**. IEEE Cloud Computing, 4(5):22–32, 2017.

[12] FRANCESCO, P. D.; LAGO, P. ; MALAVOLTA, I.. **Migrating towards microservice architectures: An industrial survey**. In: INTERNATIONAL CONFERENCE ON SOFTWARE ARCHITECTURE, p. 29–2909, 2018.

[13] MAZLAMI, G.; CITO, J. ; LEITNER, P.. **Extraction of microservices from monolithic software architectures**. In: INTERNATIONAL CONFERENCE ON WEB SERVICES, p. 524–531, 2017.

[14] ESCOBAR, D.; CÁRDENAS, D.; AMARILLO, R.; CASTRO, E.; GARCÉS, K.; PARRA, C. ; CASALLAS, R.. **Towards the understanding and evolution of monolithic applications as microservices**. In: LATIN AMERICAN COMPUTING CONFERENCE, p. 1–11, 2016.

[15] JIN, W.; LIU, T.; ZHENG, Q.; CUI, D. ; CAI, Y.. **Functionality-oriented microservice extraction based on execution trace clustering**. In: INTERNATIONAL CONFERENCE ON WEB SERVICES (ICWS), p. 211–218, 2018.

[16] JIN, W.; LIU, T.; CAI, Y.; KAZMAN, R.; MO, R. ; ZHENG, Q.. **Service candidate identification from monolithic systems based on execution traces**. IEEE Transactions on Software Engineering, p. 1–1, 2019.

[17] TAIBI, D.; LENARDUZZI, V. ; PAHL, C.. **Microservices Anti-patterns: A Taxonomy**, p. 111–128. Springer International Publishing, Cham, 2020.

[18] CANDELA, I.; BAVOTA, G.; RUSSO, B. ; OLIVETO, R.. **Using cohesion and coupling for software remodularization: Is it enough?** ACM Transactions on Software Engineering and Methodology, 25(3):24:1–24:28, June 2016.

[19] BASS, L.; CLEMENTS, P. ; KAZMAN, R.. **Software Architecture in Practice**. Addison-Wesley Professional, 3rd edition, 2012.

[20] ROZANSKI, N.; WOODS, E.. **Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives**. Addison-Wesley Professional, 2 edition, 2011.

[21] APEL, S.; BATORY, D.; KSTNER, C. ; SAAKE, G.. **Feature-Oriented Software Product Lines: Concepts and Implementation**. Springer Publishing Company, Incorporated, 2013.

[22] PARNAS, D. L.. **On the design and development of program families**. IEEE Transactions on Software Engineering, SE-2(1):1–9, March 1976.

[23] LIEBIG, J.; KÄSTNER, C. ; APEL, S.. **Analyzing the discipline of preprocessor annotations in 30 million lines of c code**. In: PROCEEDINGS OF THE TENTH INTERNATIONAL CONFERENCE ON ASPECT-ORIENTED SOFTWARE DEVELOPMENT, p. 191–202, New York, NY, USA, 2011. Association for Computing Machinery.

[24] MEDEIROS, F.; RIBEIRO, M.; GHEYI, R.; APEL, S.; KÄSTNER, C.; FERREIRA, B.; CARVALHO, L. ; FONSECA, B.. **Discipline matters: Refactoring of preprocessor directives in the #ifdef hell**. IEEE Trans. Software Eng., 44(5):453–469, 2018.

[25] BERGER, T.; RUBLACK, R.; NAIR, D.; ATLEE, J. M.; BECKER, M.; CZARNECKI, K. ; WĄSOWSKI, A.. **A survey of variability modeling in industrial practice**. In: INTERNATIONAL WORKSHOP ON VARIABILITY MODELLING OF SOFTWARE-INTENSIVE SYSTEMS, VaMoS '13, New York, NY, USA, 2013. Association for Computing Machinery.

[26] MARTINEZ, J.; ASSUNÇÃO, W. K. G. ; ZIADI, T.. **Espla: A catalog of extractive spl adoption case studies**. In: INTERNATIONAL SYSTEMS AND SOFTWARE PRODUCT LINE CONFERENCE - VOLUME B, SPLC '17, p. 38–41, New York, NY, USA, 2017. Association for Computing Machinery.

[27] FRANCESCO, P. D.; LAGO, P. ; MALAVOLTA, I.. **Architecting with microservices: A systematic mapping study**. Journal of Systems and Software, 150:77 – 97, 2019.

[28] KNOCHE, H.; HASSELBRING, W.. **Using microservices for legacy software modernization**. IEEE Software, 35(3):44–49, May 2018.

[29] BALL, T.. **The concept of dynamic analysis**. In: 7TH EUROPEAN SOFTWARE ENGINEERING CONFERENCE HELD JOINTLY WITH THE 7TH ACM SIGSOFT INTERNATIONAL SYMPOSIUM ON FOUNDATIONS OF SOFTWARE ENGINEERING, ESEC/FSE-7, p. 216–234, Berlin, Heidelberg, 1999. Springer-Verlag.

[30] HARMAN, M.; JONES, B. F.. **Search-based software engineering**. Information and Software Technology, 43(14):833 – 839, 2001.

[31] HARMAN, M.; MANSOURI, S. A. ; ZHANG, Y.. **Search-based software engineering: Trends, techniques and applications**. ACM Computing Surveys, 45(1):11:1–11:61, Dec. 2012.

[32] DEB, K.; JAIN, H.. **An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints**. IEEE Transactions on Evolutionary Computation, 18(4):577–601, Aug 2014.

[33] ESKI, S.; BUZLUCA, F.. **An automatic extraction approach: Transition to microservices architecture from monolithic application**. In: INTERNATIONAL CONFERENCE ON AGILE SOFTWARE DEVELOPMENT: COMPANION, XP '18, p. 25:1–25:6, New York, NY, USA, 2018. ACM.

[34] ALSHUQAYRAN, N.; ALI, N. ; EVANS, R.. **A systematic mapping study in microservice architecture**. In: INTERNATIONAL CONFERENCE ON SERVICE-ORIENTED COMPUTING AND APPLICATIONS, p. 44–51, 2016.

[35] PAHL, C.; JAMSHIDI, P.. **Microservices: A systematic mapping study**. In: INTERNATIONAL CONFERENCE ON CLOUD COMPUTING AND SERVICES SCIENCE, p. 137–146, 2016.

[36] **ISO/IEC/IEEE 24765: 2017(E): ISO/IEC/IEEE International Standard - Systems and software engineering–Vocabulary**. IEEE, 2017.

[37] SCARBOROUGH, W.; ARNOLD, C. ; DAHAN, M.. **Case study: Microservice evolution and software lifecycle of the xsede user portal api**. In: CONFERENCE ON DIVERSITY, BIG DATA, AND SCIENCE AT SCALE, p. 47:1–47:5, New York, NY, USA, 2016. ACM.

[38] KNOCHE, H.. **Sustaining runtime performance while incrementally modernizing transactional monolithic software towards microservices**. ICPE '16, p. 121–124, New York, NY, USA, 2016. ACM.

[39] ELMASRI, R.; NAVATHE, S.. **Fundamentals of Database Systems**. Addison-Wesley Publishing Company, USA, 6th edition, 2010.

[40] CHIKOFSKY, E. J.; CROSS, J. H.. **Reverse engineering and design recovery: a taxonomy**. IEEE Software, 7(1):13–17, 1990.

[41] LINAKER, J.; SULAMAN, S. M.; MAIANI DE MELLO, R.; HÖST, M. ; RUNESON, P.. **Guidelines for conducting surveys in software engineering**. 2015.

[42] DE MELLO, R. M.; TRAVASSOS, G. H.. **Surveys in software engineering: Identifying representative samples**. In: INTERNATIONAL SYMPOSIUM ON EMPIRICAL SOFTWARE ENGINEERING AND MEASUREMENT, p. 55. ACM, 2016.

[43] WOHLIN, C.. **Guidelines for snowballing in systematic literature studies and a replication in software engineering**. In: INTERNATIONAL CONFERENCE ON EVALUATION AND ASSESSMENT IN SOFTWARE ENGINEERING, p. 38:1–38:10, New York, NY, USA, 2014. ACM.

[44] TAHIR, A.; MACDONELL, S. G.. **A systematic mapping study on dynamic metrics and software quality**. In: INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE, p. 326–335, 2012.

[45] Broy, M.; Denert, E., editors. **Software Pioneers: Contributions to Software Engineering**. Springer-Verlag, Berlin, Heidelberg, 2002.

[46] ELMASRI, R.; NAVATHE, S. B.. **Fundamentals of Database Systems**. Pearson, 7th edition, 2015.

[47] BROOKS, F. P.. **The Mythical Man-Month (Anniversary Ed.)**. Addison-Wesley Longman Publishing Co., Inc., USA, 1995.

[48] CHONG, F.; CARRARO, G.. **Architecture strategies for catching the long tail**. MSDN Library, Microsoft Corporation, p. 9–10, 2006.

[49] VON RHEIN, A.; GREBHAHN, A.; APEL, S.; SIEGMUND, N.; BEYER, D. ; BERGER, T.. **Presence-condition simplification in highly configurable systems**. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING - VOLUME 1, ICSE '15, p. 178–188, Piscataway, NJ, USA, 2015. IEEE Press.

[50] MIETZNER, R.; METZGER, A.; LEYMANN, F. ; POHL, K.. **Variability modeling to support customization and deployment of multi-tenant-aware software as a service applications**. In: WORKSHOP ON PRINCIPLES OF ENGINEERING SERVICE ORIENTED SYSTEMS, PESOS '09, p. 18–25, Washington, DC, USA, 2009. IEEE Computer Society.

[51] BACHMANN, F.; CLEMENTS, P. C.. **Variability in software product lines**. Technical Report CMU/SEI-2005-TR-012, Carnegie Mellon University - Software Engineering Institute, 2005.

[52] CAPILLA, R.; BOSCH, J. ; KANG, K.-C.. **Systems and Software Variability Management: Concepts, Tools and Experiences**. Springer Publishing Company, Incorporated, 2013.

[53] DUBINSKY, Y.; RUBIN, J.; BERGER, T.; DUSZYNSKI, S.; BECKER, M. ; CZARNECKI, K.. **An exploratory study of cloning in industrial software product lines**. In: European Conference on Software Maintenance and Reengineering, p. 25–34. IEEE, 2013.

[54] C. MARTIN, R.. **Agile Software Development, Principles, Patterns, and Practices**. Pearson, 1st edition, 2002.

[55] BASS, L.; WEBER, I. ; ZHU, L.. **DevOps: A software architect's perspective**. Addison-Wesley Professional, 2015.

[56] RAHMAN, M. T.; QUEREL, L.-P.; RIGBY, P. C. ; ADAMS, B.. **Feature toggles: Practitioner practices and a case study**. In: INTERNATIONAL CONFERENCE ON MINING SOFTWARE REPOSITORIES, MSR '16, p. 201–211, New York, NY, USA, 2016. ACM.

[57] MELLA, F. P.; MÁRQUEZ, G. ; ASTUDILLO, H.. **Migrating from monolithic architecture to microservices: A rapid review**. In: INTERNATIONAL CONFERENCE OF THE CHILEAN COMPUTER SCIENCE SOCIETY, 2019.

[58] NEWMAN, M. E. J.; GIRVAN, M.. **Finding and evaluating community structure in networks**. Physical Review, 69:026113, Feb 2004.

[59] CHATTERJEE, M.; DAS, S. K. ; TURGUT, D.. **Wca: A weighted clustering algorithm for mobile ad hoc networks**. Cluster Computing, 5(2):193–204, Apr 2002.

[60] RUNESON, P.; HÖST, M.. **Guidelines for conducting and reporting case study research in software engineering**. Empirical Software Engineering, 14(2):131, Dec 2008.

[61] DRAGONI, N.; GIALLORENZO, S.; LAFUENTE, A. L.; MAZZARA, M.; MONTESI, F.; MUSTAFIN, R. ; SAFINA, L.. **Microservices: Yesterday, Today, and Tomorrow**, p. 195–216. Springer International Publishing, Cham, 2017.

[62] ARCURI, A.. **Restful api automated test case generation with evomaster**. ACM Transactions on Software Engineering and Methodology, 28(1):3:1–3:37, Jan. 2019.

[63] COLANZI, T. E.. **Search based design of software product lines architectures**. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, p. 1507–1510, June 2012.

[64] ASSUNÇÃO, W. K. G.; COLANZI, T. E.; VERGILIO, S. R. ; POZO, A.. **A multi-objective optimization approach for the integration and test order problem**. Information Sciences, 267:119 – 139, 2014.

[65] HOLLAND, J. H.. **Adaptation in Natural and Artificial Systems**. University of Michigan Press, Ann Arbor, MI, 1975. second edition, 1992.

[66] DEB, K.; PRATAP, A.; AGARWAL, S. ; MEYARIVAN, T.. **A fast and elitist multiobjective genetic algorithm: Nsga-ii**. IEEE Transactions on Evolutionary Computation, 6(2):182–197, April 2002.

[67] CARVALHO, L.; GARCIA, A.; ASSUNÇÃO, W. K. G.; DE MELLO, R. ; DE LIMA, M. J.. **Analysis of the criteria adopted in industry to extract microservices**. In: JOINT 7TH INTERNATIONAL WORKSHOP ON CONDUCTING EMPIRICAL STUDIES IN INDUSTRY AND 6TH INTERNATIONAL WORKSHOP ON SOFTWARE ENGINEERING RESEARCH AND INDUSTRIAL PRACTICE, CESSER-IP '19, p. 22–29, Piscataway, NJ, USA, 2019. IEEE Press.

[68] MITCHELL, B. S.; MANCORIDIS, S.. **On the automatic modularization of software systems using the bunch tool**. IEEE Transactions on Software Engineering, 32(3):193–208, March 2006.

[69] CHIDAMBER, S. R.; KEMERER, C. F.. **A metrics suite for object oriented design**. IEEE Transactions on Software Engineering, 20(6):476–493, June 1994.

[70] CAPILLA, R.; GALLINA, B.; CETINA, C. ; FAVARO, J.. **Opportunities for software reuse in an uncertain world: From past to emerging trends**. Journal of Software: Evolution and Process, 31(8):1–22, August 2019.

[71] **Refactoring: Improving the Design of Existing Code**. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[72] DURILLO, J. J.; NEBRO, A. J.. **jmetal: A java framework for multi-objective optimization**. Advances in Engineering Software, 42(10):760–771, 2011.

[73] DEB, K.; JAIN, H.. **An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints**. IEEE Transactions on Evolutionary Computation, 18(4):577–601, 2014.

[74] COLANZI, T. E.; ASSUNÇÃO, W. K. G.; FARAH, P. R.; VERGILIO, S. R. ; GUIZZO, G.. **A review of ten years of the symposium on search-based software engineering**. In: Nejati, S.; Gay, G., editors, SEARCH-BASED SOFTWARE ENGINEERING, p. 42–57, Cham, 2019. Springer International Publishing.

[75] ZITZLER, E.; THIELE, L.; LAUMANNS, M.; FONSECA, C. M. ; DA FONSECA, V. G.. **Performance assessment of multiobjective optimizers: An analysis and review**. IEEE Transactions on Evolutionary Computation, 7:117–132, 2003.

[76] FONSECA, C. M.; PAQUETE, L. ; LOPEZ-IBANEZ, M.. **An improved dimension-sweep algorithm for the hypervolume indicator**. In: IEEE INTERN. CONFERENCE ON EVOLUTIONARY COMPUTATION, p. 1157–1163, 2006.

[77] DEB, K.; PRATAP, A.; AGARWAL, S. ; MEYARIVAN, T.. **A fast and elitist multiobjective genetic algorithm: Nsga-ii**. IEEE Transactions on Evolutionary Computation, 6(2):182–197, 2002.

[78] BERGMANN, R.; LUDBROOK, J. ; SPOOREN, W. P. J. M.. **Different Outcomes of the Wilcoxon-Mann-Whitney Test from Different Statistics Packages**. The American Statistician, 54(1):72–77, 2000.

[79] VARGHA, A.; DELANEY, H.. **A critique and improvement of the cl common language effect size statistics of McGraw and Wong**. J. of Educational and Behavioral Statistics, 25(2):101–132, 2000.

[80] ARCURI, A.; BRIAND, L.. **A Hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering**. Software Testing, Verification and Reliability, 24(3):219–250, 2014.

[81] CARVALHO, L.; GARCIA, A.; ASSUNÇÃO, W. K. G.; BONIFÁCIO, R.; TIZZEI, L. P. ; COLANZI, T. E.. **Extraction of configurable and**

reusable microservices from legacy systems: An exploratory study. In: INTERNATIONAL SYSTEMS AND SOFTWARE PRODUCT LINE CONFERENCE - VOLUME A, SPLC '19, p. 26–31, New York, NY, USA, 2019. ACM.

[82] PERRY, D. E.; WOLF, A. L.. **Foundations for the study of software architecture**. SIGSOFT Software Engineering Notes, 17(4):40–52, Oct. 1992.

[83] KONTIO, J.; BRAGGE, J. ; LEHTOLA, L.. **The Focus Group Method as an Empirical Tool in Software Engineering**, p. 93–116. Springer London, London, 2008.

[84] MORGAN, D. L.. **Focus groups**. Annual Review of Sociology, 22(1):129–152, 1996.

[85] COCHRANE, J.; ZELENY, M.. **Multiple Criteria Decision Making**. University of South Carolina Press, Columbia, 1973.

[86] COELLO, C. A. C.. **Handling preferences in evolutionary multiobjective optimization: a survey**. In: PROCEEDINGS OF THE 2000 CONGRESS ON EVOLUTIONARY COMPUTATION. CEC00 (CAT. NO.00TH8512), volumen 1, p. 30–37 vol.1, 2000.

[87] CORNE, D. W.; KNOWLES, J. D.. **Techniques for highly multiobjective optimisation: Some nondominated points are better than others**. In: PROCEEDINGS OF THE 9TH ANNUAL CONFERENCE ON GENETIC AND EVOLUTIONARY COMPUTATION, GECCO '07, p. 773–780, New York, NY, USA, 2007. Association for Computing Machinery.

[88] FRASER, G.; ARCURI, A.. **Whole test suite generation**. IEEE Transactions on Software Engineering, 39(2):276 –291, feb. 2013.

[89] CORNELISSEN, B.; ZAIDMAN, A.; VAN DEURSEN, A.; MOONEN, L. ; KOSCHKE, R.. **A systematic survey of program comprehension through dynamic analysis**. IEEE Transactions on Software Engineering, 35(5):684–702, Sep. 2009.

[90] DIT, B.; REVELLE, M.; GETHERS, M. ; POSHYVANYK, D.. **Feature location in source code: a taxonomy and survey**. J. Softw. Evol. Process., 25(1):53–95, 2013.

[91] NAMIOT; SNEPS-SNEPPE, M.. **On micro-services architecture**. International Journal of Open Information Technologies, 2(9), 2014.

[92] SEAMAN, C. B.. **Qualitative methods in empirical studies of software engineering**. IEEE Transactions on Software Engineering, 25(4):557–572, July 1999.