On the Performance and Adoption of Search-Based Microservice Identification with toMicroservices

Luiz Carvalho*, Alessandro Garcia*, Thelma Elita Colanzi†, Wesley K. G. Assunção‡, Juliana Alves Pereira*, Baldoino Fonseca§, Márcio Ribeiro§, Maria Julia de Lima¶, Carlos Lucena*

*Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil. {lmcarvalho,afgarcia,jpereira,lucena}@inf.puc-rio.br

†State University of Maringá, Maringá, Brazil. thelma@din.uem.br

‡Federal University of Technology - Paraná, Toledo, Brazil. wesleyk@utfpr.edu.br

§Federal University of Alagoas, Maceió, Brazil. {baldoino,marcio}@ic.ufal.br

¶Tecgraf Institute, Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil. mjulia@tecgraf.puc-rio.br

***Abstract*—The expensive maintenance of legacy systems leads companies to migrate such systems to microservice architectures. This migration requires the identification of system's legacy parts to become microservices. However, the successful identification of microservices, which are promising to be adoptable in practice, requires the simultaneous satisfaction of many criteria, such as coupling, cohesion, reuse and communication overhead. Search-based microservice identification has been recently investigated to address this problem. However, state-of-the-art search-based approaches are limited as they only consider one or two criteria (namely cohesion and coupling), possibly not fulfilling the practical needs of developers. To overcome these limitations, we propose `toMicroservices`, a many-objective search-based approach that considers five criteria, the most cited by practitioners in recent studies. Our approach was evaluated in a real-life industrial legacy system undergoing a microservice migration process. The performance of `toMicroservices` was quantitatively compared to a baseline. We also gathered qualitative evidence based on developers' perceptions, who judged the adoptability of the recommended microservices. The results show that our approach is both: (i) very similar to the most recent proposed approach on optimizing the traditional criteria of coupling and cohesion, but (ii) much better when taking into account all the five criteria. Finally, most of the microservice candidates were considered adoptable by practitioners.**

***Index Terms*—microservice architecture, legacy systems, microservice identification, search-based software engineering**

I. Introduction

Legacy systems are commonly found in the industry and represent a long-term massive investment [1], [2]. Despite their business importance, these systems often depend on obsolete technologies [3], [4]. 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 [1], [2]. Many industries are increasingly modernizing legacy systems by extracting their features into *microservices* [3]–[8]. Nowadays this is a key strategy to increase the longevity of legacy systems, while offering new business opportunities to organizations [4], [9].

The identification of microservice candidates from legacy code is a complex, time-consuming, and error-prone task [3], [4], [10]–[13]. To reap the benefits of microservices, developers must reason about multiple criteria while identifying legacy features as microservice candidates. A criterion is an information in the legacy system used to support the decision-making process to transform (either partially or fully) certain features in microservices. Recent studies show that practitioners have to simultaneously consider five typical criteria – cohesion, coupling, feature modularization, reuse and communication overhead [12], [14] – along the decision-making process.

In fact, these five criteria are directly related to definition of what is a microservice. *Microservices* are *small* and *autonomous* services that work together by using lightweight network protocols [15]. The notion of *small* refers to the need of producing cohesive, fine-grained microservices [16] with each modularizing a single feature [17]. The notion of *autonomous* implies that each microservice should be highly decoupled, reusable, whereas reducing the risk of unacceptable network overhead [9], [18].

Thus, project teams have to consider many criteria along microservice identification. This in turn requires the identification and analysis of a wide range of possible microservice candidates, becoming a complex and error-prone task without proper automated support [10], [12]. In fact, there are many cases reported where the microservification of certain legacy features were considered a failure in the long run [19].

However, search-based approaches [20]–[24] are limited as they only consider one or two criteria (namely cohesion and coupling), possibly not fulfilling the practical needs of developers. Even worse, there is no explicit knowledge about the performance of these approaches in an industrial case study. The performance in properly finding solutions with optimized criteria of such approaches is purely assessed quantitatively and in the context of: (i) simple, academic, already well-modularized systems [22], or (ii) an open-source project where they do not have any access to the actual developers [20], [21], [23]. Thus, the performance of existing search-based approaches in an industrial legacy system is unknown.

Moreover, 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 where many features are tangled to each other. In other words, methods in a legacy module are often implementing intertwined features, thereby complicating the identification of microservice.

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 IV). Differently from existing approaches, `toMicroservices` is designed to identifying microservices scattered in methods, rather than simply modules (files). Our quantitative and qualitative evaluation of `toMicroservices` (Section V) is the first to be performed in the context of an industrial *in-situ* case study (Section III). We analyzed the performance of our approach against a baseline approach and the potential adoption of the microservices candidates by the developers of the target system. The baseline only considers the coupling and cohesion criteria.

The main findings of this work are (Section VI): *(i)* `toMicroservices` outperforms recent state-of-the-art approaches (our baselines), *(ii)* it is able to find a set of microservices that simultaneously satisfy well all the five criteria; 53% of which were considered as adoptable by the developers, *(iii)* it is able to successfully identify microservices that modularize features otherwise tangled and scattered through many modules of the legacy code, and *(iv)* some opportunities to evolve the legacy system were easily identified by developers. The complementary material is available online[1].

II. Background and Related Work

In this section, we introduce the basic concepts of microservices and motivate the use of search-based techniques.

A. Microservices Identification in Legacy Code

Monolithic software systems tend to accumulate design problems over time and, at some point, can become unmaintainable or even discontinued [25]–[28]. Such design problems are hard to be identified [29]–[32] and refactored [33], [34] in in monoliths. Recent studies have reported several benefits of migrating legacy monolithic systems to a microservice architecture, such as improved maintainability and evolvability [4], [8], [10], [11], [35]. In this order, approaches to identify microservices has been proposed, where each identified microservice is a mapping to legacy code elements. Overall, the benefits of an automatic approach to identify microservices are apparent. They free the developer to go into the details of the whole legacy code-base manually and allow them to focus on specific well-modularized parts of the microservice candidates. However, the few existing approaches rely on one or two quality criteria [20]–[24]. Coupling is the most adopted criterion [20]–[23], and recently cohesion has been also used to provide better microservice candidates [23].

Yet, a recent exploratory study with practitioners has revealed that the efficiency of re-engineering legacy systems depends on several criteria beyond coupling and cohesion [12], [14] *e.g.*, the criterion network overhead is relevant given the architectural style of distributed components. In case of important criteria that are not included in an approach, its solutions can hardly align with the desired benefits of microservices. To overcome this limitation, we propose a microservice identification approach that relies on the optimization of five criteria.

B. Search-Based Software Engineering

In the problem of microservice identification, manually evaluating all possible microservice candidates from a legacy system is impractical, especially if many conflicting decision criteria are involved. Thus, this problem has been recently treated by search-based algorithms [20], [21], [23], [24]. Search-based software engineering (SBSE) is the field in which search-based optimization techniques are used to address software engineering problems [36]. In such problems, optimal or near-optimal solutions are sought in a large search space of candidate solutions, guided by an objective function that distinguishes between better and worse solutions [36]. SBSE has proved to be an applicable and successful field, with many studies across the software engineering life cycle, from requirements to maintenance and evolution [36]–[41]. In SBSE, there is also increasing evidence of industrial interest in many software-centric organizations including Facebook [42], IBM [43], and Microsoft [44].

Existing search-based microservice solutions apply evolutionary algorithms such as genetic algorithms [21], [23], to optimize some source code quality criteria extracted from execution traces, and thus generate a set of microservice candidates. These solutions commonly make use of one or two criteria [20], [21], [23], [24]. In contrast, we make use of five criteria. This implies that we turn the search-based microservice identification problem into a many-objective problem[2]. To the best of our knowledge, there is no effort to solve such SBSE problem based on many objectives to formally represent the several criteria involved in the microservices identification. This new perspective represents an important challenge in this field. Thus, our work has a comparison of performance, that is, how the criteria are properly optimized by `toMicroservices` and the baselines. The performance comparison considers different criteria and search-based algorithms commonly used. In addition, we conducted a qualitative study with developers to understand the potential adoption of microservice candidates identified by `toMicroservices`.

III. Industrial Case Study

Our case study relies on a Java legacy system in the domain of oil and gas industry that has been maintained for more than 15 years. 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

[1]https://zenodo.org/record/4001153

[2]Search-based problems with four or more objectives [45].

a database, the search for keywords in different files is very slow. Moreover, the system exhibits all the legacy problems mentioned in Section I.

Microservices Identification Context. Aiming at mitigating the difficulties to maintain the legacy system, the developers involved in its maintenance started analyzing its migration to a microservice architecture. Initially, the developers tried to perform a manual analysis of the source code to identify possible features to be migrating into microservices. However, this manual analysis required a high effort from developers due to system complexity and size. To reduce effort, the developers used visual tools that represent the legacy system classes and their relations as a graph. The developers stated that the use of such tools was not helpful due to poor, complex modularization of the software features. As an alternative strategy, the developers also considered applying state-of-the-art search-based approaches (Section II) to identify microservice candidates. However, they struggled to use them for the reasons mentioned in the previous section. Moreover, the quality of the approaches' results was considered largely unsatisfactory. Thus, we present `toMicroservices` (Section IV) to our industry partner as a possible alternative to automate the identification of microservice candidates for the legacy system. The developers agreed to analyze the microservice candidates generated by `toMicroservices` along with their migration to a microservice architecture.

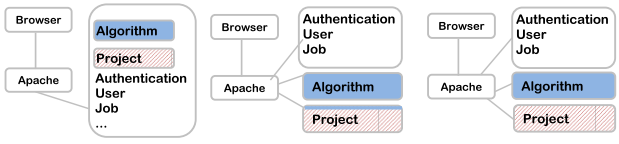
"Microservification" of certain legacy features. The developers decided for the "microservification" of three important features: *Algorithm*, *Authentication*, and *Project*. The three features have at least two subfeatures, 180 classes, and 1,041 methods associated with the implementation in the legacy code.

- *Algorithm*: provides algorithms information by a REST API, including parameters, binary, documents, and connection points with other algorithms. In addition, this information can be stored using different resources.
- *Authentication*: authenticates and provides information of system users. This includes the creation and validation of tokens, verification of login and password, update of password, and related simple information about users.
- *Project*: responsible for the concept of a collaborative environment between the system's users. This collaborative environment includes shared files and metadata between different users or types of users.

Developers responsible for the legacy system were asked about the number of desired microservices to the three features aforementioned, the developers usually answered between 5 and 10 microservices. In this way, we configured two different scenarios of `toMicroservices` to better investigate the adoption of its solutions.

Example of Many-Criteria Analysis. In the context of our case study, Figure 1(a) depicts the current monolithic architecture, which is the source of information. Figures 1(b) and 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 hachured elements are related *Project*.

Let us consider the Alternative 1 (Figure 1(b)). This architecture has one microservice with implementation exclusively of the feature *Algorithm*. This architecture also 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*. As assumed by existing approaches, we could infer that this solution is good in accordance with coupling and cohesion, which are low and high, respectively. However, these simplistic assumptions that features of the existing legacy system usually have well-modularized features in files do not hold in practice. Here we can observe that features are not well-modularized, since the implementation of *Algorithm* is scattered in two microservices, and more importantly, *Algorithm* is tangled with *Project* in the second microservice. Yet, 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 prohibitive network overhead. On the other hand, Alternative 2 (Figure 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 a problem since they are in the same microservice.



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

Fig. 1. Alternative Architectures for the Legacy System

Both alternatives illustrate that more than two criteria are needed to achieve satisfactory microservice identification. For legacy systems, as our case study, approaches should consider several criteria and optimize them to obtain a suitable microservice architecture.

IV. Proposed Approach

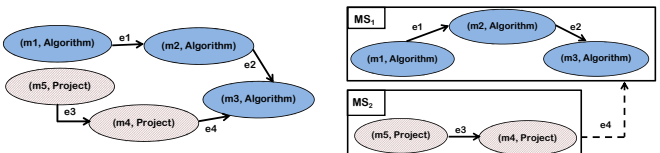
In this paper, we introduce `toMicroservices` - a search-based microservice identification approach. Due to the complexity of the problem, the approach relies on many-objective optimization (Section II-B). `toMicroservices` was conceived based on five objective functions considered by developers as useful in previous studies [12], [14]. The criteria are coupling, cohesion, feature modularization, network overhead, and reuse. The approach's details are presented next.

A. 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 the 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, as well as syntactic and dynamic calls between them. Figure 2(a) 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*.



(a) Legacy's representation

(b) Microservice candidates

Fig. 2. Excerpt of the Legacy System

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 2(b), where: (i) $m1$, $m2$, and $m3$ are part of MS_1, and (ii) $m4$ and $m5$ are included in MS_2. In this case, the communication between $m3$ and $m4$ ($e4$) in the solution representation realizes the communication between microservices MS_1 and MS_2. The values for data communication, syntactic and dynamic calls related to each edge were omitted in Figures 2(a) and 2(b).

B. Objective Functions

Next, we describe each criterion adopted as an objective function. The criteria of coupling and cohesion were based on related work, but adapted in our approach to the fine-grained (method) 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. Next, we describe these criteria. From now on, MSA (MicroServices Architecture) refers to the graph of the microservice candidates (vertices) and their communications (edges) generated by `toMicroservices`. That is, it is a solution generated by `toMicroservices` and also an element in the Pareto set. MS_C (MicroService Candidate) refers to a vertex in MSA.

1. Coupling: this criterion is computed by using static information similarly to [46]. The individual coupling for each microservice candidate MS_c (δ function), showed in Equation 1, is the number of static calls from methods within an MS_c to another microservice candidate MS_c in the same MSA. δ function 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 belong to MS_c). In Equation 1, sc is the number of calls present in the body of v_i method and particularly made to the v_j method, and $(v_i, v_j) \in E$ (the edges set in the graph that represent the legacy system).

$$\delta(MS_c) = \sum^{v_i \in MS_c \wedge v_j \notin MS_c} sc(v_i, v_j) \quad (1)$$

The total coupling of a solution is the sum of the values of coupling associated with every MS_c in an MSA (see Equation 2). The lower the coupling, the better is the result.

$$Coupling = \sum^{\forall MS_c \in MSA} \delta(MS_c) \quad (2)$$

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 MS_c) by all possible existing static calls (similarly to [46]). Hence, this measure indicates how strongly related the methods are within a microservice candidate. To compute cohesion, the ce function is defined in Equation 3 as a boolean function indicating the existence of at least a static call.

$$ce(v_i, v_j) = \begin{cases} 1, & if \text{ sc(vi, vj) > 0} \\ 0, & otherwise \end{cases} \quad (3)$$

The cohesion of a microservice candidate is presented in Equation 4, where $|MS_c|$ is the cardinality (number of methods) of an MS_c. Equation 4 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 is the combination two-by-two of all methods within an MS_c.

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

The total cohesion of a solution is the sum of the cohesion associated with every MS_c in an MSA according to Equation 5. The higher the cohesion better.

$$Cohesion = \sum^{\forall c \in MSA} C(c) \quad (5)$$

3. Feature modularization: We propose a strategy to indicate the responsibility of microservice candidates based on the features associated with executions of the legacy system. To compute feature modularization, the user provides a list of feature 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 feature 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 number of features per microservices. Equation 6 defines the predominant feature (pf function) of an MS_c, where F_{M_c} contains the number of occurrences of each feature in an MS_c.

$$pf(MS_c) \ = \ max \ _{\forall k \in F_{MS_c}}\{k\} \tag{6}$$

$$f(MS_c) \ = \ \frac{pf(MS_c)}{\sum^{\forall k \in F_{MS_c}} k} \tag{7}$$

Thus, the feature modularization of a microservice candidate was defined in Equation 7, as the measure of the number of occurrences of the most common feature divided by the sum of all features occurrences within a microservice candidate.

Regarding the feature modularization in the proposed microservice architecture, Equation 7 introduced the feature per microservices. This equation avoids the fact that each microservice candidate has largely different features.

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 in the MSA. Equation 8 shows the measurement of feature modularization where $FMSA$ is the set of different predominant features in the MSA. In order to avoid separation of the same feature by different microservice candidates, the division in Equation 8 of FMSA and MSA cardinality reduce the feature scattered. In addition, the occurrences of the predominant feature in each microservice candidate of the MSA are summed. It is desired a degree of feature modularization as high as possible.

$$F \ = \ \sum_{\forall c \in MSA} f(c) \ + \ \frac{|FMSA|}{|MSA|} \tag{8}$$

4. Network overhead: Some non-functional requirements, such as performance, may be negatively affected by the network overhead of the identified (and further extracted) microservices. In order to minimize that problem, 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, thus, the size of this header is considered in our estimation.

The network overhead measurement is showed in Equation 9 where the function $P(v_j)$ returns the set of arguments used in an 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. Data traffic function (dt) is computed as shown in Equation 10, where dc function is the total of calls from method v_i to method v_j in execution time.

$$overhead(v_i, v_j, m) = \sum^{\forall p \in P(v_j)} sizeOf(p, m) \tag{9}$$

$$dt((v_i, v_j)) \ = \ max_{m=1}^{m=dc(v_i,v_j)}(overhead(v_i, v_j, m)) \tag{10}$$

The network overhead is defined as the sum of the sizes of the network traffic data to each MS_c (see Equation 11). The lower the network overhead, the better is the result.

$$\begin{aligned} O(MS_c) &= \sum^{\forall v_i \in MS_c \wedge \forall v_j \notin MS_c} dt((v_i, v_j)) \\ Overhead \ &= \ \sum^{\forall MS_c \in MSA} O(MS_c) \end{aligned} \tag{11}$$

5. Reuse: In `toMicroservices`, a microservice candidate is considered reusable when either other microservices or the user call it. In this order, we combine static and dynamic analysis to measure the level of reuse of an MS_c. The static analysis computes the number of calls to microservice candidates. The dynamic analysis computes how many times each microservice candidate is called by the user (*e.g.*, calling the API or user interface). This concept is captured by the mdu function (**m**icroservice **d**irectly called by the **u**ser). mdu considers the system executions that allow identifying dynamic calls between vertices, including start points by the user.

Equation 12 measures whether each microservice is statically or dynamically reusable. Ideally, a microservice should be reused as much as possible or at least twice [47]. Thus, whenever a microservice candidate is reused at least twice, the result of Equation 12 indicates an adequate reuse level of the microservice (*i.e.*, $r(MS_c) = 1$).

$$r(MS_c) \ = \ \begin{cases} 1, if \ \sum^{v_i \in MS_c \wedge v_j \notin MS_c} sc(v_j, v_i) + mdu(MS_c) > 1 \\ 0, \ otherwise \end{cases} \tag{12}$$

The reuse of a microservice architecture is defined in Equation 13. $|MSA|$ represents the number of microservices. $Reuse$ 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 to maximize reuse.

$$Reuse \ = \ \frac{\sum^{\forall MS_c \in MSA} r(MS_c)}{|MSA|} \tag{13}$$

C. Implementation Aspects

Due to the interdependence and conflicting nature of many-objective criteria, we relied on *Non-dominated Sorting Genetic Algorithm III* (NSGA-III) [48]. NSGA-III is a genetic algorithm, which is a method for solving both constrained and unconstrained optimization problems that reflect the process of natural selection [49]. This algorithm repeatedly selects the fittest individuals for reproduction to produce offspring of

the next generation. Over successive generations, the individuals evolve toward a set of optimal solutions. To implement NSGA-III, we used jMetal, which is an object-oriented Java-based framework that includes modern state-of-the-art optimizers [50].

By its complex nature, optimization of designs using evolutionary algorithms can easily lead to architectural violations. Harman and Tratt [51] pointed out that it is difficult to guarantee the accuracy and consistency of solutions after the application of the crossover operator, which has as goal combination of two different designs. To avoid this problem, the first version of `toMicroservices` relies exclusively on the use of the mutation operator. This mutation operator is based on the operator presented in related works [21], [23]. It consists of moving methods from one microservice candidate to another one in an individual of the population. In a simplified form, this mutation operator can be seen as an analogy of the move method refactoring [52]. The fraction of methods to be moved is configurable. The methods and microservice candidates are randomly selected and any kind of method can be moved.

A pre-processing traceability step was performed before the optimization process to label each vertex with the feature that it implements. The feature labeling was configured by the manual analysis of the legacy system execution. During this analysis, an expert on the legacy system informed the entry points in the trace execution of the selected features or subfeature. Each entry point in the trace execution was used to label the vertices of the graph representation (Section IV-A).

V. 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 III.

A. Research Questions

We aim at answering the following two research questions:

RQ1- *How is the performance of* `toMicroservices` *during the optimization process in comparison to the baseline approaches?* To answer this research question, we evaluate the performance of `toMicroservices` (Section V-B) firstly in comparison with a baseline that represents the best state-of-the-art approach [21], [23], optimizing two traditional criteria, namely coupling and cohesion. Considering that there is no existing approach optimizing five criteria, secondly, we compared `toMicroservices` with a random search algorithm. Arcuri and Briand [53] state that a search algorithm can always be compared against at least a random search to check that its success is not due to the simplicity of solving the posed problem. In such evaluations we analyzed: (i) the results according to traditional quality indicators used in multi-objective optimization (Section V-C), and (ii) how the criteria are optimized by search algorithms.

RQ2- *Do the developers judge the solutions found by toMicroservices adoptable?* We want to investigate whether the microservices identified by our approach are in accordance with practical needs of developers. So, we performed a qualitative analysis to verify if the developers would adopt a microservice candidate as an actual solution to be used in practice. To do so, eight experienced developers in the target system were divided into two groups to evaluate the adoptability of microservices generated by `toMicroservices` in two different levels of granularity. The design of the qualitative study is presented in Section V-D.

B. Study Configurations

To answer the posed RQs, we implement different algorithms in two different studies, as follows.

1) Study to Answer RQ1: We implement four algorithm configurations: (i) *Baseline NSGA-II*, (ii) *Baseline NSGA-III*, (iii) `toMicroservices` and (iv) *Random Search*. *Baseline NSGA-II* and *Baseline NSGA-III* optimize the traditional criteria of coupling and cohesion. The former is based on the related works [20], [21], [23] and uses NSGA-II [54], widely used in SBSE studies [40]. The latter uses NSGA-III, which is a new version of the algorithm used in the former configuration. Our goal in comparing *Baseline NSGA-II* with *Baseline NSGA-III* is to reason about which one performs better when optimizing only coupling and cohesion, supporting the adoption of NSGA-III for our approach. For the optimization of the five criteria, we implement `toMicroservices`, which is the proposed approach, and a *Random Search*, which is commonly adopted for comparisons when there is no existing approach in the literature for direct comparison [53]. We implemented a random search with a random selection of vertex per each microservice. After each iteration, the five criteria were evaluated and the non-dominated solutions were stored to be used in the next iterations.

The analysis of these configurations was based on two quality indicators [55] traditionally used to compare the algorithm´s performance, namely Hypervolume (HV) and Inverted Generational Distance (IGD), described in details in the next subsection. Despite our focus being the performance on optimizing the five criteria, an initial analysis between *Baseline NSGA-II* and *Baseline NSGA-III* using ED and IGD only considered coupling and cohesion was performed, as aforementioned, to allow us observing the behavior of NSGA-III in comparison to state-of-the-art approach using NSGA-II [23]. For the analysis of performance with the five criteria, we compared `toMicroservices` and *Random Search* firstly using ED and IGD, and secondly the trade-off among solutions depicting the values of all five objective functions in the search space. For this, we also included *Baseline NSGA-III*, which optimized only two objective functions, but we computed the remaining values of the additional three criteria after the optimization for the sake of analysis.

All algorithms, except for the *Random Search*, the parameters were the same. The population size was defined as 100 individuals. The maximum number of iterations was set to 10,000. This last parameter is also the stopping criterion. Both parameters are higher than the previous study that used genetic algorithm [23]. In addition to these traditional parameters,

there are three more parameters related to our problem: *(i)* the fraction of methods exchanged by the mutation operator, that was set to the minimum of 1 to the maximum of 50% of all microservice methods; *(ii)* the number of microservice candidates was set to 10; and *(iii)* number of methods allocated in each microservice was set to between 3% and 16% of the total number of methods. We used the configuration of 10 microservices for the analysis of performance as it is a more challenging setting by having a higher search-space in comparison to the configuration of 5 microservices. Given the randomness of genetic algorithms, we executed 30 independent runs for each approach, according to guidelines for SBSE studies [40], [53].

2) Study to Answer RQ2: We configured two different scenarios of `toMicroservices` to better investigate the adoption of its solutions. These scenarios are: (i) *Scenario-5MS*, in which the number of desired microservices was set to 5, and the number of methods allocated in each microservice was set to between 5% and 50% of the total number of methods; and (ii) *Scenario-10MS*, that similarly to configurations to answer RQ1, had the number of microservices set to 10, and the number of methods allocated in each microservice was set to between 3% and 16% of the total number of methods.

C. Quantitative Comparison of Performance

For the quantitative comparison of the algorithm configurations of the study designed to answer RQ1 – *Baseline NSGA-II* vs *Baseline NSGA-III* using two criteria, and `toMicroservices` vs *Random Search* using five criteria – we used two quality indicators commonly applied in the field of SBSE when optimizing many objectives. Our quantitative analysis is based on the traditional indicators of *Hypervolume* (HV) [56] and *Inverted Generational Distance* (IGD) [57].

HV measures the area of the objective space from a reference point to a front of solutions [56]. In this study, we use the HV computed by a recursive and dimension-sweep algorithm [58]. The front of the solutions was normalized between [0,1] to compute this indicator. The reference point adopted had the value of 1.1 for all the five objectives, to make it strictly dominated by all solutions of the normalized front. Notice that all algorithms were implemented to deal with all objectives as a minimization problem.

IGD is used to calculate the distance from the *True Pareto Front* (PF_{true}) to an approximation of the Pareto Front found by the algorithm (PF_{known}). For this indicator, values closer to 0 are desired, since 0 indicates that the PF_{known} is the same than the PF_{true}. As PF_{true} is not known for the problem tackled in this paper, and finding an optimal solution is NP-hard, we adopted a common way to estimate PF_{true} that consists of using the non-dominated solutions found by all algorithms in all runs [56].

To analyze the statistical difference between `toMicroservices` and the Baseline, we used the Wilcoxon signed-rank test [59] as the data sets have non-normal distribution. Furthermore, we also compute the effect size with the Vargha-Delaney's $\hat{A}_{12}$ measure [60]. Both tests are widely used to assess approaches in SBSE [40], [53].

D. Qualitative Evaluation with Developers

This section describes a qualitative study conducted to evaluate the potential adoption of microservice candidates obtained with `toMicroservices`. The study involved interviewing developers of the legacy system with respect to individual microservices generated by `toMicroservices` to answer *RQ2*. Next, we detail the study design.

1) We selected eight developers who are specialists in the legacy system. They were randomly divided into two groups, one for each aforementioned scenario in Section V-B.

2) For each scenario (*Scenario-5MS* and *Scenario-10MS*), 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 objective criteria.

3) Developers indicated, using the Likert scale, how specialist they are for each feature considered by our approach.

4) During a semi-structured interview (with closed and open questions), each developer assessed four microservice candidates from two different solutions, which were selected from the set of solutions created in step (2). The first selected solution has the best value of the objective criteria pointed as the most useful by the developer. The second one has the best value of the objective criteria pointed as the less useful. Given the selected solutions, two microservices were chosen from each solution, according to the feature expertise of the developer indicated in step (3). For each microservice, we asked the developer: *Would you adopt this microservice? Why?* The adoptability was collected on a five-point Likert scale and the corresponding motivation (explanation) was summarized in a textual file. The five points Likert scale are: (i) I would not adopt at all, (ii) I would not adopt, (iii) I would adopt partially, (iv) I would adopt, and (v) I would promptly adopt. The participant was instructed to adopt partially whenever he would modify the microservice elements with not less than 20% of modifications (e.g., moving a method from one microservice to another).

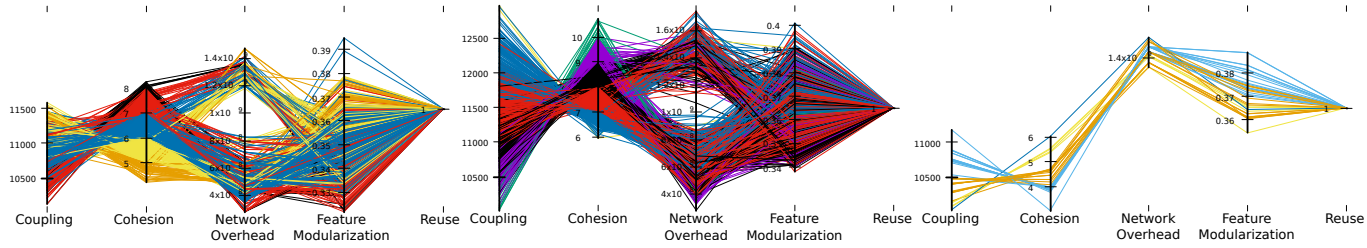
5) In the last step, all data were collected and analyzed by three of the authors. The analysis of the adoption motivation was conducted using a coding strategy. Three researchers (two professors and one master student) performed the encoding to all the motivations in an isolated way. After that, they compared the encoding in a meeting and converged to a single encoding.

VI. Results and Discussion

In the next sections, we present the results regarding the performance of `toMicroservices` (*RQ1*, Section VI-A) and the potential adoption of the microservice candidates (*RQ2*, Section VI-B).

A. RQ1 - Performance of `toMicroservices`

For the initial analysis between *Baseline NSGA-II* and *Baseline NSGA-III*, Table I presents the results of HV and



(a) toMicroservices (b) RandomSearch (c) Baseline NSGA-III

Fig. 3. Comparison of objectives

IGD (see Section V-C) for the 30 independent runs of each algorithm. Wilcoxon pointed statistical difference (p-value $\leq$ 0.05) for both indicators. Furthermore, the effect size measure also pointed difference of large magnitude for HV and IGD. The cells with the best values are highlighted in gray. For all cases, *Baseline NSGA-III* reached better results. This indicates that even for two objectives, NSGA-III can explore the search space of solutions better than the baseline. Based on this evidence, we confirm that NSGA-III is the best choice for the problem of identifying microservices.

TABLE I
RESULTS OF HYPERVOLUME (HV) AND INVERTED GENERATIONAL DISTANCE (IGD). OPTIMIZING COUPLING AND COHESION.

Indicator	Average (Std dev.)		Wilcoxon p-value	$\hat{A}_{12}$ Effect Size	
	Baseline NSGA-II	Baseline NSGA-III		Baseline NSGA-II	Baseline NSGA-III
HV	0.400123 (0.185375)	0.646086 (0.063585)	4.79E-08	11.89%	88.11%
IGD	0.102097 (0.033851)	0.037710 (0.008241)	7.61E-16	0.78%	99.22%

Regarding the optimization using the five criteria for identifying microservices, `toMicroservices` was compared to a *Random Search*. Table II presents the results of the indicators. The results obtained by `toMicroservices` are better than the results of the *Random Search* in all cases. Corroborating with the results that NSGA-III was the best algorithm for the baseline, which is the same algorithm used by our approach, we can clearly attest that `toMicroservices` has the best performance in comparison to the *Random Search* and the state-of-the-art approaches, when optimizing five criteria simultaneously. This comparison among solutions of `toMicroservices`, *Random Search*, and *Baseline NSGA-III* is further detailed next.

To reason about the trade-off among the criteria and the benefits of using five criteria instead of two criteria, Figure 3 depicts parallel coordinate charts. The first two charts (*i.e.*, Figures 3(a) and 3(b)) depict the solutions of `toMicroservices` and *Random Search* approaches to optimize five criteria. The last chart (*i.e.*, Figure 3(c)) depicts the solutions of the *Baseline NSGA-III*. As aforementioned, the Baseline approach optimizes only two objectives (coupling and cohesion). However, for the sake of comparison, we collected the corresponding values of the additional criteria after the optimization process. This allows us to reason about

TABLE II
RESULTS OF HYPERVOLUME (HV) AND INVERTED GENERATIONAL DISTANCE (IGD). OPTIMIZING THE CRITERIA OF COUPLING, COHESION, NETWORK OVERHEAD, FEATURE MODULARIZATION, AND REUSE.

Indicator	Average (Std dev.)		Wilcoxon p-value	$\hat{A}_{12}$ Effect Size	
	Random Search	toMicro-services		Random Search	toMicro-services
HV	0.051593 (0.001111)	0.074534 (0.007403)	1.69E-17	0%	100.00%
IGD	0.100091 (0.002061)	0.092351 (0.017592)	4.00E-04	24.00%	76.00%

the advantages of optimizing only coupling and cohesion in comparison to our many-criteria approach.

At first, we can see that the range of values for the criteria of coupling and cohesion covered by all approaches are concentrated in the best area for *Baseline NSGA-III*, as expected since it exclusively optimizes these two objectives. On the other hand, values of the other three criteria (*i.e.*, feature modularization, overhead and reuse) for *Baseline NSGA-III* are all placed in the worse area. We can also observe that despite the values of coupling between `toMicroservices` and *Baseline NSGA-III* have the solutions in a very similar range of values, the latter has some solutions with better values of cohesion than the former. However, as we previously discussed, these solutions lead to a high degradation of the other criteria. Thus, `toMicroservices` has the overall best performance, since it optimizes all criteria simultaneously.

We investigate the benefit of using three additional objectives equally relevant to identify microservices. We can observe the difference in the solutions regarding the criteria feature modularization and network overhead in Figures 3(a) and 3(c). For feature modularization, the great majority of solutions found by `toMicroservices` is better than the solutions found by the Baseline approach. In addition, almost all solutions of `toMicroservices` are also better regarding the criterion of network overhead, since they are mostly grouped in the mid-lower area of the graph in Figures 3(a), which correspond to lower network overhead. Finally, reuse is a boolean criterion, in which all solutions of all approaches reached values equal to 1, indicating that the microservice candidates found by the approaches are reused in other parts of the architecture (see Section IV-B(5)).

Now we compare the two five-objective approaches, *i.e.*, `toMicroservices` and *Random Search*. The solutions of

toMicroservices (Figure 3(a)) are more concentrated in a better range of values when compared to *Random Search* (Figure 3(b)). For example, the blue lines in Figure 3(a) show a better trade-off among the four first criteria with average values of coupling and cohesion, and low values of network overhead and feature modularization for most of the solutions. For the *Random Search* (Figure 3(b)), there is not standard behavior, as in toMicroservices.

> toMicroservices has better performance than the *Baseline* and the *Random Search* for the search-based microservice identification. Yet, the additional criteria optimized by our approach introduce a new perspective during the optimization of the solutions, which is not achieved when optimizing only coupling and cohesion.

B. RQ2 - Adoptability of Microservice Candidates

We inquired eight participants about the potential adoption of the microservices generated by toMicroservices. Seven participants are developers of the legacy system and one is a team leader. The participants are mostly experienced developers, with a median time experience in software development of 12.5 years. Regarding the experience with the target system, we have both experienced developers (8-20 years of experience), and recent developers (0.5-3 years).

As described in Section V-B, we analyze two scenarios: Scenario-5MS and Scenario-10MS with five and ten microservice candidates, respectively. Participants P1 to P4 were inquired about the adoptability of the microservices identified in Scenario-5MS, while the participants P5 to P8 were inquired about the adoptability of the microservices identified in Scenario-10MS. Each participant was asked to analyze the adoptability of four microservices according to his expertise.

Interestingly, regardless of their experience in the legacy system, all participants could identify the predominant features being modularized within the microservices. The participants also recognized new features (including subfeatures), which were not previously informed as known by them, but identified during the adoptability analysis of the microservice candidates. However, participants with more experience in the target system (P4, P5, P6, P8) were able to recognize more features and new features than moderately experienced participants [3]. A participant (P7) of Scenario-10MS stated the following with respect to the feature algorithm and the 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-5MS indicated that some microservices would not be adopted in practice because of their large size. According to them, these microservices had too many methods as part of a coarse-grained feature, and different features undesirably tangled in the same candidate microservice. One participant (P1) stated: *"I would not adopt this microservice because it is very large with too many methods, which realize different, unrelated features, i.e., project and authentication"*. The quantitative results indicated that 7 microservices of the proposed microservices are partially adoptable or adoptable. A participant argues: *"I would adopt this microservice since it realizes the management of project files, and it is highly reusable by different tenants"*. Also, the participant P2 argues: *"'I would adopt this microservice since it 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 it is"*.

For Scenario-10MS, the analysis of the solutions by the developers pointed out that they would fully or partially adopt 63% (10 microservices) of the proposed microservices. The participants indicated 3 microservices as not interesting to be adopted in practice, mainly because the features modularized by them are too small. A participant (P6) said: *"This isolated microservice does not make sense. I believe that it should be merged into the microservice that manages project files"*. However, the same microservice, which modularized the aforementioned subfeature, was accepted by another participant. Similar contradicting cases occurred among other practitioners. These cases evidence a divergence among participants and their analysis of the adequate level of granularity for the microservice boundaries.

In addition to the limiting of different developers profiles regarding granularity level, another point that impacted the adoptability of microservices was methods in the wrong microservices. Part of them are related to badly modularized transversal features, such as authentication and log. Overall, toMicroservices was able to generate 53% of microservices that would be adoptable in a practical sense by the developers of the system under analysis.

The microservices generated by toMicroservices indicated opportunities to evolve the legacy system. Regarding them, new technologies were pointed out by developers. The participant P5 stated that some generated microservices modularize the subfeature of the project file management, and they can be modified to be a facade to a file database. As mentioned in Section III, the current version of the legacy system does not use a database.

Microservices generated by toMicroservices also helped developers to identify customization opportunities to evolve the legacy system, as shown by the team leader: *"one possible customization for this microservice is to allow using centralized or decentralized files for each project"*. Other opportunities mentioned involve three microservices (Algorithm Flow Builder, Project Files Manager, and Project Metadata) and they can be done in different levels of abstraction: *(i)* customizing the interface of a microservice, *(ii)* changing a microservice by specialization, and *(iii)* completely disabling a microservice. Such observation corroborates the finding of a previous study [14] that the migration to a microservice architecture contributes to the system's customization.

[3]Detailed results in https://zenodo.org/record/4001153

Overall, the developers found the solutions generated by `toMicroservices` adoptable. Four participants would adopt two analyzed microservices, three participants would adopt at least one microservice, and only one participant would not adopt any microservice.

C. Other Lessons Learned

Scattered features need further attention. For instance, the feature *authentication* was not captured by a microservice as the main feature. The methods implementing this feature were scattered among several microservices for both scenarios presented in Section VI-B. 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 objective values since in a microservice architecture is common to find methods regarding *e.g.*, *authentication* or *logging* that are highly-coupled with other microservices.

At the end of the interview, the developers were inquired about improvements that could help them in the analysis. Half of the participants pointed out that a visual representation of the methods inside a microservice could make the analysis easier. In addition, the participants 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.

The results of the evaluation also indicate the importance of adding user interaction during the evolutionary process. In particular, `toMicroservices` should be able to allow developers to perform certain modifications on microservice candidates (*e.g.*, add, delete and move methods) and freeze the microservice that they judge properly designed.

VII. Threats to Validity

Next, we discuss the threats to the validity of this study.

Internal Validity. A threat to the internal validity of this study is the selection of the evolutionary algorithm and its parameter settings. We used the state-of-the-art genetic algorithm based on NSGA III [48] which has shown high accuracy to solve many-objective problems in the SBSE field. Moreover, we set parameters based on previous work [23] or to outperform it.

Due to the many-objective nature of NSGA-III, the solutions may converge to a different set of local optimum (*i.e.*, microservice candidates) in each run. To mitigate any bias, we set the execution of the algorithm to 30 independent runs, and we selected the best solution among the solutions produced in each run according to the *Euclidean distance* quality indicator.

Our results can also be subject to execution traces bias, since all execution traces are generated using given test cases. To mitigate this threat, the developer in charge of microservice identification has sufficient knowledge of the target system, including test cases and parameter settings.

External Validity. There is an inherent threat to external validity in generalizing our findings for other legacy systems. We mitigate this threat by selecting a consolidated real-world legacy system with more than 15 years of existence under the process of migration to microservices. We focused on a single system to be able at making robust and reliable statements about the performance of our approach. Once we are able to demonstrate evidence to the considered system, we can then perform such an analysis also over other systems and adding new objective criteria by extending our current implementation to generalize our findings and improve our results.

The second threat is related to the discussion of the results. Three authors of the paper analyzed the developers' answers to mitigate potential problems in the coding process. In addition, we conducted a discussion with all authors to avoid any confusing interpretation or misunderstanding of the results.

The different levels of experience of subjects participating in the experimentation is another external threat to validity. Different groups of subjects may have different opinions about the microservice candidates. We tried to minimize this threat by involving experienced developers who were knowledgeable about the legacy system.

VIII. 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 to a baseline approach in an industrial case study. The baseline considered only two traditional criteria, namely coupling and cohesion.

The quantitative results pointed out that the performance of `toMicroservices` is better than the baseline, reinforcing the need to adopt more criteria than traditional ones. We clearly observed that the criteria of feature modularization, network overhead, and reuse introduced a new perspective in the optimization of the solutions, and that 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 solutions generated by `toMicroservices` indicated opportunities to better maintain and evolve the legacy system.

The lessons learned with the industrial case study include: the importance of providing proper visualization of the microservice candidates as well as the need of adapting some objective functions to consider highly-crosscutting features.

Acknowledgment

This work is supported by CNPq (Grants: 141278/2020-0, 428994/2018-0, 427787/2018-1, 421306/2018-1, 434969/2018-4, 309844/2018-5, 408356/2018-9, and 312149/2016-6), FAPPR (Grants: 51152 and 51435), CAPES (Grants: 88887.473590/2020-00, 88887.177710/2018-00,

175956, and Proex - PUC-Rio), and FAPERJ (Grants: 010002285/2019 and 200773/2019). The authors thank Tecgraf Institute of PUC-Rio for its support in the industrial case study.

REFERENCES

[1] J. Bisbal, D. Lawless, B. Wu, and J. Grimson, "Legacy information systems: Issues and directions," *IEEE Software*, vol. 16, no. 5, pp. 103–111, Sep. 1999.

[2] J. Ransom, I. Somerville, and I. Warren, "A method for assessing legacy systems for evolution," in *Proceedings of the Second Euromicro Conference on Software Maintenance and Reengineering*, March 1998, pp. 128–134.

[3] L. P. Tizzei, M. Nery, V. C. V. B. Segura, and R. F. G. Cerqueira, "Using microservices and software product line engineering to support reuse of evolving multi-tenant saas," in *21st International Systems and Software Product Line Conference (SPLC)*. New York, NY, USA: ACM, 2017, pp. 205–214.

[4] W. Luz, E. Agilar, M. C. de Oliveira, C. E. R. de Melo, G. Pinto, and R. Bonifácio, "An experience report on the adoption of microservices in three brazilian government institutions," in *XXXII Brazilian Symposium on Software Engineering (SBES)*. New York, NY, USA: ACM, 2018, pp. 32–41.

[5] S. Fowler, *Production-Ready Microservices*, 1st ed. O'Reilly Media, 2016.

[6] C. Watson, S. Emmons, and B. Gregg. (2015) A microscope on microservices. [Online]. Available: http://techblog.netflix.com/2015/02/a-microscope-on-microservices.html

[7] J. Gouigoux and D. Tamzalit, "From monolith to microservices: Lessons learned on an industrial migration to a web oriented architecture," in *International Conference on Software Architecture Workshops (ICSAW)*, 2017, pp. 62–65.

[8] A. Bucchiarone, N. Dragoni, S. Dustdar, S. T. Larsen, and M. Mazzara, "From monolithic to microservices: An experience report from the banking domain," *IEEE Software*, vol. 35, no. 3, pp. 50–55, 2018.

[9] D. Taibi, V. Lenarduzzi, and C. Pahl, "Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation," *IEEE Cloud Computing*, vol. 4, no. 5, pp. 22–32, 2017.

[10] P. D. Francesco, P. Lago, and I. Malavolta, "Migrating towards microservice architectures: An industrial survey," in *International Conference on Software Architecture (ICSA)*, 2018, pp. 29:01–29:09.

[11] H. Knoche and W. Hasselbring, "Using microservices for legacy software modernization," *IEEE Software*, vol. 35, no. 3, pp. 44–49, May 2018.

[12] L. Carvalho, A. Garcia, W. K. G. Assunção, R. de Mello, and M. J. 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*, ser. CESSER-IP '19. Piscataway, NJ, USA: IEEE Press, 2019, pp. 22–29.

[13] J. Fritzsch, J. Bogner, A. Zimmermann, and S. Wagner, "From monolith to microservices: A classification of refactoring approaches," in *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*, J.-M. Bruel, M. Mazzara, and B. Meyer, Eds. Cham: Springer International Publishing, 2019, pp. 128–141.

[14] L. Carvalho, A. Garcia, W. K. G. Assunção, R. Bonifácio, L. P. Tizzei, and T. E. Colanzi, "Extraction of configurable and reusable microservices from legacy systems: An exploratory study," in *Proceedings of the 23rd International Systems and Software Product Line Conference - Volume A*, ser. SPLC '19. New York, NY, USA: ACM, 2019, pp. 26–31.

[15] S. Newman, *Building microservices: designing fine-grained systems*. O'Reilly Media, Inc., 2015.

[16] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, *Microservices: Yesterday, Today, and Tomorrow*. Cham: Springer International Publishing, 2017, pp. 195–216.

[17] J. Lewis and M. Fowler. (2014) Microservices. [Online]. Available: https://martinfowler.com/articles/microservices.html

[18] P. D. Francesco, I. Malavolta, and P. Lago, "Research on architecting microservices: Trends, focus, and potential for industrial adoption," in *IEEE International Conference on Software Architecture (ICSA)*, April 2017, pp. 21–30.

[19] N. Ford, *The State of Microservices Maturity: Survey Results*. CA: O'Reilly Media, Inc., 2018.

[20] G. Mazlami, J. Cito, and P. Leitner, "Extraction of microservices from monolithic software architectures," in *International Conference on Web Services (ICWS)*. IEEE, 2017, pp. 524–531.

[21] W. Jin, T. Liu, Q. Zheng, D. Cui, and Y. Cai, "Functionality-oriented microservice extraction based on execution trace clustering," in *International Conference on Web Services (ICWS)*. IEEE, 2018, pp. 211–218.

[22] S. Eski and F. Buzluca, "An automatic extraction approach: Transition to microservices architecture from monolithic application," in *19th International Conference on Agile Software Development: Companion*, ser. XP '18. New York, NY, USA: ACM, 2018, pp. 25:1–25:6.

[23] W. Jin, T. Liu, Y. Cai, R. Kazman, R. Mo, and Q. Zheng, "Service candidate identification from monolithic systems based on execution traces," *IEEE Transactions on Software Engineering*, pp. 1–1, 2019.

[24] D. Escobar, D. Cárdenas, R. Amarillo, E. Castro, K. Garcés, C. Parra, and R. Casallas, "Towards the understanding and evolution of monolithic applications as microservices," in *XLII Latin American Computing Conference (CLEI)*. IEEE, 2016, pp. 1–11.

[25] W. N. Oizumi, L. da Silva Sousa, A. Oliveira, L. Carvalho, A. Garcia, T. E. Colanzi, and R. F. Oliveira, "On the density and diversity of degradation symptoms in refactored classes: A multi-case study," in *30th IEEE International Symposium on Software Reliability Engineering, ISSRE 2019, Berlin, Germany, October 28-31, 2019*, K. Wolter, I. Schieferdecker, B. Gallina, M. Cukier, R. Natella, N. Ivaki, and N. Laranjeiro, Eds. IEEE, 2019, pp. 346–357.

[26] W. N. Oizumi, A. F. Garcia, L. da Silva Sousa, B. B. P. Cafeo, and Y. Zhao, "Code anomalies flock together: exploring code anomaly agglomerations for locating design problems," in *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016*, L. K. Dillon, W. Visser, and L. Williams, Eds. ACM, 2016, pp. 440–451.

[27] W. N. Oizumi, A. F. Garcia, T. E. Colanzi, M. Ferreira, and A. von Staa, "When code-anomaly agglomerations represent architectural problems? an exploratory study," in *2014 Brazilian Symposium on Software Engineering, Maceió, Brazil, September 28 - October 3, 2014*. IEEE Computer Society, 2014, pp. 91–100.

[28] I. M. Bertran, R. Arcoverde, A. Garcia, C. Chavez, and A. von Staa, "On the relevance of code anomalies for identifying architecture degradation symptoms," in *16th European Conference on Software Maintenance and Reengineering, CSMR 2012, Szeged, Hungary, March 27-30, 2012*, T. Mens, A. Cleve, and R. Ferenc, Eds. IEEE Computer Society, 2012, pp. 277–286.

[29] W. N. Oizumi, L. da Silva Sousa, A. Oliveira, A. Garcia, O. I. A. B. Agbachi, R. F. Oliveira, and C. Lucena, "On the identification of design problems in stinky code: experiences and tool support," *J. Braz. Comput. Soc.*, vol. 24, no. 1, pp. 13:1–13:30, 2018.

[30] L. da Silva Sousa, A. Oliveira, W. N. Oizumi, S. D. J. Barbosa, A. Garcia, J. Lee, M. Kalinowski, R. M. de Mello, B. Fonseca, R. F. Oliveira, C. Lucena, and R. B. de Paes, "Identifying design problems in the source code: a grounded theory," in *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, M. Chaudron, I. Crnkovic, M. Chechik, and M. Harman, Eds. ACM, 2018, pp. 921–931.

[31] R. F. Oliveira, L. da Silva Sousa, R. M. de Mello, N. M. C. Valentim, A. Lopes, T. Conte, A. F. Garcia, E. C. C. de Oliveira, and C. J. P. de Lucena, "Collaborative identification of code smells: A multi-case study," in *39th IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice Track, ICSE-SEIP 2017, Buenos Aires, Argentina, May 20-28, 2017*. IEEE Computer Society, 2017, pp. 33–42.

[32] W. N. Oizumi, L. da Silva Sousa, A. Garcia, R. F. Oliveira, A. Oliveira, O. I. A. B. Agbachi, and C. Lucena, "Revealing design problems in stinky code: a mixed-method study," in *Proceedings of the 11th Brazilian Symposium on Software Components, Architectures and Reuse, SBCARS 2017, Fortaleza, CE, Brazil, September 18 - 19, 2017*. ACM, 2017, pp. 5:1–5:10.

[33] D. Cedrim, A. Garcia, M. Mongiovi, R. Gheyi, L. da Silva Sousa, R. M. de Mello, B. Fonseca, M. Ribeiro, and A. Chávez, "Understanding the impact of refactoring on smells: a longitudinal study of 23 software

projects," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*, E. Bodden, W. Schäfer, A. van Deursen, and A. Zisman, Eds. ACM, 2017, pp. 465–475.

[34] M. Ferreira, E. A. Barbosa, I. M. Bertran, R. Arcoverde, and A. Garcia, "Detecting architecturally-relevant code anomalies: a case study of effectiveness and effort," in *Symposium on Applied Computing, SAC 2014, Gyeongju, Republic of Korea - March 24 - 28, 2014*, Y. Cho, S. Y. Shin, S. Kim, C. Hung, and J. Hong, Eds. ACM, 2014, pp. 1158–1163.

[35] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices architecture enables devops: Migration to a cloud-native architecture," *Ieee Software*, vol. 33, no. 3, pp. 42–52, 2016.

[36] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based software engineering: Trends, techniques and applications," *ACM Computing Surveys (CSUR)*, vol. 45, no. 1, pp. 1–61, 2012.

[37] M. Harman, "The current state and future of search based software engineering," in *Future of Software Engineering (FOSE'07)*. IEEE, 2007, pp. 342–357.

[38] M. Harman, P. McMinn, J. T. De Souza, and S. Yoo, "Search based software engineering: Techniques, taxonomy, tutorial," in *Empirical software engineering and verification*. Springer, 2010, pp. 1–59.

[39] O. Räihä, "A survey on search-based software design," *Computer Science Review*, vol. 4, no. 4, pp. 203–249, 2010.

[40] T. E. Colanzi, W. K. G. Assunção, P. R. Farah, S. R. Vergilio, and G. Guizzo, "A review of ten years of the symposium on search-based software engineering," in *Search-Based Software Engineering*, S. Nejati and G. Gay, Eds. Cham: Springer International Publishing, 2019, pp. 42–57.

[41] A. Ramirez, J. R. Romero, and S. Ventura, "A survey of many-objective optimisation in search-based software engineering," *Journal of Systems and Software*, vol. 149, pp. 382–395, 2019.

[42] N. Alshahwan, X. Gao, M. Harman, Y. Jia, K. Mao, A. Mols, T. Tei, and I. Zorin, "Deploying search based software engineering with sapienz at facebook," in *Search-Based Software Engineering*, T. E. Colanzi and P. McMinn, Eds. Cham: Springer International Publishing, 2018, pp. 3–45.

[43] S. Yoo, M. Harman, and S. Ur, "Highly scalable multi objective test suite minimisation using graphics cards," in *International Symposium on Search Based Software Engineering*. Springer, 2011, pp. 219–236.

[44] K. Lakhotia, N. Tillmann, M. Harman, and J. De Halleux, "Flopsy-search-based floating point constraint solving for symbolic execution," in *IFIP International Conference on Testing Software and Systems*. Springer, 2010, pp. 142–157.

[45] S. Chand and M. Wagner, "Evolutionary many-objective optimization: A quick-start guide," *Surveys in Operations Research and Management Science*, vol. 20, no. 2, pp. 35–42, 2015.

[46] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, June 1994.

[47] R. Capilla, B. Gallina, C. Cetina, and J. Favaro, "Opportunities for software reuse in an uncertain world: From past to emerging trends," *JSEP-ICSR'18-Special Issue*, vol. 31, no. 8, pp. 1–22, August 2019. [Online]. Available: http://www.es.mdh.se/publications/5550-

[48] K. Deb and H. Jain, "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*, vol. 18, no. 4, pp. 577–601, 2014.

[49] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998.

[50] J. J. Durillo and A. J. Nebro, "jmetal: A java framework for multi-objective optimization," *Advances in Engineering Software*, vol. 42, no. 10, pp. 760–771, 2011.

[51] M. Harman and L. Tratt, "Pareto optimal search based refactoring at the design level," in *9th Annual Conference on Genetic and Evolutionary Computation*. New York, NY, USA: ACM, 2007, pp. 1106–1113.

[52] M. Fowler, *Refactoring: Improving the Design of Existing Code*. USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[53] A. Arcuri and L. Briand, "A Hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering," *Software Testing, Verification and Reliability*, vol. 24, no. 3, pp. 219–250, 2014.

[54] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[55] M. Li and X. Yao, "Quality evaluation of solution sets in multiobjective optimisation: A survey," *ACM Computing Surveys*, vol. 52, pp. 1–38, 03 2019.

[56] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca, "Performance assessment of multiobjective optimizers: An analysis and review," *IEEE Transactions on Evolutionary Computation*, vol. 7, pp. 117–132, 2003.

[57] I. Radziukyniene and A. Zilinskas, "Evolutionary Methods for Multi-Objective Portfolio Optimization," in *Proceedings of the World Congress on Engineering 2008 Vol II*, jul 2008.

[58] C. M. Fonseca, L. Paquete, and M. Lopez-Ibanez, "An improved dimension-sweep algorithm for the hypervolume indicator," in *IEEE Intern. Conference on Evolutionary Computation*, 2006, pp. 1157–1163.

[59] R. Bergmann, J. Ludbrook, and W. P. J. M. Spooren, "Different Outcomes of the Wilcoxon-Mann-Whitney Test from Different Statistics Packages," *The American Statistician*, vol. 54, no. 1, pp. 72–77, 2000.

[60] A. Vargha and H. Delaney, "A critique and improvement of the cl common language effect size statistics of McGraw and Wong," *J. of Educational and Behavioral Statistics*, vol. 25, no. 2, pp. 101–132, 2000.