Defining and measuring microservice granularity – a literature overview

Fredy H Vera-Rivera [Corresp., Equal first author, 1, 2, 3] , **Carlos Gaona** [Equal first author, 2] , **Hernán Astudillo** [4]

[1] GIA Research Group, Universidad Francisco de Paula Santander, Cúcuta, Norte de Santander, Colombia

[2] GEDI Research Group, Universidad del Valle, Santiago de Cali, Valle del Cauca, Colombia

[3] Materials and Technology Research Group, Foundation of Researchers in Science and Technology of Materials - FORISTOM, Bucaramanga, Santander, Colombia

[4] Toeska Research Team, Universidad Técnica Federico Santa María, Valparaiso, Valparaiso, Chile

Corresponding Author: Fredy H Vera-Rivera
Email address: fredyhumbertovera@ufps.edu.co

Background. Microservices are an architectural approach of growing use, and the optimal granularity of a microservice directly affects the application's quality attributes and usage of computational resources. Determine microservice granularity is an open research topic.

Methodology. We conducted a systematic literature review to analyze literature that address the definition of microservice granularity. We searched in IEEE Xplore, ACM Digital Library and Scopus. The research questions were: Which approaches have been proposed to define microservice granularity and determine the microservices' size? Which metrics are used to evaluate microservice granularity? Which quality attributes are addressed when researching microservice granularity?

Results. We found 326 papers and selected 29 articles after applying inclusion and exclusion criteria. The quality attributes most often addressed are runtime properties (e.g. scalability and performance), not development properties (e.g. maintainability). Most proposed metrics were about the product, both static (coupling, cohesion, complexity, source code) and runtime (performance, and usage of computational resources), and a few were about the development team and process. The most used techniques for defining microservices granularity were machine learning (clustering), semantic similarity, genetic programming, and domain engineering. Most papers were concerned with migration from monoliths to microservices; and a few addressed green-field development, but none address improvement of granularity in existing microservice-based systems.

Conclusions. Methodologically speaking, microservice granularity research is at a Wild West stage: no standard definition, no clear development – operation trade-offs, and scarce conceptual reuse (e.g., few methods seem applicable or replicable in projects other than their initial proposal). These gaps in granularity research offer clear options to investigate on continuous improvement of the development and operation of microservice-based systems.

1 # Defining and measuring microservice granularity – a
2 # literature overview.

4 Fredy H. Vera-Rivera[1, 2, 3], Carlos Gaona[3], Hernán Astudillo[4]

6 [1] GIA Research Group, Universidad Francisco de Paula Santander, San José de Cúcuta, Norte de
7 Santander, Colombia
8 [2] Materials and Technology Research Group, Foundation of Researchers in Science and
9 Technology of Materials - Foristom, Bucaramanga, Santander, Colombia.
10 [3] GEDI Research Group, Universidad del Valle, Santiago de Cali, Valle del Cauca, Colombia.
11 [4] Toeska Research Team, Universidad Técnica Federico Santa María, Valparaíso, Valparaiso,
12 Chile.

14 Corresponding Author:
15 Fredy H. Vera Rivera[1]
16 Av. 0AE # 3AN – 19 Apto 901 Edificio Cámbulos Gold, San José de Cúcuta, Norte de
17 Santander, 540001, Colombia
18 Email address: fredyhumbertovera@ufps.edu.co

20 ## Abstract

21 **Background.** Microservices are an architectural approach of growing use, and the optimal
22 granularity of a microservice directly affects the application's quality attributes and usage of
23 computational resources. Determine microservice granularity is an open research topic.
24 **Methodology.** We conducted a systematic literature review to analyze literature that address the
25 definition of microservice granularity. We searched in IEEE Xplore, ACM Digital Library and
26 Scopus. The research questions were: Which approaches have been proposed to define
27 microservice granularity and determine the microservices' size? Which metrics are used to
28 evaluate microservice granularity? Which quality attributes are addressed when researching
29 microservice granularity?
30 **Results.** We found 326 papers and selected 29 articles after applying inclusion and exclusion
31 criteria. The quality attributes most often addressed are runtime properties (e.g. scalability and
32 performance), not development properties (e.g. maintainability). Most proposed metrics were
33 about the product, both static (coupling, cohesion, complexity, source code) and runtime
34 (performance, and usage of computational resources), and a few were about the development
35 team and process. The most used techniques for defining microservices granularity were
36 machine learning (clustering), semantic similarity, genetic programming, and domain
37 engineering. Most papers were concerned with migration from monoliths to microservices; and a
38 few addressed green-field development, but none address improvement of granularity in existing
39 microservice-based systems.

40   **Conclusions.** Methodologically speaking, microservice granularity research is at a Wild West
41   stage: no standard definition, no clear development – operation trade-offs, and scarce conceptual
42   reuse (e.g., few methods seem applicable or replicable in projects other than their initial
43   proposal). These gaps in granularity research offer clear options to investigate on continuous
44   improvement of the development and operation of microservice-based systems.

1. Introduction

47   Microservices are an architectural style that change the way applications are created, tested,
48   implemented, and maintained. Microservices facilitate the migration of applications to the cloud
49   infrastructure, since they allow automatic scaling, load balancing, and fault tolerance. By using
50   microservices, a large application can be implemented as a set of small applications that can be
51   developed, deployed, expanded, managed, and monitored independently. Agility, cost reduction
52   and granular scalability entail some of the challenges such as the complexity of managing
53   distributed systems (Villamizar et al., 2015).
54   The agile practices highlighting DevOps (Dev: Developers, Ops: Operations) and microservices
55   are current trends in the development and deployment of applications in the cloud; their use by
56   recognized technology companies and world leaders of the microservice architectural style has
57   grown considerably. Companies such as Netflix, Amazon, eBay, PayPal, and Google, use
58   microservices, some with more than 1000 microservices deployed and supporting a considerable
59   number of concurrent users. Microservice research is in a formative stage and evolution in this
60   field is continuous. Some companies have adopted microservices to deploy their businesses.
61   Microservices are characterized by rapid industrial adoption, and the main advances are taking
62   place in the technology industry. However, the trend in publications and scientific advances from
63   the academy is growing (Soldani, Tamburri & Heuvel, 2018).
64   Microservices have been presented as an implementation approach of service-oriented
65   architecture (SOA), aiming to improve its disadvantages and problems (Zimmermann, 2017),
66   (Pautasso et al., 2017). Microservices architecture features include business orientation, polyglot
67   programming in multiple paradigms and languages, fault-tolerant design, decentralization, and
68   automation. Microservices are independent applications and distributed systems. SOA and
69   microservices inherit the characteristics of distributed systems and their complexities. In this
70   article we focus exclusively on microservices, not including SOA, web services or mobile
71   services, which are implemented, tested, and deployed very differently than microservices.
72   Resilience, scalability, fast software delivery and use of fewer resources are essential
73   characteristics of current applications. Microservices architecture came to fulfill those
74   expectations (Salah et al., 2016), but many challenges still exist, such as the definition of the
75   granularity, the complexity of managing small distributed systems, network latency and lack of
76   reliability, fault tolerance, coherence and integration of data, distributed transaction management,
77   communication layers, load balancing, orchestration, monitoring and security (Pautasso et al.,
78   2017). The main objective of this work is to analyze the problem of defining the granularity of
79   microservices. Microservices granularity is defined mainly, first by its size or dimensions,

80   meaning the number of operations exposed by the microservice, along with the number of
81   microservices that are part of the whole application, and second by its complexity and
82   dependencies. The goal is to have low coupling, low complexity, and high cohesion between
83   microservices. Hassan, Bahsoon, and Kasman (2020) stated that a granularity level determines
84   "the service size and the scope of functionality a service exposes (Kulkarni & Dwivedi, 2008)".
85   Granularity adaptation entails merging or decomposing microservices thereby moving to a finer
86   or more coarse-grained granularity level (Hassan, Bahsoon, & Kazman, 2020). Homay et al.
87   (2020) stated that "the problem in finding service granularity is to identify a correct boundary
88   (size) for each service in the system. In other words, each service in the system needs to have a
89   concrete purpose, as decoupled as possible, and add value to the system. A service has a correct
90   or good granularity if it maximizes system modularity while minimizing the complexity.
91   Modularity in the sense of flexibility, scalability, maintainability, and traceability, whereas
92   complexity in terms of dependency, communication, and data processing" (Homay et al., 2020).
93   The quality of a microservices-based system is influenced by the granularity of its microservices,
94   because their size and number directly affect the system's quality attributes. The optimal size or
95   granularity of a microservice directly affects application performance, maintainability, storage
96   (transactions and distributed queries), and usage and consumption of computational resources
97   (mainly in the cloud, the usual platform to deploy and execute microservices). Although the size
98   of microservice or optimal granularity is a discussion topic, few patterns, methods, or models
99   exist to determine how small a microservice should be, as others have already pointed out:
100   Soldani et al. (2018) noticed the difficulty of identifying the business capacities and delimited
101   contexts that can be assigned to each microservice (Soldani, Tamburri & Heuvel, 2018). Bogner
102   et al. (2017) claimed that "the appropriate microservice granularity is still one of the most
103   discussed properties (How small is small enough?), as shown in the difficulty of defining
104   acceptable value ranges for source code metrics" (Bogner, Wagner & Zimmermann, 2017a).
105   Zimmermann (2017) indicated that professionals request more concrete guidance than the
106   frequent advice to "define a limited context for each domain concept that will be exposed as a
107   service" (Zimmermann, 2017). Jamshidi et al. (2018) affirmed that the real challenge is finding
108   the right modules, with the correct size, the correct assignment of responsibilities, and well-
109   designed interfaces, and besides, no agreement on the correct size of microservices exist
110   (Jamshidi et al., 2018).The aim of this article is to identify the main approaches in the literature
111   that define microservice granularity or that use it in the process of designing microservice-based
112   systems, either from scratch or migrated from monoliths. A systematic literature review was
113   carried out on key scientific computing literature databases (IEEE Xplore, ACM Digital Library,
114   and Scopus); we formulated three research questions, we defined inclusion and exclusion
115   criteria, and current research trends to identify relevant works, challenges and gaps for future
116   research were identified. Research papers that address the problem of microservice granularity
117   are detailed; the research questions are (RQ1) which approaches have been proposed to define
118   microservice granularity? (RQ2) which metrics are used to evaluate microservice granularity?
119   and (RQ3) which quality attributes are addressed when researching microservice granularity?

120   Very few previous works have reviewed the definition of microservices granularity; we did not
121   find any review that details the techniques, methods or methodologies used to define granularity,
122   none describe the metrics used to evaluate it, and few addresses the quality attributes considered
123   to define it.
124   Contributions of this work are as follow: (1) we identified and classified research papers that
125   address the problem of microservice granularity, therefore we defined the state of the art; (2) we
126   identified and defined the metrics currently used to assess the granularity of microservices-based
127   systems; (3) we identified the quality attributes that researchers studied to define microservice
128   granularity; (4) we identified the case studies used to validate the methods, which can serve as a
129   dataset for future evaluations of methods or techniques to define granularity.
130   The remainder of this article is organized as follows: Section 2 defines previous related works;
131   Section 3 presents the survey design; Section 4 organizes the results; Section 5 discusses the
132   trends and research gaps; and Section 6 summarizes and concludes.
133

134   2. Related work

135   A small number of literature reviews have been published on microservice architecture research,
136   some of these papers include analysis of application modeling and architecture; design and
137   development patterns; industrial adoption; state of practice; grey literature review; and analysis
138   and interviews with industry leaders, software architects, and application developers of
139   microservices-based applications; whereas two papers focused on microservice granularity
140   specifically (Hassan, Bahsoon & Kazman, 2020) and (Schmidt & Thiry, 2020). The literature
141   reviews are described in chronological order below.
142   Di Francesco (2017), Di Francesco, Lago and Malavolta (2019) focused on determining the
143   publication trends of research studies on architecture with microservices and on the potential for
144   industrial adoption of existing research. They point out that few studies have focused on design
145   patterns and architectural languages for microservices, and research gaps exist in areas related to
146   quality attributes (Di Francesco, 2017) (Di Francesco, Lago & Malavolta, 2019).
147   Zimmermann (2017) extracted the principles of microservices from the literature and makes a
148   comparison with SOA and highlights the critical points in the research on microservices, as a
149   result of the review and discussions with industry opinion leaders, developers and members of
150   the service-oriented community. He raises five research issues: (1) service interface design
151   (contracting and versioning), (2) microservice assembly and hosting, (3) microservice integration
152   and discovery, (4) service dependency management, and (5) service and end client application
153   testing (Zimmermann, 2017).
154   Jamshidi et al. (2018) presented a technological and architectural perspective on the evolution of
155   microservices. Their editorial introduction also set out future research challenges: (1) service
156   modularization and refactoring; (2) service granularity; (3) front-end integration; (4) resource
157   monitoring and management; (5) failure, recovery and self-repair; and (6) organizational culture
158   and coordination (Jamshidi et al., 2018).

159   Soldani, Tamburri, and Heuvel (2018) systematically analyzed the grey industrial literature on
160   microservices to define the state of practice, identified the technical and operational problems
161   and benefits of the architectural style based on microservices at an industrial level. When
162   designing a microservice-based application, key issues involve determining the right granularity
163   of its microservices and the design of its security policies. During development time, managing
164   distributed storage and application testing, is challenging. Another pain was usage of network
165   and computing resources during operation (Soldani, Tamburri & Heuvel, 2018).
166   Ghofrani and Lübke (2018) focused on identify challenges and gaps in the design and
167   development of microservices, they describe the main reasons leveraging and preventing the
168   usage of systematic approaches in microservice architectures; and the suggestions or solutions
169   that improve aspects of the microservices architecture. Ghofrani and Lübke provided an updated
170   map of the state of practice in microservice architecture and its complexities for future research.
171   According to the results of their survey, optimization in security, response time, and performance
172   had higher priorities than resilience, reliability, fault tolerance, and memory usage are research
173   gaps (Ghofrani & Lübke, 2018).
174   Osses, Márquez, and Astudillo (2018) summarized the 44 architectural patterns of microservices,
175   and proposed a microservice architectural pattern taxonomy: front-end, back-end, orchestration,
176   migration, internet of things, and DevOps. There was no specific pattern to define the adequate
177   microservice granularity, and they just proposed designing the application as a set of modules,
178   each one an independent business function with its data, developed by a separate team and
179   deployed in a separate process (Osses, Márquez & Astudillo, 2018).
180   Hamzehloui, Sahibuddin & Salah (2019) aimed to identify the common trends and direction of
181   research in microservices. They stated that infrastructure-related issues were more common than
182   software-related issues, and the cloud was the most common platform for running microservices.
183   At the infrastructure level, automation and monitoring require more research, as do software
184   development and design in microservices; safety, maintenance, and costs were three other areas
185   that have been studied relatively less compared to other topics (Hamzehloui, Sahibuddin &
186   Salah, 2019).
187   Vera-Rivera, Gaona, and Astudillo (2019) identified the challenges and research trends present
188   in the phases of the development process and in the management of quality attributes of
189   microservice-based applications (Vera-Rivera, Gaona Cuevas & Astudillo, 2019). This article
190   was more general, it did not emphasize in granularity.
191   Hassan, Bahsoon, and Kazman (2020) carried out a systematic mapping study to provide a better
192   understanding of the transition to microservices; they consolidated various views (industrial,
193   research/academic) of the principles, methods, and techniques commonly adopted to assist in the
194   transition to microservices. They identified gaps in the state of the art and the practice related to
195   reasoning about microservice granularity. In particular, they identified possible research topics
196   concerning (1) systematic architecture-oriented modeling support for microservice granularity,
197   (2) a dynamic architectural assessment approach for reasoning about the cost and benefit of
198   granularity adaptation, and (3) effective decision support for informing reasoning about

199 microservice granularity at runtime (Hassan, Bahsoon & Kazman, 2020). They focused on
200 understanding the transition to microservices and the microservice granularity problem (a direct
201 antecedent of this study). They considered quality attributes but not metrics. Their sources were
202 gray literature (blog articles, presentations, and videos, as means of reporting first-hand
203 industrial experiences) and research papers, whereas our study put emphasis only on white
204 literature (articles published in journals and scientific events). Our work was more specific and
205 detailed evaluating the methods and techniques to define granularity, whereas their work detailed
206 those methods and techniques in general. Our work are complementary to their work and take a
207 deeper look at the definition of granularity.
208 Schmidt and Thiry (2020) carried out a systematic literature review, they found proposals for
209 identification, decomposition, partitioning or breaking down the application domain to reach an
210 adequate granularity for microservices. Moreover, the research aims to highlight the usage of
211 Model-Driven Engineering (MDE) or Domain-Driven Design (DDD) approaches (Schmidt &
212 Thiry, 2020). They emphasized on DDD and MDE; whether the selected studies cover DDD or
213 apply MDE; and which elements, principles, practices, and patterns authors applied; they did not
214 include metrics and quality attributes. Therefore, our work is complementary to their work.
215 Most previous literature reviews do not emphasize granularity, they concern general topics of
216 microservice architecture. To our knowledge, this is the first study focuses specifically on
217 classified and detailed research papers of microservice granularity, including quality attributes
218 that motivate working on it, methods/techniques to improve it, and metrics to measure it.
219

3. Survey methodology

221 A systematic literature review was carried out following the approach introduced by Kitchenham
222 define systematic literature reviews as "a form of secondary study that uses a well-defined
223 methodology to identify, analyze and interpret all available evidence related to a specific
224 research question in a way that is unbiased and (to a degree) repeatable" (Kitchenham, 2004).
225

3.1. Planning the systematic literature review

227 The objectives of this systematic literature review are defined as follows: first, to identify the
228 proposals that address the microservice granularity problem; second, to identify the metrics that
229 have been used to evaluate microservice granularity; and third, to analyze the quality attributes
230 addressed in those works to evaluate microservice granularity. Few studies or reviews
231 specifically address the problem of microservice granularity, and very few identify the metrics
232 along with the quality attributes addressed to assess microservice granularity.
233 A review protocol specifies the methods that will be used to undertake a specific systematic
234 review. We selected research papers through two queries strings used in the IEEE Xplore, ACM
235 Digital Library, and Scopus; then the papers were screening and reviewed, we applied inclusion
236 and exclusion criteria, next we tabulated the papers, the contribution of each selected paper was
237 detailed, we classified the papers, the metrics were described, and the quality attributes were
238 specified. The protocol components are listed below:

239
240   1. Define research questions.
241   2. Search Strategy.
242   3. Data extraction strategy
243   4. Study selection criteria and procedures.
244   5. Synthesis of the extracting data.
245
246   **1. Define the research questions.** The research questions (RQs) covered in this systematic
247   literature review were:
248
249   **RQ1:** Which approaches have been proposed to define microservice granularity?
250   **RQ2:** Which metrics are used to evaluate microservice granularity?
251   **RQ3:** Which quality attributes are addressed when researching microservice granularity?
252
253   The definition of the correct size and the functionalities that each microservice must contain,
254   affects the quality attributes of the system, and affects testing, deployment, and maintenance;
255   therefore, identify the metrics that are being used to evaluate the granularity is very important,
256   describe them and use them as references to propose other methods, techniques or methodologies
257   that allow defining the granularity of the microservices in a more objective way. Identifying the
258   quality attributes studied and how they were evaluated is an important reference for future work
259   to define the granularity of the microservices that are part of an application.
260
261   **2. Search strategy:** Two query strings were defined, which included alternative spellings for
262   "microservice" and any of the following words: granularity, size, dimensioning, or
263   decomposition. Additionally, to make the search terms that correspond to the research questions
264   more precise, another search string was included. In addition to granularity, it contained the
265   words "method", "technique", and "methodology". Therefore, the search strings were as follow:
266
267   **Query string 1 (QS1):** ("micro service" OR microservice) AND (granularity OR size OR
268   dimensioning OR decomposition).
269   According to Hassan, Bahsoon, and Kasman (2020), it is established that the granularity of
270   microservices is related to the size and dimension of the microservice, so these terms are
271   included in the search string. Additionally, the decomposition of monolithic applications to
272   microservices is an important research topic, so we include this word.
273
274   **Query string 2 (QS2):** ("micro service" OR microservice) AND "granularity" AND ("method"
275   OR "technique" OR "methodology"), targeting only research papers.
276   The main objective of the work was to identify the methods, techniques or methodologies used to
277   determine the microservices granularity.

278 QS1 and QS2 addresses all research question; for each of the proposals selected in each QS, the
279 metrics used, and the quality attributes addressed were identified. The query strings were used in
280 IEEE Xplore[1], ACM Digital Library[2] and Scopus[3], searching for papers' titles, abstracts and
281 keywords. The search in these databases, yield 969 results for QS1 and 146 results for QS2. The
282 search was performed in July 2020.
283
284 **3. Data extraction strategy.** First, papers were tabulated; second, duplicated papers were
285 removed; third, title, abstract, and conclusions of all papers were reviewed and analyzed. Each
286 coauthor of this report carried out this process.
287
288 **4. Study selection criteria and procedures.** We selected primary research papers that make a
289 specific proposal (methodology, model, technique, or method) about microservices granularity,
290 including migrations from monolith to microservices and decompositions of systems in
291 microservices. After obtaining the relevant studies, the inclusion and exclusion criteria were
292 applied (see table 1). We excluded any paper about monolith migrations that were not directly
293 related to the definition of microservice granularity. We also excluded papers that proposed
294 methods, techniques, or models for SOA, web services or mobile services.
295
296 **5. Synthesis of the extracting data.** Each of the selected papers was evaluated in full-text form,
297 taking detailed note of the work and the contributions made.
298

3.2. Conducting the systematic literature review

300 The review process was carried out as follow:
301
302 1. We download the full-text paper.
303 2. Each co-author read and review the paper.
304 3. Each co-author uses the classification criteria on the paper, using the table presented in
305   the appendix A; this was carried out by each co-author independently.
306 4. We discussed and analyzed the results obtained by each author, resolving doubts and
307   contradictions and the results are presented in appendix A.
308
309 The results of applying the research protocol is presented in figure 1. To analyze the works
310 presenting definitions of the granularity of microservices, classification criteria were defined.
311 These criteria were based on the classification performed by (Wieringa et al., 2006), and have
312 been widely used in previous systematic literature reviews: (Di Francesco, Lago & Malavolta,
313 2019), (Hamzehloui, Sahibuddin & Salah, 2019), (Hassan, Bahsoon & Kazman, 2020), (Vural,
314 Koyuncu & Guney, 2017). To answer the research questions, we added the classification criteria

---

[1] IEEE Xplore: https://ieeexplore.ieee.org/
[2] ACM Digital Library: https://dl.acm.org/
[3] Scopus: https://www.scopus.com/search/form.uri?display=basic

315 in each paper: metrics, stage of the development process, technique used, and quality attributes
316 studied or analyzed; namely:

317

318 • **Metrics used:** Which metrics are used to define the granularity of microservices?
319 • **Development process phases:** Phases of the development process on which the work
320   focuses.
321 • **Research strategies:** Includes solution proposal, validation research, experience paper,
322   opinion paper, philosophical paper, and evaluation research.
323 • **Approach:** Structural or behavioral aspects proposed in the papers to define the
324   granularity of microservices (Hassan, Bahsoon & Kazman, 2020).
325 • **Quality attribute studied:** The Quality attributes considered in the proposal, such as
326   performance, availability, reliability, scalability, maintainability, security, and complexity.
327 • **Research contribution:** Type of contribution made in the article; namely, method,
328   application, problem formulation, reference architecture, middleware, architectural
329   language, design pattern, evaluation, or comparison.
330 • **Experimentation type:** Type of experimentation used to validate the proposal; namely
331   experiment, case study, theoretical, or other.
332 • **Technique used:** This criterion describes the technique, method or model used to define
333   the granularity of the microservices.
334 • **Input data:** Type of input data used to identify the microservices (i.e. uses cases, logs,
335   source code, execution traces, among others)
336 • **Type of case study:** This criterion determines if the case study is a toy example
337   (hypothetical case) or a real-life case study. We identified the case study.
338 • **Automatization level:** This criterion determines the level of automation of the proposed
339   technique, if it is manual, automatic, or semi-automatic.

340

341 Finally, results were presented in four sections: first, the classification of the selected papers;
342 second, the main contributions and research gaps in sizing and definition of microservice
343 granularity were detailed; third, metrics were described an ordered by year and type; and fourth,
344 quality attributes were detailed, and results were discussed, leading to conclusions presented in
345 this article.

346

4. Results

348 The search process took place in July 2020. The search in the databases of scientific publications
349 when applying the search strings (QS1 and QS2) related to the granularity of the microservices
350 yield 969 and 146 works respectively (see table 2).

351

352 After applying the inclusion and exclusion criteria, 29 papers were selected that address the
353 definition of the granularity of microservices. (see table 3). The summarized results of this
354 systematic literature review are synthesized in figure 2.

355 For RQ1, we identified the papers that propose a method, model, or methodology to define the
356 microservice granularity; metrics are fundamental because they allow one to measure, monitor,
357 and evaluate any aspect of a microservice, thus defining or determining the appropriate
358 granularity of a microservice. For RQ2, we identified metrics used to evaluate microservice
359 granularity and their decomposition. Figure 2 shows the type and number of metrics and whether
360 it was applied to microservice, system, development process, or development team. These
361 metrics are detailed in section 4.3. Finally, for RQ3 we synthesized the works that address
362 quality attributes to evaluate microservices granularity.
363

4.1. Classification of the selected papers

365 Appendix A shows the tabulated data and the results of the evaluation of classification criteria.
366 Most papers were published in conference (86%), and only four (14%) were published in
367 journals. All selected papers were published between 2016 and the beginning of 2020 (2 in 2016,
368 7 in 2017, 6 in 2018, 12 in 2019, and 2 in 2020).
369 The development process phases addressed by each proposal are shown in figure 3. Several
370 papers emphasize more than one phase, (e.g., P10 focuses on development and deployment, as
371 befits a method for migration from monolith to microservices).
372 Most of the proposed methods focus on the design (79%) and development (38%) phase, with
373 only one addressing testing (3%). Migrations from monolithic architectures to microservices are
374 very common and important 19 of 29 papers (66%). The papers that do not address migration do
375 focus on identifying microservices in the design phase; therefore, defining the size and
376 granularity of microservices from the design phase on is key, because it has implications for
377 development, testing and deployment.
378 Further, most papers (79%) focus on the design phase; implicitly or explicitly, they suggest that
379 defining the "right" microservices granularity from the design phase on is fundamental.
380 However, some authors affirm that reasoning about microservices size and performance is not
381 possible at design time; indeed, (Hassan, Ali & Bahsoon, 2017) affirm that the expected behavior
382 of the system cannot be fully captured at design time.
383 On the research strategy (see figure 4), validation research and solution proposals account for
384 almost all (14 and 11, respectively); proposals that have been tested and validated in practice are
385 very few, namely P5 (a reference architecture) and P16 (a method for candidates microservice
386 identification from monolithic systems).
387 On the type of contribution (see figure 5), the vast majority (17 papers) proposed methods (59%),
388 some proposed methodologies (24%), few proposed reference architectures (7%) and problem
389 formulation (7%), Only one propose an evaluation or comparison (3%).
390 On the validation approach (see figure 6), most papers (69%) used case studies for validation and
391 evaluation, other papers use experiments (37%), and most of also used case studies.
392 More than half of studies (13 of 29) validated their proposals using realistic (but not real -
393 hypothetical) case studies, and the remaining almost-half (14 of 29) used real-life case studies,
394 real-life case studies achieve better validation than hypothetical case studies. Even better, some

395 studies (8) used actual open-source projects. The case studies found in the reviewed articles are
396 summarized in table 4; they are valuable resources to validate future research and to compare
397 new methods with those identified in this review.
398 In any case, other microservice-based datasets have been found to be beyond the reach of this
399 study; for example, (Rahman, Panichella & Taibi, 2019) shared a dataset composed of 20 open-
400 source projects using specific microservice architecture patterns; and (Marquez & Astudillo,
401 2018) shared a dataset of open source microservice-based projects when investigating actual use
402 of architectural patterns.
403 The most used case studies to validate the proposals were Kanban boards (P6, P20, P28) and
404 Money transfer (P6, P20, P28); they were used by 3 papers, followed by JPetsStore (P16, P29)
405 and Cargo tracking (P6, P24) which was used by 2 papers.
406

407 ## 5.2. RQ1: Approaches to define microservices granularity.
408 The granularity of microservices involves defining their size and the number that will be part of
409 the application. From the proposal of S. Newman (Newman, 2015), microservices follow the
410 principle of simple responsibility that says "Gather things that change for the same reason and
411 separate things that change for different reasons". The size, dimension, or granularity of
412 microservices have traditionally been defined as follows:
413

414 1. Trial and error, depending on the experience of the architect or developer.
415 2. According to the number of lines of code.
416 3. By implementation units.
417 4. By business capabilities.
418 5. By capabilities of the development team or teams.
419 6. Using domain-driven design.
420 7. Number of methods or exposed interfaces.
421

422 Richardson (2020) proposed four decomposition patterns, which allow for the decomposition of
423 an application into services: (1) Decompose by business capability: define services
424 corresponding to business capabilities; (2) decompose by subdomain: define services
425 corresponding to DDD subdomains; (3) self-contained service: design services to handle
426 synchronous requests without waiting for other services to respond. (4) service per team: each
427 service is owned by a team, which has sole responsibility for making changes, and ideally each
428 team has only one service (Richardson & microservices.io).
429 Zimmermann et al. (2019) proposed a microservice API patter (MAP) for API design and
430 evolution. The patterns are divided in five categories: (1) foundation, (2) responsibility, (3)
431 structure, (4) quality, and (5) evolution. These patterns are an important reference for developing
432 microservice-based applications. There is no specific pattern that helps to determine the number
433 of microservices and their size, that is, the number of operations it must contain (Zimmermann et
434 al., 2019).

435 The size of the microservice or optimal granularity is one of the most discussed properties and
436 there are few patterns, methods, or models to determine how small a microservice should be. In
437 this respect, some authors have addressed this problem and proposed the solutions summarized
438 in table 5.
439 The proposed techniques were classified into manual, semi-automatic, or automatic techniques;
440 manual techniques are methods, procedures, or methodologies performed by the architect or
441 developer decomposing systems following a few steps. Automatic techniques use some type of
442 algorithm to generate decomposition, and the system generates the decomposition. Semi-
443 automatic combine one part made manually and with another made automatically. Most papers
444 proposed manual procedures to identify the microservice granularity (15 papers); some proposals
445 were automatic (8 papers) and few proposals paper were semi-automatic (6 papers). The most
446 used case studies to validate the proposals were Kanban boards and Money Transfer (P6, P20,
447 P28).
448 The papers from 2017 and 2018 are mostly manual methods or methodologies that detail the way
449 to decompose or determine microservices, using DDD, domain engineering, or a specific
450 methodology. Later, the papers from 2019, and 2020 propose semi-automatic, and automatic
451 methods that use intelligent algorithms and machine learning mostly focused on migrations from
452 monolith to microservices. We can observe a chronological evolution in the proposals, the type
453 of techniques used to define the granularity of the microservices that are part of an application
454 are presented in figure 7, semantic similarity, machine learning, and genetic programing were the
455 most important techniques.
456
457

458 ## 5.3. RQ2: Metrics to evaluate the microservice granularity.
459 Software metrics allow us to measure and monitor different aspects and characteristics of a
460 software process and product, and there are metrics at the level of design, implementation,
461 testing, maintenance, and deployment. These metrics allow understanding, controling, and
462 improving of what happens during the development and maintenance of the software, to take
463 corrective and preventive actions.
464 In the methods and models identified, most of them (59%) used some metrics to determine
465 microservices granularity. We would have expected greater importance for metrics in automatic
466 methods to validate granularity in microservice-based applications and evaluate decompositions
467 yield by methods.
468 We identify metrics for coupling, cohesion, granularity, complexity, performance, use of
469 computational resources, development team, source code, and so on (see Table 6). We classified
470 them into four groups: about development team, about microservices development process, about
471 the system, and about each microservice. Most identified metrics (40) focused on a microservice,
472 and only two address the microservice development processes. There is a research gap for
473 metrics to evaluate the full development process of microservice-based applications and their
474 impact on the granularity of microservices.

475 The most used metrics are related to coupling (14 proposed metrics), followed by performance
476 and cohesion (13 metrics), next, computational resources metrics (8 metrics), and complexity
477 and source code (7 metrics); (see figure 8). Nine papers used coupling metrics (P11, P12, P13,
478 P14, P15, P16, P21, P22, P24), and seven papers used cohesion metrics (P12, P14, P16, P21,
479 P24, P27, P28), whereas performance metrics was used by five papers (P4, P10, P12, P21, P23);
480 Complexity metrics were considered by two papers (P8, P22), although they are fundamental
481 characteristic of microservices.
482 More proposals that include more complexity metrics are required, as well as metrics related to
483 the microservice development process. The other metrics were used by only one paper each. We
484 found that 11 papers used coupling or cohesion metrics, and 5 papers used both. Only one (P24)
485 used coupling, cohesion, and complexity metrics.
486 The size and number of microservices that compose an application directly affects its
487 maintainability. Automation of tests, continuous integration and deployment are essential
488 especially when microservices and many distributed systems must be managed independently by
489 each microservice.
490 Bogner, Wagner & Zimmermann (2017) performed a literature review to measure the
491 maintainability of software and identified metrics in four dominant design properties: size,
492 complexity, coupling, and cohesion. For service-based systems, they also analyzed its
493 application to systems based on microservices and presented a maintainability model for services
494 (MM4S), consisting of service properties related to automatically collectible service metrics
495 (Bogner, Wagner & Zimmermann, 2017b). The metrics proposed by them can be used or
496 adapted to determine the adequate granularity of the microservices that are going to be part of an
497 application.
498 Considering Bogner, Wagner & Zimmermann (2017), Candela et al. (2016) and related papers,
499 we detail the following metrics, which can be used or adapted to define the right granularity of
500 the microservices and to evaluate decompositions.
501
502 **5.2.1. Coupling metrics**
503 The coupling measures the degree of dependence of one software component in relation to
504 another. If there is a high degree of coupling, the software component cannot function properly
505 without the other component; furthermore, when we change a software component, we must
506 obligatorily change the other component. For these reasons when designing microservice-based
507 applications, we should look for a low degree of coupling between each microservice.
508 Mazlami, Cito & Leitner (2017) represented the information in the monolith and create an
509 undirected, edge-weighted graph G. Each graph edge has a weight defined by the weight
510 function; this weight function determines how strong the coupling between the neighboring
511 edges is, according to the coupling strategy in use (Mazlami, Cito & Leitner, 2017). These
512 coupling strategies can be used as metrics to define the granularity. These metrics are defined as
513 follows:
514

515 **Dependency weight.** Ahmadvand & Ibrahim (2016) said "dependency weight indicates the
516 frequency of using the dependency. For example, the dependency weight between a billing and
517 shopping cart is high, because with each call to the former a call is required to the latter. On the
518 other hand, the dependency weight between the billing service and a service managing the meta-
519 data of payment methods is low, because the former calls the latter only once a day"
520 (Ahmadvand & Ibrahim, 2016).
521
522 **Logical coupling.** Gall, Jazayeri & Krajewski (2003) coined the term logical coupling as a
523 retrospective measure of implicit coupling based on the revision history of an application source
524 code (Gall, Jazayeri & Krajewski, 2003). Mazlami, Cito & Leitner (2017) define the value of the
525 logical coupling is 1 if classes (C_1, C_2) have changed together in a certain commit. They use the
526 logical coupling aggregate which is the sum of the logical coupling for each pair of classes
527 (Mazlami, Cito & Leitner, 2017).
528
529 **Semantic coupling.** Basically, semantic coupling couples together classes that contain code
530 about the same things, i.e., domain model entities. The semantic coupling strategy can compute a
531 score that indicates how related two files are, in terms of domain concepts or "things" expressed
532 in code and identifiers (Mazlami, Cito & Leitner, 2017).
533
534 **Contributor count and contributor coupling.** The contributor coupling strategy aims to
535 incorporate the team-based factors into a formal procedure that can be used to cluster class files
536 according to the team-based factors (reduce communication overhead to external teams and
537 maximize internal communication and cohesion inside developer teams). It does so by analyzing
538 the authors of changes on the class files in the version control history of the monolith. The
539 procedure to compute the contributor coupling is applied to all class files. In the graph G
540 representing the original monolith M, the weight on any edge is equal to the contributor coupling
541 between two classes C_i and C_j that are connected in the graph. The weight is defined as the
542 cardinality of the intersection of the sets of developers that contributed to class C_i and C_j
543 (Mazlami, Cito & Leitner, 2017).
544
545 **Structural coupling.** Structural coupling consists of the number of classes outside package P_j
546 referenced by classes in the package P_j divided by the number of packages (Candela et al., 2016).
547
548
549 **Afferent coupling (Ca).** The number of classes in other packages (services) that depend upon
550 classes within the package (service) itself, as such it indicates the package's (service's)
551 responsibility (Martin, 2002) cited by (Li et al., 2019).
552

553   **Efferent coupling (Ce).** The number of classes in other packages (services), that the classes in a
554   package (service) depend upon, thus indicates its dependence on others (Martin, 2002) cited by
555   (Li et al., 2019).
556

557   **Instability (I).** Measures a package's (service's) resilience to change through calculating the
558   ratio of Ce and $Ce + Ca$. I = 0 indicates a completely stable package (service), whereas I = 1 a
559   completely unstable package (service) (Martin, 2002) cited by (Li et al., 2019).
560

561   **Coupling between microservice (CBM).** Taibi & Syst (2019) defined CBM extending the well-
562   known coupling between objects; this coupling can occur through method calls, field accesses,
563   inheritance, arguments, return types, and exceptions. They calculated the relative CBM for each
564   microservice by dividing the number of external links between the number of classes in the
565   microservices (Taibi & Syst, 2019).
566

567   **Integrating interface number (IFN).** The number of published interfaces of a service. The
568   smaller the IFN, the more likely the service is to assume a single responsibility. The IFN for the
569   system is the average of all IFN (Jin et al., 2019).
570

571   **Average domain redundancy (ADR).** A favorable microservice design avoids duplication of
572   responsibilities across services. Mazlami, Cito & Leitner (2017) computed the average domain
573   redundancy metric as a proxy to indicate the amount of domain-specific duplication or
574   redundancy between the services on a normalized scale between 0 and 1, where we favor service
575   recommendations with lower ADR values (Mazlami, Cito & Leitner, 2017).
576

577   **Absolute importance of the service (AIS).** The number of clients that invoke at least one
578   operation of a service's interface (Rud, Schmietendorf & Dumke, 2006).
579

580   **Absolute Dependence of the Service (ADS).** The number of other services that service S
581   depends on, i.e., the number of services from which S invokes at least one operation (Rud,
582   Schmietendorf & Dumke, 2006).
583

584   **Services interdependence in the system (SIY).** The number of service pairs that are bi-
585   directionally dependent on each other (Rud, Schmietendorf & Dumke, 2006).
586

587   **Dependencies composition.** Cojocaru, Uta & Oprescu (2019) defined dependencies composition
588   as "The test assesses how balanced outward dependencies are across the microservices, by
589   counting the outward dependencies each microservice has toward its peers. The algorithm
590   constructs a dependency graph of the system where each dependency represents a
591   communication path utilized for exchanging data between two components of the system"
592   (Cojocaru, Uta & Oprescu, 2019).

593

594 **Strongly connected components (SCC).** SCC identifies strongly connected components on the
595 previously mentioned dependency graph using Tarjan's algorithm. If cycles are detected in the
596 communication paths, then the respective services should be aggregated into one microservice.
597 The score is divided by the total number of microservices within the system (Cojocaru, Uta &
598 Oprescu, 2019).

599

600 **Silhouette score.** Nunes, Santos and Rito Silva (2019) define the silhouette score as "the
601 difference between the mean nearest-cluster distance a and the mean intra-cluster distance b
602 divided by the greatest of a and b. This score ranges its values from −1 to 1, representing
603 incorrect clustering (samples on wrong clusters) and highly dense clustering, respectively. This
604 metric creates a parallelism with the overall coupling of the clusters of the system, as our
605 objective was to obtain a high intra-cluster similarity and a low inter-cluster similarity, so the
606 partition between clusters is well defined" (Nunes, Santos & Rito Silva, 2019).

607

608 Perepletchikov et al. (2007) proposed a set of design-level metrics to measure the structural
609 attribute of coupling in service-oriented systems, which can be adapted to microservices
610 (Perepletchikov et al., 2007).

611

612 **5.2.2. Cohesion metrics**
613 Cohesion and coupling are two contrasting goals. A solution balancing high cohesion and low
614 coupling is the goal for developers. (Candela et al., 2016) employed a two-objective approach
615 aimed at maximizing package cohesion and minimizing package coupling. They used class
616 dependencies and structural information to measure the structural cohesion, which can be
617 adapted to microservices. The metrics used in the related work are listed below:

618

619 **Lack of cohesion.** Lack of cohesion of classes for the J_{th} package (P_j) measured as the number
620 of pairs of classes in P_j with no dependency between them (Candela et al., 2016). Al-Debagy and
621 Martinek (2020) used the metric in a different way as "It is based on Henderson-Sellers's lack of
622 cohesion metrics (Al-Debagy & Martinek, 2020). But the proposed version is modified to be
623 applicable for microservices' APIs. It works by finding how many times a microservice has used
624 a specific operation's parameter, divided by the product of the number of operations multiplied
625 by the number of unique parameters". Candela et al. (2016) used lack of structural cohesion to
626 measure the lack of cohesion between classes divided by the number of packages, and they also
627 defined a lack of conceptual cohesion as a metric (Candela et al., 2016).

628

629 **Relational cohesion.** Relational cohesion was defined as "the ratio between the number of
630 internal relations and the number of types in a package (service). Internal relations include
631 inheritance between classes, invocation of methods, access to class attributes, and explicit

632   references like creating a class instance. Higher numbers of RC indicate higher cohesion of a
633   package (service)**"** (Larman, 2012) cite by (Li et al., 2019)**.**

634

635   **Cohesion at the domain level (CHD).** The cohesiveness of interfaces provided by a service at
636   the domain level. The higher the CHD, the more functionally cohesive that service is (Jin et al.,
637   2019).

638

639   **Average cohesion at the domain level (Avg. CHD).** The average of all CHD values within the
640   system (Jin et al., 2019).

641

642   **Cohesion at the message level (CHM).** The cohesiveness of the interfaces published by a
643   service at the message level. The higher a service's CHM, the more cohesive the service is, from
644   an external perspective. CHM is the average functional cohesiveness (Jin et al., 2019).

645

646   **Service interface data cohesion (SIDC).** The cohesion of a given service S with respect to the
647   similarity of parameter data types of its interface's operations (Perepletchikov, Ryan &
648   Frampton, 2007).

649

650   **Service Interface Usage Cohesion (SIUC).** The cohesion of a given service S based on the
651   invocation behavior of clients using operations from its interface (Perepletchikov, Ryan &
652   Frampton, 2007).

653

654   **Entities composition.** According to Cojocaru, Uta, and Oprescu (2019), "entities composition
655   assesses whether the entities are equally distributed among the proposed microservices and no
656   duplicates, which might break the cohesion, exist. They define an entity as the class, or action of
657   the service" (Cojocaru, Uta & Oprescu, 2019).

658

659   **Relation composition.** According to Cojocaru, Uta, and Oprescu (2019) "relation composition
660   assesses the quantitative variation in published language per relation. It applies the concept of
661   relative assessment to entities shared between the services via their communication paths. The
662   test identifies services communicating much more data than their peers, and thus potential
663   communication bottlenecks" (Cojocaru, Uta & Oprescu, 2019).

664

665   **Responsibilities composition.** Cojocaru, Uta, and Oprescu (2019) stated that the responsibilities
666   composition "assesses to what extent the use case responsibilities are equally distributed among
667   the proposed microservices. It uses the coefficient of variation between the number of use case
668   responsibilities of each microservice. Services having relatively more responsibility may imply
669   low cohesion: a service providing multiple actions violates the single responsibility principle"
670   (Cojocaru, Uta & Oprescu, 2019).

671

672 **Semantic similarity:** According to Cojocaru, Uta, and Oprescu (2019) "semantic similarity uses
673 lexical distance assessment algorithms to flag the services that contain unrelated components or
674 unrelated actions hindering cohesion" (Cojocaru, Uta & Oprescu, 2019).
675
676 Perepletchikov, Ryan, and Frampton (2007) " reviewed categories of cohesion initially proposed
677 for object-oriented software in order to determine their conceptual relevance to service-oriented
678 designs"; and proposed a set of metrics for cohesion that can be adapted for microservices
679 (Perepletchikov, Ryan & Frampton, 2007).
680
681 **5.2.3. Complexity metrics**
682 The complexity of microservices should be low, so that they can be changed within several
683 weeks, rewritten, and improved quickly. If the complexity is high, then the cost of change is
684 higher. Measuring complexity is fundamental for developing microservice-based applications.
685 The metrics used by the authors of the papers are listed below.
686
687 **Function points.** A method for measuring the size of a software. A function point count is a
688 measurement of the amount of functionality that software will provide (Totalmetrics.com).
689
690 **COSMIC function points.** Focuses on data movements between different layers. One of the
691 benefits of the COSMIC method is that it can estimate the size in the planning phase, based on
692 the user's functional requirements. The four main data group types are: entry, exit, read and
693 write. The COSMIC function point calculation is aimed at measuring the system at the time of
694 planning. This size calculation can be used for estimating efforts (Vural, Koyuncu & Misra,
695 2018).
696
697 **Total response for service (TRS).** The sum of all responses for operation (RFO) values for all
698 operations of the interface of service S (Perepletchikov et al., 2007).
699
700 **Number of singleton clusters (NSC).** Nunes, Santos, and Rito Silva (2019) said "that having
701 more than 2 singleton clusters is considered negative. Considering a final microservice
702 architecture with clear functional boundaries established, it is likely that there are not two
703 services in which their content is a single domain entity" (Nunes, Santos & Rito Silva, 2019).
704
705 **Maximum cluster size (MCS).** Nunes, Santos, and Rito Silva, (2019) "MCS should not be
706 bigger than half of the size of the system. Even with a cluster size inside this range, there is also
707 a dependency regarding the number of entity instances that are part of the aggregate" (Nunes,
708 Santos & Rito Silva, 2019).
709
710 **5.2.4. Performance metrics**

711   Performance is a critical point of microservice-based applications. The selected papers used eight
712   performance metrics:
713

714   **The number of calls or requests.** The number of times that a microservice is called.
715

716   **The number of rejected requests.** The number of times that a microservice does not respond or
717   exceeds the time limit.
718

719   **Response or execution time.** The execution time of the invoked service.
720

721   **Interaction number (IRN).** The number of calls for methods among all pairs of extracted
722   microservices. The smaller is the IRN, the better the quality of candidate microservices as a low
723   IRN reflects loose coupling (Saidani et al., 2019).
724

725   **Number of executions.** The number of test requests sent to the system or microservices.
726

727   **Maximum request time.** The maximum time for a request (output) made from one microservice
728   to another.
729

730   **Maximum response time:** The maximum response time is that of a call (input) or request to the
731   system or microservice. It is the time to process a response to another microservice.
732

733   **Number of packets sent:** The packets sent to the system or microservice.
734

735   Ren et al. (2018) (P29) they used package analysis (PA), static structure analysis (SSA), class
736   hierarchy analysis (CHA), static call graph analysis (SCGA), and combined static and dynamic
737   analysis (CSDA) to evaluate migration performance. However, they did not explain the details of
738   the performance analysis test or the metrics they used (Ren et al., 2018).
739

740   **5.2.5. Other quality attributes metrics.**
741   Few metrics were directly related to quality attributes. The metrics proposed in the revised works
742   are defined as follows:
743

744   **Cost of quality assurance.** It can be calculated by adding up the time spent by testers validating
745   not only the new features but also the non-regression on existing ones, along with the time spent
746   on release management (Gouigoux & Tamzalit, 2017).
747

748   **Cost of deployment.** The time spent by operational teams to deploy a new release, in man-days;
749   it decreases greatly as teams automate deployment (Gouigoux & Tamzalit, 2017).
750

751 **Security impact.** The security policy applied to requirements or services. Assets and threats
752 identified lead to deployed security mechanisms, which form security policies. Ahmadvand and
753 Ibrahim (2016) mapped the identified policies to the corresponding functional requirements,
754 mainly based on their access to the system assets. Security impact is a qualitative value (low,
755 medium, high) (Ahmadvand & Ibrahim, 2016).
756
757 **Scalability impact.** Ahmadvand & Ibrahim (2016) define th Scalability impact as the required
758 level of scalability (high, medium, low) to implement a a functional requirement or service.
759 Defining the requirements at design time for a software system to be scalable is a challenging
760 task. (Ahmadvand & Ibrahim, 2016) think that a requirements engineer should answer a question
761 such as "What is the anticipated number of simultaneous users for this functionality?"
762 (Ahmadvand & Ibrahim, 2016).
763
764 **5.2.6. Computational resources metrics**
765 The computational resources are all the software and hardware elements necessary for the
766 operation of the microservice-based applications. The proposed metrics are listed.
767
768 **Average of memory.** The average memory consumption for each microservice or application.
769
770 **Average of disk.** The average of disk consumption for each microservice or application.
771
772 **Average of network.** The average network bandwidth consumption for the entire system; Kb/s
773 used by system or microservice (De Alwis et al., 2019).
774
775 **Average of CPU.** The average of the CPU consumption by the system or microservice.
776
777 **Service composition cost (SCC).** Homay et al. (2019) stated that "identifying which existing
778 functionalities in the service are consuming more resources is not an easy task. Therefore, we
779 suggest relying on each request that a service provider receives from a service consumer.
780 Because each request is a chain of stats or activities that needs to be satisfied inside of the service
781 provider to generate a related response. The cost-of-service composition for the service s will be
782 equal to the maximum cost of requests (routes)" (Homay et al., 2019).
783
784 **Service decomposition cost (SDC).** According to Homay et al., 2019, "By refining a service
785 into smaller services, we will make some drawbacks. The SDC is a function that calculates the
786 overhead of refining the service s into smaller pieces" (Homay et al., 2019).
787
788 **5.2.7. Team metrics**
789 Each microservice can be developed by a different team, and with different programming
790 languages and database engines. It is important to consider metrics that allow analysis of

791   microservices' granularity and its impacts on the development team. The proposed metrics in the
792   analyzed are as follows:

793

794   **Team size reduction (TSR).** Reduced team size translates to reduced communication overhead
795   and thus more productivity and the team's focus can be directed toward the actual domain
796   problem and service for which it is responsible. TSR is a proxy for this team-oriented quality
797   aspect. Let *RM* be a microservice recommendation for a monolith *M*. *TSR* is computed as the
798   average team size across all microservice candidates in the *RM* divided by the team size of the
799   original monolith *M* (Mazlami, Cito & Leitner, 2017).

800

801   **Commit count.** The number of commits in the code repository made by the developers.

802

803   We found very few metrics related to the development team. This can be an interesting topic for
804   future research.

805

806   **5.2.8. Source code metrics**
807   The source code is one of the most important sources for analyzing certain characteristics of an
808   application. Some authors have used it to identify microservices and define their granularity. The
809   proposed metrics are described below:

810

811   **Code size in lines of code.** The total size of the code in the repository, in terms of in lines of
812   code, microservices' lines of code, or application's lines of code.

813

814   **The number of classes per microservice.** Helps to understand how large the identified
815   microservice is and to identify if any microservice is too big compared to others. The number of
816   classes should be minimized because a smaller number of classes implies more independent
817   development of the microservice (Taibi & Syst, 2019).

818

819   **The number of duplicated classes.** In some cases, two execution traces will have several
820   classes in common. The number of duplicated classes helps one to reason about the different
821   slicing options, considering not only the size of the microservices but also the number of
822   duplications, that will be then reflected in the microservices' development. Duplicated classes
823   should be avoided since duplication adds to the system's size and maintenance(Taibi & Syst,
824   2019).

825

826   **Internal co-change frequency (ICF).** How often entities within a service change together as
827   recorded in the revision history. A higher ICF means that the entities within this service will be
828   more likely to evolve together. The ICF is the average of all ICFs within the system (Jin et al.,
829   2019).

830

831   **External co-change frequency (ECF).** How often entities assigned to different services change
832   together, according to the revision history. A lower ECF score means that entity pairs located in
833   different services are expected to evolve more independently. Similarly, ECF is the average ECF
834   value of all services within the system (Jin et al., 2019).
835

836   **The ratio of ECF to ICF (REI).** The ratio of co-change frequency across services vs. the co-
837   change frequency within services. The ratio is expected to be less than 1.0 if co-changes happen
838   more often inside a service than across different services. The smaller the ratio is, the less likely
839   co-changes are across services, and the extracted services tend to evolve more independently.
840   Ideally, all co-changes should happen inside the services. REI is calculated as ECF divided by
841   ICF (Jin et al., 2019).
842

843   **Modularity quality measure.** The modularity of a component or service can be measured from
844   multiple perspectives, such as structural, conceptual, historical, and dynamic
845   dimensions(Candela et al., 2016). They extend the modularity quality (MQ), as defined by
846   (Mancoridis et al., 1998) as structural and conceptual dependencies, using structural modularity
847   quality and conceptual modularity quality to assess the modularity of service candidates.
848   Structural modularity quality (SMQ) measures the quality of modularity from a structural
849   perspective. The higher the SMQ, the better modularized the service is. On the other hand,
850   conceptual modularity quality (CMQ), similarly measures modularity quality from a conceptual
851   perspective. The higher the CMQ, the better (Jin et al., 2019).
852

853   **5.2.9. Granularity metrics**
854   Measuring granularity is complex. Granularity is related to size, including the number of
855   functionalities or services that the application or microservice will have. It is also related to
856   coupling and cohesion. Being more granular implies that microservice has no dependencies and
857   can function independently, as an independent and encapsulated piece. Six granularity metrics
858   were identified:
859

860   **Weighted service interface count (WSIC):** WSIC(S) is the number of exposed interface
861   operations of service *S*. The default weight is set to 1. Alternate weighting methods, which need
862   to be validated empirically, can take into consideration the number and the complexity of data
863   types of parameters in each interface. (Hirzalla, Cleland-Huang & Arsanjani, 2009). WSIC(S) is
864   the number of exposed interface operations of service S. Operations can be weighted based on
865   the number of parameters or their granularity (e.g. a complex nested object) with the default
866   weight being set to 1 (Bogner, Wagner & Zimmermann, 2017a).
867

868   **Component Balance (CB):** The CB is a system-level metric to evaluate the appropriateness of
869   granularity, i.e. if the number and size uniformity of the components (in this case, services) are in
870   a favorable range for maintainability (Bouwers et al., 2011).

871
872 **Operation number (OPN):** The OPN is used to compute the average number of public
873 operations exposed by an extracted microservice to other candidate microservices. The smaller
874 the OPN is the better (Saidani et al., 2019).
875
876 **Number of microservices:** The number of microservices that are part of the system or
877 application.
878
879 **Lines of code:** The lines of code measure the number of lines of code in the microservice.
880 Additionally, it may consider the total size of the code in the repository.
881
882 **Number of nanoentities:** Cojocaru, Uta, and Oprescu (2019) stated that "the number of
883 nanoentities (attributes or fields of a class) computes the number of nanoentities assigned to each
884 proposed service, storing the result as a floating-point parameterized list. The list's length is
885 equal to the number of services found in the system model specification file" (Cojocaru, Uta &
886 Oprescu, 2019).
887
888 The fundamentals of microservices suggest that they must have low coupling, high cohesion, and
889 low complexity. Based on the described metrics, a model or method could be defined that uses
890 artificial intelligence to determine the most appropriate dimensioning and size for microservices.
891 Some have already been defined, mainly for migrations from monoliths to microservices. The
892 number of microservices, their size, and their computational complexity directly affect the use of
893 computational resources and therefore their cost of deployment. This is an interesting topic for
894 future research.
895 In conclusion, some papers used metrics to evaluate the granularity of the microservices,
896 including coupling, cohesion, number of calls, number of requests, and response time, although
897 few methods or techniques use complexity as a metric, even though it seems fundamental for
898 microservices. More research that considers design-level metrics is needed to define the
899 granularity of the microservices that are part of an application, as well as research proposing
900 models, methods, or techniques to determine the most appropriate granularity.
901

902 ## 5.4. RQ3: Quality attributes to define the microservice granularity.
903 Quality attributes are essential for today's applications. Availability, performance, automatic
904 scaling, maintainability, security, and fault tolerance are essential features that every application
905 must handle. An architecture based on microservices allows independent management of quality
906 attributes, according to the specific need of each microservice. This is one of the main
907 advantages compared to monolithic architectures.
908 The size and number of microservices that compose an application directly affect its quality
909 attributes. Creating more microservices may affect maintainability because testing costs will
910 increase, even more so if automated testing is not available. Moreover, performance may also be

911   affected by having to integrate and process data from several distributed applications. Clearly,
912   quality attributes are impacted by microservices granularity and should be considered when
913   defining a model, method, or technique to determine granularity (see Fig. 9).
914   Surprisingly, 62% of the identified proposals did not consider or report any quality attributes at
915   all. Of those that did, scalability and performance were the most considered (7 papers, 24%),
916   followed by maintainability and availability (2 papers, 7%); and lastly, reliability (fault
917   tolerance), security, functionality, and modularity with only one paper each. More research is
918   needed that considers quality attributes to define the granularity of the microservices comprising
919   an application. Security and fault tolerance are key attributes that microservice-based
920   applications must handle, few works addressed these features (see table 7).
921   We grouped the software quality attributes into the following two categories: firstly, according to
922   runtime characteristics, (scalability, performance, reliability, availability, and functionality),
923   which are observable during execution; and secondly according to software as an artifact
924   characteristic (maintainability, modularity, reusability), which are not observable during
925   execution (Bass, Clemens & Katzman, 1998) (Astudillo, 2005); run time characteristics were the
926   most used ones, having been addressed by 8 papers (P2, P4, P10, P11, P12, P13, P21 y P29);
927   only two papers addressed software artifact characteristics (P3 y P16); only one paper used both
928   artifact and runtime characteristics (P16). Therefore, more proposals are required to define
929   microservices granularity considering both runtime and software artifact characteristics.
930
931   **5.3.1. Runtime characteristics**
932   In this section, we detail the runtime quality attributes and the way they were addressed by the
933   papers, whether they used metrics to evaluate their proposals.
934   •   Scalability, performance, and reliability (fault tolerance) were used by only one paper (P2).
935   P2 proposed a re-implementation of otoo.de (a real-life case study). They defined the
936   granularity through vertical decomposition, they used DevOps including continuous
937   deployment, to deliver features quickly to customers. Team organization is crucial for
938   success, this organization was based on Conway's Law. Full automation of quality assurance
939   and software deployment allows for early fault and error detection, thus reducing repair times
940   both during development and during operations (Hasselbring & Steinacker, 2017). This
941   paper did not propose metrics for evaluation.
942   •   Scalability and performance were used by two papers (P10, P29); P10 proposed an automatic
943   decomposition method, which was based on a black-box approach that mines the application
944   access logs using a clustering method to discover URL partitions having similar performance
945   and resource requirements. Such partitions were mapped to microservices (Ahmadvand &
946   Ibrahim, 2016). The metrics used in this paper were performance metrics (response time SLO
947   violations, number of calls, number of rejected requests, and throughput) and computational
948   resource metrics (Avg. CPU, number of virtual machines used, and allocated virtual
949   machines). P29 used the source code and the runtime logs in a semi-automatic method, it
950   used granularity, performance, and source code metrics to evaluate the decompositions. They

951   presented a program analysis-based method to migrate monolith legacy applications to
952   microservices architecture; this method used a function call graph, a Markov chain model to
953   represent migration characteristics, and a k-means hierarchical clustering algorithm (Ren et
954   al., 2018).
955   •   Only performance was used by two papers (P4, P13). P4 examined the granularity problem
956   of the microservice and explored its effect on the latency of the application. Two approaches
957   for the deployment of microservices were simulated; the first one with microservices in a
958   single container, and the second one with microservices divided into separate containers.
959   They discussed the findings in the context of the Internet of Things (IoT) application
960   architectures (Shadija, Rezai & Hill, 2017); that paper corresponds to an evaluation or
961   comparison, it is not a method to define the microservice granularity; it used performance
962   metrics (response time and the number of calls). P13 presented three formal coupling
963   strategies and embedded those in a graph-based clustering algorithm: (1) logical coupling, (2)
964   semantic coupling, and (3) contributor coupling. The coupling strategies rely on meta-
965   information from monolithic code bases to construct a graph representation of the monoliths
966   that are in turn processed by the clustering algorithm to generate recommendations for
967   potential microservice candidates in a refactoring scenario; P13 was the only one that
968   proposed development team based metrics; logical coupling, average domain redundancy,
969   contributor coupling, semantic coupling, commit count, contributor count, and lines of code
970   were the metrics used by this paper.
971   •   Scalability and security were used by one paper (P11). P11 proposed a methodology,
972   consisting of a series of steps and activities that must be carried out to identify the
973   microservices that will be part of the system. It is based on the use cases and the analysis
974   made by the architect in terms of the scalability and security of each use case, as well as the
975   dependencies with the other use cases (Ahmadvand & Ibrahim, 2016); this paper used the
976   following metrics: dependency weight, security impact, and scalability impact (qualitative
977   metrics).
978   •   Scalability, performance, and availability were addressed by two papers (P12, P21). P12
979   presented discovery techniques that help identifying the appropriate parts of consumer-
980   oriented business systems that could be redesigned as microservices with desired
981   characteristics such as high cohesion, low coupling, high scalability, high availability, and
982   high processing efficiency (De Alwis et al., 2018). They proposed microservice discovery
983   algorithms and heuristics. It was an automatic method that used coupling (structural
984   coupling), cohesion (lack of cohesion), computational resources (avg. memory, avg. disk),
985   and performance metrics (number of requests, execution time). P21 was a semi-automatic
986   method, a genetic algorithm with semantic similarity based on DISCO and non-dominated
987   sorting genetic algorithm-II (NSGAII). That paper presented four microservice patterns,
988   namely object association, exclusive containment, inclusive containment, and subtyping for
989   'greenfield' (new) development of software while demonstrating the value of the patterns for
990   'brownfield' (evolving) developments by identifying prospective microservices (De Alwis et

991   al., 2019). The metrics used by that paper were: structural coupling, lack of cohesion, the
992   average CPU, the average of the network, number of executions, and the number of packets
993   sent.
994

5.3.2. Software as an artifact characteristic

996   Only two software as an artifact characteristic were used, which were maintainability and
997   modularity. Only maintainability was used by one paper (P3); whereas maintainability, and
998   modularity were used by (P16), which proposed the most complete method.
999   P3 used a balance between the cost of quality assurance and the cost of deployment for defining
1000   microservices granularity, it was a manual method. The choice of granularity should be based on
1001   the balance between the costs of quality assurance and the cost of deployment (Gouigoux &
1002   Tamzalit, 2017).
1003   P16 presented a framework that consists of three major steps: (1) extracting representative
1004   execution traces, (2) identifying entities using a search-based functional atom grouping
1005   algorithm, and (3) identifying interfaces for service candidates (Jin et al., 2019). They also
1006   presented a comprehensive measurement system to quantitatively evaluate service candidate
1007   quality in terms of functionality, modularity, and evolvability. P16 proposed an automatic
1008   method, which used a search-based functional atom grouping algorithm and a non-dominated
1009   sorting genetic algorithm-II (NSGA II). The evaluation metrics were coupling (Integrating
1010   interface number), cohesion at message level, cohesion at domain level, structural modularity
1011   quality, conceptual modularity quality, internal co-change frequency (ICF), external Co-change
1012   frequency (ECF), and the ratio of ECF to ICF.
1013

5.3.3. Quality attributes and artificial intelligence

1015   In some cases, artificial intelligence techniques are being used to improve the quality attributes
1016   of microservices. For example:
1017   Alipour and Liu (2017) proposed two machine learning algorithms and predicted the resource
1018   demand of microservice backend systems, as emulated by a Netflix workload reference
1019   application. They proposed a microservice architecture that encapsulates monitoring functions of
1020   metrics and learning of workload patterns. Then, this service architecture is used to predict the
1021   future workload for making decisions about resource provisioning (Alipour & Liu, 2017).
1022   Prachitmutita et al. (2018) proposed a new self-scaling framework based on the predicted
1023   workload, with an artificial neural network, a recurrent neural network, and a resource scaling
1024   optimization algorithm used to create an automated system to manage the entire application with
1025   Infrastructure-as-a-service (IaaS) (Prachitmutita et al., 2018).
1026   Ma et al. (2018) proposed an approach, called scenario-based microservice retrieval (SMSR), to
1027   recommend appropriate microservices for users based on the Behavior-driven Development
1028   (BDD) test scenarios written by the user. The proposed service retrieval algorithm is based on
1029   word2vec, an automatic learning method widely used in natural language processing (NLP) to
1030   perform service filtering and calculate service similarity (Ma et al., 2018).

1031 Abdullah, Iqbal and Erradi (2019) proposed a complete automated system for breaking down an
1032 application into microservices, implementing microservices using appropriate resources, and
1033 automatically scaling microservices to maintain the desired response time (Abdullah, Iqbal &
1034 Erradi, 2019).
1035 Artificial intelligence can help to improve and control different characteristics of microservices,
1036 especially those related to improving quality attributes. Some proposals have been made in this
1037 regard, but more research is needed.
1038 Finally, we identified the automatic and semi-automatic methods, which used metrics and
1039 addressed some quality attribute to define the granularity (see table 8). Only six papers meet
1040 those conditions (P10, P12, P13, P16, P21, and P29), and were the more suitable methods to
1041 define the granularity of microservices.
1042 Also, we identified semi-automatic methodologies, which used metrics to define the granularity,
1043 only two papers were found (P14 and P22); there were no automatic methodologies, most of
1044 them were manual methodologies (P11, P17, P18, and P26) and only one was semi-automatic
1045 but it did not use metrics (P25).
1046 P14 was a data flow-driven decomposition algorithm. Their methodology, first, the use case
1047 specification and business logics were analyzed based on requirements; second, the detailed
1048 dataflow diagrams (DFD) at different levels and the corresponding process-datastore version of
1049 DFD (DFD PS) are constructed from business logics based on requirement analysis; third, we
1050 designed an algorithm to automatically condense the DFD PS to a decomposable DFD, in which
1051 the sentences between processes and data stores are combined; last but not least, microservice
1052 candidates were identified and extracted automatically from the decomposable DFD (Li et al.,
1053 2019). The metrics used by P14 were coupling (afferent coupling, efferent coupling, instability)
1054 and cohesion (relational cohesion) metrics. P14 did not address any quality attribute directly.
1055 P22 proposed a clustering algorithm applied to aggregate domain entities. It used coupling
1056 (silhouette score) and complexity (number of singleton clusters, maximum cluster size) metrics.
1057 The authors proposed an approach to the migration of monolith applications to a microservice
1058 architecture that focused on the impact of the decomposition on the monolith business logic
1059 (Nunes, Santos & Rito Silva, 2019).
1060

5. Discussion

1062 The use of artificial intelligence techniques to determine the appropriate granularity or to identify
1063 the microservices that will be part of an application is a growing trend; this is especially true of
1064 machine learning clustering algorithms and genetic algorithms, with an emphasis on semantic
1065 similarity to group the microservices that refer to the same entity. Domain engineering and DDD
1066 are still among the most used techniques.
1067 Migration of software systems implies many architectural decisions that should be systematically
1068 evaluated to assess concrete trade-offs and risks (Cruz et al., 2019). In these cases, the beginning
1069 is given from a monolithic system that must be decomposed into microservices, and that
1070 monolithic system has important data sources that allow for the identification and evaluation of

1071 the candidate microservices. These sources are mainly: the source code, the use cases, the
1072 database, the logs, and the execution traces. It should be noted that the development of
1073 microservice-based applications is closely related to agile practices and DevOps, and none of the
1074 input data that are being considered in the proposed methods correspond to agile artifacts such as
1075 user stories, product backlog, iteration planning and others. Therefore, research work is needed
1076 at this point.
1077 The migration from monolith to microservices is a topic with much interest and is widely
1078 studied. In contrast, the design and development of microservice-based applications from scratch
1079 have few related proposals. The proposed methods emphasize artifacts available at run time,
1080 development, deployment, or production, which are hardly available when starting a project from
1081 scratch at design time.
1082 The development of microservice-based applications from scratch resembles component-based
1083 development (Vera-Rivera & Rojas Morales, 2010), in which microservices are reusable
1084 software components. In (Vera-Rivera, 2018) we characterized the process of developing
1085 applications based on microservices, identifying two fundamental parts, first the development of
1086 each microservice and then the development of applications based on those microservices.
1087 The definition of adequate granularity is fundamental to the development of microservices-based
1088 applications (Vera-Rivera, 2018). The granularity of a monolith is not the most optimal and
1089 defining an operation by microservice is also not optimal. Hence, if an application that offers 100
1090 operations should have 100 microservices, it would not be optimal either, due to latency,
1091 performance, and management of this large distributed system. The optimal granularity is
1092 somewhere in between the monolithic application and the operation by microservice system, this
1093 granularity should be defined according to the characteristics of the application, the development
1094 team, the non-functional requirements, the available resources, and design, development, and
1095 operation trade-offs.
1096 The research gaps focus on proposing techniques or methods that allow for the evaluation of
1097 granularity and its impact on tests, considering security controls, fault tolerance mechanisms and
1098 DevOps. By managing more microservices or larger microservices, testing can be slower and
1099 more tedious. Moreover, the pipelines of continuous integration and deployment would be more
1100 complex. Determining the appropriate number of microservices and their impact on continuous
1101 deployment is an interesting research topic. Few works address these issues.
1102 In addition, few papers use as input data or analysis units the artifacts used in agile development,
1103 such as user stories, product backlog, release planning, Kanban board and its data, to propose
1104 agile methods or new practices that allow for determination or evaluation of the microservices
1105 that will be part of the application. None of the proposed works focuses on agile software
1106 development.
1107 Several interesting works have been proposed, but there are still few specific, actionable
1108 proposals; more research is needed to propose design patterns, good practices, more complete
1109 models, methods, or tools that can be generalized to define microservices granularity considering
1110 metrics, quality attributes and trade-offs.

5.1. Research trends:

We detail de research trends according to the analyzed papers, the trends summarized as follow:

- The most used techniques to define microservice granularity included machine learning clustering, semantic similarity, genetic programming, and domain engineering.
- The most used research strategies were validation research and solution proposals.
- The most used validation method was the case study, although some studies used experimental evaluations. We summarized the case studies found in the reviewed papers because they are valuable resources with which to validate future research and to compare new methods (See table 4). The most common case studies were Kanban boards, Money transfer, JPetsStore and Cargo Tracking, which are either hypothetical or open-source projects.
- The use of metrics was evidenced to evaluate the granularity of the microservices comprising an application. Performance and coupling were the most used metrics; they help to identify microservices and their granularity more objectively.
- Migrations from monoliths to microservices have been widely studied. Methods and techniques have been proposed to decompose applications into microservices, with the source code, logs, execution traces, and even use cases used as input data. These methods are used mainly during design and development time.
- Scalability and performance were the most addressed quality attributes in the reviewed papers; they are fundamental for microservice-based applications. Finally, the main reason to migrate a monolithic application to microservices is precisely to improve performance and scalability, followed by fault tolerance, maintainability, and modularity.

5.2. Research gaps:

Research gaps allow us to propose new research works and future work, we identify the research gaps, which are listed as follow:

- Research works that include techniques or methods to evaluate granularity and its impact on tests, while also considering security controls, fault-tolerance mechanisms, and DevOps.
- Metrics were grouped into four categories: development team, development process, microservice-based application (system), and microservice itself. Few metrics were found for the development team or development process, more research is necessary in these groups.
- Few methods have been proposed to define the most adequate microservices granularity at testing or deployment time.
- More research is required that uses agile development artifacts as inputs, (i.e. user stories, product backlog, release planning, Kanban boards, and their data), to propose new agile practices to define or assess microservices' granularity. None of the proposal identified in this survey focused on agile software development.

6. Threats to validity

6.1. External validity

We express a threat to external validity regarding the search and selection of primary papers, which may not be representative of the state of the art in the definition of the granularity of microservices, to reduce this risk we used a systematic and well defined process, using two search strings so that the papers obtained are representative, to define and select the papers that were included in our review, each of the authors made their selection and tabulation independently, then in common agreement and group discussion were selected by applying the inclusion and exclusion criteria. In addition, the systematic literature review process carried out corresponds to the classic and widely used in other reviews, proposed by Kitchenham (Kitchenham, 2004), also our study significantly include research papers have undergone a rigorous peer-review process, which is a well-established requirement for high quality publications, so the selected paper may be representatives to define the state-of-the-art of microservices granularity definition. For each paper obtained from the query strings, the reason why it was included or excluded from the review was defined. We did not include grey literature. By using a systematic method already established and widely used in other reviews, the replicability of our study is guaranteed, and the process was rigorously followed to reduce this threat.

6.2. Internal validity

In order to reduce the researcher bias a pre-defined protocol were defined (See figure 1). The classification criteria for the selected papers were carefully selected and they were defined in other literature reviews, these literature reviews were explained in the related work section. We download the selected papers; they were shared with all authors for review. The papers were summarized, we detail the contribution and made the classification and analysis based on full-text papers. We specified a paper ID, the used technique, the input data, the full paper summary and synthesis, the description of the proposal, the journal or conference where it was published, and observations and comments. We tabulated the papers using the classification criteria explained in section 4, we review each selected paper based on the interpretation of the contribution raised in each one, then we grouped the papers. To reduce the selection bias this process was reviewed for each author independently.

The threats to data synthesis and results were mitigated by having a unified classification and description scheme and following a standard protocol where a systematic process was done and externally evaluated. The data extraction process was aligned with our research questions, also we applied the guidelines of a classic systematic literature review, following a research protocol, thus making our research easy to check and replicate.

Conclusions

This systematic literature review identified the main contributions and research gaps regarding the dimensioning and definition of the granularity of microservices comprising an application. Methods, methodologies, and techniques to determine the granularity of microservices were identified.

Microservice granularity research is at a Wild West stage: no standard definition exist, development-operation trade-offs are unclear, there is little notion of continuous granularity improvement, and conceptual reuse is scarce (e.g. few methods seem applicable or replicable in projects other than the first to use them). These gaps in granularity research offer clear options for research on continuous improvement of the development and operation of microservice-based systems.

We propose a microservice granularity definition, first by its size or dimensions, meaning the number of operations (services) exposed by the microservice, along with the number of microservices that are part of the whole application and by its complexity and dependencies. The goal is to have low coupling, low complexity, and high cohesion of the microservices. Defining the most optimal granularity for microservices can significantly improve performance, maintainability, scalability, network use and consumption, computational resources, and cost, because microservices mainly are deployed in the cloud.

As future work we will propose "Microservice Backlog" a model and techniques to define and evaluate the microservice granularity at design time, using metrics to evaluate the granularity. We want to develop a genetic programing technique and a semantic grouping algorithm to group the user stories of the product backlog into candidate microservices, so the architect or development team can evaluate the candidate decomposition of the application.

Acknowledgements

This work was partially supported by Colombia's Ministry of Science and Technology (Minciencias - Colciencias) through doctoral scholarship "753 - Formación de capital humano de alto nivel para el departamento Norte de Santander"; by the Francisco de Paula Santander University (Cúcuta, Colombia) throughthe doctoral studies commission number 14 of 2016; by the Universidad del Valle (Cali,Colombia); and by ANID (Chile) through PIA/APOYO AFB180002.

References

Abdullah M, Iqbal W, Erradi A. 2019. Unsupervised learning approach for web application auto-decomposition into microservices. *Journal of Systems and Software* 151:243–257. DOI: 10.1016/j.jss.2019.02.031.

Ahmadvand M, Ibrahim A. 2016. Requirements Reconciliation for Scalable and Secure Microservice (De)composition. In: *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*. IEEE, 68–73. DOI: 10.1109/REW.2016.026.

Al-Debagy O, Martinek P. 2020. Extracting Microservices' Candidates from Monolithic Applications: Interface Analysis and Evaluation Metrics Approach. In: *2020 IEEE 15th International Conference of System of Systems Engineering (SoSE)*. IEEE, 289–294. DOI:

1232   10.1109/SoSE50414.2020.9130466.
1233   Alipour H, Liu Y. 2017. Online machine learning for cloud resource provisioning of
1234   microservice backend systems. In: *2017 IEEE International Conference on Big Data (Big*
1235   *Data)*. IEEE, 2433–2441. DOI: 10.1109/BigData.2017.8258201.
1236   De Alwis AAC, Barros A, Fidge C, Polyvyanyy A. 2019. Business object centric microservices
1237   patterns. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in*
1238   *Artificial Intelligence and Lecture Notes in Bioinformatics)*. Springer, 476–495. DOI:
1239   10.1007/978-3-030-33246-4_30.
1240   De Alwis AAC, Barros A, Polyvyanyy A, Fidge C. 2018. Function-Splitting Heuristics for
1241   Discovery of Microservices in Enterprise Systems. In: 37–53. DOI: 10.1007/978-3-030-
1242   03596-9_3.
1243   Astudillo H. 2005. Five ontological levels to describe and evaluate software architectures.
1244   *Revista Facultad de Ingeniería Universidad de Tarapacá* 13:69–73.
1245   Bass L, Clemens P, Katzman R. 1998. *Software Architecture in Practice*. Addison Wesley.
1246   Bogner J, Wagner S, Zimmermann A. 2017a. Automatically measuring the maintainability of
1247   service- and microservice-based systems. In: *Proceedings of the 27th International*
1248   *Workshop on Software Measurement and 12th International Conference on Software*
1249   *Process and Product Measurement on - IWSM Mensura '17*. 107–115. DOI:
1250   10.1145/3143434.3143443.
1251   Bogner J, Wagner S, Zimmermann A. 2017b. Towards a practical maintainability quality model
1252   for service-and microservice-based systems. In: *Proceedings of the 11th European*
1253   *Conference on Software Architecture Companion Proceedings - ECSA '17*. New York, New
1254   York, USA: ACM Press, 195–198. DOI: 10.1145/3129790.3129816.
1255   Bouwers E, Correia JP, Van Deursen A, Visser J. 2011. Quantifying the analyzability of
1256   software architectures. *Proceedings - 9th Working IEEE/IFIP Conference on Software*
1257   *Architecture, WICSA 2011*:83–92. DOI: 10.1109/WICSA.2011.20.
1258   Candela I, Bavota G, Russo B, Oliveto R. 2016. Using cohesion and coupling for software
1259   remodularization: Is it enough? *ACM Transactions on Software Engineering and*
1260   *Methodology* 25. DOI: 10.1145/2928268.
1261   Cojocaru M, Uta A, Oprescu AM. 2019. MicroValid: A validation framework for automatically
1262   decomposed microservices. In: *Proceedings of the International Conference on Cloud*
1263   *Computing Technology and Science, CloudCom*. IEEE Computer Society, 78–86. DOI:
1264   10.1109/CloudCom.2019.00023.
1265   Cruz P, Astudillo H, Hilliard R, Collado M. 2019. Assessing Migration of a 20-Year-Old System
1266   to a Micro-Service Platform Using ATAM. *2019 IEEE International Conference on*
1267   *Software Architecture Companion (ICSA-C)*:174–181. DOI: 10.1109/icsa-c.2019.00039.
1268   Di Francesco P. 2017. Architecting microservices. *Proceedings - 2017 IEEE International*
1269   *Conference on Software Architecture Workshops, ICSAW 2017: Side Track*
1270   *Proceedings*:224–229. DOI: 10.1109/ICSAW.2017.65.
1271   Di Francesco P, Lago P, Malavolta I. 2019. Architecting with microservices: A systematic
1272   mapping study. *Journal of Systems and Software* 150:77–97. DOI:
1273   10.1016/j.jss.2019.01.001.
1274   Gall H, Jazayeri M, Krajewski J. 2003. CVS release history data for detecting logical couplings.
1275   In: *International Workshop on Principles of Software Evolution (IWPSE)*. Institute of
1276   Electrical and Electronics Engineers Inc., 13–23. DOI: 10.1109/IWPSE.2003.1231205.
1277   Ghofrani J, Lübke D. 2018. Challenges of Microservices Architecture : A Survey on the State of

1278   the Practice. *Zeus 2018* 10th ZEUS.
1279   Gouigoux JP, Tamzalit D. 2017. From monolith to microservices: Lessons learned on an
1280   industrial migration to a web oriented architecture. In: *Proceedings - 2017 IEEE*
1281   *International Conference on Software Architecture Workshops, ICSAW 2017: Side Track*
1282   *Proceedings*. 62–65. DOI: 10.1109/ICSAW.2017.35.
1283   Hamzehloui MS, Sahibuddin S, Salah K. 2019. A Systematic Mapping Study on Microservices
1284   Mohammad. In: Saeed F, Gazem N, Mohammed F, Busalim A eds. *IRICT: International*
1285   *Conference of Reliable Information and Communication Technology 2018*. Advances in
1286   Intelligent Systems and Computing. Cham: Springer International Publishing, 1079–1090.
1287   DOI: 10.1007/978-3-319-99007-1.
1288   Hassan S, Ali N, Bahsoon R. 2017. Microservice Ambients: An Architectural Meta-Modelling
1289   Approach for Microservice Granularity. In: *Proceedings - 2017 IEEE International*
1290   *Conference on Software Architecture, ICSA 2017*. IEEE, 1–10. DOI:
1291   10.1109/ICSA.2017.32.
1292   Hassan S, Bahsoon R, Kazman R. 2020. Microservice Transition and its Granularity Problem: A
1293   Systematic Mapping Study. *Software: Practice and Experience*:1–31. DOI:
1294   10.1002/spe.2869.
1295   Hasselbring W, Steinacker G. 2017. Microservice architectures for scalability, agility and
1296   reliability in e-commerce. In: *Proceedings - 2017 IEEE International Conference on*
1297   *Software Architecture Workshops, ICSAW 2017: Side Track Proceedings*. IEEE Computer
1298   Society, 243–246. DOI: 10.1109/ICSAW.2017.11.
1299   Hirzalla M, Cleland-Huang J, Arsanjani A. 2009. A Metrics Suite for Evaluating Flexibility and
1300   Complexity in Service Oriented Architectures. In: Springer, Berlin, Heidelberg, 41–52.
1301   DOI: 10.1007/978-3-642-01247-1_5.
1302   Homay A, de Sousa M, Zoitl A, Wollschlaeger M. 2020. Service Granularity in Industrial
1303   Automation and Control Systems. In: *2020 25th IEEE International Conference on*
1304   *Emerging Technologies and Factory Automation (ETFA)*. IEEE, 132–139. DOI:
1305   10.1109/ETFA46521.2020.9212048.
1306   Homay A, Zoitl A, De Sousa M, Wollschlaeger M, Chrysoulas C. 2019. Granularity cost
1307   analysis for function block as a service. *IEEE International Conference on Industrial*
1308   *Informatics (INDIN)* 2019-July:1199–1204. DOI: 10.1109/INDIN41052.2019.8972205.
1309   Jamshidi P, Pahl C, Mendonca NC, Lewis J, Tilkov S. 2018. Microservices: The Journey So Far
1310   and Challenges Ahead. *IEEE Software* 35:24–35. DOI: 10.1109/MS.2018.2141039.
1311   Jin W, Liu T, Cai Y, Kazman R, Mo R, Zheng Q. 2019. Service Candidate Identification from
1312   Monolithic Systems based on Execution Traces. *IEEE Transactions on Software*
1313   *Engineering* X:1–1. DOI: 10.1109/TSE.2019.2910531.
1314   Kitchenham B. 2004. *Procedures for performing systematic reviews*. DOI: 10.1.1.122.3308.
1315   Kulkarni N, Dwivedi V. 2008. The role of service granularity in a successful soa realization - A
1316   case study. In: *Proceedings - 2008 IEEE Congress on Services, SERVICES 2008*. 423–430.
1317   DOI: 10.1109/SERVICES-1.2008.86.
1318   Larman C. 2012. *Applying UML and patterns: an introduction to object oriented analysis and*
1319   *design and interative development*.
1320   Li S, Zhang H, Jia Z, Li Z, Zhang C, Li J, Gao Q, Ge J, Shan Z. 2019. A dataflow-driven
1321   approach to identifying microservices from monolithic applications. *Journal of Systems and*
1322   *Software* 157. DOI: 10.1016/j.jss.2019.07.008.
1323   Ma S-P, Chuang Y, Lan C-W, Chen H-M, Huang C-Y, Li C-Y. 2018. Scenario-Based

1324   Microservice Retrieval Using Word2Vec. In: *2018 IEEE 15th International Conference on*
1325   *e-Business Engineering (ICEBE)*. IEEE, 239–244. DOI: 10.1109/ICEBE.2018.00046.

1326 Mancoridis S, Mitchell BS, Rorres C, Chen Y, Gansner ER. 1998. Using automatic clustering to
1327   produce high-level system organizations of source code. In: *Proceedings. 6th International*
1328   *Workshop on Program Comprehension. IWPC'98 (Cat. No.98TB100242)*. IEEE Comput.
1329   Soc, 45–52. DOI: 10.1109/WPC.1998.693283.

1330 Marquez G, Astudillo H. 2018. Actual Use of Architectural Patterns in Microservices-Based
1331   Open Source Projects. In: *Proceedings - Asia-Pacific Software Engineering Conference,*
1332   *APSEC*. IEEE Computer Society, 31–40. DOI: 10.1109/APSEC.2018.00017.

1333 Martin RC. 2002. *Agile Software Development Principles, Patterns, and Practices Alan Apt*
1334   *Series*. Upper Saddle River, New Jersey 07458: Pearson Education, Inc.

1335 Mazlami G, Cito J, Leitner P. 2017. Extraction of Microservices from Monolithic Software
1336   Architectures. In: *2017 IEEE International Conference on Web Services (ICWS)*. IEEE,
1337   524–531. DOI: 10.1109/ICWS.2017.61.

1338 Newman S. 2015. *Building Microservices*. O'Reilly Media, Inc.

1339 Nunes L, Santos N, Rito Silva A. 2019. From a Monolith to a Microservices Architecture: An
1340   Approach Based on Transactional Contexts. In: *13th European Conference, ECSA 2019.*
1341   *Lectures Notes in Computer Science 11681*. Springer, 37–52. DOI: 10.1007/978-3-030-
1342   29983-5_3.

1343 Osses F, Márquez G, Astudillo H. 2018. An exploratory study of academic architectural tactics
1344   and patterns in microservices: A systematic literature review. *Avances en Ingenieria de*
1345   *Software a Nivel Iberoamericano, CIbSE 2018*:71–84.

1346 Pautasso C, Zimmermann O, Amundsen M, Lewis J, Josuttis N. 2017. Microservices in Practice,
1347   Part 1: Reality Check and Service Design. *IEEE Software* 34:91–98. DOI:
1348   10.1109/MS.2017.24.

1349 Perepletchikov M, Ryan C, Frampton K. 2007. Cohesion Metrics for Predicting Maintainability
1350   of Service-Oriented Software. In: *Seventh International Conference on Quality Software*
1351   *(QSIC 2007)*. IEEE, 328–335. DOI: 10.1109/QSIC.2007.4385516.

1352 Perepletchikov M, Ryan C, Frampton K, Tari Z. 2007. Coupling Metrics for Predicting
1353   Maintainability in Service-Oriented Designs. In: *2007 Australian Software Engineering*
1354   *Conference (ASWEC'07)*. IEEE, 329–340. DOI: 10.1109/ASWEC.2007.17.

1355 Prachitmutita I, Aittinonmongkol W, Pojjanasuksakul N, Supattatham M, Padungweang P. 2018.
1356   Auto-scaling microservices on IaaS under SLA with cost-effective framework. In: *2018*
1357   *Tenth International Conference on Advanced Computational Intelligence (ICACI)*. IEEE,
1358   583–588. DOI: 10.1109/ICACI.2018.8377525.

1359 Rahman MI, Panichella S, Taibi D. 2019. A Curated Dataset of Microservices-Based Systems.
1360   In: *Joint Proceedings of the Inforte Summer School on Software Maintenance and Evolution*
1361   *(CEUR Workshop Proceedings; Vol. 2520). CEUR-WS*. Tempere, Finland: SSSME-2019,.

1362 Ren Z, Wang W, Wu G, Gao C, Chen W, Wei J, Huang T. 2018. Migrating web applications
1363   from monolithic structure to microservices architecture. In: *ACM International Conference*
1364   *Proceeding Series*. New York, New York, USA: Association for Computing Machinery, 1–
1365   10. DOI: 10.1145/3275219.3275230.

1366 Richardson C, microservices.io.Microservice Architecture pattern. *Available at*
1367   *https://microservices.io/patterns/microservices.html* (accessed December 12, 2019).

1368 Rud D, Schmietendorf A, Dumke RR. 2006. Product Metrics for Service-Oriented
1369   Infrastructures. In: *Conference: Applied Software Measurement. Proceedings of the*

1370   *International Workshop on Software Metrics and DASMA Software Metrik Kongress*
1371   *(IWSM/MetriKon 2006)*.

1372   Saidani I, Ouni A, Mkaouer MW, Saied A. 2019. Towards Automated Microservices Extraction
1373   Using Muti-objective Evolutionary Search. In: *17th International Conference Service-*
1374   *Oriented Computing. Lectures Notes in computer science 11895*. Springer, Cham, 58–63.
1375   DOI: 10.1007/978-3-030-33702-5_5.

1376   Salah T, Jamal Zemerly M, Chan Yeob Yeun, Al-Qutayri M, Al-Hammadi Y. 2016. The
1377   evolution of distributed systems towards microservices architecture. In: *2016 11th*
1378   *International Conference for Internet Technology and Secured Transactions (ICITST)*.
1379   IEEE, 318–325. DOI: 10.1109/ICITST.2016.7856721.

1380   Schmidt RA, Thiry M. 2020. Microservices identification strategies : A review focused on
1381   Model-Driven Engineering and Domain Driven Design approaches. In: *2020 15th Iberian*
1382   *Conference on Information Systems and Technologies (CISTI)*. IEEE, 1–6. DOI:
1383   10.23919/CISTI49556.2020.9141150.

1384   Shadija D, Rezai M, Hill R. 2017. Microservices: Granularity vs. Performance. In: *UCC '17*
1385   *Companion Companion Proceedings of the10th International Conference on Utility and*
1386   *Cloud Computing*. Austin, Texas, USA: ACM, 215–220. DOI: 10.1145/3147234.3148093.

1387   Soldani J, Tamburri DA, Heuvel W-J Van Den. 2018. The Pains and Gains of Microservices: A
1388   Systematic Grey Literature Review. *Journal of Systems and Software* 146:215–232. DOI:
1389   10.1016/j.jss.2018.09.082.

1390   Taibi D, Syst K. 2019. From Monolithic Systems to Microservices: A Decomposition
1391   Framework based on Process Mining. *International Conference on Cloud Computing and*
1392   *Service Science - CLOSER 2019*.

1393   Totalmetrics.com.Total Metrics Approach - Function points. *Available at*
1394   *https://www.totalmetrics.com/our_approach* (accessed August 5, 2020).

1395   Vera-Rivera FH. 2018. A development process of enterprise applications with microservices.
1396   *Encuentro internacional de Ciencias Aplicadas e Ingeniería (Universidad - Industria) –*
1397   *EISI 2018. Journal of Physics: Conference Series* 1126:012017. DOI: 10.1088/1742-
1398   6596/1126/1/012017.

1399   Vera-Rivera FH, Gaona Cuevas CM, Astudillo H. 2019. Desarrollo de aplicaciones basadas en
1400   microservicios: tendencias y desafíos de investigación. *Revista Ibérica de Sistemas e*
1401   *Tecnologias de Informação* E23:107–120.

1402   Vera-Rivera FH, Rojas Morales FA. 2010. Propuesta de aplicación de la Ingeniería del Software
1403   Basada en Componentes en el desarrollo de software empresarial. *Revista Iteckne* 7:128–
1404   135.

1405   Villamizar M, Garcés O, Castro H, Verano M, Salamanca L, Gil S. 2015. Evaluating the
1406   Monolithic and the Microservice Architecture Pattern to Deploy Web Applications in the
1407   Cloud Evaluando el Patrón de Arquitectura Monolítica y de Micro Servicios Para Desplegar
1408   Aplicaciones en la Nube. *10th Computing Colombian Conference*:583–590. DOI:
1409   10.1109/ColumbianCC.2015.7333476.

1410   Vural H, Koyuncu M, Guney S. 2017. A Systematic Literature Review on Microservices. In:
1411   Springer, Cham, 203–217. DOI: 10.1007/978-3-319-62407-5_14.

1412   Vural H, Koyuncu M, Misra S. 2018. A Case Study on Measuring the Size of Microservices. In:
1413   Laganá A, Gavrilova ML, Kumar V, Mun Y, Tan CJK, Gervasi O eds. *International*
1414   *Conference on Computational Science and Its Applications - ICCSA 2018*. Lecture Notes in
1415   Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 454–463. DOI:

1416   10.1007/b98054.
1417 Wieringa R, Maiden N, Mead N, Rolland C. 2006. Requirements engineering paper
1418   classification and evaluation criteria: a proposal and a discussion. *Requirements*
1419   *Engineering* 11:102–107. DOI: 10.1007/s00766-005-0021-6.
1420 Zimmermann O. 2017. Microservices tenets: Agile approach to service development and
1421   deployment. *Computer Science - Research and Development* 32:301–310. DOI:
1422   10.1007/s00450-016-0337-0.
1423 Zimmermann O, Stocker M, Zdun U, Lübke D, Pautasso C. 2019.Microservice API Patterns.
1424   *Available at https://www.microservice-api-patterns.org/introduction* (accessed December
1425   17, 2019).
1426

Figure 1

Systematic literature review method.

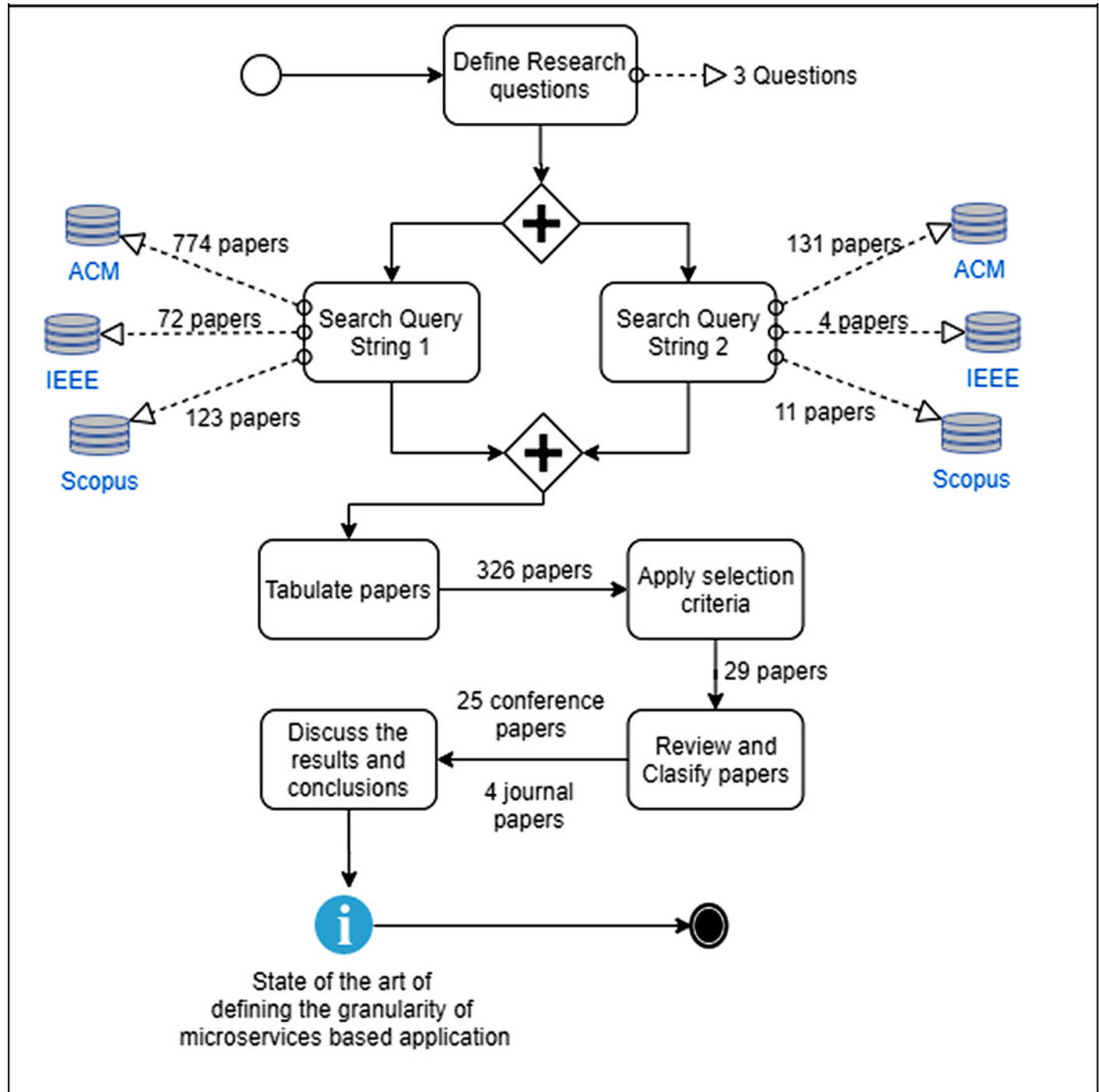



Figure 2

Infographic summary of approaches to defining microservice granularity.

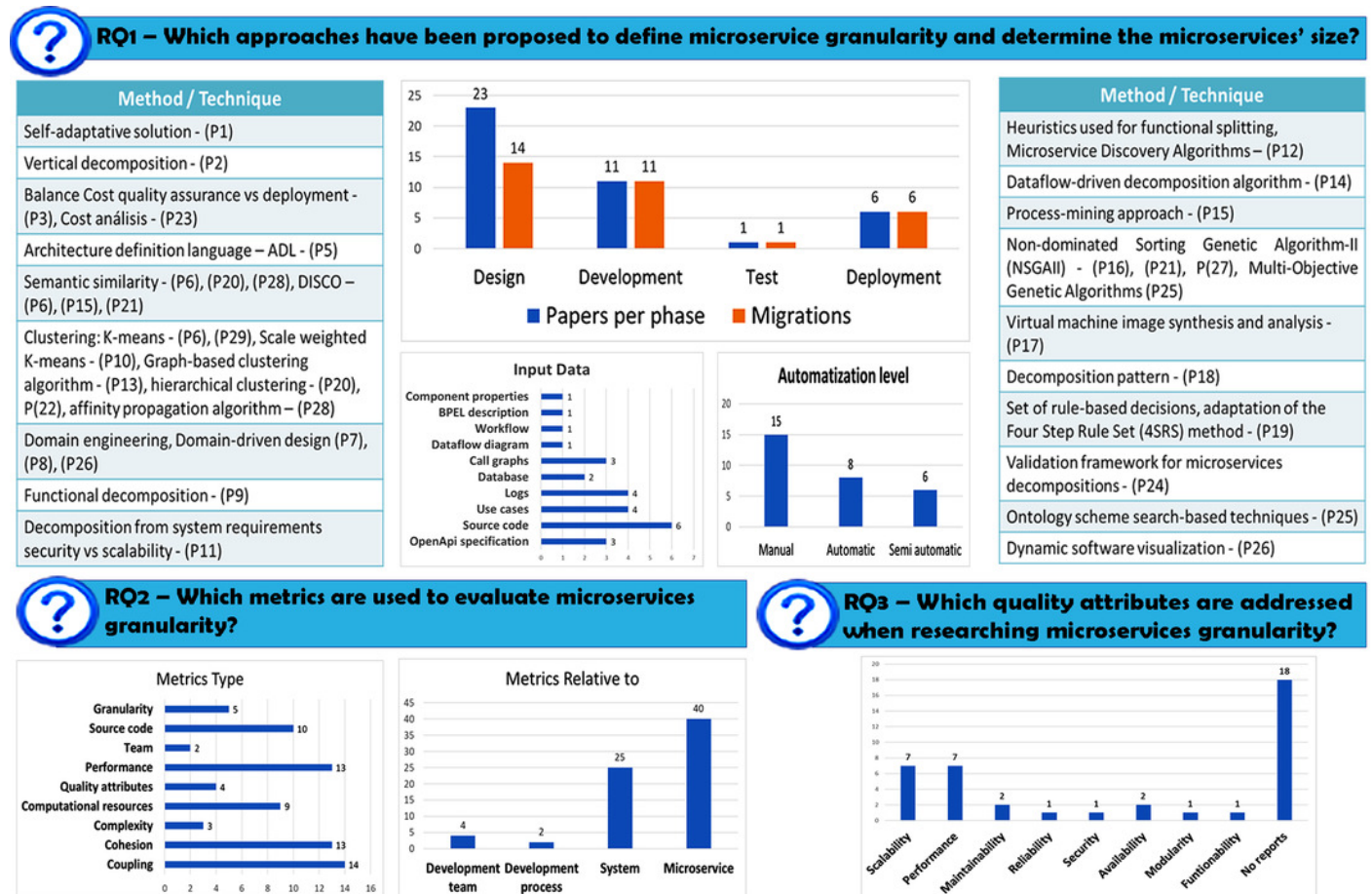



Figure 3

Amount of papers by development process phase.

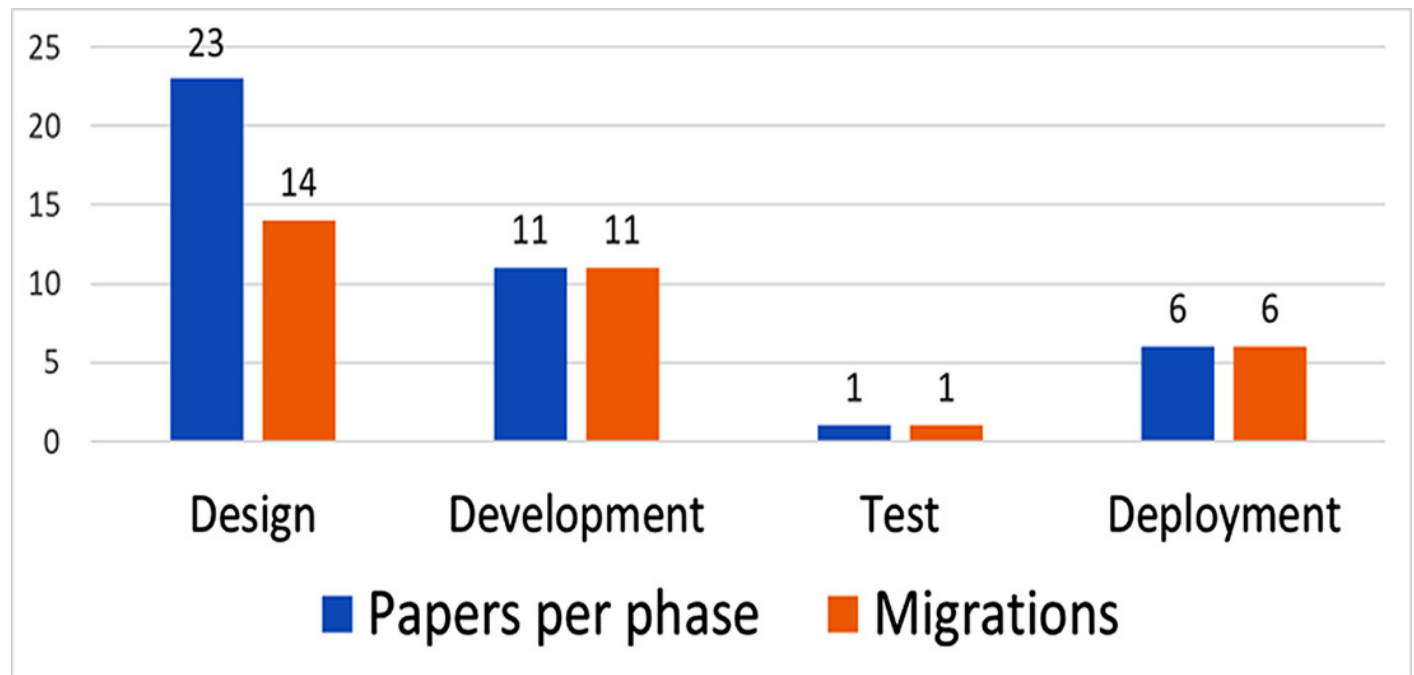



Figure 4

Number of papers by research strategy.

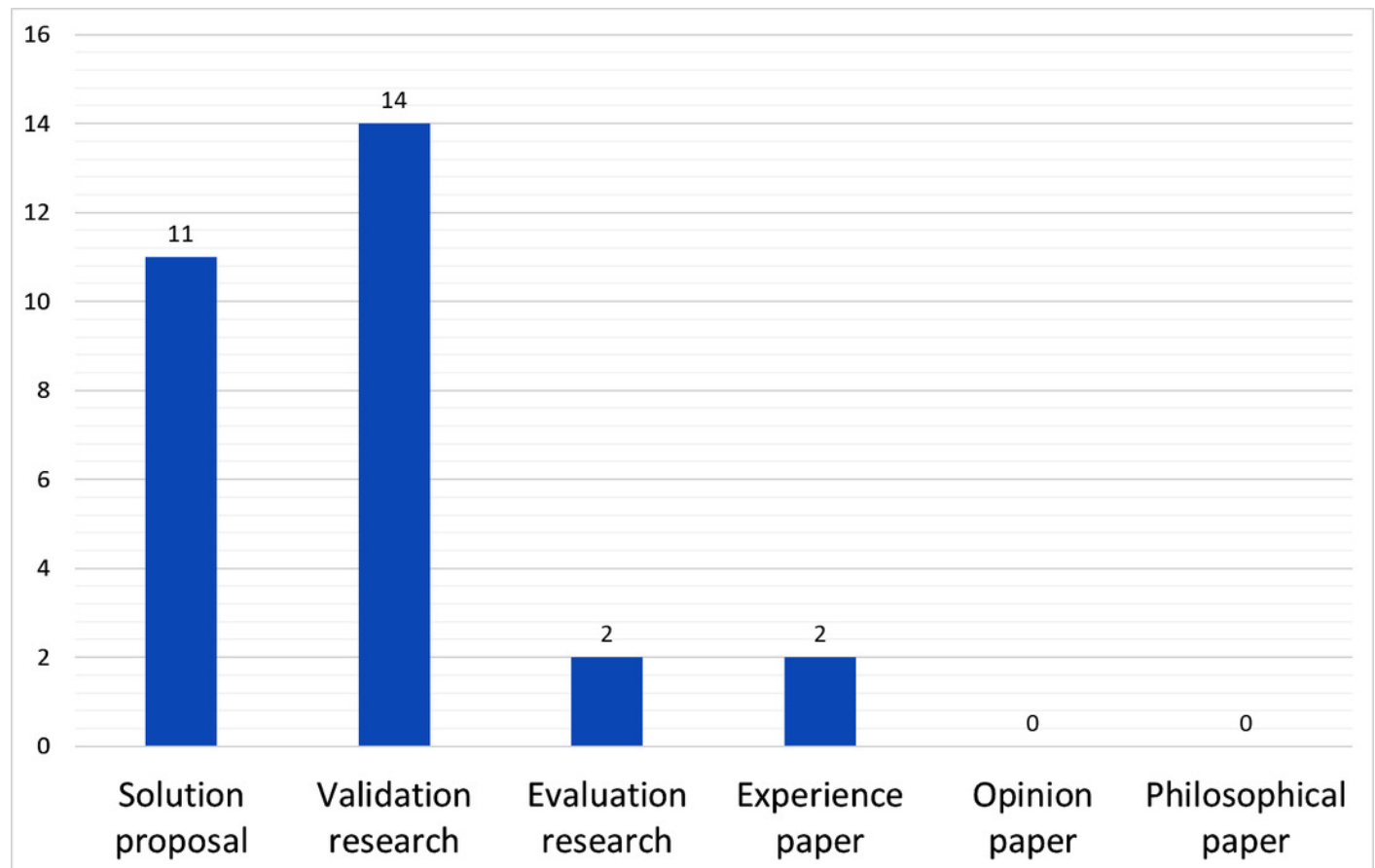



Figure 5

Number of papers by contribution type.

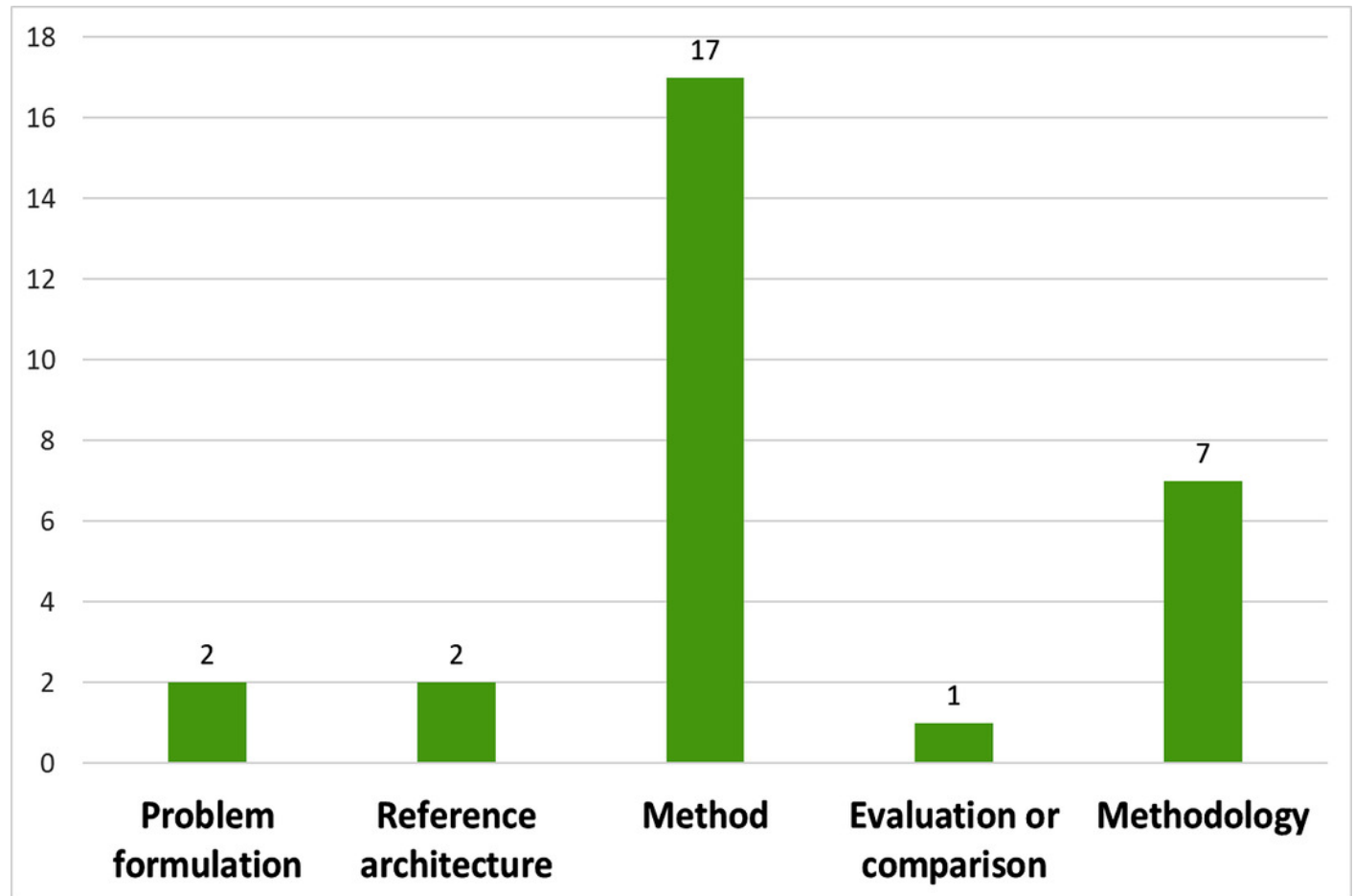



Figure 6

Number of papers by validation approach.

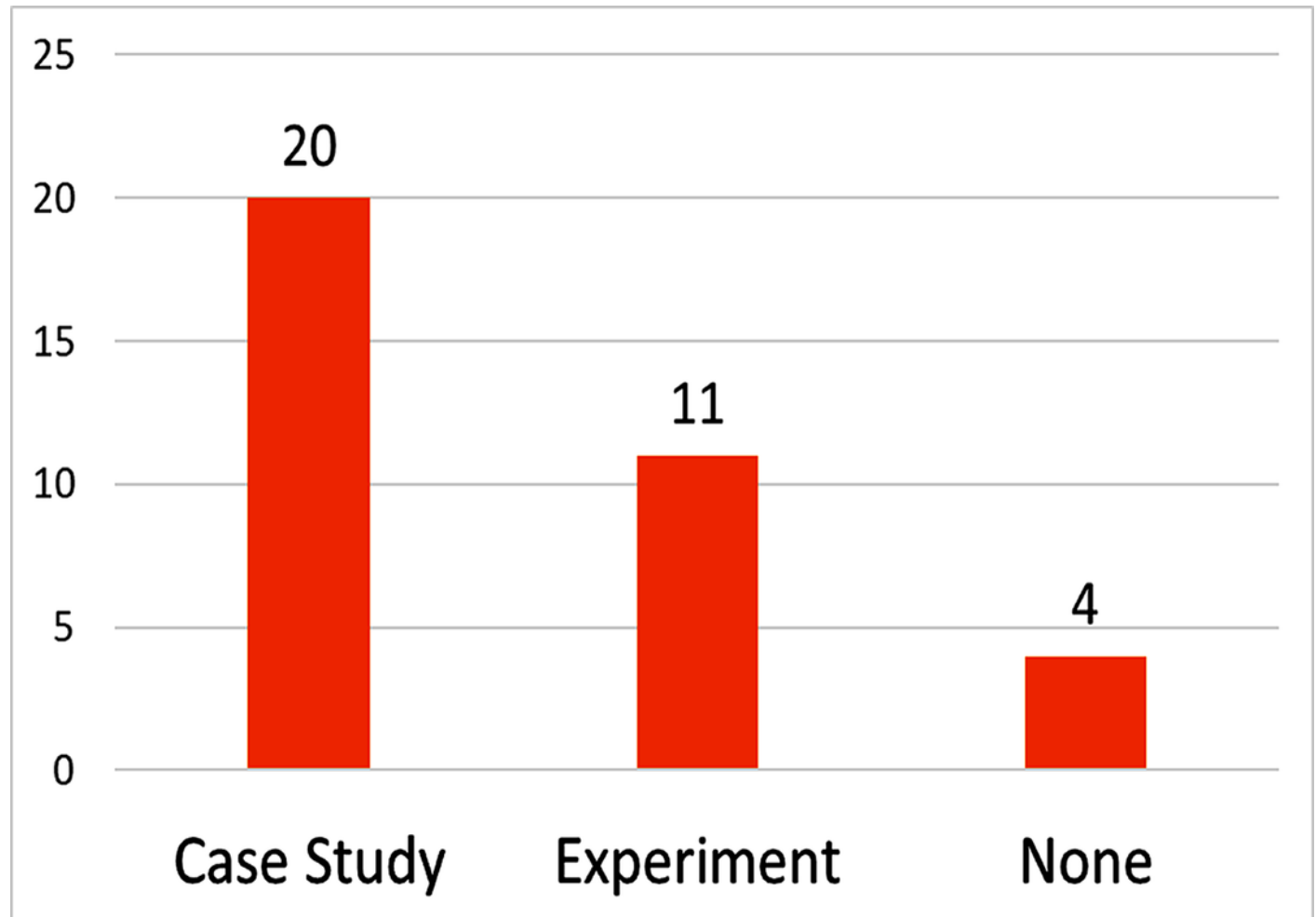



Figure 7

Type of techniques used in the selected papers.

Multiple techniques may be used by one paper. Some papers used some specific technique, these techniques were not included.

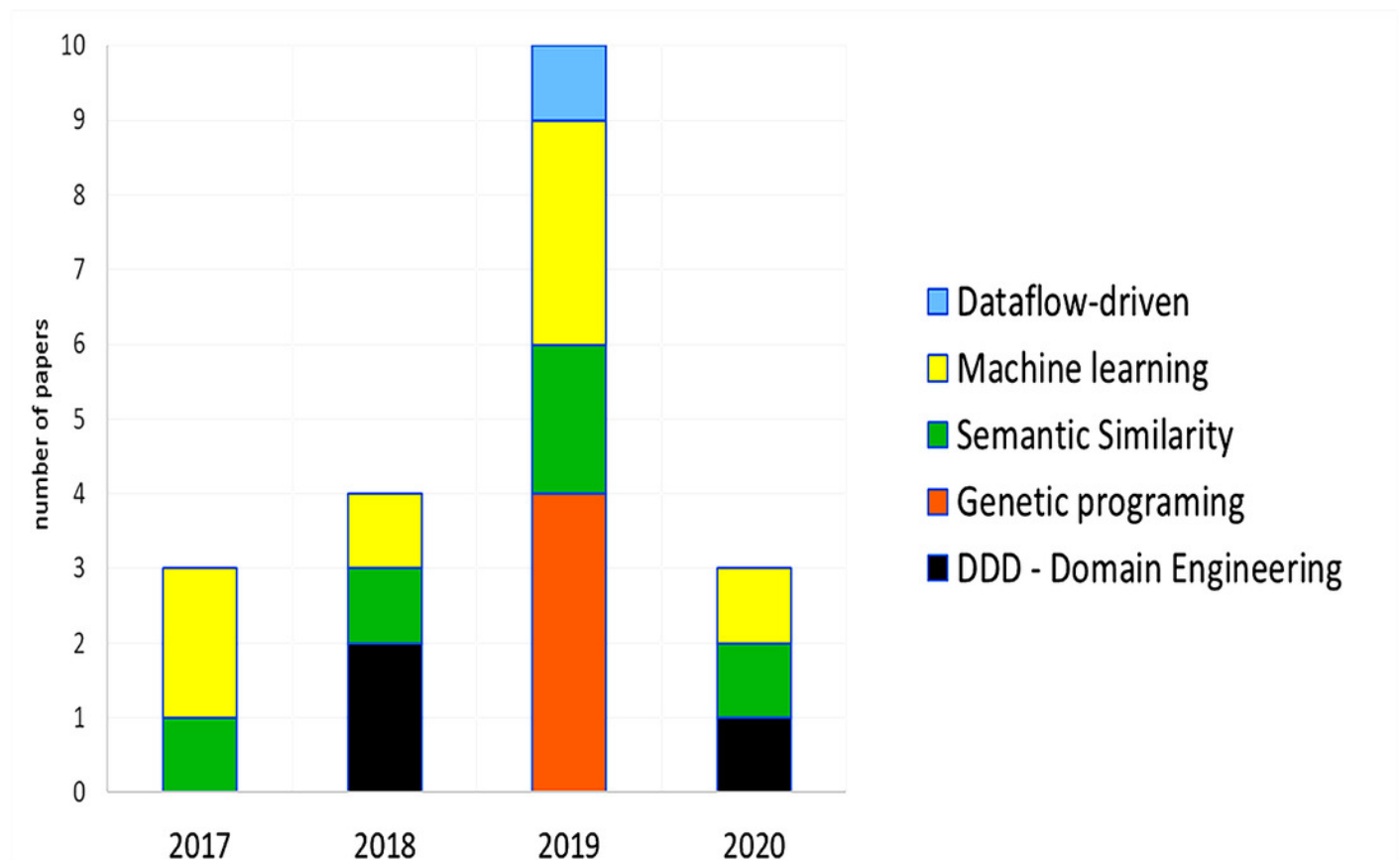



Figure 8

Amount of metrics per metric type.

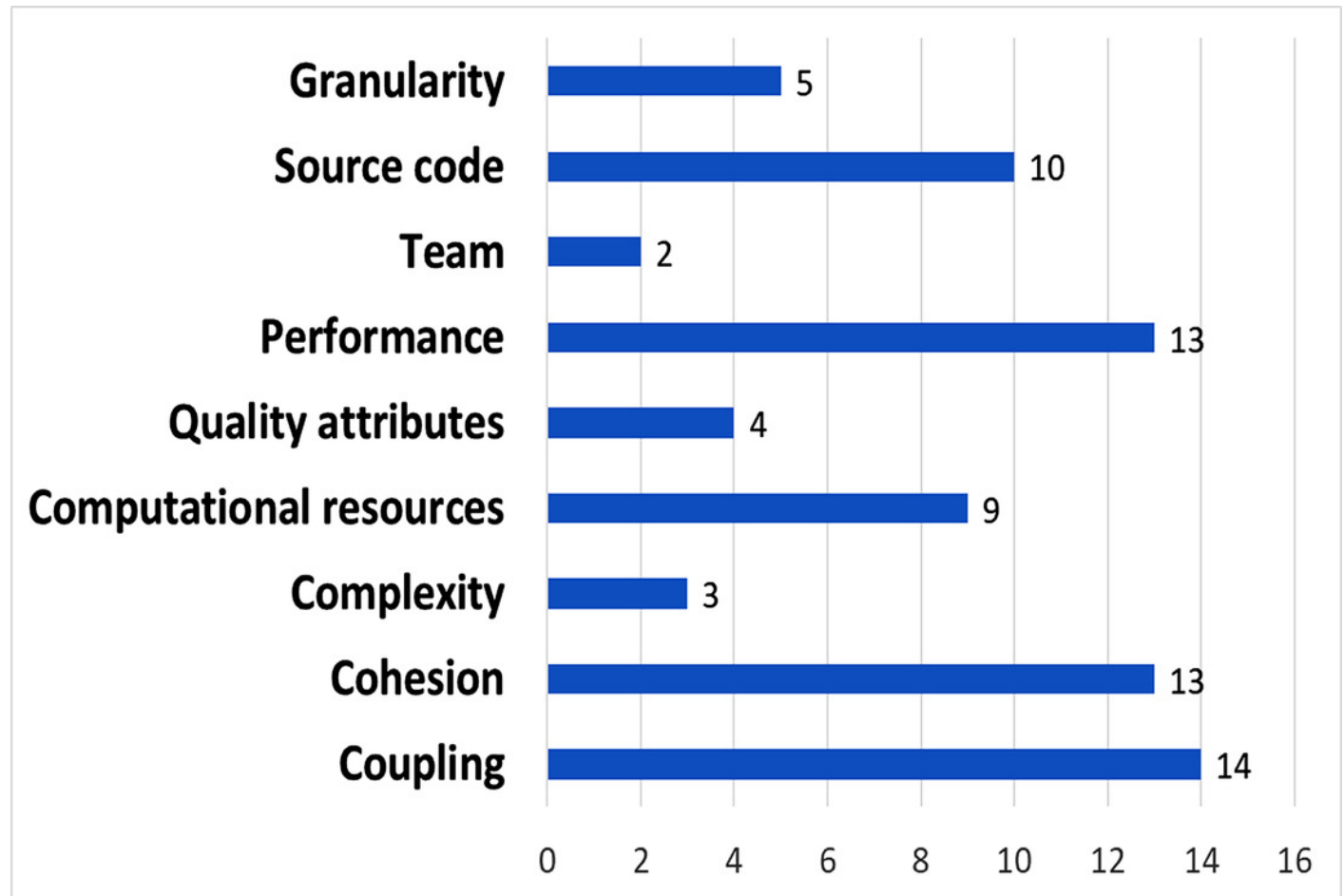
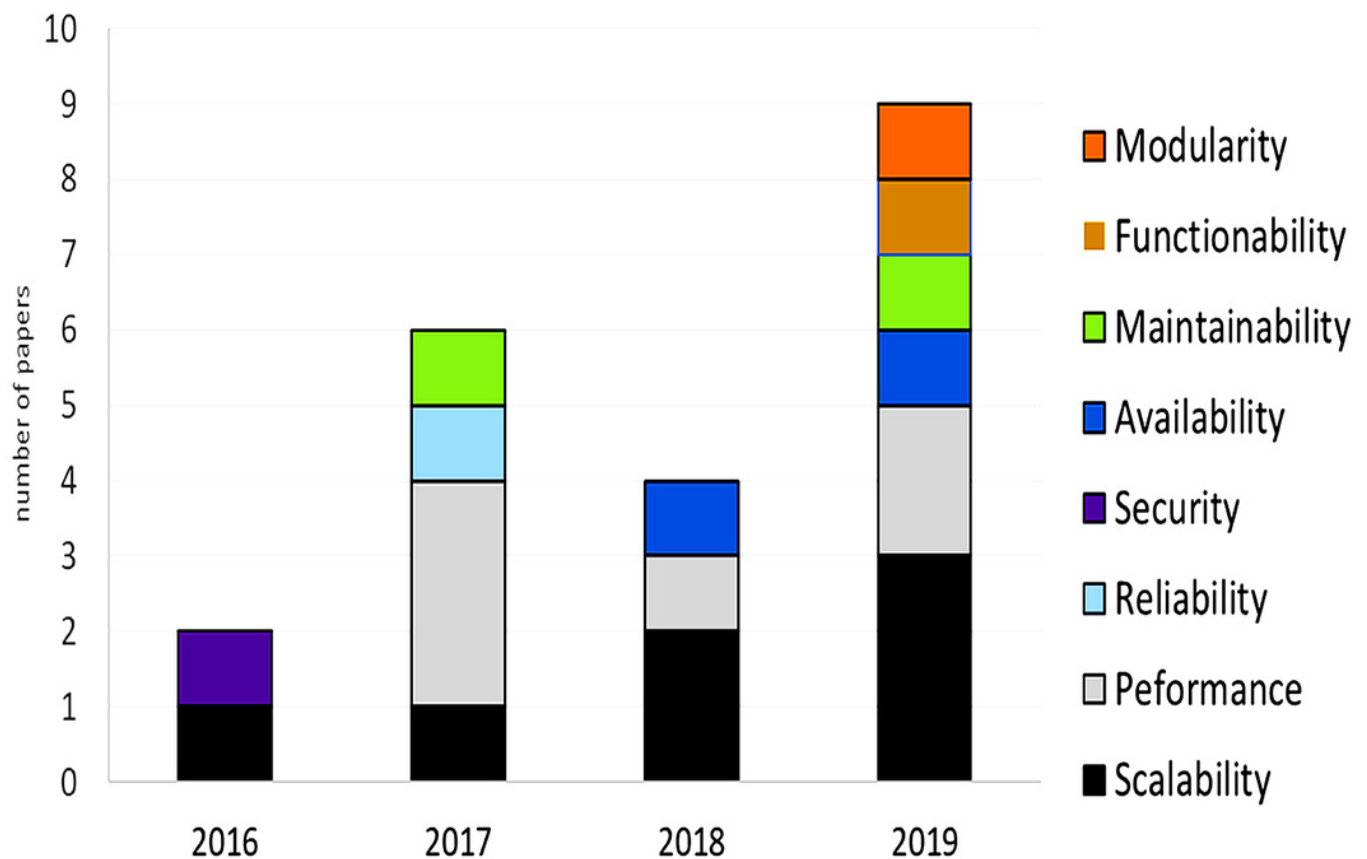



Figure 9

Amount of papers per addressed Quality Attributes and year.

Inclusion and exclusion criteria.

1 **Table 1.** Inclusion and exclusion criteria

Inclusion criteria	Description
Primary research papers that make a specific proposal about the size, granularity, or decomposition of applications to microservices.	This criterion focuses on identifying primary research papers that propose or define the size or granularity of microservices, also we include migrations from monolith to microservices that carry out a proposal to decompose the monolithic application to microservices.
Papers that propose a methodology, model, technique, or method to define granularity, size, or dimension of microservices.	The objective of the review is to identify the models, methods, methodologies, or techniques used to define the microservice granularity.
Migrations that include a methodology, model, technique, or method to define granularity, size, or dimension of microservices.	We include migrations from monolithic applications to microservices that reason about the definition of microservice granularity, those migrations that focus on other aspects are not included.
Papers published in journals and conference proceedings in the field of software architecture, software engineering and computer science.	We focus on research papers published in international journals and conferences only in software architecture, software engineering, and computer science. We include only peer-reviewed papers. We did not include gray literature.

Exclusion criteria	Description
Tutorial, example, experience, and opinion articles.	We do not include tutorials, examples, experiences, and opinion articles, because they do not correspond to primary research papers, or they do not carry out a new contribution in the definition of microservices granularity.
Survey and literature review.	We exclude survey papers, and literature reviews because they are secondary research papers that list the contributions of other authors.
Use of microservices in other areas.	The use of microservices architecture in other areas is evident and fundamental, for this review they were excluded because they do not directly address the problem of defining the microservice granularity.
Papers that do not include a methodology, model, technique, or method to define granularity, size, or dimension of microservices.	Articles related to the microservice architecture, which do not make a specific proposal on the definition of microservices granularity are excluded.
Papers which propose a specific method, technique or model for SOA, web services or mobile services.	The fundamentals of SOA, web services, and mobile services are different from the fundamentals of microservices architecture, so specific proposals in these topics are not included.
Literature only in the form of abstracts, blogs, or presentations.	We used full-text articles, excluding those that are only available in abstract, blog, or presentation form (not peer-reviewed).
Articles not written in English or Spanish.	Only we include papers written in English or Spanish in other languages are excluded.

Number of publications.

2   **Table 2.** Number of publications

Query String	Mont / Year	IEEE	ACM	Scopus	Total *
QS1	July / 2020	72	774	123	969
QS2	July / 2020	4	131	11	146

4   * Including duplicated papers.

Selected papers, related works on the definition of granularity of microservices.

1   **Table 3.** Selected papers, related works on the definition of granularity of microservices.

ID.	Paper	Year	Type
P1	Microservices and Their Design Trade-offs: A self-adaptive Roadmap (Hassan & Bahsoon, 2016) .	2016	Conference paper
P2	Microservice Architectures for Scalability, Agility and Reliability in E-Commerce (Hasselbring & Steinacker, 2017).	2017	Conference paper
P3	From Monolith to Microservices: Lessons Learned on an Industrial Migration to a Web Oriented Architecture (Gouigoux & Tamzalit, 2017).	2017	Conference paper
P4	Microservices: Granularity vs. Performance (Shadija, Rezai & Hill, 2017).	2017	Conference paper
P5	Microservice Ambients: An Architectural Meta-Modelling Approach for Microservice Granularity (Hassan, Ali & Bahsoon, 2017).	2017	Conference paper
P6	Microservices Identification Through Interface Analysis (Baresi, Garriga & De Renzis, 2017).	2017	Conference paper
P7	Partitioning Microservices: A domain engineering approach (Josélyne et al., 2018).	2018	Conference paper
P8	A Case Study on Measuring the Size of Microservices (Vural, Koyuncu & Misra, 2018)	2018	Conference paper
P9	Identifying Microservices Using Functional Decomposition (Tyszberowicz et al., 2018).	2018	Conference paper
P10	Unsupervised Learning Approach for Web Application Auto-decomposition into Microservices (Abdullah, Iqbal & Erradi, 2019).	2019	Journal paper
P11	Requirements Reconciliation for Scalable and Secure Microservice (De)composition (Ahmadvand & Ibrahim, 2016).	2016	Conference paper
P12	Function-Splitting Heuristics for Discovery of Microservices in Enterprise Systems (De Alwis et al., 2018).	2018	Conference paper
P13	Extraction of Microservices from Monolithic Software Architectures (Mazlami, Cito & Leitner, 2017).	2017	Conference paper
P14	From Monolith to Microservices: A Dataflow-Driven Approach (Chen, Li & Li, 2017).	2017	Conference paper
	A Dataflow-driven Approach to Identifying Microservices from Monolithic Applications (Li et al., 2019).	2019	Journal paper
P15	From Monolithic Systems to Microservices: A Decomposition Framework Based on Process Mining (Taibi & Syst, 2019).	2019	Conference paper
P16	Service Candidate Identification from Monolithic Systems Based on Execution Traces (Jin et al., 2019).	2019	Journal paper
P17	The ENTICE Approach to Decompose Monolithic Services into Microservices (Kecskemeti, Marosi & Kertesz, 2016).	2016	Conference paper
	Towards a Methodology to Form Microservices from Monolithic Ones (Kecskemeti, Kertesz & Marosi, 2017).	2017	
P18	Refactoring Orchestrated Web Services into Microservices Using Decomposition Pattern (Tusjunt & Vatanawood, 2018).	2018	Conference paper
P19	A logical architecture design method for microservices architectures (Santos et al., 2019).	2019	Conference paper
P20	A New Decomposition Method for Designing Microservices (Al-Debagy & Martinek, 2019).	2019	Journal paper
P21	Business Object Centric Microservices Patterns (De Alwis et al., 2019).	2019	Conference paper
P22	From a Monolith to a Microservices Architecture: An Approach Based on Transactional Contexts (Nunes, Santos & Rito Silva, 2019).	2019	Conference paper
P23	Granularity Cost Analysis for Function Block as a Service (Homay et al., 2019).	2019	Conference paper
P24	MicroValid: A Validation Framework for Automatically Decomposed Microservices (Cojocaru, Uta & Oprescu, 2019).	2019	Conference paper
P25	Migration of Software Components to Microservices: Matching and Synthesis (Christoforou, Odysseos & Andreou, 2019).	2019	Conference paper
P26	Microservice Decomposition via Static and Dynamic Analysis of the Monolith (Krause et al., 2020).	2020	Conference paper
P27	Towards Automated Microservices Extraction Using Multi-objective Evolutionary Search (Saidani et al., 2019).	2019	Conference paper
P28	Extracting Microservices' Candidates from Monolithic Applications: Interface Analysis and Evaluation Metrics Approach (Al-Debagy & Martinek, 2020).	2020	Conference paper

| P29 | Migrating Web Applications from Monolithic Structure to Microservices Architecture (Ren et al., 2018). | 2018 | Conference paper |

Types of case study used by selected papers

1 **Table 4.** Types of case study

Paper	Case Study	Type	URL
P2	An e-commerce web application called otto.	Real-life	http://www.otto.de
P3	MGDIS.	Real-life	http://www.mgdis.fr
P4	University admissions systems. Central Admission Authority UK.	Hypothetical	No report
P5	Online movie streaming subscription-based system.	Hypothetical	No report
P6	Cargo tracking, money transfer, and kanban boards.	Hypothetical	No report
P6	Real-world OpenAPI specification, from Apis.guru	Real-life	https://apis.guru/openapi-directory/
P7	Weather case study.	Hypothetical	No report
P8	E-shop	Hypothetical	No report
P9	The Common Component Modeling Example – CoCoME [56].	Hypothetical	No report
P10	Open source benchmark web application ACME Air.	Hypothetical	https://github.com/blueperf, https://github.com/blueperf/acmeair-driver
P11	Movie/TV show streaming system.	Hypothetical	No report
P12	Open-source projects: SUGAR and Church CRM.	Real-life	https://www.sugarcrm.com/. http://churchcrm.io/
P13	Open source repositories using the Git VCS. (21 projects)	Real-life	No report
P14	Case study 1: movies and comments. Case study 2: cinema, movies, plan screening, online box, office list.	Hypothetical	No report
P15	Bookkeeping for Italian tax accountants.	Real-life	No report
P16	JPetsStore, a web application.	Real-life	https://github.com/mybatis/jpetstore-6
P16	SpringBlog. Solo. jforum2. Agilefant. Xwiki.	Real-life	https://github.com/bvn13/SpringBlog https://github.com/b3log/solo https://sourceforge.net/projects/jforum2 https://roller.apache.org https://www.agilefant.com http://platform.xwiki.org
P18	Online shopping workflow.	Hypothetical	No report
P19	The Integrated Management Platform 4.0 (IMP_4.0), ERP System.	Real-life	No report
P20	Kanban boards. Money transfer.	Hypothetical	No report
P20	AWS. PayPal.	Real-life	No report
P22	LdoD. Blended workflow.	Real-life	https://ldod.uc.pt https://github.com/socialsoftware/blended-workflow.
P24	Cargo tracking. Booking system. Trading system. Movie crawler system. Ticket price comparator.	Hypothetical	No report
P26	Legacy lottery application in|FOCUS.	Real-life	No report
P27	JpetStore, SpringBlog	Hypothetical	https://github.com/mybatis/jpetstore-6. https://github.com/Raysmond/SpringBlog.
P28	Kanban boards. Money transfer.	Hypothetical	No report
P29	DayTrader, JPetStore,	Real-life	No report

TPC-W. RUBiS.
12 applications migrated.

Contributions in the definition of the granularity of the microservices by technique, contribution type and automatization level.

2   **Table 5.** Contributions in the definition of the granularity of the microservices by technique,
3   contribution type and automatization level.

Paper	Technique	Migration, From Scratch, or None	Input Data	Contribution type					Automatization level		
				Method	Methodology	Ref. Architec.	Problem.	Evaluat.	Manual	Automatic	Semi Auto.
P1	Self-adaptative solution.	None					X		X		
P2	Vertical decomposition in self-contained systems.	Migration				X			X		
P3	Balance cost quality assurance vs deployment.	Migration		X					X		
P4	Comparing the same microservices in a single container and in two containers.	Migration						X	X		
P5	Architecture definition language (ADL).	From Scratch				X			X		
P6	Semantic similarity, clustering k-means, DISCO.	From Scratch	OpenApi specification.	X						X	
P7	Domain engineering, domain-driven design.	From Scratch		X					X		
P8	Domain-driven design COSMIC function points.	From Scratch		X					X		
P9	Functional decomposition.	From Scratch	Use cases	X					X		
P10	Machine learning method, scale weighted k-means.	Migration	Access logs	X						X	
P11	Decomposition form system requirements – security vs scalability	From Scratch			X				X		
P12	Heuristics used for functional splitting, microservice discovery algorithms.	Migration	Source code, database, execution call graphs.	X						X	
P13	Graph-based clustering algorithm.	Migration	Source code.	X						X	
P14	Dataflow-driven decomposition algorithm.	Migration	Dataflow diagram, uses cases.		X						X
P15	Process-mining approach, DISCO used to identify the business processes.	Migration	Execution logs.	X							X
P16	Search-based functional atom grouping algorithm. Non-dominated sorting genetic algorithm-II (NSGA II).	Migration	Execution traces from logs.	X						X	
P17	Virtual machine image synthesis and analysis.	Migration			X				X		
P18	Scheme to refactor SOA services into microservices using decomposition pattern.	From Scratch	Scenario statements, workflow, BPEL description.		X				X		
P19	Set of rule-based decisions, adaptation of the four-step rule set (4SRS) method.	From Scratch	UML use cases model.	X					X		

ID	Description	Usage	Input	C1	C2	C3	C4	C5	C6
P20	Word embedding and hierarchical clustering of semantic similarity.	Migration	OpenApi specification	X					X
P21	Microservice discovery algorithms. Genetic algorithm, DISCO, NSGAII.	Migration	Source code, database, execution call graphs	X					X
P22	A clustering algorithm applied to aggregate domain entities.	Migration	Source code, MVC architectural style. Call graph.		X				X
P23	Service granularity cost analysis-based method, cost analysis function.	From Scratch				X	X		
P24	Validation framework for microservice decompositions.	Migration		X			X		
P25	Ontology scheme search-based techniques, multi-objective genetic algorithm.	Migration	Component and microservices properties.		X				X
P26	Domain-driven design, architectural design via dynamic software visualization.	Migration	None		X		X		
P27	Non-dominated sorting genetic algorithm-II (NSGA II).	Migration	Source code	X				X	
P28	Fast test model, clustering using affinity propagation algorithm, and clustering of semantically similar API operations.	Migration	OpenAPI specification	X				X	
P29	Function call graph, Markov chain model to represent migration characteristics, k-means hierarchical clustering.	Migration	Source code, runtime logs	X					X

5   Abreviatures: Migration: The proposal is used in migration from monolith to microservices.

6   From Scratch: The proposal can be used for development from scratch. Ref. Architec.: reference

7   architecture, Evaluat: evaluation or comparison, Semi Auto: Semi-automatic, DISCO:

8   Distributionally related words using co-occurrences, NSGAII: Non-dominated sorting genetic

9   algorithm-II, SOA: Service-oriented architecture.

Papers and used metrics.

1 **Table 6.** Contributions in the definition of the granularity of the microservices.

Paper	Use of metrics relative to				Metric type								
	Dev. Team	Dev. Process	App. System.	MS.	Coupling	Cohesion	Granularity	Complexity	Team	Comput. Resource.	Source Code	Performance	Other Quality Attributtes
P3		X											Cost of quality assurance. Cost of deployment.
P4				X								Response time. Number of calls.	
P8			X	X				COSMIC function points.					
P10				X						Avg. CPU. Used virtual machines. Allocated virtual machines.		Response time SLO violations. Number of calls. Number of rejected request. Throughput.	
P11				X	Dependency weight.								Security impact. Scalability impact.
P12			X	X	Structural coupling.	Lack of cohesion.				Avg. Memory. Avg. Disk.		Number of requests. Execution time.	
P13	X		X	X	Logical coupling. Average domain redundance. Contributor coupling. Semantic coupling.				Commit count. Contributor count.		Lines of code		

ID									
P14	X	X	Afferent coupling. Efferent coupling. Instability.	Relational cohesion.					
P15		X	Coupling between microservice.					Number of classes. Number of duplicated classes.	
P16		X	Integrating interface number.	Cohesion at message level. Cohesion at domain level.				Structural modularity Quality. Conceptual modularity Quality. Internal and external Co-change Frequency (ICF, ECF). Ration of ECF to ICF.	
P21	X	X	Structural coupling.	Lack of cohesion.			Avg. CPU Avg. Network.		Number of executions. Number of packets send.
P22	X		Silhouette Score.			Number of singleton clusters. Maximum cluster size.			
P23							Service composition cost. Service decomposition cost.		Request time
P24	X	X	Dependencies composition. Strongly	Nano-entities composition. Relation	Number of nano-entities.				

	connected components.		composition. Responsibilities composition. Semantic similarity.			
P27		X	Cohesion at Message level. Cohesion at Domain level.	Operation number.		Interaction number.
P28	X	X	Lack of cohesion.	Operation number.		
P29	X			Number of microservices. Number of interfaces.	Lines of code. Package analysis. Class hierarchy analysis.	Static structure analysis. Static call graph analysis. Combined static and dynamic analysis.

3   Abbreviatures: Paper: paper number, Dev. Team: Development team, Dev. Process: Development process, App. System: Application
4   or system, MS: Microservice, Comput. Resource: Computational resource, Avg.: Average, COSMIC: Common Software Measurement
5   International Consortium, SLO: Service-level objective.
6

Papers and quality attributes grouped by runtime and artifact software characteristics.

1   **Table 7.** Papers and quality attributes

Paper	Runtime characteristics						Software artifact characteristics	
	Scalab.	Perfor.	Reli.	Avail.	Security	Funct.	Maint.	Modul.
P2	X	X	X					
P3							X	
P4		X						
P10	X	X						
P11	X				X			
P12	X	X		X				
P13		X						
P16	X					X	X	X
P21	X	X		X				
P29	X	X						

4   **Abreviatures.** Scalab: Scalability, Perfor: Performance, Reli: Reliability, Avail: Availability,
5   Funct: functionality, Main: Maintainability, Evol: Evolvability, Modul: Modularity.

Semi-automatic methods, which used metrics and addressed some quality attribute to define the granularity .

1 **Table 8.** Automatic or Semi-automatic methods, which used metrics and addressed some quality
2 attribute to define the granularity.

Paper	Metrics	Quality Attributes
P10	Response time. Number of calls. Number of rejected request.	Scalability and performance
P12	Structural coupling. Lack of cohesion. Avg. Memory. Avg. Disk. Number of requests. Execution time.	Scalability, performance, and availability
P13	Logical coupling. Average domain redundance. Contributor coupling. Semantic coupling. Commit count. Contributor count. Lines of code	Performance
P16	Integrating interface number. Cohesion at message level. Cohesion at domain level. Structural modularity Quality. Conceptual modularity Quality. Internal and extermal Co-change Frequency (ICF, ECF). Ration of ECF to ICF.	Scalability, functionality, maintainability, evolvability, and modularity
P21	Structural coupling. Lack of cohesion. Avg. CPU. Avg. Network. Number of executions. Number of packets send.	Scalability, performance, and availability
P29	Number of microservices. Number of interfaces. Lines of code. Package analysis. Class hierarchy analysis. Static structure analysis. Static call graph analysis. Combined static and dynamic analysis.	Scalability and performance.