Continuously Managing Microservice Granularity: An Evidence-Based Industrial Approach

Yan Justino
CESAR SCHOOL
Recife, Brazil
contact@yanjustino.com

Carlos Eduardo da Silva
Sheffield Hallam University
Sheffield, UK
C.DaSilva@shu.ac.uk

Rafael Batista Duarte
CESAR SCHOOL
Recife, Brazil
rbd@cesar.school

ABSTRACT

Defining and managing appropriate service granularity remains a recurring challenge in the design and evolution of microservices-based systems, directly affecting modularity, maintenance, and operational efficiency. This paper presents early empirical evidence supporting Granulify, a continuous granularity-management approach that dynamically adjusts service boundaries throughout the system lifecycle. Although still preliminary, the results already point to tangible benefits in modularity, maintainability, and operational cost, underscoring the method's industrial applicability. The proposal is being validated through the reengineering of a real-world investment management platform from a major financial institution in Brazil. The analysed platform manages over 40 million transactions monthly and supports more than 250K internal users across investment, compliance, and trading domains. Preliminary results indicate patterns of architectural fragmentation, productivity impacts, and signs of granularity saturation. We believe that continuous application of this approach will contribute to more informed architectural decisions, balancing modularity, maintenance effort, and operational costs. This ongoing research aims to consolidate Granulify as a practical solution to support teams in the evolutionary management of granularity in microservices architectures.

CCS CONCEPTS

• **Software and its engineering → Software architectures**; **Software maintenance**; **Software evolution**; *Distributed systems organizing principles*;

KEYWORDS

Microservices, Service Granularity, Continuous Architecture, Software Maintenance, Software Evolution, Case Study, Granularity Management Approach

1 INTRODUCTION

Accurately establishing the size and scope of a microservice has been widely recognised as one of the main software engineering challenges faced by development teams. Several studies highlight the complexity involved in this decision, which directly impacts aspects such as modularity, performance, maintenance, and the evolution of microservice-based systems [3, 5, 7, 12, 15, 17, 20, 22, 24, 27, 28, 30].

This decision is fundamental to the software modelling process, as an inadequate definition of granularity may result in the excessive creation of components and automation mechanisms. Such a scenario tends to intensify challenges such as increased cost and operational complexity, the need for greater coordination effort for functionality testing, the emergence of communication failures and inconsistencies, as well as a high frequency of maintenance activities focused on compatibility with technologies and implementation platforms [2, 4].

In this context, this paper joins other academic reports that highlight the inherent challenges in the process of migrating systems to a microservices architecture. To this end, this study presents an approach for the continuous management of granularity in distributed systems. The proposal allows for dynamic adjustments throughout the application lifecycle, incorporating progressive modularisation and architectural adaptation.

This approach has been iteratively developed and refined through practical application in a real-world scenario involving the migration of a large investment platform in the Brazilian financial sector. In this context, the absence of mechanisms for revisiting service granularity over time has negatively affected maintenance and operational efficiency.

The central contribution of this article lies in the definition and application of a specific set of metrics to support the continuous management of granularity, and in the introduction of the **Granularity Saturation Method**, which helps in the assessment of the saturation level in the microservice granularity. We define "granularity saturation" as the point at which further splitting a system into services or modules no longer delivers net benefits, because the added coordination and complexity costs outweigh the gains in isolation.

The focus of this study lies in the formulation of the metrics and the empirical analysis of their practical application. For that we have applied our proposed metrics in a real industrial case, performing an analysis of a real system evolution of a period of two years, and demonstrated how our approach supports decision making in granularity management.

The remainder of this paper is organised as follows. Section 2 presents the fundamental concepts and related work on microservice granularity. Section 3 describes the context of the case study used to conduct this research. In Section 4, we detail the process of extracting the Granulify approach from the case study, followed by Section 5, which discusses the main results and analyses. Finally, Section 6 presents the conclusions and directions for future work.

2 RELATED WORK

This section presents the essential concepts that underpin the proposal of this work, focusing on the relationship between microservice granularity and its impacts on the evolution of distributed systems, while discussing the current gap in support for continuous granularity management. The definition of granularity in its classical conception is related to the level of detail or abstraction of a

software component, reflecting the size, functional scope, and degree of coupling between system units [18, 21].

In the context of microservice-based architectures, Vera-Rivera et al. [26] propose a specialisation of this concept by stating that defining the granularity of a system means correctly identifying the boundaries of each service. However, choosing the appropriate level of granularity is recognised in the literature as one of the greatest challenges in the design and evolution of microservice-based systems [15, 20, 26]. This is because, while finer-grained services may increase modularity and scalability, they also introduce additional challenges such as architectural complexity, increased communication, and operational costs [15, 20].

These adverse effects mark the point of granularity saturation. Furthermore, the overhead associated with the coordination and orchestration of multiple fine-grained services can negatively impact both performance and the productivity of development teams. In this sense, the definition of granularity depends on multiple factors, such as: Domain boundaries (Domain-Driven Design) [2, 27]; System maintainability and evolvability [3, 22]; Operational infrastructure cost [14, 26]; and Service independence flexibility [15].

Recent research highlights that granularity should not be treated as a one-time, immutable decision, but rather as a dimension that can and should be dynamically adjusted throughout the application lifecycle [4, 5, 8, 22]. In practice, selecting the appropriate level of granularity involves several trade-offs [7, 8, 15, 16]. For instance, while finer granularity enables selective scalability, it often increases orchestration and operational costs. Similarly, smaller and more modular services improve isolated development and deployment, yet require greater coordination among teams and components. Additionally, although modularity supports separation of concerns, the resulting distribution of logic may hinder system-level understanding and maintainability.

It is important to emphasise that there is no single "ideal" metric for evaluating granularity. The choice of metrics depends on the goals of the evaluation, the type of system, and the quality attributes that are considered most important [8]. Furthermore, Vera-Rivera et al. [26] and Hassan et al. [15] suggest that analysing the historical evolution of the system can reveal important patterns to guide decisions about granularity (re)adjustments. To support the management and monitoring of granularity in microservice systems, several studies propose the use of structural and process metrics such as size metrics (of software and teams), distribution, coupling, cohesion, complexity, maintainability, and performance [15, 22, 26].

In this study, these metrics are examined in relation to three key trade-offs commonly observed in service granularity decisions: scalability vs. costs, modularity vs. coordination effort, and modularity vs. maintainability. These trade-offs informed the selection and refinement of the metric set employed, aiming to ensure alignment with practical challenges in system evolution. A comparative analysis between existing metrics and those introduced in this work is presented in Section 4, justifying the adjustments made to better capture granularity saturation over time.

According to Hassan et al. [14], the state of the art in scalability evaluation when reasoning about granularity adaptation is often ad hoc. Supporting this view, recent literature highlights the need for models that enable continuous granularity management throughout system evolution [8, 26, 30]. This management aims to identify when a system is: **Undermodularised** (insufficient granularity); **Overmodularised** (excessive granularity); or **Saturated** (inefficient or ineffective growth).

Approaches and tools that incorporate monitoring and adaptation of granularity, such as the one described in this work, are considered a current gap in the literature [15, 26], especially in the practical application of systematic methods to support continuous architectural decision-making. Recent studies have addressed the challenges related to decomposing monolithic systems into microservice architectures. These investigations reveal a gap in support for continuous granularity management, especially in industrial environments.

Becker and Lucrédio [3] investigated the impact of adopting microservices in the evolution of a *Software Product Line* (SPL), highlighting the effects of poorly defined granularity on product maintenance and evolution. Hassan et al. [15] conducted a systematic mapping study showing that the problem of granularity definition is among the most critical in architectural transitions, but few studies propose approaches for its management over time. Bogner et al. [5] analysed industrial practices related to the evolvability of microservice-based systems, concluding that the absence of metrics and methods for continuous granularity monitoring undermines the sustainable evolution capability of applications. Vera-Rivera et al. [26], in turn, conducted a literature review focused on methods for defining and measuring service granularity, but also identified the limitation of most studies in treating granularity as a static decision.

Complementing these analyses, Taibi and Lenarduzzi [22] discussed the so-called *microservice bad smells*, highlighting how poor granularity decisions negatively impact system maintenance and operation. Although they address recurring problems in practice, these studies do not present systematic solutions for continuous granularity management throughout the system lifecycle. The work of Vural and Koyuncu [27] explores the use of *Domain-Driven Design* (DDD) to support the initial definition of granularity, proposing that domain boundaries serve as natural guides for modularisation. Despite the usefulness of DDD, the authors acknowledge that organisational and business changes may require dynamic granularity revisions, reinforcing the need for adaptive approaches.

By proposing a structured and iterative approach, this work seeks to fill this gap by supporting continuous granularity management, integrating quantitative metrics and qualitative analyses based on real system evolution data.

3 THE CASE STUDY

In order to understand the impacts of granularity on the evolution of systems based on microservice architecture, this case study analyses a modernisation project of a large-scale investment management platform from one of the largest private financial institutions in Brazil. The platform operates at significant scale, managing over 40 million transactions monthly and supporting more than 250,000 internal users across investment, compliance, and trading domains. Its selection was due to its relevance to the fields of monolithic-to-microservices architecture migration and microservice granularity. Additionally, its modular structure and the need

to adapt to increasing scalability demands make it a representative example of the real challenges of modern software engineering.

Originally designed to support a limited daily transaction load, the institution's *Asset Management System* evolved over time to meet growing demands in the investment sector. Initially monolithic, the system's architecture was gradually fragmented to address requirements for scalability, improved maintainability, greater robustness, and increased business alignment. Between January 2023 and January 2025, a new phase of migration was conducted to further advance the modernisation of the system.

The team observed in this case study was responsible for migrating three critical business capabilities: *Financial Instrument Management*, *Trade Position Management*, and *Investment Portfolio Management* [1]. As a strategy, the team chose to segment the monolithic system into parts that were easier to understand and maintain, adopting a microservice-based architecture design and implementation.

Throughout this process, the team faced the reality that there are few established patterns to guide the definition of how granular a microservice should be. In this context, the modernisation effort encountered challenges related to productivity and operational costs, making its trajectory particularly relevant to the discussion on continuous granularity management in microservice-based architectures. Based on this, the main contribution of this work is the definition of an approach focused on continuous granularity, identified over recent years during the migration of monolithic systems to microservices.

In the following section, we detail how this case study, combined with a set of prior experiences, served as the basis for extracting the proposed approach.

4 GRANULIFY EXTRACTION

Granulify is an approach currently under development in the context of industry-applied research, aiming to support systems in evolving dynamically along the granularity spectrum. In Subsection 4.1, the core principles and structuring mechanisms of the approach are presented. Subsection 4.2 describes the associated metrics and evaluation criteria. The following subsections (4.3, 4.4, and 4.5) detail the procedures adopted during the initial stages of development and validation. Given that this research is still ongoing, only the early phases of the method are presented in this paper.

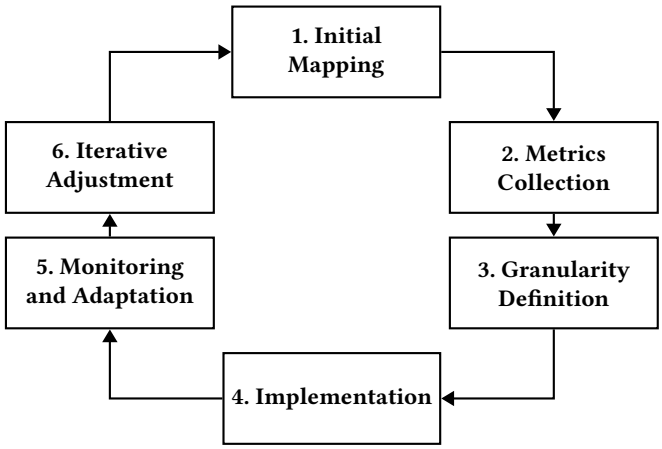
4.1 Principles and Processes

The Granulify approach integrates principles of distributed design, iterative refinement, and metric-based evaluation to support architectural decisions related to service granularity. Its design explicitly considers the trade-offs involved in these decisions, balancing the pursuit of modularity, scalability, and maintainability with the associated coordination overhead, operational costs, and increased system complexity. By aligning metric interpretation with these opposing forces, the approach aims to assist teams in making more informed and context-sensitive granularity adjustments throughout the system lifecycle. Its core principles are:

i **Granularity as a dynamic spectrum:** granularity is not a fixed point but a continuum that can be adjusted over time. It should be revisited regularly to ensure alignment with system and business needs.
ii **Continuous review:** granularity should be constantly monitored and adapted to balance modularity, performance, and management complexity, taking into account changes in technical and organisational demands.
iii **Trade-offs between flexibility and control:** the choice of granularity involves compromises between scalability, operational cost, and governance, requiring iterative adjustments throughout the application lifecycle.
iv **Domain orientation and functional decomposition:** domains and subdomains define natural boundaries for modularisation, but these boundaries should be periodically reassessed to avoid unnecessary coupling or excessive fragmentation.

Additionally, Granulify provides an iterative granularity adjustment process consisting of the following steps: **Initial Mapping**, to diagnose the current granularity; **Metrics Collection**, to compile logs, quantitative metrics (coupling, LOC, frequent changes, etc.), and team feedback; **Granularity Definition**, to evaluate trade-offs and determine whether a service should be more granular (splitting) or less granular (merging); **Implementation**, to perform refactorings and adjust granularity as needed; and finally, **Monitoring and Adaptation**, to detect granularity saturation and conduct periodic reviews. Figure 1 illustrates the flow of this process.

Figure 1: Granulify – Continuous Granularity Management Process



4.2 Metrics and Evaluation Criteria

The selection of metrics and evaluation criteria used in *Granulify* was based on two pillars: (i) adherence to best practices and empirical evidence found in the literature on microservices, granularity, and software maintenance, and (ii) availability and feasibility of data collection in the real context of the evaluated platform. Additionally, the definition of the metrics used in this study was guided by their direct alignment with the strategic objectives of

[1] The original business capabilities were renamed using terms from the BIAN model (Banking Industry Architecture Network) - bian.org

the case study organisation, which establishes explicit goals related to the quality, efficiency, speed, and competitiveness of technological solutions—aligning with pillars such as resilience, financial efficiency, and delivery acceleration. The granularity, effort, change, complexity, and operational cost metrics adopted by *Granulify* directly reflect indicators that support the achievement of these strategic goals.

For example, reducing delivery time (*Lead Time*), improving productivity (*throughput*), optimising costs, and architectural resilience are corporate objectives that are intrinsically related to granularity decisions and the system's ability to adapt sustainably over time. In this context, the proposed model was developed to support the generation of insights and guide architectural adaptations in line with the organisation's modernisation and technological efficiency plans. To this end, a set of metrics was selected to quantitatively reflect these dimensions, enabling the observation of the relationship between granularity and architectural evolution throughout the system lifecycle. In addition to conventional metrics, two new aggregate indicators were proposed — the Relative Variation Index (RVI) and the Elasticity ($\mathcal{E}$) — aiming to capture dynamic patterns of growth, stability, or decline in granularity over time.

The RVI enables the quantification of weighted relative variations across different dimensions (granularity, process, operation, code, and financial), while $\mathcal{E}$ measures the sensitivity of these variations in relation to changes in the number of microservices. Together, RVI and $\mathcal{E}$ compose the Granularity Saturation Index (GSI), which serves as a composite instrument for classifying granularity evolution into zones (high, reduced, saturated, or adverse), indicating whether architectural changes lead to improvement, stagnation, or degradation.

Table 1 summarises the metrics and indices employed, their objectives, and how they relate to the aspects under investigation. For organisational clarity, these indicators are grouped into five main dimensions: (i) **Size and granularity**, which tracks the total number of microservices and their fragmentation (NMS, TMS); (ii) **Development effort**, encompassing team effort and task-execution efficiency (TMB, EST, ACT, EFT); (iii) **Operation**, assessing stability and change intensity (NCH, FCH); (iv) **Code**, reflecting structural complexity and software quality (LOC, CYC, COV); and (v) **Financial**, evaluating infrastructure costs (CST). Continuous monitoring of these dimensions over time makes it possible to detect saturation signals and potential imbalances in architectural evolution.

Table 1: Classification of Granularity Metrics and Indices

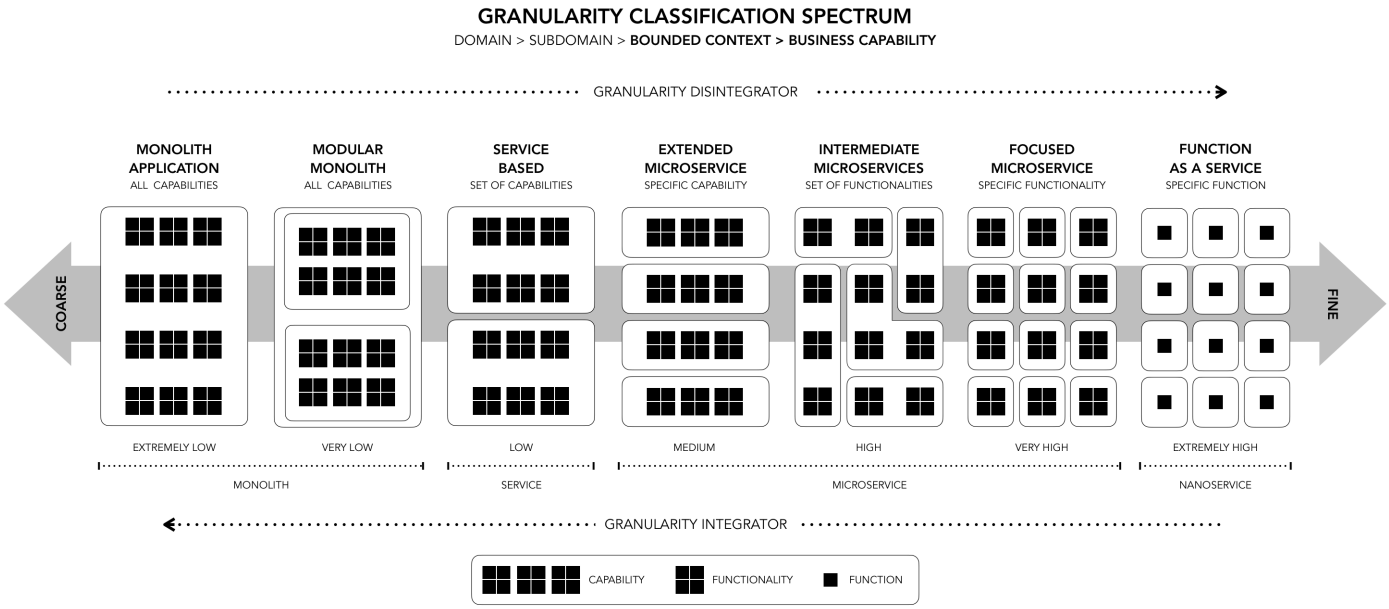
Base Metrics		
Metric	**Rationale**	**Source**
NMS	Number of active microservices; directly expresses the architectural granularity level.	[6, 26]
TMS	Total count of microservices (including internal fragmentations); indicates structural complexity and fragmentation.	[1]
TMB	Team members allocated to evolution tasks; captures the human factor related to adapting to a given granularity.	[26]
EST	Median estimated effort for completed tasks; evaluates teams' predictive capability with respect to adopted granularity.	[15, 30]
ACT	Median actual effort for completed tasks; measures the direct productivity impact of granularity.	[3, 30]
EFT	Effort factor (ACT/EST); gauges planning efficiency relative to actual effort required.	[3, 30]
NCH	Total number of significant code changes; reflects maintenance volume and intensity attributable to granularity.	[8]
FCH	Average change frequency per service; indicates architectural stability and cohesion.	[8, 30]
LOC	Total lines of code across microservices; captures overall system size and code complexity.	[6, 8]
CYC	Mean cyclomatic complexity of services; acts as a proxy for internal modularity and structural complexity.	[6]
COV	Percentage of code covered by automated tests; indicates testability and safety in the face of changes.	[15, 22]
CST	Total operational and infrastructure cost (in thousands USD); enables financial-impact analysis related to granularity.	[14]
Granularity Saturation Index (GSI)		
Metric	**Rationale**	**Source**
RVI	Weighted *Relative Variation Index* aggregating key metrics; measures system instability or maturity trends over time.	Proposed
$\mathcal{E}$	Elasticity: sensitivity of RVI to variation in NMS; classifies granularity behaviour into high, reduced, saturated or adverse zones.	Proposed

4.3 Initial Mapping

The goal of this step is to understand the current state of granularity in the system. To that end, we aim to identify the existing level of granularity and potential points for optimisation. Additionally, the goal is to assess the degree of modularity within the system's architecture. The outcome of this step should be an initial diagnostic of the system's granularity. To support this step, we developed an artefact called the **Granularity Classification Spectrum (GCS)**.

Unlike studies that cast granularity as a simple "monolith vs. microservices" dichotomy [1, 4, 15, 25], the GCS arranges architectural styles on a single continuum of deployment scope and business responsibility. At the coarsest extreme, the granularity integrators—Monolith, Modular Monolith, and Service-Based Architecture—consolidate multiple business capabilities into one deployable artefact.

Figure 2: Granularity Classification Spectrum (GCS): from Monolith to Nanoservice



As functional decomposition deepens, a system progresses through three microservice sub-classes: Extended (one business capability), Intermediate (a cohesive cluster of related functionalities), and Focused (a single functionality). The spectrum culminates in the Nanoservice/Function-as-a-Service (FaaS) tier, where each short-lived, event-triggered unit encapsulates a lone business function.

By replacing the binary view with this richer vocabulary, the GCS supports continuous analysis of how varying granularity levels affect modularity, modifiability, and software evolution; crucially, the term nanoservice denotes only the finest granularity and is not inherently an anti-pattern—its [23, 27] suitability depends on context-specific cost-benefit trade-offs. Furthermore, the model highlights two processes: the Granularity Disintegrator, which fragments services to increase modularity, and the Granularity Integrator, which consolidates services to adjust granularity as needed [10]. Figure 2 presents this visual classification model, representing the transition between different granularity levels in distributed systems. Table 2 details these levels and their respective classification criteria.

As a practical example, consider the business capability **Trade Position Management**, responsible for handling asset-trading positions. In the original monolithic system, this functionality resided in a component that bundled several investment-management capabilities. Using the Granularity Classification Spectrum (GCS) and the criteria in Table 2, the initial mapping showed that the capability had been refined architecturally and decomposed into fifteen separate components. Their distribution across the spectrum is presented in Table 3.

This diagnostic process was conducted collaboratively among architects, software engineers, and technical leaders involved in the platform modernisation, ensuring that the classifications reflected both the technical structure and the business domain organisation.

4.4 Metrics Collection

The collection of metrics used in the case study was carried out systematically from various organisational sources. The Number of MSA was obtained through version control systems and the *Configuration Management Database* (CMDB), considering the count of active microservices deployed each quarter. The total number of microservices (TMS), in turn, represented the sum of all services, including internal fragmentations associated with architectural evolution.

Estimated effort (EST) was extracted from the *agile project management platform*, which integrates issue tracking, sprint planning, and estimation features. The values were based on the average effort estimations provided by development teams for both implementation and maintenance tasks. Actual effort (ACT) was obtained from task tracking logs within the same platform, reflecting the actual time recorded for task execution. Based on these data, the effort factor (EFT) was calculated as the ratio between actual and estimated effort (ACT/EST), and used to assess planning accuracy in light of the complexity imposed by service granularity.

Maintenance-related metrics were collected directly from code repositories. The number of changes (NCH) corresponds to the count of significant commits or code alterations recorded each quarter, while the frequency of change (FCH) represents the average number of changes per service, reflecting the stability and cohesion of the architecture. Code structural metrics were obtained using static analysis tools. Lines of code (LOC) were captured automatically by these tools, while cyclomatic complexity (CYC) was

Table 2: Granularity levels grouped into *Monolith, Service, Microservice* and *NanoService* categories.

Level	Rationale	Source
	MONOLITH	
Monolithic	Single application that consolidates multiple business capabilities; internal modularity may range from nonexistent to high.	[3, 7, 8, 28]
Modular Monolith	Single deployable application with cohesive internal modules and explicit boundaries; low coupling between modules, though without independent deployment.	[3, 7, 8, 28]
	SERVICE	
Service-Based	Broad services representing multiple business capabilities; grouped by high-level domains yet still tightly coupled and with low separation of concerns.	[9, 11, 19, 23]
	MICROSERVICE	
Extended	Services organised by individual business capabilities; low internal granularity and few external integration points.	[4, 25]
Intermediate	Services encapsulating cohesive functional subsets; own interfaces and data but retaining significant internal dependencies.	[3, 13, 17]
Focused	Services centred on specific isolated functionalities; strong context isolation with explicit inter-service communication.	[3, 8]
	NANOSERVICE	
FaaS	Independent decoupled functions executed on demand; maximum granularity, very high fragmentation, and strong dependency on serverless platforms.	[7, 29]

computed as the average cyclomatic complexity of the services, indicating the typical structural complexity observed across the system. Test coverage (COV), expressed as a percentage, was calculated from reports generated by CI/CD pipelines, indicating the system's degree of testability.

Finally, infrastructure cost (CST) was obtained from corporate infrastructure cost monitoring dashboards, reflecting the total cost of running and maintaining services in production. All metrics were collected quarterly throughout the analysed period, ensuring comparability and visibility of architectural evolution in regular cycles. Table 4 consolidates this data.

Table 3: Granularity levels, styles, and investment services components.

Level	Components
Very Low	**Modular Monolith**
	1. Portfolio Management
Low	**Service-Based**
	1. Client Investment Services
Medium	**Extended Microservice**
	1. Transaction Logging;
	2. Asset Registration Management
High	**Intermediate Microservice**
	1. Position Aggregation
Very High	**Focused Microservice**
	1. Position Maintenance;
	2. Investor Profile Management
Extremely High	**Functions as a Service (FaaS)**
	1. Position Event Consumer;
	2. Position Event Publisher;
	3. Asset Validation Service;
	4. Investor Data Enrichment;
	5. Position Workflow Orchestrator;
	6. Portfolio Risk Evaluation;
	7. Compliance Rule Validator;
	8. Market Data Synchronisation

In the context of this investigation, no significant operational overhead was observed when enabling continuous metric collection. The partner institution already had both an observability stack and historically structured data pipelines, eliminating the need for additional hardware or licensing investments. The only incremental cost identified amounted to approximately 24 hours of work per quarter by a senior engineer — allocated to selecting the relevant time-series, exporting snapshots, and normalising the data used in this analysis.

The quarterly evolution analysis of the system revealed the direct impact of increasing granularity across different technical and organisational dimensions. Table 4 presents an overview of the variation in key metrics from 23Q2 to 24Q4. In the Granularity dimension, the number of active microservices (NMS) increased from 7 in 23Q2 to 10 in 24Q4, while the total number of services (TMS), including fragmentations and internal subdivisions, grew from 15 to 71. This significant growth reveals a progressive decomposition strategy aligned with modularity and component autonomy goals.

In the Process dimension, a nonlinear pattern was observed in team productivity. The average number of team members per quarter (TMB) remained relatively stable, varying between 11 and 14.

Table 4: Quarterly Evolution of Metrics by Dimension (Granularity, Development Effort, Operation, Code, Financial)

Quarter	Granularity		Development Effort				Operation		Code			Financial
-	NMS	TMS	TMB	EST	ACT	EFT	NCH	FCH	LOC	CYC	COV (%)	CST (k$)
23Q2	07	15	14	10	30	3.00	87	0.98	100	10	80	18
23Q3	15	30	12	15	34	2.28	183	2.03	110	18	89	20
23Q4	08	38	11	10	29	2.90	118	1.31	130	18	89	85
24Q1	08	42	12	05	13	2.80	143	1.58	151	19	90	115
24Q2	08	50	13	10	18	1.80	108	1.20	172	19	89	170
24Q3	11	61	13	10	17	1.70	107	1.18	210	21	88	175
24Q4	10	71	13	10	20	2.00	201	2.23	279	22	87	155

However, estimated effort (EST) and actual effort (ACT) showed notable fluctuations. In 23Q2, the median actual effort exceeded the estimate threefold (EFT = 3.00), indicating initial difficulties in team adaptation to the new architectural model. From 24Q2 onward, the Effort Factor stabilised (EFT $\approx$ 1.7–2.0), suggesting that teams began operating with greater predictability amid increasing granularity. Although an EFT value of 1 represents the ideal scenario—indicating perfect alignment between estimated and actual effort—values in the observed range can still reflect a reasonable level of planning accuracy, particularly in complex and evolving architectures.

Regarding Operation, the number of changes (NCH) increased steadily, peaking at 201 changes in 24Q4. The frequency of changes per service (FCH), which started at 0.98, surpassed 2.23 by the end of the period. This pattern indicates greater maintenance dynamism, possibly linked to fine-grained adjustments in more specialised services. The Code dimension also showed clear impacts. Lines of code (LOC) more than doubled, rising from 100k to 279k lines over the analysed period. Cyclomatic complexity (CYC) followed a similar trend, indicating a growth in logical density. Test coverage (COV), although remaining relatively high, showed a slight decline—from 90% to 87%—which, when considered alongside the growth in service fragmentation and complexity, may reflect increasing demands on automated testing practices.

Finally, the Financial dimension showed a sharp rise in operational costs (CST), from USD 18k in 23Q2 to USD 155k in 24Q4. This increase is associated with higher computational resource usage, infrastructure complexity, and orchestration of highly fragmented services.

4.5 Indices to Support Granularity Definition

To identify growth patterns as well as the degree of system stabilisation or degradation, a method called the *Granularity Saturation Method* was developed. Its purpose is to evaluate whether the analysed indicators suggest a saturation level in the microservice granularity. The diagram (Figure 3) provides a view of the traceability between raw data and the final indicator, showing the calculation process of the **Total Relative Variation Index** (RVI_{TOT}) and the **Granularity Saturation Elasticity** ($\mathcal{E}$).

To measure temporal changes in software quality, we define the *Relative Variation Index* (RVI), a composite metric that synthesises fluctuations across multiple software engineering dimensions. The calculation begins at the metric level, where the relative variation of metric j at quarter i is given by:

$$\gamma_{j_i} = \frac{x_{j_i} - x_{j_{i-1}}}{x_{j_{i-1}} + \epsilon} \tag{1}$$

where x_{j_i} is the value of metric j in quarter i, and ϵ is a small constant to prevent division by zero. This index γ_{j_i} expresses the normalised variation between two consecutive quarters. Table 5 shows the computed values of γ_{j_i} for a representative set of metrics grouped across five semantic dimensions: Granularity (TMS), Development Effort (TMB, EFT), Operation (FCH), Code (LOC, CYC, COV), and Financial (CST).

Table 5: Metric-level relative variation (γ_{j_i}) per quarter and assigned weights (w_j)

Q	TMS	TMB	EFT	FCH	LOC	CYC	COV	CST
23Q2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
23Q3	1.00	-0.24	-0.14	1.07	0.10	0.80	-0.11	0.11
23Q4	0.26	0.27	-0.08	-0.35	0.18	0.00	-0.00	3.25
24Q1	0.10	-0.03	0.09	0.20	0.16	0.05	-0.01	0.35
24Q2	0.19	-0.35	0.08	-0.24	0.13	0.00	0.01	0.47
24Q3	0.22	-0.05	0.00	-0.01	0.22	0.10	0.01	0.02
24Q4	0.16	0.17	0.00	0.88	0.32	0.04	0.01	-0.11
w_j	1	1	1	1	1	1	**-1**	1

Each metric is associated with a directional weight (w_j), indicating whether an increase represents a positive or negative impact. For instance, test coverage (COV) receives a negative weight ($w_{\text{COV}} = -1$) because increases in coverage are beneficial, while increases in most other metrics imply greater effort, complexity, or cost. To compute the *dimension-level variation index* for each quarter i, we aggregate the weighted variations of all metrics j within the same semantic dimension d, normalised by the total absolute weight:

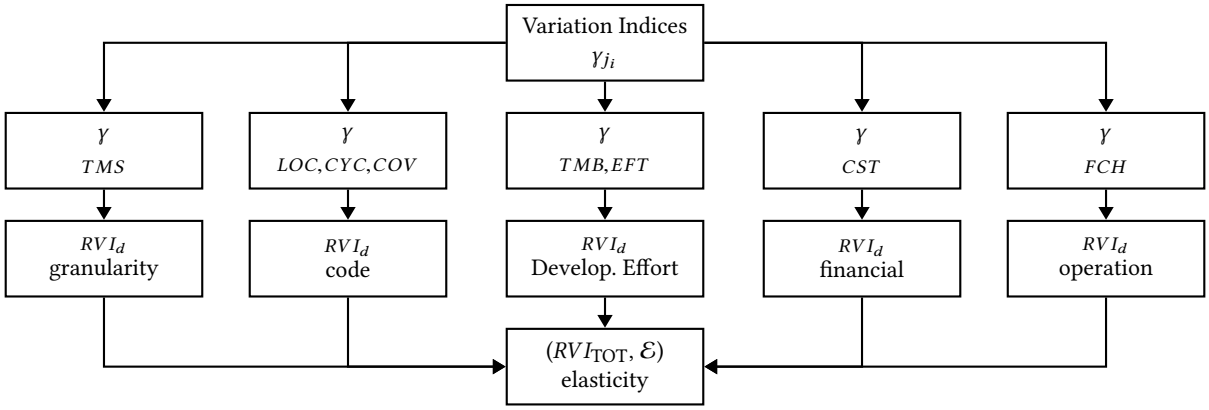
$$RVI_{d,i} = \frac{\sum_{j \in d} (\gamma_{j_i} \cdot w_j)}{\sum_{j \in d} |w_j|} \tag{2}$$

Finally, the *Total Relative Variation Index* for quarter i is defined as the sum of all partial indices:

$$RVI_{\text{TOT},i} = \sum_{d} RVI_{d,i} \tag{3}$$

Table 6 presents the dimension-level and total RVI values for each quarter, alongside the number of active microservices (MSA) and an auxiliary metric $\mathcal{E}$, used for exploratory interpretation. These

Figure 3: Granularity Saturation Method



values provide a temporal perspective on how architectural characteristics evolved, particularly in relation to the increasing fragmentation observed through the number of microservices.

Table 6: Relative Variation Index (RVI) per dimension and quarter, including microservice count (MSA)

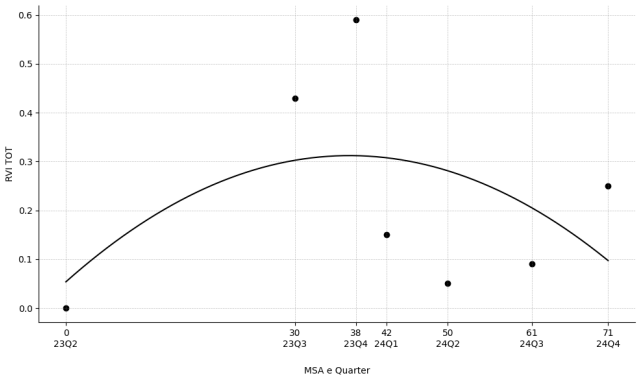
		RVI						
Q	**MSA**	**GRA**	**EFT**	**OPE**	**COD**	**FIN**	**TOT**	$\mathcal{E}$
23Q2	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00
23Q3	30	0.17	-0.06	0.18	0.13	0.02	0.43	0.72
23Q4	38	0.04	0.03	-0.06	0.03	0.54	0.59	1.32
system in production								
24Q1	42	0.02	0.01	0.03	0.03	0.06	0.15	-13.37
24Q2	50	0.03	-0.05	-0.04	0.03	0.08	0.05	-6.38
24Q3	61	0.04	-0.01	-0.00	0.06	0.00	0.09	2.64
24Q4	71	0.03	0.03	0.15	0.06	-0.02	0.25	7.06

MSA = Number of microservices; GRA = Granularity; EFT = Development Effort; OPE = Operation; COD = Code; FIN = Financial; TOT = Total

These data points suggest that RVI can be used to capture phases of architectural transition. In early quarters, high variation reflects structural change and system realignment. To further explore this relationship, Figure 4 plots a polynomial regression between the number of microservices (MSA) and the total RVI (TOT). While the visual trend may suggest inflection points in architectural saturation or recovery, such interpretations should be approached with caution. The aim of this graph is not to prescribe an ideal number of microservices, but to support exploratory analysis of how variation behaves as fragmentation increases. More robust conclusions would require broader empirical validation.

To formalise this observation, we introduce the **Granularity Saturation Elasticity** ($\mathcal{E}$), a metric that captures the sensitivity of relative variation growth with respect to the growth in microservice count. Conceptually, this elasticity quantifies whether architectural variation is growing proportionally, slower, or faster than the system's fragmentation. The elasticity is computed as the ratio between the logarithmic growth rates of total variation (RVI_i) and microservice count (MSA_i) over time:

Figure 4: Polynomial regression between MSA and RVI TOT



$$\mathcal{E}_i = \frac{\Delta \ln(RVI_i)}{\Delta \ln(MSA_i)} = \frac{\ln(RVI_i + \epsilon) - \ln(RVI_{i-1} + \epsilon)}{\ln(MSA_i + \epsilon) - \ln(MSA_{i-1} + \epsilon)} \quad (4)$$

This formulation helps identify distinct behavioural zones:

- **High Variation Zone** ($\mathcal{E} > 1$): Indicates simultaneous growth across multiple dimensions (effort, complexity, change, cost), usually associated with increased architectural instability and overhead.
- **Reduced Variation Zone** ($0 < \mathcal{E} < 1$): Reflects decelerating variation, suggesting the system may be approaching structural balance.
- **Saturation Zone** ($\mathcal{E} \approx 0$): Reveals a point at which fragmentation continues to grow, but without additional systemic variation, indicating stabilisation.
- **Adverse Effect Zone** ($\mathcal{E} < 0$): Suggests that increased granularity is degrading software quality or maintainability, possibly requiring architectural intervention.

This section has focused primarily on defining and detailing the indices used to support decision-making regarding microservice granularity. The presented indices, particularly the Relative Variation Index (RVI) and Granularity Saturation Elasticity ($\mathcal{E}$), provide a robust quantitative foundation for assessing architectural

trends and identifying critical saturation points in service granularity. However, it is important to emphasise that the practical procedures for applying these indices in making effective granularity adjustment decisions require further elaboration. In future publications, we plan to comprehensively detail this step of the Granulify process, explicitly addressing activities, qualitative criteria, and the decision-making workflow involved in continuous granularity definition and management.

5 EVALUATION

This section presents the empirical evaluation of the proposed granularity management metrics in the context of a real-world case study. The goal is to investigate the extent to which the introduced indicators—*Total Relative Variation Index* (*RVI*) and *Saturation Elasticity* ($\mathcal{E}$)—are able to capture architectural shifts, granularity trends, and indicators of saturation over time. These metrics are interpreted in light of the trade-offs identified earlier, particularly those concerning modularity, coordination effort, and maintainability. Where appropriate, existing structural metrics from the literature are referenced to contrast their coverage and sensitivity in relation to the proposed indicators.

5.1 Exploratory Growth (23Q2–23Q4)

In the first two analysed quarters, the system showed consistent variation across metrics, with the *RVI* increasing from 0.00 (23Q2) to 0.43 (23Q3) and 0.59 (23Q4), accompanied by an increase in granularity elasticity, which reached $\mathcal{E} = 1.32$. These values characterise the **High Variation Zone**, indicating a phase of accelerated growth in granularity, effort, changes, and complexity.

This period corresponds to the initial migration phase (23Q2 - 23Q4), during which the system had not yet been deployed in a production environment. The technical team chose to adopt a highly granular approach with strong use of serverless functions (FaaS), aiming to maximise component independence and enable more agile event orchestration. This decision was based on load simulations and assumptions about future needs for scalability and resilience.

The observed metrics faithfully reflect this strategy: the increase in the total number of services (from 15 to 38) and structural complexity, without perceptible penalties in productivity or stability, suggests that the extreme granularity model was, at that time, compatible with the exploratory stage of the project. The empirical validation of the metrics at this stage demonstrates that the transitioning architecture was aligned with the designed technical objectives — even though these decisions had not yet been tested in a real operational environment.

5.2 Production Impact (24Q1)

From quarter 24Q1 onward, a critical inflection point in the system's behaviour is observed. Although the total number of microservices increased from 38 to 42, granularity elasticity plummeted to $\mathcal{E} = -13.37$, signalling that structural growth no longer translated into organisational or technical benefits. This scenario fits within the **Adverse Effect Zone**, suggesting that continued fragmentation began to compromise system productivity and stability.

This quarter marks the beginning of operation in a real production environment. The introduction of actual load, business demands, and operational incidents revealed the practical limits of the extremely granular architecture conceived up to that point. The metrics accurately reflect this transition: the FCH increased significantly, indicating instability, and operational costs surged, reflecting orchestration overhead and high consumption of computing resources. Moreover, several bugs were recorded, along with multiple redeployment cycles — a clear sign that, although theoretically well-planned, the structure was not resilient to operational challenges.

In this context, the metrics demonstrated their utility as diagnostic instruments: by highlighting deterioration in key aspects (EFT, FCH, CST), they signalled the inadequacy of the current architecture and prompted critical reflection within the team. The data not only corroborated the qualitative perceptions of the engineers involved, but also grounded internal discussions about the need to reassess the chosen granularity level.

5.3 Architectural Reassessment (24Q2)

In the following quarter (24Q2), elasticity remained negative ($\mathcal{E} = -6.38$) and the relative variation index remained low ($RVI = 0.05$), indicating persistent adverse effects. This period was marked by a strategic repositioning: explicitly recognising the negative impacts of excessive fragmentation — such as maintenance difficulties, recurring production failures, and rising costs — the team began a deliberate restructuring effort.

Services with low autonomy or high functional coupling, whose maintenance demanded disproportionate effort, were identified. In response, the organisation started consolidating such microservices and deactivating redundant or underutilised instances, reorienting the system toward a more sustainable granularity.

This decision represents a turning point in architectural governance and reinforces the practical value of metrics in supporting decision-making. In retrospect, it can be said that the indicators faithfully captured the mismatch between the planned architecture and operational reality, functioning as tools for continuous feedback.

5.4 Recovery and Alignment (24Q3–24Q4)

This architectural consolidation marked a paradigm shift: the strategy moved from pursuing maximum granularity — guided by principles of functional independence and scalability — to a more pragmatic approach, centred on the idea of sustainable granularity. The effectiveness of this shift began to appear in the data in quarter 24Q3, when elasticity returned to a positive value ($\mathcal{E} = 2.64$), once again placing the system in the **High Variation Zone**.

This recovery suggests a possible correlation between the granularity adjustments and initial improvements observed in productivity, stability, and efficiency — although these relationships require further validation. In the subsequent quarter (24Q4), this trend solidifies: elasticity reached $\mathcal{E} = 7.06$ and the *RVI* rose to 0.25. These values indicate that the interventions — such as integrating tightly coupled services and removing redundant components — produced perceptible structural effects.

Improvements in metrics, especially in change frequency, team effort, and operational costs, point to a realignment between architectural decisions and the organisation's strategic goals. In this case, the metrics not only reflect the system's state but also feed back into the decision-making process, serving as continuous instruments of technical governance.

5.5 Comparison with Existing Metrics and Insights from the Approach

Compared to existing structural metrics commonly found in the literature—such as service count, coupling, cohesion, and complexity—the proposed indicators offer a more integrative and temporal perspective. Traditional metrics tend to provide static snapshots of system structure, whereas the *Total Relative Variation Index* (*RVI*) captures the cumulative rate of architectural changes over time, reflecting the pace and intensity of structural evolution.

Similarly, the *Saturation Elasticity* ($\mathcal{E}$) incorporates variation thresholds and growth dynamics to identify potential saturation points, a scenario not addressed by conventional metrics. Together, these indicators extend current metric-based approaches by emphasising system dynamics, enabling a more proactive strategy for granularity management aligned with architectural stability and maintainability concerns.

The patterns observed over time strengthen the central hypothesis of this study: the absence of systematic mechanisms for continuous granularity management can lead distributed systems into stages of stagnation, saturation, or even operational regression. On the other hand, adopting an evidence-based and iterative model — as proposed by the Granulify — enables dynamic monitoring and adjustment of granularity levels, balancing modularity, performance, and operational cost.

The application of Granulify in this case study demonstrated its practical utility: empirical data collected over seven quarters enabled the identification of different zones of structural behaviour and the connection of those zones to real events and decisions within the organisation. As such, the proposed approach proves to be a promising tool to support architectural decision-making, contributing to more adaptive and metrics-driven governance in system modernisation contexts.

In the next phases of this research, the approach will be expanded and validated in new domains and projects within the organisation, aiming to consolidate its large-scale applicability and assess its contribution to the evolutionary sustainability of microservice architectures.

6 CONCLUSION

This study analysed the impact of granularity on the evolution of a microservice-based system within the real-world context of a financial institution. Through the proposal and application of the **Granulify**, it was possible to demonstrate the need for continuous granularity management throughout the system's lifecycle, enabling iterative adjustments based on metrics and quantitative indicators.

The results indicated that architectural fragmentation, although essential for modularisation and scalability, initially led to increased unpredictability in effort and change frequency. Over time, a stabilisation trend was observed, suggesting that teams and processes progressively adapted to the new architectural reality.

The approach also enabled the identification of signs of granularity saturation, reinforcing that architectural evolution must consider not only technical aspects, but also organisational and operational constraints.

6.1 Study Limitations

This study presents some limitations. First, it was conducted in a single organisational context, which may restrict the generalisability of the results. Furthermore, the approach has not yet been validated across multiple teams or different domains, a gap that will be addressed in the next phases of the research.

For these reasons, the findings should be regarded as indicative evidence whose generalisability hinges on replications across multiple domains and organisations. Another limitation concerns the nature of the metrics used, which, although widely recognised in the literature, may not fully capture qualitative aspects such as the subjective experience of teams or organisational factors that are not directly measurable.

6.2 Future Work

The next stages of this research include replicating Granulify in other industrial scenarios, focusing on different domains and organisations. Plans also include enhancing the indicators used, incorporating architectural quality metrics and dependency analysis. Another relevant aspect will be the investigation of the impacts of continuous granularity management on the governance of distributed systems and the organisational processes that support their evolution. We expect this work to not only deepen the understanding of how granularity affects the evolution of microservice-based systems but also to foster the development of more robust and adaptable methods for architectural management in contemporary software engineering.

ARTEFACTS AVAILABILITY

The data and artefacts used in this study originate from a real-world investment management platform of a major financial institution in Brazil. Due to strict confidentiality agreements and internal compliance policies, the raw data and system artefacts cannot be publicly shared. However, they are available for the review process under a non-disclosure basis if required.

ACKNOWLEDGMENTS

For the purpose of open access, the author has applied a Creative Commons Attribution (CC BY) licence to any Author Accepted Manuscript version arising from this submission.

REFERENCES

[1] Yalemisew Abgaz, Andrew McCarren, Peter Elger, David Solan, Neil Lapuz, Marin Bivol, Glenn Jackson, Murat Yilmaz, Jim Buckley, and Paul Clarke. 2023. Decomposition of Monolith Applications Into Microservices Architectures: A Systematic Review. *IEEE Transactions on Software Engineering* 49, 8 (Aug. 2023), 4213–4242. https://doi.org/10.1109/tse.2023.3287297

[2] Mehdi Ait SAID, Abdellah EZZATI, Soukaina MIHI, and Lahcen BELOUADDANE. 2024. Microservices Adoption: An Industrial Inquiry into Factors Influencing Decisions and Implementation Strategies. *International Journal of Computing and Digital Systems* 15, 1 (March 2024), 1417–1432. https://doi.org/10.12785/ijcds/1501100

[3] Alex Malmann Becker and Daniel Lucrédio. 2020. The Impact of Microservices on the Evolution of a Software Product Line. In *Proceedings of the 14th Brazilian Symposium on Software Components, Architectures, and Reuse (SBCARS '20)*. ACM. https://doi.org/10.1145/3425269.3425275

[4] Shanay Behrad, David Espes, Philippe Bertin, and Cao-Thanh Phan. 2021. Impacts of Service Decomposition Models on Security Attributes: A Case Study with 5G Network Repository Function. In *2021 IEEE 7th International Conference on Network Softwarization (NetSoft)*. IEEE. https://doi.org/10.1109/netsoft51509.2021.9492620

[5] Justus Bogner, Jonas Fritzsch, Stefan Wagner, and Alfred Zimmermann. 2021. Industry practices and challenges for the evolvability assurance of microservices: An interview study and systematic grey literature review. *Empirical Software Engineering* 26, 5 (July 2021). https://doi.org/10.1007/s10664-021-09999-9

[6] Justus Bogner, Stefan Wagner, and Alfred Zimmermann. 2019. On the impact of service-oriented patterns on software evolvability: a controlled experiment and metric-based analysis. *PeerJ Computer Science* 5 (Aug. 2019), e213. https://doi.org/10.7717/peerj-cs.213

[7] Thelma Colanzi, Aline Amaral, Wesley Assunção, Arthur Zavadski, Douglas Tanno, Alessandro Garcia, and Carlos Lucena. 2021. Are we speaking the industry language? The practice and literature of modernizing legacy systems with microservices. In *15th Brazilian Symposium on Software Components, Architectures, and Reuse (SBCARS '21)*. ACM. https://doi.org/10.1145/3483899.3483904

[8] Famke Driessen, Luís Ferreira Pires, João Luiz Rebelo Moreira, Paul Verhoeven, and Sander van den Bosch. 2024. A Quantitative Assessment Method for Microservices Granularity to Improve Maintainability. In *Enterprise Design, Operations, and Computing. EDOC 2023 Workshops*, Tiago Prince Sales, Sybren de Kinderen, Henderik A. Proper, Luise Pufahl, Dimka Karastoyanova, and Marten van Sinderen (Eds.). Springer Nature Switzerland, Cham, 211–226. https://doi.org/10.1007/978-3-031-54712-6_13

[9] Thomas Erl. 2017. *Service-oriented architecture* (second edition ed.). Prentice Hall, Upper Saddle River, NJ. Includes index.

[10] Neal Ford. 2021. *Software architecture* (first edition ed.). The MIT Press, Cambridge, Massachusetts. Made available through: Safari, an O'Reilly Media Company.

[11] Neal Ford. 2023. *Building evolutionary architectures* (second edition ed.). O'Reilly, Beijing.

[12] Jonas Fritzsch, Justus Bogner, Stefan Wagner, and Alfred Zimmermann. 2019. Microservices Migration in Industry: Intentions, Strategies, and Challenges. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE. https://doi.org/10.1109/icsme.2019.00081

[13] Jonas Fritzsch, Justus Bogner, Alfred Zimmermann, and Stefan Wagner. 2019. *From Monolith to Microservices: A Classification of Refactoring Approaches*. Springer International Publishing, 128–141. https://doi.org/10.1007/978-3-030-06019-0_10

[14] Sara Hassan, Rami Bahsoon, and Rajkumar Buyya. 2022. Systematic scalability analysis for microservices granularity adaptation design decisions. *Software: Practice and Experience* 52, 6 (Jan. 2022), 1378–1401. https://doi.org/10.1002/spe.3069

[15] Sara Hassan, Rami Bahsoon, and Rick Kazman. 2020. Microservice transition and its granularity problem: A systematic mapping study. *Software: Practice and Experience* 50, 9 (June 2020), 1651–1681. https://doi.org/10.1002/spe.2869

[16] Munezero Immaculée Josélyne, Doreen Tuheirwe-Mukasa, Benjamin Kanagwa, and Joseph Balikuddembe. 2018. Partitioning microservices: a domain engineering approach. In *Proceedings of the 2018 International Conference on Software Engineering in Africa (ICSE '18)*. ACM. https://doi.org/10.1145/3195528.3195535

[17] Luís Nunes, Nuno Santos, and António Rito Silva. 2019. *From a Monolith to a Microservices Architecture: An Approach Based on Transactional Contexts*. Springer International Publishing, 37–52. https://doi.org/10.1007/978-3-030-29983-5_3

[18] D. L. Parnas. 1972. On the criteria to be used in decomposing systems into modules. *Commun. ACM* 15, 12 (Dec. 1972), 1053–1058. https://doi.org/10.1145/361598.361623

[19] Mark Richards. 2015. *Microservices vs. service-oriented architecture*. O'Reilly Media.

[20] Sh. Salii, J. Ajdari, and Xh. Zenuni. 2023. Migrating to a microservice architecture: benefits and challenges. In *2023 46th MIPRO ICT and Electronics Convention (MIPRO)*. IEEE. https://doi.org/10.23919/mipro57284.2023.10159894

[21] Mary Shaw and David Garlan. 1996. *Software architecture*. Prentice Hall, Upper Saddle River, NJ. Literaturverz. S. 227 - 237.

[22] Davide Taibi and Valentina Lenarduzzi. 2018. On the Definition of Microservice Bad Smells. *IEEE Software* 35, 3 (May 2018), 56–62. https://doi.org/10.1109/ms.2018.2141031

[23] Rafik Tighilt, Manel Abdellatif, Imen Trabelsi, Loïc Madern, Naouel Moha, and Yann-Gaël Guéhéneuc. 2023. On the maintenance support for microservice-based systems through the specification and the detection of microservice antipatterns. *Journal of Systems and Software* 204 (Oct. 2023), 111755. https://doi.org/10.1016/j.jss.2023.111755

[24] Mathawee Tusjunt and Wiwat Vatanawood. 2018. Refactoring Orchestrated Web Services into Microservices Using Decomposition Pattern. In *2018 IEEE 4th International Conference on Computer and Communications (ICCC)*. IEEE. https://doi.org/10.1109/compcomm.2018.8781036

[25] Victor Velepucha and Pamela Flores. 2023. A Survey on Microservices Architecture: Principles, Patterns and Migration Challenges. *IEEE Access* 11 (2023), 88339–88358. https://doi.org/10.1109/access.2023.3305687

[26] Fredy H. Vera-Rivera, Carlos Gaona, and Hernán Astudillo. 2021. Defining and measuring microservice granularity—a literature overview. *PeerJ Computer Science* 7 (Sept. 2021), e695. https://doi.org/10.7717/peerj-cs.695

[27] Hulya Vural and Murat Koyuncu. 2021. Does Domain-Driven Design Lead to Finding the Optimal Modularity of a Microservice? *IEEE Access* 9 (2021), 32721–32733. https://doi.org/10.1109/access.2021.3060895

[28] Muhammad Waseem, Peng Liang, Mojtaba Shahin, Amleto Di Salle, and Gastón Márquez. 2021. Design, monitoring, and testing of microservices systems: The practitioners' perspective. *Journal of Systems and Software* 182 (Dec. 2021), 111061. https://doi.org/10.1016/j.jss.2021.111061

[29] Yang Zhao, Ran Mo, Yao Zhang, Siyuan Zhang, and Pu Xiong. 2022. Exploring and understanding cross-service code clones in microservice projects. In *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension (ICPC '22)*. ACM. https://doi.org/10.1145/3524610.3527925

[30] Xin Zhou, Shanshan Li, Lingli Cao, He Zhang, Zijia Jia, Chenxing Zhong, Zhihao Shan, and Muhammad Ali Babar. 2023. Revisiting the practices and pains of microservice architecture in reality: An industrial inquiry. *Journal of Systems and Software* 195 (Jan. 2023), 111521. https://doi.org/10.1016/j.jss.2022.111521

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009