# Human in the loop for microservice identification: An exploratory study

João Castanho
State University of Maringá
Maringá, Brazil
joao.staub42@gmail.com

Cezar G. Britez
State University of Maringá
Maringá, Brazil
cezar.cgb18@gmail.com

Willian M. Freire
Federal University of Technology-Paraná
Campo Mourão, Brazil
willianfreire@utfpr.edu.br

Aline M. M. M. Amaral
State University of Maringá
Maringá, Brazil
ammmamaral@uem.br

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

Wesley K. G. Assunção
NC State University
Raleigh, USA
wguezas@ncsu.edu

## Abstract

Legacy monolithic systems face scalability and maintainability challenges, driving organizations toward microservice architectures. Automated Search-Based Software Engineering (SBSE) techniques generate candidate decompositions by balancing structural metrics, but operate in isolation from architects tacit domain knowledge, often yielding unbalanced, low-adoption partitions. We propose *i-toMicroservices*, an interactive extension of the method-level toMicroservices approach. Grounded in Interactive SBSE, it embeds a pause-and-intervene mechanism that lets decision-makers (DMs) inspect intermediate architectures and apply four refinement operators (Freeze, Split, Merge, Move), while an intra-cluster propagation mechanism reduces cognitive fatigue by preserving population diversity. An exploratory study with three software engineers shows that human-in-the-loop intervention can help correct automated partition imbalances, increase topological similarity to reference architectures by up to 23.7% in the best-performing case, and encapsulate cohesive business sub-domains that automated optimization alone fails to isolate. A qualitative analysis by two experts across six dimensions suggests decomposition quality may be shaped by the DM's profile.

## Keywords

Software Modernization, Interactive Search-Based Software Engineering, Monolith decomposition, Multi-objective optimization

## 1 Introduction

The technology industry has undergone a significant transformation in distributed systems architecture, driven by organizations that migrated from monolithic to microservice-based architectures [15]. Monolithic applications, built as tightly integrated units sharing a common codebase, simplify initial development but introduce scalability, startup, and changeability limitations as the system grows [8, 25, 37]. Microservice architectures address these issues by decomposing the application into smaller, capability-scoped services communicating through interfaces such as REST APIs [47], yielding loosely coupled systems that support independent deployment, horizontal scalability, continuous delivery, and flexible technology stacks [15].

Despite that, decomposing a monolithic system into microservices remains a challenge [2, 45, 46]. Traditionally, software architects rely on manual source-code analysis and subjective expertise to redraw system boundaries, an approach that scales poorly as the target software grows in size and architectural density [19]. To mitigate this, Search-Based Software Engineering (SBSE) [21] enables the use of multi-objective optimization algorithms, such as NSGA-III, for generating service decompositions. A representative approach is toMicroservices [5], which models a legacy monolith as a fine-grained method-level dependency graph and employs NSGA-III to balance trade-offs among coupling, cohesion, network overhead, functionality modularization, and service reuse, with edges encoding static invocations, dynamic runtime traces, and estimated payload sizes. In comparison to other related SBSE approaches [50], toMicroservices is the best performing approach, achieving high Business Context Purity (61.8%) and Domain Independence (74.6%).

Although these automated approaches excel at mathematically sound structural decompositions, they evaluate solutions solely through rigid objective functions, isolated from the contextual and strategic business requirements known only to the human designer [10, 49]. Recent empirical evaluations confirm that fully automated techniques frequently produce separate domain elements that developers prefer unified, or generate architectures with low practical adoptability [10]. Specifically, Wang et al. [50] found that, despite the best results achieved by toMicroservices in comparison to related approaches, its method-level decomposition tends to generate unbalanced partitions with low adherence to reference architectures, suggesting that interactive approaches keeping human in the loop could complement what automated analyses miss.

To incorporate human in the loop, Interactive SBSE (iSBSE) proposes integrating the Decision Maker (DM) preferences directly into the optimization cycle [39, 40], periodically pausing the evolutionary process so the DM can inspect intermediate candidates and guide the search by applying refinement constraints, reallocating elements, or locking optimal components [9]. While iSBSE has been successfully explored in product line design [18] and requirements prioritization [4], its systematic integration into the microservice identification cycle remains vastly underexplored [10].

To address these limitations, we propose *i-toMicroservices* [13], extending toMicroservices, a state-of-the-art baseline. Following the iSBSE pause-and-intervene model [9, 39], it pauses NSGA-III

at user-defined checkpoints to present candidate decompositions via a GUI, allowing DMs to apply four refinement operators: *freeze*, *split*, *merge*, and *move*. To reduce fatigue and ensure population consistency, an intelligent change propagation mechanism based on intra-cluster grouping is used [7]. We evaluated our approach with three participants performing interactive interventions on JPetStore and Spring PetClinic [50], followed by a quanti-qualitative analysis with two experts against the automated baseline. Results show human intervention helps correct partition imbalances, improves similarity to reference architectures, and establishes high-fidelity domain boundaries missed by automated criteria.

In summary, this paper contributes: (i) an empirical study on iSBSE refactoring, showing architectural proximity gains up to 23.7% relative to ideal designs; (ii) practical insights for interactive tool engineering, highlighting the need for structural constraints to protect human-refined choices from algorithmic drift; and (iii) a two-dimensional evaluation revealing a misalignment between automated structural metrics and developer-perceived quality, showing that topological balance alone cannot guarantee cohesive, independently deployable microservices.

## 2 Background

SBSE applies search-based optimization techniques to solve complex software engineering problems [22], modeling them as optimization problems with candidate solutions evaluated via objective functions, iteratively explored through selection, crossover, and mutation. Multi-objective algorithms produce a set of non-dominated solutions (the Pareto front) [48]; problems with more than three objectives are classified as many-objective. NSGA-III [48] addresses this by using reference points on a normalized hyperplane to promote a well-distributed, convergent approximation of the Pareto front in high-dimensional objective spaces.

Interactive Search-Based Software Engineering (iSBSE) [16, 39] incorporates the DM's preferences into the optimization process. Unlike fully automated approaches, which often fail to capture tacit domain knowledge through predefined fitness functions, iSBSE enables preference integration before, after, or during optimization. In interactive settings, the algorithm periodically presents intermediate solutions for evaluation, allowing the DM to guide the search by refining constraints, reallocating elements, or preserving desirable components [18]. Although this interaction steers the search toward contextually relevant solutions, repeated manual evaluations can lead to fatigue and reduced decision consistency. Thus, recent research has focused on reducing human effort through architectural strategies, structural constraints, and machine learning techniques that decrease the frequency and complexity of user interventions.

**The toMicroservices approach**. toMicroservices [12] is a fully automated approach that assists architects in defining microservice boundaries for legacy monoliths. Operating at a method level, it models the system as a directed, weighted graph $G = (V, E)$ [5]. Vertices $v \in V$ represent internal methods mapped to business capabilities, while directed edges $e \in E$ capture structural and operational dependencies, including static invocations, dynamic traces, and execution payload sizes.

To explore the search space of potential software partitions, toMicroservices utilizes NSGA-III. The core optimization engine is designed to simultaneously balance five conflicting software engineering metrics, formalizing architectural quality attributes into objective functions: (i) **Cohesion:** Measures the degree of internal relationship within an identified microservice by evaluating the ratio of actual static method invocations inside the service boundaries to the total number of possible internal calls, which is maximized to promote highly focused services; (ii) **Coupling:** Evaluates the external dependencies of a microservice by quantifying the total number of static calls established between its internal methods and methods residing in external services, which is minimized to foster loosely coupled components; (iii) **Network Overhead:** Approximates data transmission performance by calculating parameter payload sizes exchanged at runtime, factoring in protocol metadata (e.g., HTTP headers) and minimizing it to prevent network bottlenecks; (iv) **Functionality Modularization:** Aligns technical method groupings with cohesive business capabilities by prioritizing solutions where each microservice holds a dominant responsibility, minimizing domain duplication across partitions; and (v) **Reuse:** Evaluates partition utility based on external component reuse or direct end-user invocation (minimum twice) within execution traces, maximizing overall system modularity [5].

The toMicroservices approach has a namesake tool, which implements its application. In its original, purely automated execution flow, the tool operates through distinct *a priori* and *a posteriori* phases, lacking a mechanism for active human-algorithm collaboration during optimization. *A priori*, the architect provides the execution log of the legacy system, configures evolutionary parameters (such as mutation rates and the number of generations), and predefines the strict number of target microservices to be generated. The NSGA-III algorithm then runs autonomously until the generation budget is exhausted. Finally, *a posteriori*, the approach outputs a set of Pareto-optimal candidate microservice decompositions.

## 3 Proposal of an Interactive Approach

To propose our interactive approach, we extend the original, fully automated toMicroservices approach [12] to incorporate human expertise directly into the optimization process.

In this sense, our interactive approach called *i-toMicroservices* integrates the DM into the NSGA-III workflow via a pause-and-intervene interaction model, as illustrated in Figure 1, which distinguishes the DM's manual action (green) from the automated algorithmic steps (yellow) and the automated propagation mechanism (blue). A tool was developed to enable the use of our approach; its implementation and public repository are described in [13] and the supplementary material includes some screenshots for illustration [23]. Initially, the DM defines the execution parameters: the target number of microservices, the total number of generations (acting as the global stopping criterion), and the interaction step size, which determines the exact number of generations the evolutionary algorithm executes autonomously before pausing. Upon reaching a predefined checkpoint, execution halts, and the current population of Pareto-optimal candidates is compiled and presented to the DM via a graphical user interface, allowing inspection of

either the full non-dominated front or specific extreme solutions for each objective.

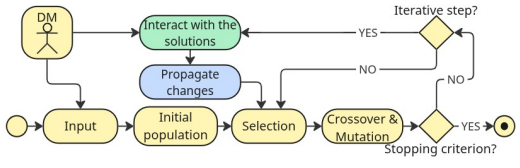


**Figure 1: Interactive process of *i-toMicroservices*.**

Within this interactive loop, human interventions operate directly on the adopted graph-based architectural representation. When inspecting a candidate architecture, the DM can intervene by applying a suite of four specialized refinement operators: *Freeze*, *Split*, *Merge*, and *Move* (represented by the green node in Figure 1). Collectively, these operators target distinct design concerns and address the primary architectural manipulation needs identified in the literature [6, 10–12] and confirmed by practitioner feedback: protecting stable structures, decomposing oversized services, consolidating fragmented components, and reallocating individual methods across service boundaries.

To minimize human effort, these manual adjustments are then automatically propagated—without further DM involvement—to structurally similar solutions throughout the population via an intra-cluster mechanism (the blue node in Figure 1). Rather than being treated as mere suggestions, the resulting human-guided solutions are injected directly into the population during the replacement phase of the NSGA-III algorithm. After the DM defines the size of the subsequent interaction step, the algorithm resumes from its checkpoint, operating on a population enriched and steered by human domain expertise. This collaborative cycle repeats until the global generation budget is exhausted, allowing automated search and human judgment to operate in tandem.

## 3.1 Refining Operators

**Structural integrity constraints.** In *i-toMicroservices*, all four refinement operators share a common design principle: every action is subject to strict structural validation prior to execution. An operation that would leave any microservice empty, target an actively frozen service, or attempt to merge a service with itself is silently rejected, ensuring that no intervention can produce a degenerate or inconsistent architectural state. This defensive design is intentional: because modified individuals are reinjected directly into an active evolutionary loop, they must be expressive enough to capture the DM's semantic intent while preventing structural anomalies.

**Freeze.** The freeze operator immobilizes a microservice partition, shielding it from any subsequent modification by either the evolutionary operators or other manual refinement actions. When the DM identifies a partition that already satisfies domain or non-functional requirements, freezing it effectively reduces the active search space, concentrating the remaining computational budget on architectural regions that still require convergence. Any subsequent refinement operator that targets a frozen microservice terminates immediately without altering its state. By permanently locking validated domain blocks, the architect neutralizes *algorithmic drift*, the tendency of stochastic mutation to reintroduce technical dependencies that the DM has explicitly discarded. Visually, frozen microservices are highlighted with a distinct color in the graphical interface, making their protected status immediately perceptible.

**Split.** The split operator fragments an existing microservice into two independent operational units, addressing one of the most common failure modes of purely automated decomposition: the tendency to produce oversized services that conflate multiple business responsibilities into a single partition due to algorithmic reliance on technical call frequencies rather than semantic domain boundaries [50]. The split operator provides the DM with a mechanism to enforce the Single Responsibility Principle (SRP) [3] directly on the candidate architecture, breaking dense monolithic classes into fine-grained, responsibility-aligned services. Structurally, the operator validates that the source microservice exists and is unfrozen, the chosen method subset is non-empty, and at least one method remains in the original service to prevent depletion.

**Merge.** The merge operator consolidates two distinct microservices into a single cohesive unit, serving as the natural counterpart to split. It allows the DM to counteract excessive service proliferation or over-fragmentation by consolidating components that logically belong to the same sub-domain, making them practical for real-world independent deployment. To minimize disruption to the existing graph topology and preserve current optimization metrics, the operator requires that both target services exist, are unfrozen, and are distinct entities. Upon execution, the operator automatically identifies the smaller service based on method count and sequentially absorbs its vertices into the larger service before purging the redundant, empty container from the architecture.

**Move.** The move operator transfers a specified subset of vertices from a source microservice to a destination microservice, providing the finest level of structural granularity available for DM intervention. It supports the full range of precision-tuning tasks, directly addressing *algorithmic myopia*—the tendency of the optimizer to misallocate technical utilities or to distribute getter/setter methods of a single domain entity across separate services simply because they co-occur in execution flows. The operator allows the DM to reunite these elements under a single bounded context. The system enforces that both the source and destination are unfrozen, distinct entities, and that at least one vertex remains permanently in the origin service to prevent accidental partition deletion. Upon validation, the underlying dependency edges and multi-objective fitness functions are automatically recalculated in real time.

## 3.2 Change Propagation Mechanism

To ensure that manual interventions are not restricted to a single individual in the population—which would force the DM to repeat adjustments across multiple candidates—*i-toMicroservices* incorporates an intelligent change propagation mechanism powered by intra-cluster grouping. After each interaction, the current population is partitioned into a fixed number of $k = 5$ clusters using the K-means++ algorithm [7, 33]. Although the experimental design originally targeted 5 clusters (one for each objective plus one for the best overall trade-off), the Reuse objective consistently yielded a constant value of 1 due to its structural nature; thus, its redundant cluster was omitted to prevent empty groupings. Each

cluster groups candidate solutions by attributes corresponding to the normalized values of the five objective functions, under the assumption that mathematically proximate candidates share structural resemblances that make the DM's edits mutually relevant.

When the DM modifies a candidate solution by applying one or more operators, the changes are automatically propagated to all other candidates within the same cluster. The precision of this propagation is guaranteed by a fundamental aspect of the graph model: every method in the legacy system's dependency graph is assigned a Universally Unique Identifier (UUID) at graph construction time, and this UUID is consistently preserved across all candidate solutions throughout the evolutionary run. Consequently, when the DM moves the method identified by `UUID_X` from one service to another in the selected candidate, the system locates that same method in every other candidate of the cluster and replicates the operation. If the method does not reside in the expected source service in a given candidate, due to natural population diversity, a semantic fallback strategy is activated: the system searches for `UUID_X` across all services of that candidate and relocates it to the destination service that presents the highest method overlap with the DM's intended target, preserving the architectural intent even when an exact structural match is unavailable. The propagated solutions are then injected into the population during the NSGA-III replacement phase, ensuring that the human-guided structural changes persist into subsequent generations rather than being overwritten by the algorithm's stochastic operators.

## 3.3 Integration with the Evolutionary Cycle

The new four operators are not applied in isolation; they are embedded in a pause-and-intervene mechanism that interrupts the NSGA-III execution at user-defined generational checkpoints. At each checkpoint, the current candidates are presented to the DM through a graphical interface (See [23]), in which each operator is accessible via dedicated controls applied directly over the microservice graph visualization. The DM may inspect any candidate, apply one or more operators, and confirm the changes. The modified solutions are then injected directly into the population before the algorithm resumes, ensuring that the DM's interventions can influence the subsequent search rather than being discarded at the end of the process as in a purely *a posteriori* interaction model.

# 4 Study Design

To evaluate the effectiveness and practical viability of *i-toMicroservices*, four Research Questions (RQs) were formulated. These questions investigate the architectural impact of human intervention on the optimization process and on the DM's perception during system decomposition.

**RQ1: To what extent do the refinement operators improve the structural quality of microservice decompositions compared to the fully automated toMicroservices approach?** The objective of this question is to measure the direct benefit of injecting DM tacit knowledge into the evolutionary cycle of the NSGA-III algorithm, specifically in terms of partition size balance and topological proximity to established reference designs. We compare the solutions generated in our study with *i-toMicroservices* against the purely automated solutions generated by both the original toMicroservices tool and the baseline configurations (reference architectures) obtained in an independent study [50]. So that, we evaluate whether real-time human intervention effectively accelerates convergence toward microservice boundaries that reduce structural imbalances and increase domain adherence.

**RQ2: How does the DM perceive the utility of each refinement operator when decomposing monolithic systems into microservices?** This question examines the DM's view of the utility, clarity, and effectiveness of the refinement operators. Evaluating this perception helps identify which commands most efficiently reduce mental effort and highlights potential usability issues or interface flaws.

**RQ3: How do different DM profiles influence the application strategy and outcomes of the refinement operators?** DMs possess different levels of market experience, domain familiarity, and design preferences. This question investigates how the professional profile (for instance, industry developers versus academic researchers) affects both the frequency of operator use and the solutions generated (microservice decompositions).

**RQ4: What is the architectural quality of the microservice decompositions produced through interactive refinement from a developer's perspective?** Quantitative metrics capture structural properties such as coupling and cohesion, but do not assess whether the resulting microservices form cohesive, independently deployable business units. This RQ evaluates the decompositions through expert assessment across qualitative dimensions, domain alignment, deploy independence, and class integrity, providing a practitioner-grounded complement to RQ1.

To address RQs, the study was conducted in three sequential phases (Figure 2) using two open-source monolithic benchmarks: JPetStore [36] (24 classes, 298 methods, 1,409 of source lines of code) and Spring PetClinic [43] (23 classes, 90 methods, 752 of source lines of code), both with reference architectures of three microservices. Participants with varying experience levels operated *i-toMicroservices* on these systems (**RQ3**), allowing us to evaluate structural outcomes (**RQ1**) and their qualitative perceptions of the refinement operators (**RQ2**, **RQ4**).

**Phase I: Participant Preparation.** In the initial phase, participants underwent preparation and training to enable them to conduct the interactive study independently. Three participants with distinct academic and professional backgrounds were selected to perform the interactive refinement cycle using the tool on the two monolithic systems. *Participant 1* (P1) is a Ph.D. candidate in Computer Science with more than seven years of industry experience in large-scale software development and Java refactoring. *Participant 2* (P2) is a Master's student in Computer Science with over seven years of professional experience as a full-stack developer, including work on legacy system migrations and microservice decomposition initiatives. Complementing these industry-oriented perspectives, *Participant 3* (P3) is a Ph.D. candidate in Computer Science with a predominantly academic background in Software Engineering, whose architectural decision-making is primarily informed by theoretical foundations and software design principles.

This phase comprised four activities (Figure 2): analyzing the source code of each system to understand its business logic and dependencies; attending a lecture on microservice modernization

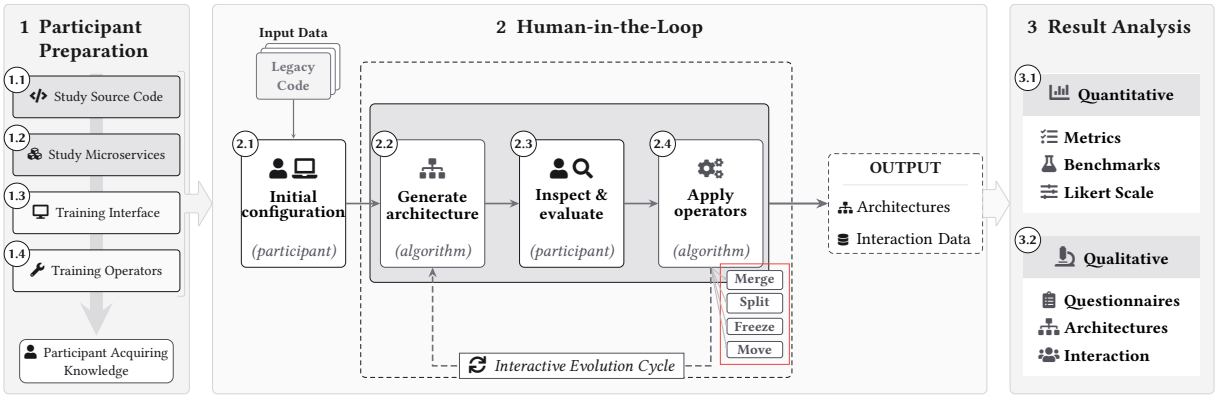


**Figure 2: Phases of the Study.**

concepts and migration strategies; a guided walkthrough of the tool's interface and workflow; and training on the four refinement operators, including their purpose and usage scenarios.

**Phase II: Human-in-the-Loop.** The second phase corresponds to the **Human-in-the-Loop** interactive cycle, which is organized into a sequence of iterative stages that enable participants to progressively refine the generated architectural decompositions. The cycle begins with **(2.1) Initial Configuration**, in which the participant specifies the target number of microservices, the total number of generations to be executed (i.e., the algorithm's stopping criterion), and the first interaction point, defining at which generation the algorithm will pause for user intervention. At each iteration, the **(2.2) Generate Architecture** stage advances the search process from the current population until the next predefined interaction checkpoint, producing a set of candidate architectural decompositions. During **(2.3) Inspect & Evaluate**, the participant analyzes the generated candidates, assesses their architectural quality, and selects a decomposition to refine using the available refinement operators. The DM may inspect any candidate from the current Pareto front, filter the best solution for each objective function, or select the solution with the best trade-off among the objective functions; this allows targeted analysis of specific architectural trade-offs. To improve the traceability and understanding of each intervention, participants were required to provide a written justification for every operator application, documenting the architectural rationale behind their decisions. Next, in **(2.4) Apply Operators**, the selected modifications are propagated to other candidates in the same cluster via the intra-cluster propagation mechanism, allowing the participant's feedback to influence the ongoing search. The participant then defines the next interaction point, determining when the algorithm will pause again for further inspection and refinement.

**Phase III: Result Analysis.** The third phase focused on analyzing the artifacts produced during the interactive cycle using a mixed-methods approach, combining quantitative and qualitative techniques to provide a comprehensive assessment of the proposed approach. The **(3.1) Quantitative Analysis** was based on three complementary sources of evidence. First, architectural metrics were extracted from the decompositions generated by the participants. Second, these metrics were compared against a baseline corresponding to the fully automated, non-interactive execution of the toMicroservices tool, enabling an objective assessment of the structural quality of the resulting decompositions (**RQ1**). Third, participants completed a post-study questionnaire using a Likert scale to evaluate their perceptions of the refinement operators and the overall interactive process, providing quantitative evidence for **RQ2** and **RQ3**.

The **(3.2) Qualitative Analysis** was conducted along three complementary dimensions. First, participants' responses to open-ended questions were analyzed to capture their perceptions, challenges, and suggestions regarding the use of the tool and its refinement operators (**RQ2** and **RQ3**). Second, the best architectural decompositions (solutions) generated during our study were independently assessed by two experts, experienced software professionals from a practitioner's perspective. The evaluators assigned scores across predefined dimensions and subsequently discussed their assessments until reaching a consensus, providing evidence for **RQ4**. Finally, the interaction logs collected throughout the study were analyzed to investigate how participants with different backgrounds and levels of expertise employed the refinement operators during the evolutionary process, providing insights into user behavior and operator usage patterns (**RQ3**).

The second dimension, the qualitative evaluation of the generated architectures (**RQ4**), was conducted independently by two experts with complementary academic and industrial backgrounds. Each expert analyzed the participants' architectural decompositions from a software development perspective and assigned scores across the six predefined evaluation dimensions. *Expert 1* (E1) is a software engineer with more than six years of professional experience, including over four years working with microservice-based systems. *Expert 2* (E2) is a Software Engineer with a Ph.D. in Computer Science, with vast research experience in SBSE and microservice architectures. The practitioners compared their ratings and discussed any discrepancies until reaching a consensus for each evaluated dimension. This process was adopted to reduce potential individual biases and improve the reliability of the qualitative assessment. Further details regarding the evaluation procedure and the resulting consensus are presented in Section 5.

**Table 1: *i-toMicroservices* configuration and interaction adopted by each participant.**

| ID | System | # MS | Gen. | Interactions | Clusters | Operators | Strategy | Outcome |
|---|---|---|---|---|---|---|---|---|
| P1 | JPetStore | 4 | 500 | 83, 433, 440 | Coupling | Move, Freeze | Deep restructure + domain seeding | Local optimum frustration |
| | Spring PetClinic | 4 | 2000 | 700–1900 | Coupling | Move | Continuous refinement | Algorithm resistant to changes |
| P2 | JPetStore | 8 | 384 | 64, 128, 192, 320 | Coupling, Func., Overhead | Split, Move | Granular decomposition + SRP | High granularity |
| | Spring PetClinic | 8 | 96 | 16, 48, 80 | Cohesion, Overhead, Tradeoff | Split, Move | Specialization via Split | Well-defined responsibilities |
| P3 | JPetStore | 5 | 600 | 200, 400, 550 | Overhead, Cohesion, Tradeoff | Move, Split | Iterative refinement + decoupling | Visible business boundaries |
| | Spring PetClinic | 4 | 600 | 100, 200, 300, 400, 500 | Func., Coupling, Overhead | Move, Split | Constant restructuring | Algorithm struggled to assimilate |

**Table 2: Evaluation results for decomposition metrics.**

| System | Tool | # Parts | Partition Size | | | | # Ents. Obs. (%) | CiD | CMod | BCP | DI | DTP | TC | LC | MoJoFM | $c2c_{cvg}$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Min | Mean | Median | Max | | | | | | | | | | 10% | 33% | 50% |
| **JPetStore** | Reference (method-level) | 3 | 73 | 103.3 | 105 | 132 | 302 (100%) | 66.7 | 77.5 | 50.5 | 59.8 | 96.2 | 33.1 | 33.1 | - | - | - | - |
| | toMicroservices | 3 | 8 | 76.3 | 8 | 213 | 229 (75.8%) | 66.7 | 34.6 | **61.2** | **95.1** | **59.7** | **33** | **33** | 51.3 | 50 | **50** | 0 |
| | *i-toMicroservices (P1)* | 4 | 53 | 54.7 | 54.5 | 57 | 219 (72.5%) | 16.7 | 29.5 | 16.7 | 0 | 19.5 | 24.6 | 24.6 | 49 | **100** | 20 | 0 |
| | *i-toMicroservices (P2)* | 8 | 5 | 27.4 | 9.5 | 65 | 219 (72.5%) | **78.6** | 9.3 | 27.8 | 0 | 10.7 | 12 | 12 | 48 | 44.4 | 11.1 | 0 |
| | *i-toMicroservices (P3)* | 5 | 15 | 43.8 | 53 | 61 | 219 (72.5%) | 70 | **41.9** | 55 | 28.9 | 41.3 | 19.7 | 19.7 | **75** | **100** | 50 | **16.7** |
| **Spring-PetClinic** | Reference (method-level) | 3 | 27 | 44 | 30 | 75 | 112 (100%) | 66.7 | 79.8 | 48.9 | 65.3 | 73 | 34 | 36.8 | - | - | - | - |
| | toMicroservices | 3 | 9 | 22.7 | 9 | 50 | 68 (60.7%) | **100** | **60.3** | **33.5** | **55.5** | **68.1** | **32.7** | **33.9** | 56.5 | **100** | **50** | **25** |
| | *i-toMicroservices (P1)* | 4 | 16 | 16.7 | 17 | 17 | 67 (59.8%) | 50 | 20.3 | 0 | 0 | 14.3 | 23.8 | 23.4 | 48.1 | **100** | 20 | 0 |
| | *i-toMicroservices (P2)* | 12 | 2 | 5.6 | 2 | 19 | 67 (59.8%) | 97 | 5.9 | 16.9 | 0 | 15 | 6.9 | 6.6 | 47.2 | 23 | 7.7 | 0 |
| | *i-toMicroservices (P3)* | 5 | 2 | 13.4 | 17 | 20 | 67 (59.8%) | **100** | 35.6 | 23.2 | 3.8 | 59.3 | 21.6 | 19.5 | **61.1** | 83.3 | 50 | 0 |

**Ethical Procedures.** The research received the Ethics Committee approval (number 8.533.538), ensuring compliance with current ethical principles.

## 5 Results

Table 1 summarizes each participant's tool configuration, interactive process, strategies, and results. **ID** identifies the participant, and **System** indicates the benchmark used. The column **# MS** represents the targeted number of microservices predefined by the participant, while **Gen.** denotes the total generation budget allocated for the NSGA-III evolutionary search. Participants freely defined **# MS** at Initial Configuration (Section 4) based on their own reading of each system; P3's choice of 5 microservices for JPetStore versus 4 for Spring PetClinic reflects this system-specific judgment rather than an inconsistency, and does not undermine cross-system comparisons since each participant's results are evaluated against that system's own baseline (Table 2). The **Interactions** column lists the generation numbers at which the participant paused the algorithm to intervene and inspect candidates.

The **Clusters** column indicates which cluster each inspected solution belonged to, reflecting the multi-objective optimization profile of the solutions each DM chose to interact with. The **Operators** column details the refinement actions applied at each checkpoint, **Strategy** outlines the structural logic followed by the participant, and **Outcome** captures the resulting architectural or algorithmic consequence, whether the interaction achieved a well-distributed topology (e.g., "High granularity" or "Visible business boundaries") or faced constraints imposed by the optimization cycle, such as the algorithm resisting human-induced changes—a phenomenon described as algorithmic drift.

### 5.1 Structural Quality and Adherence (RQ1)

The architectural decompositions were evaluated by comparing the participants' refined solutions with baseline solutions. The quantitative results are summarized in Table 2. The baseline evaluation data (i.e., the original toMicroservices tool and the reference architectures) were sourced from the independent study by Wang et al. [50]. These architectural solutions are assessed across structural, domain, and organizational dimensions. Regarding structural and domain integrity, **CiD (Cyclic Independence)** measures the absence of cyclic dependencies among microservices, helping to reduce the risk of cascading runtime failures. **CMod (Code Modularity)** evaluates the quality of the decomposition using the **TurboMQ** metric [35], which quantifies the balance between internal cohesion and external coupling.

To capture semantic alignment, **BCP (Business Context Purity)** measures how strongly a microservice is associated with a specific business context using Shannon Entropy [42]. **DI (Domain Independence)** evaluates whether use cases are encapsulated within individual microservices rather than dispersed across the architecture [29]. **DTP (Database Transaction Purity)** assesses transactional integrity via the degree of table sharing across microservices, thereby supporting data sovereignty and service autonomy [1, 14].

In terms of organizational impact and adherence to baseline models, **TC (Team Contributors)** and **LC (Lifecycle Commits)**

measure operational independence by analyzing whether the service boundaries minimize cross-team code interference and allow isolated repository deployment commits, respectively [50]. Finally, baseline adherence is quantified through **MoJoFM (Topological Similarity)** [51], which computes the proximity to the baseline (0% to 100%) based on the minimum move or join operations required to transform the generated topology into the ideal design, and **c2c$_{cvg}$ (Cluster-to-Cluster Coverage)** [20, 31, 32], which measures boundary accuracy by calculating the percentage of microservices that achieve a specific method overlap threshold (10%, 33%, or 50%) against the reference clusters, where a 50% score denotes high-fidelity encapsulation.

**Topography and Partition Homogeneity.** The structural data suggest that, in some cases human-in-the-loop intervention can reshape the architectural topology toward a more balanced and deployable distribution of responsibilities. As shown in Table 2, when DMs were actively engaged in the optimization process, they tended to reduce the disproportionate concentrations of methods generated by the automated engine.

This particularly evident in Participant 3's (P3) strategy on the JPetStore system. Employing a structured regime of Move and Split operators at generations 200, 400, and 550, P3 targeted oversized services to enforce a decoupling strategy. This interactive refinement guided the algorithm to converge into a well-distributed topology, narrowing the partition range to a sustainable 15–61 entities and allowing the architect to stabilize the system's modularity around homogeneous boundaries, rather than a purely mathematical grouping (Table 2).

**Direct Gains in Structural Adherence for P3.** The benefit of guiding the optimization search is illustrated by P3's structural adherence metrics relative to the baseline model.

**Topological Similarity (MoJoFM):** While the automated engine achieved a MoJoFM score of only 51.3% on JPetStore in isolation , the interactive loop enabled P3 to elevate this architectural proximity to 75.0% (Table 2). This metric shift indicates that the iterative reallocation of methods discarded false technical execution dependencies in favor of domain-aligned boundaries. A consistent improvement was also observed in the Spring PetClinic system, where P3 constant restructuring strategy (Table 1) propelled the MoJoFM score to 61.1%, outperforming the automated baseline of 56.5% (Table 2).

**High-Fidelity Cluster Coverage ($c2c_{cvg}$):** Under a strict 50% similarity threshold, the automated tool scored 0.0% on *JPetStore*, meaning it failed to establish even a single service partition that mirrored the ideal baseline design (Table 2). Through interactive refinement, P3 successfully guided the evolutionary search to achieve a 16.7% high-fidelity coverage (Table 2). This suggests that, in this case, human intervention can encapsulate cohesive sub-domains that the objectives cannot automatically isolate.

**Trade-offs and Behavioral Constraints.** The empirical results also reveal a compelling trade-off regarding service granularity. P2, leveraging high technical seniority (Table 1), utilized the Split operator intensively as an explicit mechanism to enforce the SRP [34] over dense domain classes. On Spring PetClinic, this specialization strategy generated a high-granularity architecture of 12 microservices, which deliberately lowered localized transactional purity metrics (Table 2) in exchange for structural flexibility and independent deployability.

Furthermore, the interactive records in Table 1 highlight how the operators function as defensive mechanisms against algorithmic drift. When the algorithm reached local optima and resisted manual adjustments, as experienced by P1 and P3, who noted that the automated steps occasionally attempted to reintroduce discarded technical dependencies, the Freeze operator emerged as a critical capability. By permanently locking validated domain blocks (such as P1 freezing core services in JPetStore), the architect was able to neutralize stochastic mutation, forcing the remaining generations to optimize around the established human design.

**Answer for RQ1:** In this exploratory study, interactive refinement indicates potential structural improvements, primarily driven by P3's strategy: topological similarity to baseline designs increased by up to 23.7% (MoJoFM), and domain boundaries were established that the automated approach alone did not achieve. This gain was not observed uniformly across participants.

## 5.2 Operator Usefulness and Workload (RQ2)

To investigate how DM perceive the usefulness of each refinement operator, quantitative ratings were collected on a 5-point Likert scale (1-Not very useful to 5-Extremely Useful), along with qualitative insights from open-ended responses and interaction logs [23]. The empirical findings show that while there is a general consensus on the utility of interactive refinement, the operators are valued differently depending on the specific architectural scenario and the DM's strategic preferences. Table 3 summarizes the individual ratings alongside their median scores.

The Move operator received the highest absolute acceptance, with two of the three participants rating it as extremely useful. DMs viewed it as an essential mechanism for fine-grained adjustments to service boundaries. Architects noted that it provides the precision needed to overcome *algorithmic myopia*, enabling manual realignment of implementation methods based on domain, whereas mathematical objectives optimize only technical execution paths.

Split exhibited a balanced utility distribution and was primarily perceived as a strategic mechanism for resolving architectural deadlocks. Participants viewed it as an effective way to enforce the SRP [34] by decomposing large, monolithic-like classes. For example, P2 used it to separate dense business logic from shared technical infrastructure. However, its effectiveness was context-dependent, relying on the initial decomposition generated by NSGA-III and the scale of the legacy system.

Freeze received neutral to positive ratings (3–5). DMs viewed it as a valuable mechanism for preserving optimized sub-domains while focusing the remaining computational budget on unexplored components. However, its practical effectiveness was limited by an implementation issue: during evolutionary replacement, frozen solutions were sometimes discarded, causing loss of human-locked states through algorithmic drift in subsequent generations.

The Merge operator showed the greatest variability in ratings (2–4) but was the least utilized. DMs reported that consolidating multiple service containers was rarely necessary, largely because the benchmark systems were relatively small and automated optimization seldom produced excessive service fragmentation, reducing the need for merging.

**Table 3: Likert-scale ratings of the refinement operators.**

| Operator | P1 | P2 | P3 | Median |
|---|---|---|---|---|
| Freeze | 3 | 5 | 3 | 3.0 |
| Split | 3 | 5 | 4 | 4.0 |
| Merge | 3 | 2 | 4 | 3.0 |
| Move | 3 | 5 | 5 | 5.0 |

**Table 4: Experts' evaluation of the six architectural solutions across six quality dimensions.**

| Solution | # MS | BC | Dp | In | Bl | Ct | Fl | Avg |
|---|---|---|---|---|---|---|---|---|
| S1 (P1) JPetStore | 4 | 2 | 2 | 2 | **5** | 4 | 2 | 2.8 |
| S2 (P2) JPetStore | 5 | **5** | 4 | 4 | 3 | 4 | 4 | **4.0** |
| S3 (P3) JPetStore | 8 | 1 | 1 | 1 | 2 | 2 | 2 | 1.5 |
| S1 (P1) PetClinic | 4 | 1 | 2 | 2 | **5** | 4 | 1 | 2.5 |
| S2 (P2) PetClinic | 5 | **5** | **5** | 4 | 3 | 4 | **5** | **4.3** |
| S3 (P3) PetClinic | 12 | 1 | 1 | 1 | 2 | 1 | 1 | 1.2 |

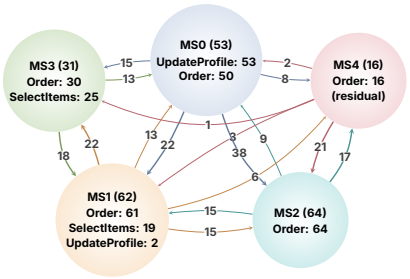


**Figure 3: Representation of the best solution for JPetStore.**

**Answer for RQ2:** DMs perceived the refinement operators as effective for injecting domain knowledge and controlling service granularity, with Move and Split as the primary mechanisms for adjusting automated partitions. Freeze and Merge held strategic value, but their utility was context-dependent, relying on the initial topology generated by NSGA-III and system scale.

## 5.3 Impact of DM Profiles on Strategies (RQ3)

To evaluate how different DMs profiles shape interactive refinement, we examine the operational patterns in Table 1, contrasting the industry-driven strategies of P1 and P2 against the theoretically guided approach of P3. The empirical data reveal that professional background, industry seniority, and refactoring familiarity are associated with differences in how architects select solutions, apply operators, and negotiate trade-offs with the automated engine.

**Industry-Seasoned Profiles (P1 and P2).** Both participants (7+ years of experience) exhibited a strong inclination toward rigorous structural control, though through different operator cadences. P1 combined PhD research with industry seniority, pursuing a "deep restructuring and domain seeding" strategy on JPetStore: letting the algorithm run until generation 83 to clean basic execution calls, then applying Move to enforce domain boundaries, and finally Freeze at generation 440 to lock validated structures. On Spring PetClinic, P1 shifted to "continuous refinement" focused on the Coupling cluster, but faced "local optimum frustration" as the engine resisted complex human-induced shifts. P2, a seasoned full-stack developer, prioritized aggressive decoupling via the Split operator, targeting 8 microservices and initiating splits as early as generation 16 on Spring PetClinic to enforce the SRP—yielding "high granularity" with "well-defined responsibilities" and suggesting that maintainability-focused profiles may proactively drive granularity choices rather than passively refining automated solutions.

**Academic Profile and Iterative Domain Alignment (P3).** P3, with a purely academic background and no industry experience, adopted an iterative, exploratory approach: distributing interactions evenly across fixed generational milestones and exploring multiple clusters (Overhead, Cohesion, Functionality, and Tradeoff). Using Move and Split for "constant restructuring", P3 iteratively shifted methods to isolate business functionalities, yielding "visible business boundaries" on JPetStore. However, on Spring PetClinic, the "algorithm struggled to assimilate" the constant changes, suggesting that a theoretically guided profile may be more attuned to identifying macro-level domain trade-offs but, lacking industry refactoring experience, remain more reliant on the algorithm's willingness to adapt, leaving the strategy vulnerable to local optima.

**Answer for RQ3:** In this exploratory study, DM profiles appeared to shape interaction dynamics differently: industry-seasoned architects applied operators (e.g., Split, Freeze) more assertively to enforce maintainability principles (e.g., SRP) and lock down granularity early, while the academic profile relied on more uniform, iterative adjustments (Move) to progressively map business boundaries, facing higher friction when the algorithm resisted changes. Given the sample size (n=3), these patterns are preliminary indications rather than generalizable claims.

## 5.4 Qualitative Architecture Evaluation (RQ4)

The quantitative metrics in Table 2 capture the structural properties of each solution but do not evaluate whether the resulting microservices (MS) are cohesive as independent business units. To address this limitation, the best solutions generated by each participant (P1, P2, and P3) were qualitatively evaluated by two experts (E1 and E2) (see Section 4). We consider the best solution to be the one that offers the best trade-off among the five objective functions. Both experts qualitatively evaluated each solution across six dimensions, scored from 1 (strongly disagree) to 5 (strongly agree): **Bounded Contexts (BC)** means each MS corresponds to a clear and recognizable business domain; **Deploy Independence (Dp)** means a modification in a functionality requires the deployment of at most one MS; **Class Integrity (In)** means a domain class resides entirely within a single MS, without separating getters and setters; **Balance (Bl)** means the class distribution among MS allows for teams of similar size; **Operational Cost (Ct)** means the number of MS is proportional to the system size, avoiding nano-services; **Flow Autonomy (Fl)** means a complete business flow is resolved within a single MS; and **Avg** column reports the arithmetic mean of the six dimensions, providing an overall quality indicator for each solution. Table 4 consolidates the consensus scores of the six solutions evaluated by the experts.

*5.4.1 JPetStore.* **Solution 1:** This solution distributes the classes nearly identically across four MS. No MS is clearly responsible for Account, Order, Cart, or Catalog: the classes of these domains are mixed across the four partitions. The most evident issue lies within the Order class: the method that reads the delivery address is in one MS, whereas the method that modifies it is in another. A simple

change, such as altering the format of the address field, requires two teams to coordinate the modification and deploy two MS concurrently. The Item class was split across four MS, scattering its access and modification methods throughout the system. Consequently, any evolution of a business entity requires deploying three to four MS simultaneously. The user login spans two MS.

**Solution 2:** This solution achieved the best average score among all candidate architectures and was therefore selected for detailed presentation. The resulting architecture is shown in Figure 3. MS0 concentrates nearly all classes of the Account domain, including authentication and user profile. MS1 groups the shopping cart functionality (Cart and Item), while MS2 encapsulates the order processing flow (Order and LineItem). MS3 isolates the product catalog (Catalog and Product). With this organization, modifying the discount logic in the shopping cart requires deploying only MS1, whereas changes to user registration are confined to MS0. The only exception is a fifth residual microservice, visible in Figure 3 in a distinct color, comprising 16 Order-related classes that were not incorporated into MS2. Consequently, user authentication is handled entirely within MS0, and the core business domains are largely encapsulated into independent microservices.

**Solution 3:** This solution partitions the system into eight microservices, five of which contain 13 classes or fewer. Several of these microservices exhibit low cohesion. For example, MS0 contains only six classes but combines functionality from three unrelated domains (Account, Cart, and Order). In addition, the Order domain is fragmented across all eight microservices. The CartItem class is also heavily decomposed, with its methods distributed among four different microservices. As a result, a simple operation such as incrementing the quantity of a product in the shopping cart requires interactions with four separate services, whereas the monolithic system performed the same operation through a single local call. Overall, maintaining eight independent deployment pipelines for a system comprising only 226 classes introduces considerable operational overhead with limited architectural benefit.

*5.4.2 Spring PetClinic.* **Solution 1**: This solution exhibits the same domain-fragmentation pattern observed in JPetStore, although on a smaller scale. The Owner, Pet, and Visit domains are distributed across four microservices, reducing domain cohesion. Even basic functionality is fragmented: the getter and setter methods for a person's name reside in different microservices, and several methods of the Owner class are also scattered across microservices. Thus, retrieving complete information about an owner requires interacting with three distinct microservices. Similarly, the workflow for registering a veterinary visit is split across three microservices, increasing service communication for a routine business operation.

**Solution 2:** This is the highest-scoring decomposition of the study. MS0 contains the entire flow of veterinary visits, from the form to the processing phase. MS1 provides comprehensive pet management, with all related classes in one place. MS2 groups the veterinarians, and MS4 groups the owners. MS3 contains only 2 residual classes. With this structure, each system functionality is resolved within a single MS, without relying on other services.

**Solution 3:** This solution presents the lowest score overall. Twelve MS for 68 classes: an average of fewer than six classes per MS, with six of these MS containing three classes or fewer. Some group unrelated methods: MS7, for instance, joins the owner's telephone number with the description of a veterinary visit, two pieces of information that share no commonality. The base class of the model, which provides methods used by all entities in the system, was partitioned across three different MS.

**Answer for RQ4:** From an expert's perspective, architectural quality varied substantially across solutions. Solution 2 achieved the highest scores in both systems, exhibiting clear domain boundaries that enable independent deployment, while Solution 3 suffered from excessive fragmentation, inflating operational cost. Domain alignment, deploy independence, and flow autonomy most distinguished adequate from inadequate decompositions. Notably, Solution 1 achieved balanced partitions yet mixed unrelated domains, confirming that structural balance alone is an unreliable indicator of developer-perceived quality.

## 5.5 Lessons Learned and Practical Implications

The experiments revealed three main findings. First, a persistent friction exists between the DM's semantic decisions and the optimization objectives of the NSGA-III algorithm, which at times reintroduced dependencies previously removed by the participants, highlighting **the need for mechanisms that preserve human-guided refinements**.

Second, **the effectiveness of the interactive operators depends on system scale:** although the Move operator proved valuable for fine-grained adjustments, method-level modifications became cognitively demanding in large systems, indicating the need for batch operations and higher-level interaction mechanisms. In this context, the Freeze operator emerged as an essential feature for protecting validated architectural decisions from stochastic changes.

Finally, the comparison between quantitative metrics and qualitative expert evaluation exposed a clear mismatch: **solutions with the best structural metrics were not necessarily those preferred by experts**, suggesting that architectural quality should be assessed through both automated metrics and expert judgment.

Practitioners should note two key lessons: (1) purely automated decomposition yields structurally balanced but domain-fragmented architectures, so interactive refinement may help achieve deployable microservices; (2) architects can focus on Move and Split operations for boundary correction, as they offer fine-grained control over domain encapsulation.

# 6 Threats to Validity

This section addresses the potential limitations of our study.

**Internal validity** threats related to human factors were mitigated by design adjustments. Cognitive workload and fatigue, stemming from method-level choices and capable of compromising decision consistency over long cycles, were mitigated by allowing participants to define their own interaction intervals and milestones based on focus. To address the learning curve and prevent suboptimal actions from tool unfamiliarity, a supervised demonstration and standardized training scenario were conducted before execution.

**External validity** threats concerning the generalizability of the experimental findings to broader industrial contexts were addressed through targeted sampling and multi-method data collection. The

primary threat stems from the limited scale and complexity of the open-source benchmarks (JPetStore and Spring PetClinic) relative to production legacy environments. This was mitigated by selecting software engineering specialists with extensive industry seniority who can extrapolate tool behavior to dense, large-scale codebases. Additionally, to counteract the constraints imposed by the small sample size of three participants, a multi-method evaluation model was implemented—integrating quantitative Likert-scale data, qualitative open-ended responses, interaction logs, and structural metrics—to construct a comprehensive, profile-by-profile analysis.

**Construct validity** evaluates whether the chosen evaluation metrics accurately reflect the real-world properties they are intended to measure. Relying on automated structural metrics—such as cohesion, coupling, or transaction purity—in isolation can be misleading, as a solution might achieve a high architectural balance score merely by uniformly distributing unrelated classes while concealing severe domain fragmentation. To mitigate this threat, a two-dimensional evaluation was carried out, combining quantitative adherence and modularity metrics (including MoJoFM, TurboMQ, and $c2c_{cvg}$) from a baseline with a six-dimensional qualitative assessment covering core software design principles such as Bounded Contexts and deploy independence.

**Conclusion validity** is threatened by the small sample size (n=3): no inferential statistical tests were applied, and reported differences should not be read as statistically significant. This limitation affects RQs differently. For RQ1, the topological similarity gains (up to 23.7% MoJoFM) were driven primarily by one participant (P3); the other two scored below the automated baseline (Table 2), indicating a strategy-dependent rather than uniform effect. For RQ3, professional background and industry experience are confounded in our sample, preventing clean attribution of observed patterns to either factor. We therefore treat all findings as preliminary and hypothesis-generating.

## 7 Related Work

In several related work, such as fully automated approaches, fine-grained structural optimization and API synthesis are achieved without DM intervention [17], dynamic execution traces are clustered and aggregated via modified NSGA-II metaheuristics [24], and security constraints are balanced with modularity objectives to yield stable architectures [30]. Other automated approaches alter the data scope or level of abstraction by partitioning high-level business models using MOEA/D algorithms [26] or by decomposing monolithic databases using relational access logs and spectral clustering [52]. To handle subjective criteria and reduce search spaces, recent frameworks incorporate the DM either *a posteriori*—using IBEA and graphical interfaces for final exploration [41] or reference lines and NSGA-III to filter solutions for manual customization [27, 28]—or interactively during the evolutionary process. Real-time interactive loop optimization is achieved by modeling system decomposition as Mixed Integer Linear Programming problems that allow iterative model and constraint refinement but without a mechanism to propagate human-induced changes across the population [38, 44], or by pairing an NSGA-II engine with constraint-satisfaction techniques to enforce rigid must-link and can-not-link class dependencies under real-time metric feedback [53]. In particular, this last approach targets the refactoring of existing microservice architectures rather than monolith decomposition, and equally limits interaction to rigid class-level constraints.

As presented, our work extends toMicroservices [12] into a fine-grained, method-level interactive SBSE approach that operates under a human-in-the-loop paradigm during the evolutionary cycle. While existing interactive methods either restrict human input to *a posteriori* customization, rely on rigid structural constraints during the evolutionary process, or address fundamentally different problem domains, our approach empowers the DM to make dynamic, real-time adjustments to intermediate solutions of a legacy monolith decomposition. We introduce four specialized architectural refinement operators that operate directly on a method-dependency graph, allowing architects to incorporate intuitive domain insights that rigid fitness functions fail to capture. Furthermore, to mitigate human fatigue and prevent the loss of DMs' decisions to algorithmic drift, we incorporate a change propagation mechanism based on intra-cluster grouping, ensuring that structural adjustments are consistently propagated and sustained across mathematically similar candidate solutions.

## 8 Conclusion and Future Work

This paper introduced *i-toMicroservices*, an interactive approach that enables software architects to actively guide the decomposition of monolithic systems into microservices during the optimization process. The proposed approach contributes a human-in-the-loop optimization workflow supported by four specialized refinement operators (Freeze, Split, Merge, and Move) together with an intra-cluster change propagation mechanism that balances human intervention with evolutionary search.

This exploratory study provides preliminary evidence of the approach's potential: human intervention helped correct partition imbalances partition and guides the search toward realistic boundaries, yielding adherence gains of up to 23.7% on reference architecture topology proximity relative to state-of-the-art approaches. The qualitative evaluation across six design dimensions further suggests that interactive operators can support the encapsulation of cohesive business sub-domains that automated criteria did not isolate in this study, while highlighting the freeze operator as a potentially valuable mechanism against algorithmic drift.

Our future work intends to: (i) provide real-time feedback on architectural metrics during interaction; (ii) strengthen the persistence of human-induced refinements across generations to mitigate algorithmic drift; (iii) improve the graphical interface based on participant feedback; and (iv) assess the approach on production-scale monolithic systems to further assess its external validity.

## Artifact Availability

The artifacts produced in this study, including the generated architectural decompositions, the post-study questionnaire, and interface screenshots, are publicly available at https://figshare.com/s/df621ed9530441707ac1.

## Acknowledgements

The work is supported by CAPES (Code 001), CNPq (404027/2023-7, 306774/2025-9), Fundação Araucária (Grant No. 792/2024).

## References


[1] Shivali Agarwal, Raunak Sinha, Giriprasad Sridhara, Pratap Das, Utkarsh Desai, Srikanth Tamilselvam, Amith Singhee, and Hiroaki Nakamuro. 2021. Monolith to microservice candidates using business functionality inference. In *2021 IEEE International Conference on Web Services (ICWS)*. IEEE, 758–763.

[2] Omar Al-Debagy and Peter Martinek. 2018. A comparative review of microservices and monolithic architectures. In *2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI)*. IEEE, 000149–000154.

[3] Apostolos Ampatzoglou, Angeliki-Agathi Tsintzira, Elvira-Maria Arvanitou, Alexander Chatzigeorgiou, Ioannis Stamelos, Alexandru Moga, Robert Heb, Oliviu Matei, Nikolaos Tsiridis, and Dionisis Kehagias. 2019. Applying the Single Responsibility Principle in Industry: Modularity Benefits and Trade-offs. In *Proceedings of the 23rd International Conference on Evaluation and Assessment in Software Engineering* (Copenhagen, Denmark) *(EASE '19)*. Association for Computing Machinery, New York, NY, USA, 347–352. doi:10.1145/3319008.3320125

[4] Allysson Allex Araújo, Matheus Paixao, Italo Yeltsin, Altino Dantas, and Jerffeson Souza. 2017. An architecture based on interactive optimization and machine learning applied to the next release problem. *Automated Software Engineering* 24, 3 (2017), 623–671.

[5] Wesley KG Assunção, Thelma Elita Colanzi, Luiz Carvalho, Alessandro Garcia, Juliana Alves Pereira, Maria Julia de Lima, and Carlos Lucena. 2022. Analysis of a many-objective optimization approach for identifying microservices from legacy systems. *Empirical Software Engineering* 27, 2 (2022), 51.

[6] Wesley K. G. Assunção, Thelma Elita Colanzi, Luiz Carvalho, Juliana Alves Pereira, Alessandro Garcia, Maria Julia de Lima, and Carlos Lucena. 2021. A Multi-Criteria Strategy for Redesigning Legacy Features as Microservices: An Industrial Case Study. In *IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 377–387. doi:10.1109/SANER50967.2021.00042

[7] Carlos Vinicius Bindewald, Willian M. Freire, Aline M. M. Miotto Amaral, and Thelma Elita Colanzi. 2020. Supporting user preferences in search-based product line architecture design using Machine Learning. In *Proceedings of the 14th Brazilian Symposium on Software Components, Architectures, and Reuse* (Natal, Brazil) *(SBCARS '20)*. Association for Computing Machinery, New York, NY, USA, 11–20. doi:10.1145/3425269.3425278

[8] Grzegorz Blinowski, Anna Ojdowska, and Adam Przybyłek. 2022. Monolithic vs. microservice architecture: A performance and scalability evaluation. *IEEE Access* 10 (2022), 20357–20374.

[9] Jürgen Branke. 2008. *Multiobjective optimization: Interactive and evolutionary approaches*. Vol. 5252. Springer Science & Business Media.

[10] Luiz Carvalho, Thelma Elita Colanzi, Wesley K. G. Assunção, Alessandro Garcia, Juliana Alves Pereira, Marcos Kalinowski, Rafael Maiani de Mello, Maria Julia de Lima, and Carlos Lucena. 2024. On the Usefulness of Automatically Generated Microservice Architectures. *IEEE Transactions on Software Engineering* 50, 3 (2024), 651–667. doi:10.1109/TSE.2024.3361209

[11] Luiz Carvalho, Alessandro Garcia, Thelma Elita Colanzi, Wesley K. G. Assunção, Maria Julia Lima, Baldoino Fonseca, Mârcio Ribeiro, and Carlos Lucena. 2020. Search-based many-criteria identification of microservices from legacy systems. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion* (Cancún, Mexico) *(GECCO '20)*. Association for Computing Machinery, New York, NY, USA, 305–306. doi:10.1145/3377929.3390030

[12] Luiz Carvalho, Alessandro Garcia, Thelma Elita Colanzi, Wesley K. G. Assunção, Juliana Alves Pereira, Baldoino Fonseca, Márcio Ribeiro, Maria Julia de Lima, and Carlos Lucena. 2020. On the Performance and Adoption of Search-Based Microservice Identification with toMicroservices. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 569–580. doi:10.1109/ICSME46990.2020.00060

[13] João Castanho, João Gomes, Thelma E. Colanzi, Aline M. M. M. Amaral, Wesley K. G. Assunção, and Alessandro Garcia. 2026. i-toMicroservices: Empowering Architects in the Interactive Modernization of Monolithic Systems. In *Proceedings of the 2026 Brazilian Symposium on Software Engineering - Tools Track (SBES-Tools 2026)*. Brazil.

[14] Adambarage Anuruddha Chathuranga De Alwis, Alistair Barros, Colin Fidge, and Artem Polyvyanyy. 2021. Microservice remodularisation of monolithic enterprise systems for embedding in industrial IoT networks. In *International Conference on Advanced Information Systems Engineering*. Springer, 432–448.

[15] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. 2017. *Microservices: Yesterday, Today, and Tomorrow*. Springer International Publishing, Cham, 195–216. doi:10.1007/978-3-319-67425-4_12

[16] Thiago Nascimento Ferreira, Silvia Regina Vergilio, and Jerffeson Teixeira de Souza. 2017. Incorporating user preferences in search-based software engineering: A systematic mapping study. *Information and Software Technology* 90 (2017), 55–69.

[17] Gianluca Filippone, Marco Autili, Fabrizio Rossi, and Massimo Tivoli. 2021. Migration of monoliths through the synthesis of microservices using combinatorial optimization. In *2021 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 144–147.

[18] Willian Marques Freire, Cláudia Tupan Rosa, Aline Maria Malachini Miotto Amaral, and Thelma Elita Colanzi. 2024. Interactive search-based Product Line Architecture design. *Automated Software Engineering* 31, 2 (2024), 59.

[19] 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)*. 481–490. doi:10.1109/ICSME.2019.00081

[20] Joshua Garcia, Igor Ivkovic, and Nenad Medvidovic. 2013. A comparative analysis of software architecture recovery techniques. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 486–496.

[21] Mark Harman, S. Afshin Mansouri, and Yuanyuan Zhang. 2012. Search-based software engineering: Trends, techniques and applications. *ACM Comput. Surv.* 45, 1, Article 11 (Dec. 2012), 61 pages. doi:10.1145/2379776.2379787

[22] Mark Harman, S Afshin Mansouri, and Yuanyuan Zhang. 2012. Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys (CSUR)* 45, 1 (2012), 1–61.

[23] i toMicroservices. 2026. Supplementary Material. https://figshare.com/s/df621ed9530441707ac1,.

[24] Wuxia Jin, Ting Liu, Yuanfang Cai, Rick Kazman, Ran Mo, and Qinghua Zheng. 2021. Service Candidate Identification from Monolithic Systems Based on Execution Traces. *IEEE Transactions on Software Engineering* 47, 5 (2021), 987–1007. doi:10.1109/TSE.2019.2910531

[25] Arjun Kamisetty, Deekshith Narsina, Marcus Rodriguez, Srinikhita Kothapalli, and Jaya Chandra Srikanth Gummadi. 2023. Microservices vs. Monoliths: Comparative Analysis for Scalable Software Architecture Design. *Engineering International* 11, 2 (2023), 99–112.

[26] Sedigheh Khoshnevis. 2023. A search-based identification of variable microservices for enterprise SaaS. *Frontiers of Computer Science* 17, 3 (2023), 173208.

[27] Takahiro Kinoshita and Hideyuki Kanuka. 2022. Automated microservice decomposition method as multi-objective optimization. In *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 112–115.

[28] Takahiro Kinoshita and Hideyuki Kanuka. 2024. Enhancing automated microservice decomposition via multi-objective optimization. *IEEE Access* 12 (2024), 55697–55710.

[29] James Lewis and Martin Fowler. 2014. Microservices: a definition of this new architectural term. *MartinFowler. com* 25, 14-26 (2014), 12.

[30] Xiaodong Liu, Zhikun Chen, Yu Qian, Chenxing Zhong, Huang Huang, Shanshan Li, and Dong Shao. 2024. Towards a security-optimized approach for the microservice-oriented decomposition. *Journal of Software: Evolution and Process* 36, 10 (2024), e2670.

[31] Thibaud Lutellier, Devin Chollak, Joshua Garcia, Lin Tan, Derek Rayside, Nenad Medvidovic, and Robert Kroeger. 2015. Comparing software architecture recovery techniques using accurate dependencies. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 2. IEEE, 69–78.

[32] Thibaud Lutellier, Devin Chollak, Joshua Garcia, Lin Tan, Derek Rayside, Nenad Medvidović, and Robert Kroeger. 2017. Measuring the impact of code dependencies on software architecture recovery techniques. *IEEE Transactions on Software Engineering* 44, 2 (2017), 159–181.

[33] Willian Marques Freire, Carlos Vinícius Bindewald, Aline M. M. Miotto Amaral, and Thelma Elita Colanzi. 2019. Supporting Decision Makers in Search-Based Product Line Architecture Design using Clustering. In *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 1. 139–148. doi:10.1109/COMPSAC.2019.00028

[34] Robert Martin and Robert C Martin. 2013. *Agile software development, principles, patterns, and practices*. BoD–Books on Demand.

[35] Brian S Mitchell and Spiros Mancoridis. 2006. On the automatic modularization of software systems using the bunch tool. *IEEE Transactions on Software Engineering* 32, 3 (2006), 193–208.

[36] MyBatis. 2024. JPetStore: A full web application built on top of MyBatis. https://github.com/mybatis/jpetstore-6.

[37] Kiran Kumar Pappula. 2022. Modular Monoliths in Practice: A Middle Ground for Growing Product Teams. *International Journal of Emerging Trends in Computer Science and Information Technology* 3, 4 (2022), 53–63.

[38] Giovanni Quattrocchi, Davide Cocco, Simone Staffa, Alessandro Margara, and Gianpaolo Cugola. 2024. Cromlech: Semi-automated monolith decomposition into microservices. *IEEE Transactions on Services Computing* 17, 2 (2024), 466–481.

[39] Aurora Ramirez, Jose Raul Romero, and Christopher L Simons. 2018. A systematic review of interaction in search-based software engineering. *IEEE Transactions on Software Engineering* 45, 8 (2018), 760–781.

[40] Aurora Ramírez, José Raúl Romero, and Christopher L. Simons. 2019. A Systematic Review of Interaction in Search-Based Software Engineering. *IEEE Transactions on Software Engineering* 45, 8 (2019), 760–781. doi:10.1109/TSE.2018.2803055

[41] Khaled Sellami, Ali Ouni, Mohamed Aymen Saied, Salah Bouktif, and Mohamed Wiem Mkaouer. 2022. Improving microservices extraction using evolutionary search. *Information and Software Technology* 151 (2022), 106996. doi:10.1016/j.infsof.2022.106996

[42] Claude Elwood Shannon. 1948. A mathematical theory of communication. *The Bell system technical journal* 27, 3 (1948), 379–423.

[43] Spring Projects. 2024. Spring PetClinic: A sample Spring-based application. https://github.com/spring-projects/spring-petclinic.

[44] Simone Staffa, Giovanni Quattrocchi, Alessandro Margara, and Gianpaolo Cugola. 2021. Pangaea: Semi-automated monolith decomposition into microservices. In *International Conference on Service-Oriented Computing*. Springer, 830–838.

[45] Davide Taibi, Valentina Lenarduzzi, Claus Pahl, and Andrea Janes. 2017. Microservices in agile software development: a workshop-based study into issues, advantages, and disadvantages. In *Proceedings of the XP2017 Scientific Workshops*. 1–5.

[46] Davide Taibi and Kari Systä. 2020. A Decomposition and Metric-Based Evaluation Framework for Microservices. In *Cloud Computing and Services Science*, Donald Ferguson, Víctor Méndez Muñoz, Claus Pahl, and Markus Helfert (Eds.). Springer International Publishing, Cham, 133–149.

[47] Victor Velepucha and Pamela Flores. 2023. A survey on microservices architecture: Principles, patterns and migration challenges. *IEEE access* 11 (2023), 88339–88358.

[48] Yash Vesikar, Kalyanmoy Deb, and Julian Blank. 2018. Reference point based NSGA-III for preferred solutions. In *2018 IEEE symposium series on computational intelligence (SSCI)*. IEEE, 1587–1594.

[49] Shuai Wang, Shaukat Ali, Tao Yue, Yan Li, and Marius Liaaen. 2016. A practical guide to select quality indicators for assessing pareto-based search algorithms in search-based software engineering. In *Proceedings of the 38th International Conference on Software Engineering* (Austin, Texas) *(ICSE '16)*. Association for Computing Machinery, New York, NY, USA, 631–642. doi:10.1145/2884781.2884880

[50] Yingying Wang, Sarah Bornais, and Julia Rubin. 2024. Microservice Decomposition Techniques: An Independent Tool Comparison. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering* (Sacramento, CA, USA) *(ASE '24)*. Association for Computing Machinery, New York, NY, USA, 1295–1307. doi:10.1145/3691620.3695504

[51] Zhihua Wen and Vassilios Tzerpos. 2004. An Effectiveness Measure for Software Clustering Algorithms.. In *IWPC*, Vol. 4. 194–203.

[52] Peng Zhang, Weigang Li, Ruihan Gao, Henan Zhang, and Junsheng Wu. 2024. Research on database splitting methodology for microservicization of monolithic applications. In *2024 4th International Conference on Communication Technology and Information Technology (ICCTIT)*. IEEE, 562–566.

[53] Chenxing Zhong, Shanshan Li, He Zhang, Huang Huang, Lanxin Yang, and Yuanfang Cai. 2024. Refactoring microservices to microservices in support of evolutionary design. *IEEE Transactions on Software Engineering* (2024).