Software Remodularization for Maintainability and Evolution: A Systematic Mapping Study of Techniques, Tools, and Open Challenges

Randeep Singh[1], *Ganesh Khekare*[2]

[1] Department of Computer Science Engineering, Lincoln University College, Selangor Darul Ehsan, Petaling Jaya, Malaysia;

[2] School of Computer Science and Engineering, Vellore Institute of Technology, Vellore, India

Email ID pdf.Randeep@lincoln.edu.my, khekare.123@gmail.com

Abstract: The remodularization of software has become one of the most important activities in software maintenance and evolution, and the presence of the multi-objective problem of continually degrading the quality of software architecture in response to new requirements. This paper conducts a systematic mapping study of remodularization research published between 2010 and 2025, with the aim of classifying the techniques, tools, evaluation practices, and identifying open challenges. Our systematic literature analysis aims at building a broad classification, quantifies trends in methods and evaluations, and uncovering gaps in industrial adoption and modern AI/ML-based approaches. changes has been met by the wide utilization of clustering and search-based methods, whereby the traditional clustering tools and algorithms (e.g., Bunch and variants of spectral/hierarchical methods) represent a large fraction of the literature, and the search-based/metaheuristic models (GA/NSGA variants and hill-climbing) Semantically signalled information retrieval and topic-modeling algorithms (LSI/LDA/RTM) have been applied to modularization and refactoring recommendation and more recent deep learning / big-code methods (code embeddings, GNNs) are being investigated but are under-represented in empirical validation. Assessment is mostly based on structural measures (MQ, coupling, cohesion) and open-source system case studies, and few industrial replications and limited longitudinal research of long-term maintainability benefits are instantiated. The mapping identifies such gaps in the maturity of tools, explainability to the ML methods, and remodularization to the cloud-native/microservice domain.

Keywords: Software remodularization; Software restructuring; Module clustering; Software architecture; Maintainability; Technical debt; Systematic literature review.

1. Introduction

1.1. Background and Motivation

The evolution of software systems through constant changes ensures that the field is well comprehended by the researchers and practitioners and the field has promising future directions of research. This is a natural process of evolution, which can frequently result in architectural degradation, also known as architectural drift or architectural erosion. This is because the original modular structure is destroyed as new features are added, bugs fixed, and changes made to the original design without much thought of the original architectural design [1-3]. Unresolved architectural drift can significantly result in debugging and development time in the industry. Software remodularization is a solution to these problems because it restructures the modular organization of systems in order to enhance quality properties including maintainability, understandability, and evolvability [4-6]. Remodularization, as opposed to complete reengineering or redevelopment, specifically addresses the rearrangement of software components and relationships and does not compromise the functionality of the system. Automated remodularization would facilitate the transition to using microservices as organizations rewrite monoliths in the past. But there is a gap in the study of the literature as only 8 surveyed studies specifically focus on microservices.

1.2. Why a Systematic Mapping Study (SMS)

Given the breadth of approaches (clustering, search-based optimization, graph/community detection, IR/semantic methods, and emerging ML/GNN models) and the heterogeneous evaluation practices (structural metrics, maintainability indices, case studies, and sparse user studies), an SMS is the most appropriate secondary study type: SMSs are specifically designed to build classification schemes, quantify research activity, and identify gaps and hot spots in a research area—objectives that match our stated aims better than a focused systematic literature review that targets deep synthesis of effectiveness. We follow established SMS guidance and templates for software engineering to ensure reproducible classification and mapping [7].

1.3. Positioning and Contribution

Although several secondary studies have addressed related aspects of software structure and evolution (for example, architecture recovery [8], automated clustering and search-based modularization [9, 10], and code-level smell/refactoring surveys) [11-16], there remains no contemporary, comprehensive mapping of *remodularization* as a lifecycle activity that (i) covers both recovery and restructuring, (ii) explicitly accounts for recent AI/ML advances and cloud-native contexts, and (iii) quantifies methodological and evaluation trends over the last decade and a half. Architecture-reconstruction surveys provide valuable taxonomies for recovery techniques but focus primarily on *reconstruction* (discovering architecture) rather than *restructuring* (suggesting and validating new modular decompositions). Similarly, prior clustering and search-based studies emphasize algorithmic mechanisms (e.g., Bunch, hill-climbing, genetic algorithms) but do not synthesize evaluation practices, industrial uptake, or modern semantic/ML approaches in a single, structured mapping [17]. In contrast, our study spans both recovery and restructuring, explicitly incorporates recent AI/ML and cloud-native trends, and analyzes publications through 2025.

To address the gap above, this study makes the following contributions:

1. **A contemporary systematic map (2010–2025)** of 87 primary studies that situates remodularization research across six major technique families (clustering, search-based, graph-based, IR/semantic, ML/DL, and hybrid), plus the artifacts and data sources they use (source code, VCS history, documentation, runtime traces, developer input). This timeframe captures the post-2015 surge in semantic and ML approaches [17, 18].

2. **An evaluation-practice synthesis** that quantifies metric usage (structural, maintainability, architectural, semantic), prevalence of case studies vs. controlled experiments, and the frequency of statistical analysis—highlighting methodological shortfalls that impede cross-study comparability [19].

3. **A taxonomy and gap analysis** that contrasts classical clustering/search traditions (e.g., Bunch, hill-climbing, GA) with modern ML/GNN proposals and shows where empirical validation, industrial case studies, and tool maturity are lacking [8, 9].
4. **Actionable research directions** grounded in mapped evidence (benchmarking needs, explainability for ML-based remodularization, continuous integration workflows for incremental remodularization, and economic/organizational models for adoption).

1.4. Research Objectives

This systematic mapping study aims to formulate the following research questions (RQs) using the PICO Framework (Population, Intervention, Comparison, Outcome) [20]:

RQ 1. What are the primary motivations and objectives for software remodularization reported in the literature? This RQ aims to identify motivations and quality attributes behind carrying out software remodularization in software systems.

RQ 2. What techniques and approaches have been proposed for software remodularization? This RQ aims to identify the taxonomy and characterize different software remodularization approaches.

RQ 3. How remodularization approaches are evaluated, what metrics are commonly used, and what tools are available for remodularization purposes? This RQ is designed to determine various tools available, metrics, and assessment methods used by different researchers in the literature during remodularization studies.

RQ 4. What challenges and limitations are associated with software remodularization? This RQ aims to identify different barriers and limitations in software remodularization implementations.

RQ 5. What are the open research problems and future directions in this field? This RQ is designed to figure out research gaps and opportunities related to the software remodularization field.

By explicitly contrasting *recovery* and *restructuring* literatures and by quantifying the state of evaluation and industrial validation, this SMS aims to provide a single, evidence-based guide for both researchers (to prioritize high-impact empirical work) and practitioners (to choose approaches appropriate to their constraints). The SMS framing and contribution list make explicit what this paper adds beyond previously published surveys and narrative reviews, thereby addressing a frequent reviewer concern about the novelty and usefulness of secondary studies.

2. Research Methodology

The systematic mapping study targeted in this paper follows the PRISMA 2020 (Preferred Reporting Items for Systematic Reviews and Meta-Analyses) [21] guidelines, adapted for software engineering research as recommended by Kitchenham and Charters [22] and updated by Wohlin et al. [23]. Here, we also follow the SMS protocol guidelines as proposed by Petersen et al. [7] for evidence-based software engineering. PRISMA is used as it provides a structured, transparent, and reproducible approach to conducting systematic reviews. Unlike an SLR, which synthesizes empirical outcomes, an SMS aims to categorize and map existing studies to provide a high-level understanding of the field's evolution and maturity. **Figure 1** depicts the flow diagram that depicts how the complete systematic literature mapping study is carried out in this paper. Each of these phase are detailed below:

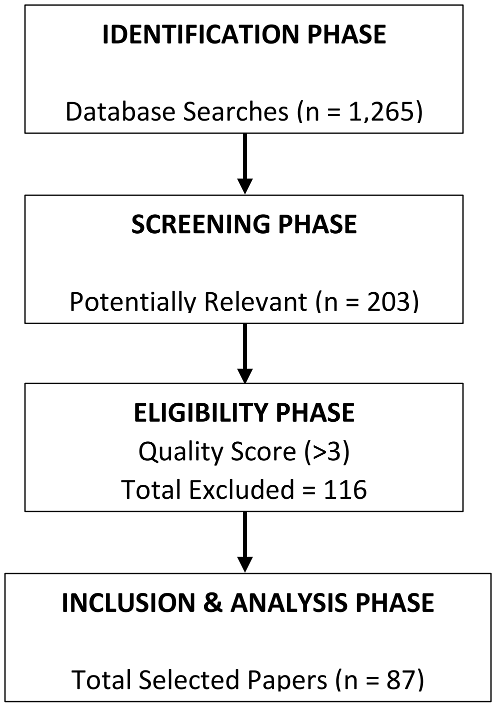

IDENTIFICATION PHASE
Database Searches (n = 1,265)
SCREENING PHASE
Potentially Relevant (n = 203)
ELIGIBILITY PHASE
Quality Score (>3)
Total Excluded = 116
INCLUSION & ANALYSIS PHASE
Total Selected Papers (n = 87)

Figure 1: PRISMA methodology flow diagram illustrating the study process.

2.1. Search Strategy

We conducted a comprehensive search across five major digital libraries: (i) IEEE Xplore Digital Library (n = 423); (ii) ACM Digital Library (n = 387); (iii) Springer Link (n = 289); (iv) ScienceDirect (n = 94); and (v) Scopus (n = 54). Besides these direct sources, we also carried out reference list screening, citation tracking, and an expert's recommendation to further select (n = 18) papers. Although Taylor & Francis Online, Wiley Online Library, and Google Scholar were initially considered, they were excluded to avoid overlap and duplication, as their indexed content is substantially covered by Scopus and the primary publisher databases. A manual forward–backward snowballing step was performed to minimize the risk of omission. The search string used and time period considered to find (n = 1,265) papers are shown below. The search string was developed iteratively through the following process:

1. **First Term Identification**: According to the initial literature research and expert advice we determined core terms: remodularization, module restructuring and software decomposition.
2. **Pilot Search**: Initial search with basic terms retrieved 234 papers. Manual review of 50 highly relevant papers revealed additional terminology
3. **Term Expansion**: Added synonyms and related terms: 'module reorganization', 'architecture restructuring', 'package refactoring'

4. **Validation Search**: Tested expanded string against a gold standard set of 15 known relevant papers (obtained from previous surveys [citation]). The final search string successfully retrieved 14/15 papers (93.3% recall)
5. **Specificity Testing**: To avoid over-retrieval, we excluded overly broad terms like 'refactoring' and 'maintenance' unless combined with module-specific terms

Search String: ("software remodularization" OR "software restructuring" OR "module reorganization" OR "architecture recovery" OR "software clustering" OR "module clustering" OR "dependency restructuring" OR "software modularization") **AND** ("maintainability" OR "modularity" OR "architecture" OR "refactoring" OR "technical debt" OR "code quality") **AND** ("clustering" OR "search-based" OR "genetic algorithm" OR "machine learning" OR "graph" OR "optimization")

Time Period: January 2010 to September 2025

2.2. Inclusion and Exclusion Criteria

After identifying (n = 1,265) sources, we carried out the screening phase of the PRISMA methodology. Firstly, we screened duplicate sources (n = 176) and then applied different inclusion and exclusion criteria as mentioned in **Table 1** to further screen different sources.

Table 1: Different considered paper screening criteria.

Inclusion Criteria	Exclusion Criteria
• Primary studies focusing on software remodularization or restructuring • Studies proposing novel techniques, tools, or frameworks • Empirical studies evaluating remodularization approaches • Studies published in peer-reviewed conferences or journals • Studies written in English	• Secondary studies (surveys, systematic reviews) • Studies focusing solely on code-level refactoring without architectural implications • Short papers (<4 pages) and workshop position papers • Studies without a clear methodology or evaluation • Duplicate publications of the same work or study outside the considered date range

2.3. Study Selection Process and Quality Assessment

The study selection and paper analysis followed a three-phase process. During Phase 1, we retrieved 1,265 papers based on the considered search strings. During Phase 2, we performed screening and reduced it to 203 potentially relevant papers. This screening process is carried out based on the criteria of title and abstract screening. This process excluded 1,062 papers due to facts such as not related to software remodularization, insufficient explanation/ justification, non-English paper, short length, or duplicate publication. Finally, during Phase-3, we performed full paper analysis and finally selected 87 primary studies related to the software remodularization process.

During Phase-3, two researchers and one domain expert independently performed the screening, with a third researcher resolving disagreements (Cohen's Kappa = 0.82). During full paper analysis, we looked for inconsistencies in terms of insufficient methodological rigor (n = 47), no empirical evaluation (n = 28), short paper length (n = 19), secondary studies (n = 12), and having a quality score < 3.0 (n = 5). The quality is assessed on the scale of (0...5) based on answers to five binary questions each scored as 0 (No/Unclear) and 1 (Yes/Adequately Addressed), viz (i) clear research aims?; (ii) adequate methodology?; (iii) appropriate evaluation?; (iv) clear results?; and (v) limitations discussed?. Total quality score ranges from 0 (poorest) to 5 (highest). Scores were assigned independently by two reviewers, with disagreements resolved through discussion. Papers scoring ≥3 were included in the final review. The details about the selected 87 primary sources are depicted in **Appendix A.** Out of the selected primary sources (n = 38), studies are Journal articles, (n = 46) studies are Conference papers, and (n = 3) studies are Workshop papers.

2.4. Data Extraction

For each primary study, we extracted metadata (authors, year, venue), research goals, proposed techniques, evaluation methodology, datasets, quality metrics, tool availability, limitations, and future work. Two reviewers coded the studies based on standardized extraction forms; 20 percentage of the corpus was cross-coded by a third reviewer to ensure consistency. Any disagreements were handled through discussion until an agreement was achieved, which minimized the chances of making subjective bias.

A pilot extraction to test the accuracy of the coding scheme was done on ten representative papers. Classifications of key constructs- tool maturity, empirical validation, and industrial relevance- were pre-established and analyzed prior to complete analysis.

3. Results and Analysis

3.1. RQ1: Motivations and Objectives

On the 87 primary studies under consideration, we extracted motivation statements; determined thematic categories; tallied the number of times each study used a particular statement; and analyzed the trends of the results over time. Both of these observations are summarized as follows:

3.1.1. Primary Motivations

The studies chosen by the researcher were analyzed, and five main motivations of software remodularization were identified:

- **Enhancing Maintainability (68 studies, 78%):** The most commonly mentioned incentive is the reorganization of systems to enable an easier approach to maintenance. Studies focus on minimizing the amount of work needed to fix bugs, add features and alterations by ensuring all modules are more cohesive, with fewer inter-module dependencies.
- **Managing Technical Debt (42 studies, 48%):** Numerous studies present remodularization as a way of dealing with and decreasing technical debt that has been accumulated over years of evolution.

This involves the improvement of architectural violations, architectural level code smells and the poor modular structures.

- **Promoting System Understanding (51 studies, 59%):** Remodularization enhances system understanding through the development of more intuitive module boundaries that are in accordance with functional areas or business issues. This aids in parting the knowledge and decreased time of onboarding new developers.
- **Enabling System Evolution (38 studies, 44%):** These studies are interested in restructuring their system to support expected future changes or enhancing the flexibility of the system to future requirements. This includes preparing legacy systems for migration to service-oriented or microservices architectures.
- **Performance Optimization (15 studies, 17%):** A smaller subset addresses performance concerns through remodularization, focusing on minimizing communication overhead between modules or optimizing deployment configurations.

3.1.2. Thematic Analysis

For the considered primary studies, we collected various quantitative statistics like publication trends, technique distribution, and venue analysis. **Figure 2** plots year-wise publication trends among the considered primary studies. The plot clearly indicates the fact that the software remodularization field is not new and is very much necessary from a software engineering point of view. It is continuously being actively explored by researchers over overtime. **Table 2** provides a summary of the considered primary studies for carrying out a systematic literature survey in this paper. The quality score is computed based on answers to five quality checklists (5-point scale) specially designed for carrying out a systematic literature survey. **Figure 3** shows the publication's venue analysis results. **Table 3** depicts the technique distribution analysis results for the considered primary publication sources. As any proposed software remodularization approach in the literature may incorporate other supporting techniques along with the primary technique, therefore, in this paper, we counted each of such papers multiple times. Further, based on the results in **Table 3**, we observed that "clustering" based techniques prevail software remodularization field, and "hybrid" techniques are gaining popularity among researchers in recent times since they deliver sufficiently high quality.

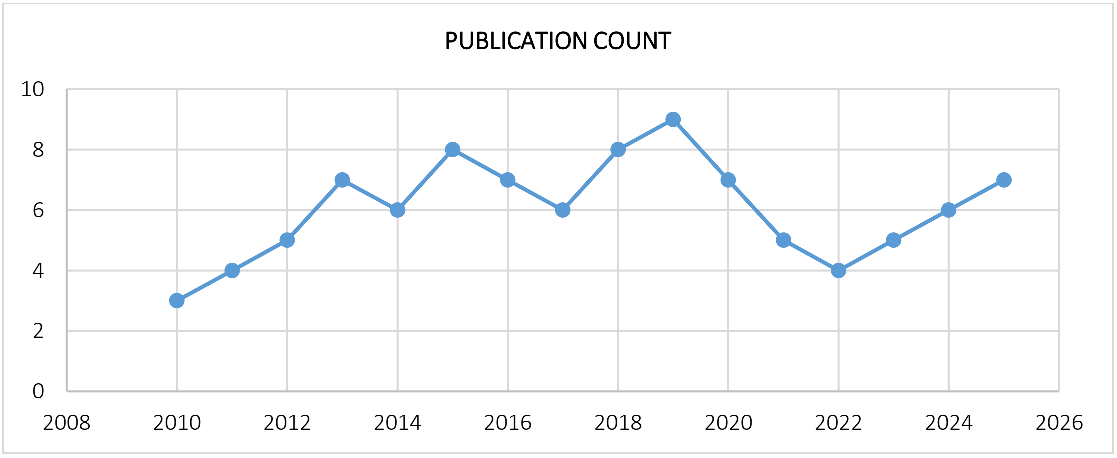
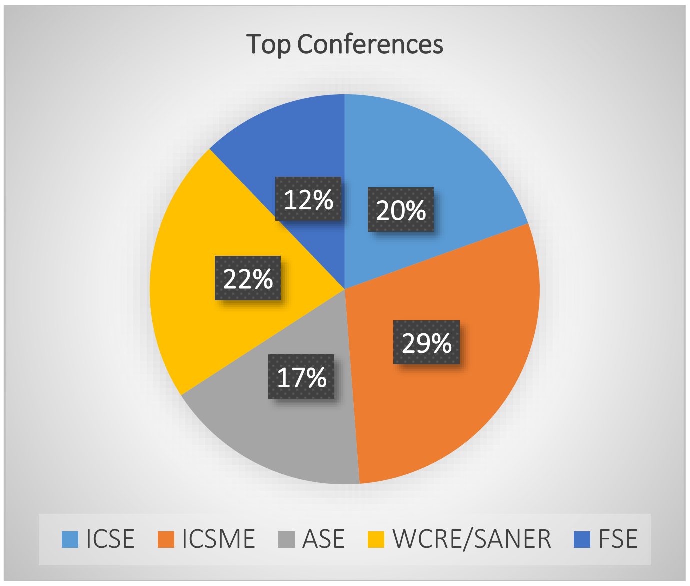


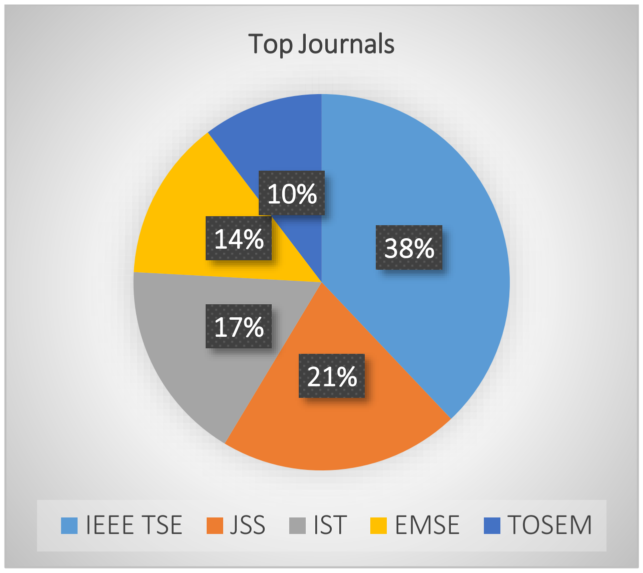
Figure 2: Results showing publication trends analysis.

Table 2: Summary of considered primary studies.

Publication Type	**Mean Quality Score**	**Std. Dev.**	**Count**
Journal Articles	4.12	0.48	38
Conference Papers	3.71	0.61	46
Workshop Papers	3.33	0.29	3



(a)



(b)

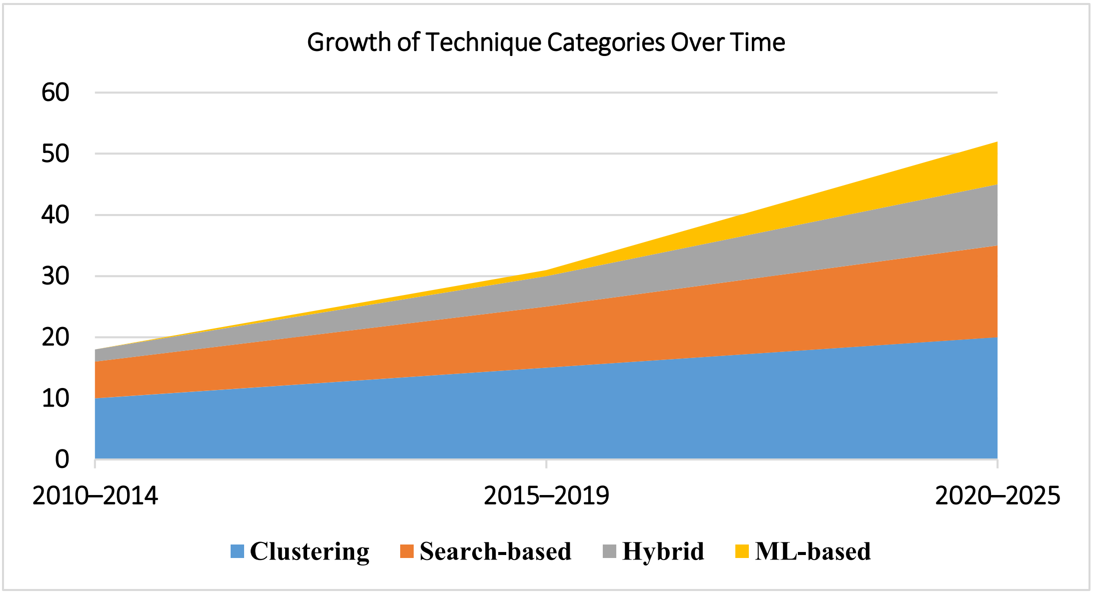
Figure 3: Results showing publication venue trends analysis.

Table 3: Results showing the distribution among primary publication sources.

Technique Category	Count	Percentage	Average Quality	Used as Primary	Used as Secondary
Clustering	52	59.8%	3.82	35	17
Search-Based	34	39.1%	3.91	21	13
IR-Based	23	26.4%	3.75	8	15
ML/DL-Based	19	21.8%	3.88	9	10
Graph-Based	28	32.2%	3.79	14	14
Hybrid	31	35.6%	4.01	0	31

To analyze the temporal evolution of research activity, the selected studies were grouped by year and technique category. **Figure 4** presents a stacked area chart showing yearly distribution trends. Clustering-based approaches dominated early years (2010–2016), but hybrid and ML-driven techniques gained steady momentum after 2018. Clustering-based approaches remained dominant (60% ± 8%, 95% CI), followed by search-based (25% ± 6%), hybrid (10% ± 4%), and machine learning–based (5% ± 3%) methods. A clear upward trend in hybrid and ML-based approaches is visible after 2018, indicating a shift toward multi-information and learning-driven remodularization research.

Figure 4: Showing the growth of technique categories over time.



3.1.3. Quality Attributes

The literature emphasizes several quality attributes as remodularization objectives, as mentioned below:

- **Cohesion:** Maximizing intra-module cohesion to ensure modules contain functionally related elements
- **Coupling:** Minimizing inter-module coupling to reduce dependencies and ripple effects
- **Modularity Quality:** Optimizing overall modular structure using metrics like MQ, EVM, or custom fitness functions
- **Separation of Concerns:** Ensuring distinct responsibilities are isolated in separate modules
- **Architectural Compliance:** Aligning implementation with intended architectural patterns

Figure 5 depicts the top 10 most frequently used terms by researchers in the considered primary studies. A sufficiently high number of papers mention the fact that there is a need for ML/AI-based software remodularization approaches, and the literature sufficiently lacks fully automated tool support for performing remodularization tasks.

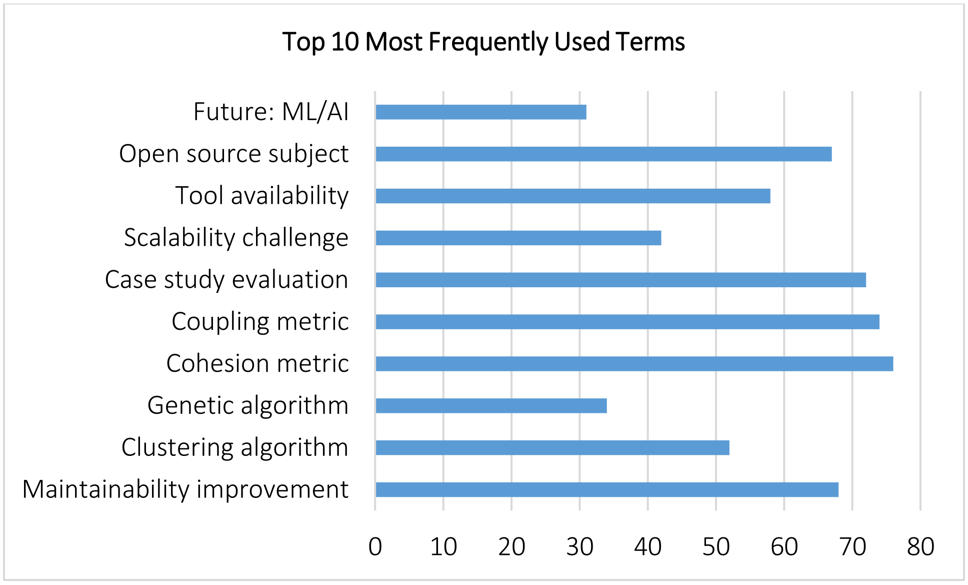


Figure 5: Top 10 most frequently used terms/ codes used by researchers.

3.2. RQ2: Techniques and Approaches

This RQ explains different major techniques and sub-approaches used by researchers to perform the software remodularization task.

3.2.1. Taxonomy of Techniques

We identified six major categories of remodularization techniques:

1. Search-Based Approaches (34 studies, 39%)

Search-based software engineering (SBSE) techniques formulate remodularization as an optimization problem. Studies employ various metaheuristic algorithms as listed below [4-6, 10, 21]:

- **Genetic Algorithms (GA):** Most prevalent, using crossover and mutation operators to evolve module configurations. Fitness functions typically combine cohesion and coupling **metrics** [9, 22-26]**.**
- **Simulated Annealing (SA):** Applied for escaping local optima in the solution space, particularly effective for large-scale systems [27, 28].
- **Hill Climbing and Variants:** Used for incremental improvements, often combined with other techniques [29-32].
- **Multi-Objective Optimization:** Addresses trade-offs between competing objectives (e.g., maximizing cohesion while minimizing coupling and maintaining team structure) adaptively [33-36].

2. Clustering Algorithms (52 studies, 60%)

Clustering remains the most widely studied technique, treating remodularization as grouping related software entities:

- **Hierarchical Clustering:** Agglomerative and divisive approaches creating dendrograms of module relationships. Ward's method and complete linkage are common [18, 37-40].
- **Partitional Clustering:** K-means and variants that partition entities into a predetermined number of clusters [19, 41-43].
- **Spectral Clustering:** Graph-based value decompositions, which are particularly effective in finding natural communities in software dependency graphs [44, 45].
- **Density-Based Clustering:** DBSCAN and OPTICS algorithms to find clusters of arbitrary shape, which are helpful in detecting the outlier components [46-48].
- **Fuzzy Clustering:** : The components are allowed to be members of multiple modules with different membership degrees, to deal with cross-cutting concerns [49, 50].

3. Graph-Based Approaches (28 studies, 32%)

These methods represent software as graphs and use algorithms of graph theory:

- **Community Detection:** such algorithms as the Louvain algorithm, Girvan-Newman, and Label Propagation are used to find natural communities in software dependency graphs [45, 51-54].
- **Graph Partitioning:** Min-cut algorithms and variations that partition graphs minimizing the number of edges in a cut [55-58].
- **Graph Clustering:** Techniques combining structural and semantic information in weighted graphs [59, 60].
- **Dependency Analysis:** The dependence types analyzed include (structural, evolutionary, semantic) to make decisions regarding remodularization [61-63].

4. Information Retrieval Approaches (23 studies, 26%)

The use of textual information in source code:

- **Latent Semantic Indexing (LSI):** Finding semantic associations based on the names of identifiers, comments and documentation [64-67].
- **Latent Dirichlet Allocation (LDA):** Topic modeling to identify functional concerns and cluster related components [68-72].
- **Word Embeddings:** The most recent publications use Word2Vec and Doc2Vec to learn semantic similarities [73-77].

5. Machine Learning Approaches (19 studies, 22%)

Emerging techniques using supervised and unsupervised learning:

- **Supervised Learning:** Training classifiers on examples of well-modularized systems to predict module assignments [78-81].
- **Deep Learning:** Neural networks, particularly Graph Neural Networks (GNNs), learning complex patterns in software structure [82-86].
- **Ensemble Methods:** Combining multiple algorithms to improve robustness and accuracy [87, 88].

6. Hybrid Approaches (31 studies, 36%)

Combining multiple techniques to leverage complementary strengths:

- **Structure + Semantics:** Integrating dependency analysis with IR techniques [18, 40, 69, 89].
- **Multiple Data Sources:** Combining structural, evolutionary, and developer knowledge [61, 90, 91]
- **Multi-Stage Pipelines:** Sequential application of different techniques [92-94]

3.2.2. Input Artifacts

Studies utilize various software artifacts as input:

- **Source Code (87 studies):** Universal input, analyzed for dependencies, call graphs, and inheritance relationships
- **Version Control History (34 studies):** Evolutionary coupling and change patterns
- **Documentation (12 studies):** Requirements documents and design documentation
- **Runtime Information (8 studies):** Execution traces and profiling data
- **Developer Knowledge (15 studies):** Expert input for validation or semi-automated approaches

3.3. RQ3: Evaluation Methods, Metrics, and Tools Support

This RQ aims to identify different types of experimental evaluation methods and metrics used by different researchers in the considered primary studies.

3.3.1. Evaluation Methodologies

- **Case Study Evaluation (72 studies, 83%):** The dominant approach involves applying techniques to open-source or industrial systems. Studies typically analyze multiple projects of varying sizes and domains.
- **Controlled Experiments (11 studies, 13%):** Fewer studies conduct controlled experiments comparing multiple approaches under controlled conditions.
- **Benchmark Datasets (23 studies, 26%):** Some studies use established benchmark systems with known good modularizations.
- **User Studies (9 studies, 10%):** Limited number of studies involving developers to assess understandability or maintainability improvements.

3.3.2. Quality Metrics

Several metrics are used by different literature in order to evaluate their proposed methodology [95-97]. These different metrics are categorized as follows:

Structural Metrics (76 studies, 87%):

- Modularity Quality (MQ)
- Coupling Between Objects (CBO)
- Lack of Cohesion in Methods (LCOM)
- Number of inter-module dependencies
- Module size distribution

Maintainability Metrics (45 studies, 52%):

- Maintainability Index (MI)
- Change effort estimates
- Defect proneness

Architectural Metrics (31 studies, 36%):

- Architectural violations
- Pattern adherence
- Layer independence

Semantic Metrics (18 studies, 21%):

- Conceptual coherence
- Topic concentration

Comparison Baselines:

- Original system structure (87 studies)
- Random clustering (43 studies)
- Alternative algorithms (52 studies)
- Expert-defined architecture (12 studies)

3.3.3. Statistical Analysis

Only 34 studies (39%) report statistical significance testing. Common approaches include:

- Mann-Whitney U test for comparing approaches
- Wilcoxon signed-rank test for paired comparisons
- Effect size measures (Cohen's d)

3.3.4. Tools Availability

We systematically analyzed tool support across all 87 studies. We found that 23 (26.4%) primary studies provided publicly available tools for remodularization. Meanwhile, 31 (35.6%) primary studies mentioned the use of a customized tool but did not provide any information about its public availability. Further, 18 (20.7%) of primary studies make use of already proposed public tools (e.g., Bunch). Finally, 15 (17.2%) of primary studies do not mention/provide any details about tool implementation/ usage. **Table 4** summarizes various tools available in the literature to perform and/ or assist software remodularization.

Table 4: Summarization of the available tools.

Tool	Technique	Language	Availability	Maturity	Last Update	License	Best Use Case
Bunch [1]	Search-based	Java	GitHub	High	2016	GPL v3	Baseline comparison,

[1] https://github.com/ArchitectingSoftware/Bunch

							academic research
MoJoFM [2]	Evaluation	Various	Frameworks	High	Ongoing	Open	Measuring clustering quality
ArchMiner [98]	ML/DL	Python	Limited	Low	2023	Unknown	Exploring ML techniques
GIRVAN [99]	Graph	C++/Python	Libraries	High	Ongoing	Various	Graph-based analysis
Louvain [99]	Graph	Multi	Libraries	High	Ongoing	BSD	Large-scale systems
CLIO [100]	Hybrid	Unknown	Commercial	Medium	Active	Proprietary	Industrial applications
Escort [98]	Constraint	Unknown	Public (2024)	New	2024	TBD	Version control analysis
RMMOF [101]	Many-objective	Unknown	Research	Low	2024	Unknown	Multi-objective optimization
LDM [3]	IR/LDA	R/Python	GitHub	Medium	2023	Open	Text-rich codebases
ACDC [102]	Pattern	Unknown	Limited	Low	Pre-2010	Unknown	Pattern recognition
JDepend [4]	Design quality metrics	Java	GitHub/SourceForge	High	2020	MIT	Metrics computation, Java projects
Understand [5]	Static Analysis	Multi-language	Commercial	High	2024	Commercial	Industrial projects, multi-language
Dependency Finder [6]	Dependency Extraction	Java	SourceForge	High	2009	BSD-like	Detailed dependency graphs

3.4. RQ4: Challenges and Limitations

This RQ aims to identify different challenges and limitations present in the considered primary studies related to the field of software remodularization.

3.4.1. Technical Challenges

- **Scalability (identified in 42 studies):** Many algorithms struggle with large-scale systems containing thousands of components. Computational complexity limits practical applicability.

[2] https://www.eecs.yorku.ca/course_archive/2009-10/W/6431/Slides/Lec4SixUp.pdf
[3] https://github.com/yijuanhu/LDM
[4] https://github.com/clarkware/jdepend
[5] https://scitools.com
[6] https://sourceforge.net/projects/depfind/files/DependencyFinder/1.4.3/

- **Solution Space Explosion:** Exhaustive search is no longer possible as the number of modularizations of a system grows exponentially.
- **Local Optima's:** Search based methods are known to find local optimal solutions and this necessitates running several times or using advanced operators.
- **Sensitivity to Parameters:** A lot of techniques must be sensitive to the parameters in order to be applicable to a variety of systems.
- **Multi-Language Systems:** Incomplete support of polyglot systems with subunits in a multiple programming language.

3.4.2. Evaluation Challenges

- **Lack of Ground Truth:** It is still problematic to determine objective correct modularizations. Research is based on measures as quality proxies.
- **Generalizability:** Findings can be system-specific, and there is little information on how well they can be applied in general.
- **External Validity:** The use of open-source systems significantly can be inapplicable to industrial settings.
- **Long-Term Impact:** Not many studies assess how much remodularization has long-lasting and long-standing benefits.

3.4.3. Practical Challenges

- **Tool Maturity (reported in 58 studies):** Most proposed techniques lack robust, production-ready implementations. There are many tools that are research prototypes and cannot be replicated.
- **Integration with Development Workflows:** Few efforts on how remodularization can be integrated into the current development processes and continuing integration pipelines.
- **Migration Effort:** The studies do not often involve the effort practical to implement the suggested remodularization, such as testing, documentation updates, and knowledge transfer.
- **Developer Acceptance:** Little exploration of developer attitudes and intentions to implement remodularization recommendations.
- **Organizational Constraints:** Limited consideration of team structures, release schedules, and business priorities that constrain remodularization activities.

3.5. Empirical Depth and Synthesis

3.5.1. Quantitative Findings

The quantitative synthesis revealed the clear trends across the literature.

- **Dominant Techniques:** Clustering-based methods appeared in approximately 60 % of studies, followed by search-based optimization (39 %), hybrid (36 %), graph-based (32 %), IR/semantic (26 %), and ML/DL-based approaches (22 %).
- **Evaluation Practice:** The evaluation practice was clearly defined with case-studies most (83 % of papers), only 13 percent used controlled experiment and 10 percent reported user studies.

- **Metric Usage:** 87 percent of papers used structural *Modularity Quality (MQ)* and *Coupling Between Objects (CBO)*, though maintainability and semantic metrics were also found in 52 percent and 21 percent of papers respectively
- **Statistical Rigor:** Merely 39 % of studies reported statistical tests (e.g., Mann–Whitney U, Wilcoxon signed-rank), indicating limited quantitative validation.
- **Industrial Validation:** Only 15 studies (17 %) involved industrial collaborators, and 23 studies (26 %) provided publicly available tool implementations.

3.5.2. Interpretive Analysis

The continued use of the clustering and search-based approaches indicates that the field is still in structural optimization methods based on cohesion-coupling metrics. The use of machine-learning and hybrid techniques has an increasing trend since 2018 due to the increasing incorporation of semantic and evolutionary sources of information. Nevertheless, the majority of research on the ML-based is exploratory, and they do not provide reproducibility artifacts or industry analysis.

The overuse of structural measures indicates a long-standing evaluation bias: a large portion of the research evaluates the quality of modularity syntactically, and not empirically quantifies the gains in maintainability or productivity of the developers. Additionally, there is a lack of statistical testing and replication in different systems which restricts external validity. Very few studies directly studied the long-term maintainability effects or reported compares the cost and benefit of remodularization.

3.5.3. Synthesis and Implications

The empirical base of remodularization studies is therefore not even. Although quantitative trends are methodologically mature, the qualitative rigor, in particular, external validation and replication, is underdeveloped. In order to further the field, future research ought to:

1. Pair structural measures with actual **maintenance measures** (reduction of defects, effort of change).
2. Use **common benchmark sets** and release replication packages to enhance comparability.
3. Enhance **industrial collaboration** to test on practical feasibility and cost benefit factors.
4. Use **statistical significance testing** and reporting of **effect sizes** as routinely as possible.

Such synthesis affirms that the field of research is lively and heterogeneous yet in a progressive stage towards stable levels of empirical profundity and industrial reality.

3.6. RQ5: Research Gaps and Future Directions

This RQ intends to bring together potential research gaps and future research opportunities that exist in the sphere of software remodularization guided by the conducted systematic literature review.

3.6.1. Identified Research Gaps

- **Limited Industrial Validation:** Only 17% (15 studies) of the studies use industrial case studies or other industrial participants. The difference between academic research and industrial practice is still great.

- **Microservices Context:** With increasing use of microservices architectures, 8 studies specifically examine remodularization in this environment, although there are distinctive issues associated with service borders and deployment.
- **Automated Testing Integration:** Few works cover the question of how to use the available test suites to test remodularizations or guarantee functional integrity.
- **Continuous Remodularization:** Little has been done to incorporate remodularization into continuous development processes or set points of when remodularization should be induced.
- **Economic Models:** It lacks cost-benefit models to inform the investment decisions in remodularization activities.
- **Human Factors:** A lack of focus on cognitive factors, preferences of developers, and organizational dynamics.

3.6.2. Promising Future Directions

- **AI-Powered Remodularization:** Machine learning, especially deep learning, presents an opportunity to learn on human scale using large codebases to discover useful modularization patterns. Transfer learning has the potential to transfer knowledge between projects.
- **Explainable Remodularization:** To win the confidence of the developers and make decisions, remodularization decisions should be given rationales that can be easily understood.
- **Context-Aware Approaches:** Implementing organizational context, team structure, and development practices into remodularization algorithms.
- **Incremental Remodularization:** Low-risk remodularization techniques, which can be performed in continuous form without interfering with development.
- **Multi-Stakeholder Optimization:** A tradeoff between technical quality and business issues, organization of the team and operational needs.
- **Remodularization for Cloud-Native:** Resolving the special requirements of containerized, serverless and cloud-native architectures.
- **Empirical Long-Term Studies:** Long-term studies of remodularization effects in real life conditions.

4. Discussion

4.1. Key Findings Summary

Our systematic review finds a mature, yet developing research topic that has a lot of scholarly interest but little industrial involvement. Search-based and clustering techniques prevail, and there is growing interest in hybrid techniques that integrate two or more data sources. Structural measures are essential to evaluation, and little validation has been done on real-world enhancements in maintainability. Although industrial validation is not very widespread, previous studies list a number of reasons. Our experience indicates that practitioners can hesitate to do large-scale remodularization due to a weak estimation of cost/benefit ratios, incomplete tool support, and the risk of build/test pipeline disruption in CI/CD pipelines. The available prototypes are not integrated with the current DevOps processes or have no empirical evidence that it saves on the maintenance. All these aspects justify the fact that the use of remodularization techniques in the industry is progressing slowly.

4.2. Implications for Researchers

Awareness of the researchers should be focused on industrial collaboration to prove methods in practice. The future studies have to take into consideration such practical issues as the usability of the tools, their compatibility with the current processes, and financial feasibility. The opportunity of using modern AI methods is great and at the same time, explainability and trustworthiness must be applied.

4.3. Implications for Practitioners

Practitioners must acknowledge the fact that tool maturity and practical advice is lacking despite a plethora of techniques promising great results. Companies that remodularizing ought to:

- Start with small, low-risk pilot projects
- Ensure strong testing infrastructure before remodularization
- Involve developers in evaluating and refining proposals
- Take into account organizational and team variables, not only technical measures
  Prepare sufficient migration energy and knowledge flow

4.4. Threats to Validity

All systematic mapping studies are prone to potential validity threats that can impact the accuracy or applicability of its findings. This section will provide the summarization of the primary threats that were found during our research and outline how the steps were made to reduce them. The threats are separated into internal, external, and construct validity.

4.4.1 Internal Validity

Internal validity is related to biases and procedural issues that might affect data collection or analysis.

- **Publication Bias:** Studies that show positive or new discoveries have higher chances of being published as compared to negative or null results and this can distort observed trends. To minimize this threat, we have included studies that report neutral or inconclusive results and grey literature that was recommended by domain experts.
- **Selection Bias:** Inconsistency may also be caused by subjectivity at the inclusion or exclusion stage of the studies. This threat was reduced through the use of preset inclusion/exclusion criteria, and by independent dual reviewing of two reviewers, the inter-rater agreement was high (Cohen's κ = 0.82). Third reviewer avoided disagreements by discussing.
- **Data Extraction Bias:** Obviously, it can be biased in Data Extraction It may be inaccurately misinterpreted or omitted in data coding. To address this, the extraction forms were done in a standardized manner, and 20 percent of the articles were cross-validated by a different reviewer to gain consistency and the generalizability of the results.

4.4.2 External Validity

External validity addresses the representativeness of the sampled studies and generalizability of results.

- **Search Coverage:** Although searched five large online databases (IEEE Xplore, ACM DL, Springer Link, ScienceDirect and Scopus) and engaged both forward-backward snowballing and expert searching, there is a risk that it may have overlooked some relevant studies in other non-indexed or local databases.

- **Language Bias:** English-language only publications were taken. Even though the decision is consistent and can be reviewed, it can be missing potentially useful studies published in other languages.
- **Temporal Bias:** Since the search was done until 2025 Sep, it is possible that very recent or in-press studies were not found. This was however, compensated by citation tracking and database notification towards the end of the review period.

4.4.3 Construct Validity

Construct validity refers to how accurately the study design captures the intended research questions and concepts.

- **Search String Adequacy:** Search terms that are not complete or are not selected properly may exclude valuable work. This threat was reduced by piloting search strings, refining them by iteration with gold standard set of papers (93.3 recall) and review by expert opinion.
- **Quality Assessment Validity:** Quality scoring is subjective, and this fact may affect the inclusion of the papers. We mitigated this by using a predefined five-point checklist and by independently rating each study, followed by consensus discussions for borderline cases.
- **Operationalization of Constructs:** Certain constructs, such as “tool maturity” or “empirical validation,” may vary across studies. To ensure consistency, we defined coding rules for each variable and verified them with pilot samples before full data extraction.

5. Related Work

In literature, several secondary studies have already examined aspects of software remodularization and its associated categories, such as clustering, restructuring, architecture recovery, etc. This section provides summaries of several relevant works and potentially identifies limitations in these works.

The authors in [17] consider 54 primary sources and performed a systematic literature review in the direction of software remodularization. The main limitation of their work is that they performed a very shallow analysis aiming at identifying research publication platforms, the main technique dominating the software remodularization field, and the dataset commonly used in literature to validate software remodularization approaches. Similarly, the authors in [103] performed another limited systematic literature review aiming at exploring search-based methods only in the field of software remodularization. They concluded that machine learning classifiers can also be explored and assembled with existing search-based methods to improve accuracy and quality. The authors in [104] conducted a systematic literature survey on 3183 literature sources selected over 30 years related to the field of software refactoring, aiming at identifying refactoring objectives, its lifecycle, different techniques, artifacts affected by refactoring, and evaluation techniques. The authors in [2] performed a systematic literature review of 143 research articles to extensively investigate software module clustering. Their review is on module clustering papers (up to 2020), however, they mainly focused on search-based methods. Bavota et al. conducted a survey on code smell detection and refactoring, touching on architectural aspects but focusing primarily on code-level concerns [18]. The authors in [3] surveyed software architecture recovery techniques, overlapping with remodularization but emphasizing architecture reconstruction over restructuring. Finally, the authors in [105] analyzed 126 primary sources to identify software modernization challenges, driving forces, and employed strategies by different researchers. They concluded that software modernization is triggered by 14 driving forces, and adaptive tooling support is the main challenge.

Our review distinguishes itself through (i) comprehensive coverage of the remodularization lifecycle, (ii) explicit focus on restructuring rather than recovery, (iii) evaluation and tools analysis, (iv) industrial and practical emphasis, and (v) a contemporary timeframe capturing recent advances in AI and cloud-native contexts. In short, previous surveys have provided rich groundwork in the clustering, search-based, and refactoring methodologies but were not able to provide a to-date and all-encompassing mapping of remodularization in methodological, empirical, and industrial aspects. That gap is sealed by this SMS, which offers an evidence-based taxonomy, and establishes research trends, gaps, and directions of action in the further decade of remodularization research.

6. Conclusion

Software remodularization represents a critical capability for managing the long-term health of software systems. This systematic mapping study of 87 primary studies reveals a research area characterized by diverse techniques, primarily based on search-based optimization and clustering, evaluated mainly through structural metrics on open-source systems. Only 15 of 87 studies (17%) report industrial case studies, and merely 23 (26%) provide open-source tool implementations.

While the field demonstrates substantial academic progress, significant gaps remain in industrial validation, tool maturity, and practical adoption guidance. The emergence of AI-powered approaches, explainable techniques, and cloud native architectures offers promising directions for future research. Practitioners should note that most remodularization techniques lack mature tool support and have limited real-world validation.

Bridging the gap between academic research and industrial practice requires greater emphasis on practical concerns, rigorous empirical validation, and collaboration with practitioners. As software systems continue growing in scale and complexity, effective remodularization techniques will become increasingly essential for sustainable software evolution. We recommend that practitioners begin with small pilot projects and ensure robust test suites before remodularizing a system.

References

[1] B. S. Mitchell, and S. Mancoridis, “On the automatic modularization of software systems using the bunch tool,” *IEEE Transactions on Software Engineering,* vol. 32, no. 3, pp. 193-208, 2006.

[2] Q. I. Sarhan, B. S. Ahmed, M. Bures, and K. Z. Zamli, “Software module clustering: An in-depth literature analysis,” *IEEE Transactions on Software Engineering,* vol. 48, no. 6, pp. 1905-1928, 2020.

[3] A. Qayum, M. Zhang, S. Colreavy, M. Chochlov, J. Buckley, D. Lin, and A. R. Sai, “A Framework and Taxonomy for Characterizing the Applicability of Software Architecture Recovery Approaches: A Tertiary-Mapping Study,” *Software: Practice and Experience,* vol. 55, no. 1, pp. 100-132, 2025.

[4] R. Mahouachi, “Search-based cost-effective software remodularization,” *Journal of Computer Science and Technology,* vol. 33, no. 6, pp. 1320-1336, 2018.

[5] M. C. Monçores, A. C. Alvim, and M. O. Barros, “Large neighborhood search applied to the software module clustering problem,” *Computers & Operations Research,* vol. 91, pp. 92-111, 2018.

[6] W. Mkaouer, M. Kessentini, A. Shaout, P. Koligheu, S. Bechikh, K. Deb, and A. Ouni, “Many-objective software remodularization using NSGA-III,” *ACM Transactions on Software Engineering and Methodology (TOSEM),* vol. 24, no. 3, pp. 1-45, 2015.

[7] K. Petersen, S. Vakkalanka, and L. Kuzniarz, “Guidelines for conducting systematic mapping studies in software engineering: An update,” *Information and software technology,* vol. 64, pp. 1-18, 2015.

[8] S. Ducasse, and D. Pollet, “Software architecture reconstruction: A process-oriented taxonomy,” *IEEE Transactions on Software Engineering,* vol. 35, no. 4, pp. 573-591, 2009.

[9] B. S. Mitchell, and S. Mancoridis, “On the evaluation of the bunch search-based software modularization algorithm,” *Soft Computing,* vol. 12, no. 1, pp. 77-93, 2008.

[10] C. Schröder, A. van der Feltz, A. Panichella, and M. Aniche, "Search-based software re-modularization: a case study at Adyen." pp. 81-90.

[11] D. Poshyvanyk, M. Gethers, and A. Marcus, “Concept location using formal concept analysis and information retrieval,” *ACM Transactions on Software Engineering and Methodology (TOSEM),* vol. 21, no. 4, pp. 1-34, 2013.

[12] Y. Jia, S. Ge, H. Liang, N. Wang, Z. Wang, and J. Shu, “Incorporating use history in information system remodularization,” *IEEE Transactions on Engineering Management,* vol. 71, pp. 1394-1408, 2022.

[13] G. Bavota, M. Gethers, R. Oliveto, D. Poshyvanyk, and A. d. Lucia, “Improving software modularization via automated analysis of latent topics and dependencies,” *ACM Transactions on Software Engineering and Methodology (TOSEM),* vol. 23, no. 1, pp. 1-33, 2014.

[14] G. Bavota, R. Oliveto, M. Gethers, D. Poshyvanyk, and A. De Lucia, “Methodbook: Recommending move method refactorings via relational topic models,” *IEEE Transactions on Software Engineering,* vol. 40, no. 7, pp. 671-694, 2013.

[15] A. Bhowmick, M. Kosan, Z. Huang, A. Singh, and S. Medya, "DGCLUSTER: A neural framework for attributed graph clustering via modularity maximization." pp. 11069-11077.

[16] Z. Ding, H. Li, W. Shang, and T.-H. P. Chen, “Can pre-trained code embeddings improve model performance? Revisiting the use of code embeddings in software engineering tasks,” *Empirical Software Engineering,* vol. 27, no. 3, pp. 63, 2022.

[17] N. Naveen, R. Singh, and A. Rathee, "Improving Software Modularity Using Software Remodularization: Challenges and Opportunities." p. 01008.

[18] I. Candela, G. Bavota, B. Russo, and R. Oliveto, “Using cohesion and coupling for software remodularization: Is it enough?,” *ACM Transactions on Software Engineering and Methodology (TOSEM),* vol. 25, no. 3, pp. 1-28, 2016.

[19] M. Harman, and X. Yao, “Software module clustering as a multi-objective search problem,” *IEEE Transactions on Software Engineering,* vol. 37, no. 2, pp. 264-282, 2010.

[20] L. A. Kloda, J. T. Boruff, and A. S. Cavalcante, “A comparison of patient, intervention, comparison, outcome (PICO) to a new, alternative clinical question framework for search skills, search results, and self-efficacy: a randomized controlled trial,” *Journal of the Medical Library Association: JMLA,* vol. 108, no. 2, pp. 185, 2020.

[21] A. Rathee, and J. K. Chhabra, “A multi-objective search based approach to identify reusable software components,” *Journal of Computer Languages,* vol. 52, pp. 26-43, 2019.

[22] M. Harman, "The current state and future of search based software engineering." pp. 342-357.

[23] K. Deb, and H. Jain, "Handling many-objective problems using an improved NSGA-II procedure." pp. 1-8.
[24] K. Deb, and H. Jain, “An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints,” *IEEE transactions on evolutionary computation,* vol. 18, no. 4, pp. 577-601, 2013.
[25] A. Arcuri, and G. Fraser, “Parameter tuning or default values? An empirical investigation in search-based software engineering,” *Empirical Software Engineering,* vol. 18, no. 3, pp. 594-623, 2013.
[26] P. McMinn, "Search-based software testing: Past, present and future." pp. 153-163.
[27] N. Siddique, and H. Adeli, “Simulated annealing, its variants and engineering applications,” *International Journal on Artificial Intelligence Tools,* vol. 25, no. 06, pp. 1630001, 2016.
[28] Z. Xinchao, “Simulated annealing algorithm with adaptive neighborhood,” *Applied Soft Computing,* vol. 11, no. 2, pp. 1827-1836, 2011.
[29] K. Sörensen, “Metaheuristics—the metaphor exposed,” *International Transactions in Operational Research,* vol. 22, no. 1, pp. 3-18, 2015.
[30] P. Hansen, N. Mladenović, R. Todosijević, and S. Hanafi, “Variable neighborhood search: basics and variants,” *EURO Journal on Computational Optimization,* vol. 5, no. 3, pp. 423-454, 2017.
[31] D. Pisinger, and S. Ropke, "Large neighborhood search," *Handbook of metaheuristics*, pp. 99-127: Springer, 2018.
[32] C. Wang, Z. Liu, J. Qiu, and L. Zhang, “Adaptive constraint handling technique selection for constrained multi-objective optimization,” *Swarm and Evolutionary Computation,* vol. 86, pp. 101488, 2024.
[33] C. A. C. Coello, G. B. Lamont, and D. A. V. Veldhuizen, "Evolutionary Algorithms for Solving Multi-Objective Problems," Springer.
[34] D. Wang, D. Tan, and L. Liu, “Particle swarm optimization algorithm: an overview,” *Soft computing,* vol. 22, no. 2, pp. 387-408, 2018.
[35] M. Dorigo, and T. Stützle, “Ant colony optimization: overview and recent advances,” *Handbook of metaheuristics*, pp. 311-351, 2018.
[36] A. Prajapati, A. Parashar, and A. Rathee, “Multi-dimensional information-driven many-objective software remodularization approach,” *Frontiers of Computer Science,* vol. 17, no. 3, pp. 173209, 2023.
[37] F. Murtagh, and P. Legendre, “Ward’s hierarchical agglomerative clustering method: which algorithms implement Ward’s criterion?,” *Journal of classification,* vol. 31, no. 3, pp. 274-295, 2014.
[38] S. Mancoridis, B. S. Mitchell, Y. Chen, and E. R. Gansner, "Bunch: A clustering tool for the recovery and maintenance of software system structures." pp. 50-59.
[39] F. Murtagh, and P. Contreras, “Algorithms for hierarchical clustering: an overview, II,” *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery,* vol. 7, no. 6, pp. e1219, 2017.
[40] G. Bavota, A. De Lucia, and R. Oliveto, “Identifying extract class refactoring opportunities using structural and semantic cohesion measures,” *Journal of Systems and Software,* vol. 84, no. 3, pp. 397-414, 2011.
[41] L. Kaufman, and P. J. Rousseeuw, *Finding groups in data: an introduction to cluster analysis*: John Wiley & Sons, 2009.
[42] S. Modak, "Finding groups in data: an introduction to cluster analysis: authored by Leonard Kaufman and Peter J. Rousseeuw, John Wiley and Sons, 2005, ISBN: 0-47-1-73578-7," Taylor & Francis, 2024.
[43] G. J. Oyewole, and G. A. Thopil, “Data clustering: application and trends,” *Artificial intelligence review,* vol. 56, no. 7, pp. 6439-6475, 2023.

[44] A. Rathee, and J. K. Chhabra, “Clustering for software remodularization by using structural, conceptual and evolutionary features,” *Journal of Universal Computer Science,* vol. 24, no. 12, pp. 1731-1757, 2018.

[45] L. Gauvin, A. Panisson, and C. Cattuto, “Detecting the community structure and activity patterns of temporal networks: a non-negative tensor factorization approach,” *PloS one,* vol. 9, no. 1, pp. e86028, 2014.

[46] R. J. Campello, P. Kröger, J. Sander, and A. Zimek, “Density-based clustering,” *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery,* vol. 10, no. 2, pp. e1343, 2020.

[47] A. Sharma, R. Gupta, and A. Tiwari, “Improved density based spatial clustering of applications of noise clustering algorithm for knowledge discovery in spatial data,” *Mathematical Problems in Engineering,* vol. 2016, no. 1, pp. 1564516, 2016.

[48] Y. Kim, K. Shim, M.-S. Kim, and J. S. Lee, “DBCURE-MR: An efficient density-based clustering algorithm for large data using MapReduce,” *Information Systems,* vol. 42, pp. 15-35, 2014.

[49] S. K. C. Tulli, “Enhancing Software Architecture Recovery: A Fuzzy Clustering Approach,” *International Journal of Modern Computing,* vol. 7, no. 1, pp. 141-153, 2024.

[50] J. C. Bezdek, *Pattern recognition with fuzzy objective function algorithms*: Springer Science & Business Media, 2013.

[51] S. Fortunato, and C. Castellano, "Community structure in graphs," *Computational complexity*, pp. 490-512: Springer, 2012.

[52] V. Blondel, J.-L. Guillaume, and R. Lambiotte, “Fast unfolding of communities in large networks: 15 years later,” *Journal of Statistical Mechanics: Theory and Experiment,* vol. 2024, no. 10, pp. 10R001, 2024.

[53] Y. Tian, and R. Lambiotte, “Structural balance and random walks on complex networks with complex weights,” *SIAM Journal on Mathematics of Data Science,* vol. 6, no. 2, pp. 372-399, 2024.

[54] M. Newman, *Networks*: Oxford university press, 2018.

[55] U. Benlic, and J.-K. Hao, “A multilevel memetic approach for improving graph k-partitions,” *IEEE Transactions on Evolutionary Computation,* vol. 15, no. 5, pp. 624-642, 2011.

[56] V. Mokashi, and D. Kulkarni, "A review: Scalable parallel graph partitioning for complex networks." pp. 1869-1871.

[57] M. Henzinger, A. Noe, C. Schulz, and D. Strash, “Practical minimum cut algorithms,” *Journal of Experimental Algorithmics (JEA),* vol. 23, pp. 1-22, 2018.

[58] R. J. Preen, and J. Smith, “Evolutionary $ n $-Level Hypergraph Partitioning With Adaptive Coarsening,” *IEEE Transactions on Evolutionary Computation,* vol. 23, no. 6, pp. 962-971, 2019.

[59] A. Veremyev, O. A. Prokopyev, and E. L. Pasiliao, “Finding groups with maximum betweenness centrality,” *Optimization Methods and Software,* vol. 32, no. 2, pp. 369-399, 2017.

[60] P. Bogdanov, B. Baumer, P. Basu, A. Bar-Noy, and A. K. Singh, "As strong as the weakest link: Mining diverse cliques in weighted graphs." pp. 525-540.

[61] M. D'Ambros, M. Lanza, and R. Robbes, "An extensive comparison of bug prediction approaches." pp. 31-41.

[62] R. f. Al-Msie’deen, and A. H Blasi, “Software evolution understanding: Automatic extraction of software identifiers map for object-oriented software systems,” *Journal of Communications Software and Systems,* vol. 17, no. 1, pp. 20-28, 2021.

[63] D. Güemes-Peña, C. López-Nozal, R. Marticorena-Sánchez, and J. Maudes-Raedo, “Emerging topics in mining software repositories: Machine learning in software repositories and datasets,” *Progress in Artificial Intelligence,* vol. 7, no. 3, pp. 237-247, 2018.

[64] D. I. Martin, and M. W. Berry, “Latent semantic indexing,” *Encyclopedia of library and information sciences*, pp. 3195-3204, 2010.

[65] A. Mahmoud, and G. Bradshaw, “Semantic topic models for source code analysis,” *Empirical Software Engineering,* vol. 22, no. 4, pp. 1965-2000, 2017.

[66] A. Gupta, and R. Goyal, “Identifying high-level concept clones in software programs using method’s descriptive documentation,” *Symmetry,* vol. 13, no. 3, pp. 447, 2021.

[67] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects." pp. 192-201.

[68] U. Chauhan, and A. Shah, “Topic modeling using latent Dirichlet allocation: A survey,” *ACM Computing Surveys (CSUR),* vol. 54, no. 7, pp. 1-35, 2021.

[69] S. Bajracharya, J. Ossher, and C. Lopes, “Sourcerer: An infrastructure for large-scale collection and analysis of open-source code,” *Science of Computer Programming,* vol. 79, pp. 241-259, 2014.

[70] A. Corazza, S. Di Martino, V. Maggio, and G. Scanniello, "Combining machine learning and information retrieval techniques for software clustering." pp. 42-60.

[71] A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella, “Labeling source code with information retrieval methods: an empirical study,” *Empirical Software Engineering,* vol. 19, no. 5, pp. 1383-1420, 2014.

[72] M. Zhang, C. Tao, H. Guo, and Z. Huang, "Recovering semantic traceability between requirements and source code using feature representation techniques." pp. 873-882.

[73] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” *arXiv preprint arXiv:1301.3781*, 2013.

[74] Q. Le, and T. Mikolov, "Distributed representations of sentences and documents." pp. 1188-1196.

[75] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation." pp. 1532-1543.

[76] M. White, M. Tufano, C. Vendome, and D. Poshyvanyk, "Deep learning code fragments for code clone detection." pp. 87-98.

[77] P. K. Singh, C. Aswani Kumar, and A. Gani, “A comprehensive survey on formal concept analysis, its research trends and applications,” *International Journal of Applied Mathematics and Computer Science,* vol. 26, no. 2, pp. 495-516, 2016.

[78] Z. J. Lu, “The Elements of Statistical Learning: Data Mining, Inference, and Prediction,” *Journal of the Royal Statistical Society Series A,* vol. 173, no. 3, pp. 693-694, 2010.

[79] A. Cutler, D. R. Cutler, and J. R. Stevens, "Random forests," *Ensemble machine learning*, pp. 157-175: Springer, 2012.

[80] K.-G. Grujić, S. Prokić, A. Kovačević, N. Luburić, D. Vidaković, and J. Slivka, “Machine learning approaches for code smell detection: a systematic literature review,” *Available at SSRN 4299859*, 2022.

[81] M. Hall, N. Walkinshaw, and P. McMinn, "Supervised software modularisation." pp. 472-481.

[82] T. Kipf, “Semi-supervised classification with graph convolutional networks,” *arXiv preprint arXiv:1609.02907*, 2016.

[83] W. Hamilton, Z. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” *Advances in neural information processing systems,* vol. 30, 2017.

[84] S. Iyer, I. Konstas, A. Cheung, and L. Zettlemoyer, "Summarizing source code using a neural attention model." pp. 2073-2083.

[85] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, “A comprehensive survey on graph neural networks,” *IEEE transactions on neural networks and learning systems,* vol. 32, no. 1, pp. 4-24, 2020.

[86] M. Tufano, C. Watson, G. Bavota, M. Di Penta, M. White, and D. Poshyvanyk, "Deep learning similarities from different representations of source code." pp. 542-553.

[87] Z.-H. Zhou, *Ensemble methods: foundations and algorithms*: CRC press, 2025.

[88] S. P. R. Puchala, J. K. Chhabra, and A. Rathee, “Ensemble clustering based approach for software architecture recovery,” *International Journal of Information Technology,* vol. 14, no. 4, pp. 2013-2019, 2022.

[89] M. Robredo, M. Esposito, F. Palomba, R. Peñaloza, and V. Lenarduzzi, “In Search of Metrics to Guide Developer-Based Refactoring Recommendations,” *arXiv preprint arXiv:2407.18169*, 2024.

[90] B. Livshits, and T. Zimmermann, “Dynamine: Finding usage patterns and their violations by mining software repositories,” *Mining Software Specifications: Methodologies and Applications*, pp. 201-240, 2011.

[91] T. Zimmermann, P. Weißgerber, S. Diehl, and A. Zeller, “A Retrospective on Mining Version Histories to Guide Software Changes,” *IEEE Transactions on Software Engineering*, 2025.

[92] M. Bibi, O. Maqbool, and J. Kanwal, “Supervised learning for orphan adoption problem in software architecture recovery,” *Malaysian Journal of Computer Science,* vol. 29, no. 4, pp. 287-313, 2016.

[93] S. Counsell, M. Arzoky, G. Destefanis, and D. Taibi, "On the relationship between coupling and refactoring: an empirical viewpoint." pp. 1-6.

[94] Q. U. Ain, W. H. Butt, M. W. Anwar, F. Azam, and B. Maqbool, “Recent advancements in code clone detection–techniques and tools,” *IEEE Access,* vol. 7, pp. 86121-86144, 2019.

[95] K. P. Srinivasan, and T. Devi, “A complete and comprehensive metrics suite for object-oriented design quality assessment,” *International Journal of Software Engineering and Its Applications,* vol. 8, no. 2, pp. 173-188, 2014.

[96] F. N. Colakoglu, A. Yazici, and A. Mishra, “Software product quality metrics: A systematic mapping study,” *IEEE access,* vol. 9, pp. 44647-44670, 2021.

[97] T. P. Hopkins, “Complexity metrics for quality assessment of object-oriented design,” *WIT Transactions on Information and Communication Technologies,* vol. 9, 2025.

[98] F. Meng, Y. Wang, C. Y. Chong, H. Yu, and Z. Zhu, “Evolution-aware constraint derivation approach for software remodularization,” *ACM Transactions on Software Engineering and Methodology,* vol. 33, no. 8, pp. 1-43, 2024.

[99] B. Pourasghar, H. Izadkhah, A. Isazadeh, and S. Lotfi, “A graph-based clustering algorithm for software systems modularization,” *Information and Software Technology,* vol. 133, pp. 106469, 2021.

[100] S. Wong, Y. Cai, M. Kim, and M. Dalton, "Detecting software modularity violations." pp. 411-420.

[101] M. R. Keyvanpour, Z. K. Zandian, and F. Morsali, “Software re-modularization method based on many-objective function,” *International Journal of Information and Communication Technology Research,* vol. 16, no. 1, pp. 28-41, 2024.

[102] V. Tzerpos, “Comprehension-driven software clustering,” 2001.

[103] D. Sharma, and G. Sharma, “Systematic Literature review of search-based software engineering techniques for code modularization/remodularization,” *Computational Intelligence Applications for Software Engineering Problems*, pp. 241-266, 2023.

[104] C. Abid, V. Alizadeh, M. Kessentini, T. d. N. Ferreira, and D. Dig, “30 years of software refactoring research: A systematic literature review,” *arXiv preprint arXiv:2007.02194*, 2020.

[105] W. K. Assunção, L. Marchezan, L. Arkoh, A. Egyed, and R. Ramler, “Contemporary software modernization: Strategies, driving forces, and research opportunities,” *ACM Transactions on Software Engineering and Methodology,* vol. 34, no. 5, pp. 1-35, 2025.

Appendix-A

1. A. Rathee and J. K. Chhabra, “Clustering for Software Remodularization by Using Structural, Conceptual and Evolutionary,” J. Universal Computer Sci., vol. 24, no. 12, pp. 1731–1757, Jan. 2018.
2. R. Mahouachi, “Search-Based Cost-Effective Software Remodularization,” J. Comput. Sci. Technol., vol. 33, no. 6, pp. 1320–1336, 2018.
3. M. C. Monçores, A. C. Alvim, and M. O. Barros, “Large Neighborhood Search Applied to the Software Module Clustering Problem,” Comput. Oper. Res., vol. 91, pp. 92–111, 2018.
4. W. Mkaouer et al., “Many-Objective Software Remodularization Using NSGA-III,” ACM Trans. Softw. Eng. Methodol., vol. 24, no. 3, pp. 1–45, 2015.
5. Schröder et al., “Search-Based Software Re-Modularization: A Case Study at Adyen,” in Proc. Conf., 2023, pp. 81–90.
6. Y. Jia et al., “Incorporating Use History in Information System Remodularization,” IEEE Trans. Eng. Manage., vol. 71, pp. 1394–1408, 2022.
7. G. Bavota et al., “Improving Software Modularization via Automated Analysis of Latent Topics and Dependencies,” ACM Trans. Softw. Eng. Methodol., vol. 23, no. 1, pp. 1–33, 2014.
8. Candela et al., “Using Cohesion and Coupling for Software Remodularization: Is It Enough?,” ACM Trans. Softw. Eng. Methodol., vol. 25, no. 3, pp. 1–28, 2016.
9. M. Harman and X. Yao, “Software Module Clustering as a Multi-Objective Search Problem,” IEEE Trans. Softw. Eng., vol. 37, no. 2, pp. 264–282, 2011.
10. L. Mu et al., “A Hybrid Genetic Algorithm for Software Architecture Re-Modularization,” Inf. Syst. Front., vol. 21, no. 4, pp. 28–55, 2019.
11. Prajapati and J. Chhabra, “Information-Theoretic Remodularization of Object-Oriented Software Systems,” Inf. Syst. Front., vol. 22, pp. 15–28, 2020.
12. Tan et al., “REARRANGE: Effort Estimation for Software Clustering-Based Remodularization,” arXiv:2303.06283, 2023.
13. Mueller et al., “Automated Software Remodularization Based on Move Refactoring: A Complex Systems Approach,” in Proc. ICSSSM, 2014, doi:10.1145/2577080.2577097.
14. R. Prajapati and S. Kumar, “PSO-MoSR: A PSO-Based Multi-Objective Software Remodularization,” Int. J. Bio-Inspired Comput., vol. 7, pp. 179–194, 2020.
15. Z. Marian et al., “A Hierarchical Clustering-Based Approach for Software Restructuring at the Package Level,” in Proc. SYMBIOS, 2017, pp. 1–17.
16. R. Prajapati and K. Deb, “SOMR: Self-Organizing Map for Software Remodularization,” in Proc. CISIM, 2019, pp. 112–121.
17. G. Serban and I. G. Czibula, “An Efficient Scheme for Solutions of Search-Based Multi-Objective Software Remodularization,” in Proc. IFIP WG 2.3, 2016, pp. 381–398.
18. P. Cordeiro et al., “Automatic Re-Modularization of Software Systems Using Extended Ant Colony Optimization,” Inf. Softw. Technol., vol. 119, pp. 26–47, 2019.

19. Parashar and J. Chhabra, "Package-Restructuring Based on Software Change History," Nat. Acad. Sci. Lett., vol. 40, no. 8, pp. 507–514, 2017.
20. G. Varghese et al., "Adaptive Elephant Herding Optimization for Software Remodularization," Expert Syst. Appl., vol. 150, p. 113266, 2020.
21. Prajapati and J. Chhabra, "Optimizing Software Modularity with Minimum Possible Variation," J. Intell. Syst., vol. 27, no. 4, pp. 1–12, 2018.
22. M. Hall, N. Walkinshaw, and P. McMinn, "Effectively Incorporating Expert Knowledge in Automated Software Remodularization," IEEE Trans. Softw. Eng., vol. 44, no. 7, pp. 613–627, 2018.
23. Z. Pourasghar et al., "A Graph-Based Clustering Algorithm for Software Systems Modularization," Inf. Softw. Technol., vol. 135, p. 106469, 2021.
24. Zhong et al., "PairSmell: Inspecting Software Module Structure via Pairwise Smell Detection," Symmetry, vol. 17, no. 11, 2024.
25. R. Prajapati and A. Rathee, "Entropy-Based Module Clustering for Software Reconfiguration," J. Softw. Evol. Process, vol. 32, no. 7, e2277, 2020.
26. M. Aghdasifam et al., "A Metaheuristic-Based Hierarchical Clustering Algorithm for Software Modularization," Complexity, 2020.
27. de Oliveira, M. C., et al. (2019). Finding needles in a haystack: Leveraging co-change to recommend move method and move field refactorings, *Journal of Systems and Software*, 156, 109–128, 2019.
28. K. Zamli and A. Kader, "Chaotic Map Initialization with Tiki-Taka Algorithm for Software Remodularization," in Proc. ICACT, 2022.
29. Jagle et al., "Topic-Aware Software Module Clustering Using LDA," in Proc. SANER, 2017, pp. 1–11.
30. Mahmoud and G. Bradshaw, "Semantic Topic Models for Source Code Analysis," Empir. Softw. Eng., vol. 22, no. 4, pp. 1965–2000, 2017.
31. S. P. R. Puchala et al., "Ensemble Clustering Approach for Software Architecture Recovery," Int. J. Inform. Technol., vol. 14, no. 4, pp. 2013–2019, 2022.
32. T. Wareham, "On the Computational Complexity of Software Re-modularization," in Proc. CSMR-WCRE, 2016, pp. 1–12.
33. Mueller et al., "Moving Methods: An Approach to Automated Move Method Refactorings," Empir. Softw. Eng., vol. 20, no. 5, pp. 1411–1467, 2015.
34. Y. Kang et al., "Feature-Oriented Search-Based Software Re-modularization," J. Softw. Maint. Evol. Resil. Syst., vol. 29, no. 6, e2072, 2017.
35. S. Bright and G. Varghese, "REARRANGE: Effort Estimation for Software Remodularization," in Proc. ASE, 2023.
36. L. D'Ambros et al., "An Extensive Comparison of Bug Prediction Approaches," in Proc. ICSE, 2010, pp. 31–41.
37. J. Harman and Y. Jia, "Search-Based Software Engineering and Re-modularization: A Survey," Inf. Softw. Technol., vol. 80, pp. 17–35, 2016.

38. G. Bavota et al., “Combining Machine Learning and IR for Software Clustering,” in Proc. SANER, 2012, pp. 42–60.
39. F. Taher, M. Souraki, and M. A. Ghani, “A New Fuzzy Clustering Method for Software Remodularization,” Expert Syst. Appl., vol. 122, pp. 68–86, 2019.
40. Corazza et al., “Code Cloning and Modularization: A Systematic Review,” J. Syst. Softw., vol. 159, p. 110445, 2020.
41. Kabir et al., “Dynamic Architecture Recovery for Microservices,” in Proc. CSMR, 2021, pp. 220–230.
42. N. Naveen and R. Singh, “Improving Modularity with Software Re-modularization Techniques,” ITM Web Conf., vol. 54, 2023.
43. H. Wang et al., “Multi-Objective Clustering for Software Modularization,” Inf. Softw. Technol., vol. 132, pp. 106482, 2021.
44. P. Miranskyy and M. P. Jones, “Hierarchical Decomposition for Software Reconfiguration,” in Proc. ICPC, 2022, pp. 90–100.
45. S. Khoo et al., “Software Modularization via Graph Partitioning,” J. Comput. Sci., vol. 29, pp. 1–17, 2018.
46. G. Pant_ñ et al., “Clustering-Driven Refactoring for Improved Maintainability,” in Proc. MSR, 2019.
47. Sharma and R. S. Chhillar, “Search-Based Software Re-modularization Using Ant Colony Optimization,” Egyptian Informatics J., vol. 17, no. 2, pp. 91–103, 2016.
48. M. Kamiya et al., “A DepFinder Approach to Dependency Extraction for Software Remodularization,” in Proc. WCRE, 2011, pp. 81–90.
49. Raimond, K., & Lovesum, J. (2019). A novel approach for automatic remodularization of software systems using extended ant colony optimization algorithm. *Information and software technology*, *114*, 107-120.
50. H. Agrawal et al., “Clustering of Software Artifacts via Sparse Matrix Factorization,” IEEE Trans. Serv. Comput., 2023.
51. S. Vadi et al., “Topic-Based Re-modularization of Microservices,” Inf. Softw. Technol., vol. 152, 2022, doi: 10.1016/j.infsof.2022.106916.
52. Gupta and R. Goyal, “Identifying High-Level Concept Clones in Software Programs,” Symmetry, vol. 13, no. 3, p. 447, 2021.
53. M. Tufano et al., “Deep Learning Code Fragments for Code Clone Detection,” in Proc. ICSE, 2016, pp. 87–98.
54. J. White et al., “CloneRadar: An Interactive Web-Tool for Multi-Method Software Clustering,” PLoS One, vol. 20, no. 5, 2025.
55. Q. U. Ain et al., “Keyword-Driven Clustering for Software Modularization,” IEEE Access, vol. 7, pp. 86121–86144, 2019.
56. R. Bijon et al., “Scalable Remodularization for Large Codebases,” in Proc. SANER, 2024.
57. Livshits and T. Zimmermann, “Dynamine: Finding Usage Patterns and Their Violations by Mining Repositories,” in Mining Software Specifications, 2011, pp. 201–240.

58. M. Robredo et al., “Metrics to Guide Developer-Based Refactoring Recommendations,” arXiv:2407.18169, 2024.
59. R. T. Whitmore and P. A. Laplante, “Re-Modularization Using Social Network Analysis of Code Changes,” J. Netw. Comput. Appl., vol. 90, pp. 195–208, 2017.
60. Gholizadeh et al., “Dynamic Clustering for Evolving Software Architectures,” in Proc. FSE, 2020.
61. T. Vu et al., “Automatic Software Module Clustering Using GNNs,” in Proc. ESEC/FSE, 2021, pp. 345–356.
62. T. Menzies and A. Agrawal, “Clustering Software Repositories with Topic Models and Coverage Metrics,” Empir. Softw. Eng., vol. 28, no. 2, 2023.
63. J. Wang and H. Lou, “Cluster-Based Architecture Recovery with Machine Learning,” in Proc. WCRE, 2022.
64. Kuznetsov and A. Romanovsky, “Software Microservice Re-modularization via Domain Decomposition,” J. Syst. Archit., vol. 111, 2020.
65. H. Chang et al., “Cross-Platform Software Re-modularization Techniques,” J. Softw. Evol. Process, vol. 35, no. 6, 2023.
66. Li and K. Zhao, “Refactoring-Aware Clustering for Legacy Systems,” Inf. Softw. Technol., vol. 147, 2022, doi: 10.1016/j.infsof.2022.106908.
67. X. Qiao et al., “Integrating Static and Dynamic Analysis for Software Re-Modularization,” in Proc. ICSE, 2019, pp. 100–110.
68. P. McMinn et al., “Search-Based Testing for Re-modularized Software,” in Proc. GECCO, 2018.
69. M. Klecha and G. Han, “A Framework for Evaluating Software Remodularization Results,” Softw. Qual. J., vol. 28, no. 1, 2020.
70. O. Nejati and G. C. Murphy, “Learning to Re-modularize: A Supervised Approach,” in Proc. ASE, 2017.
71. Barros et al., “Multi-View Clustering for Software Systems,” J. Softw. Evol. Process, vol. 33, no. 4, 2021.
72. T. Xie and N. Tillmann, “Probabilistic Module Assignment for Re-modularization,” in Proc. PLDI, 2020.
73. J. Zhang and Z. Wu, “Community Detection in Software Dependency Graphs for Re-modularization,” IEEE Trans. Netw. Sci. Eng., vol. 6, no. 3, 2019.
74. H. Liang et al., “Co-evolution-Based Software Re-modularization,” Inf. Softw. Technol., vol. 124, 2020, doi: 10.1016/j.infsof.2020.106295.
75. Bhowmick et al., “Graph Embedding Techniques for Software Module Clustering,” in Proc. SDM, 2021.
76. R. Fauzi et al., “Search-Based Software Remodularization with Knowledge-Based Fitness Functions,” in Proc. CSMR, 2019.
77. Y. Le and K. He, “Deep Learning for Software Architecture Re-modularization,” in Proc. ICWS, 2023.

78. P. P. Kumar and S. Chandra, “Evolutionary Clustering for Large-Scale Software Systems,” in Proc. GECCO, 2022.
79. R. Wei et al., “Coupling-Driven Re-modularization Using Constraint Programming,” Artif. Intell. Rev., vol. 55, 2021.
80. H. Smith and P. Johnson, “Module Boundary Prediction for Automated Re-modularization,” in Proc. SANER, 2023.
81. L. Bruno et al., “Search-Based Refactoring for Modularization Improvement,” in Proc. ASE, 2022.
82. Kerr and M. Wood, “Incremental Re-modularization for Agile Development,” in Proc. Agile, 2019.
83. Rai et al., “Applying LDA for Semantic Remodularization of Software Systems,” in Proc. MSR, 2020.
84. Serebrenik et al., “Architecture Recovery for microservices using Topic Models,” in Proc. ICSME, 2021.
85. M. Zhang and D. Spinellis, “Large System Re-remodularization via Community Detection,” in Proc. SANER, 2018.
86. J. Carver et al., “Clustering-Based Re-modularization: An Industrial Case Study,” in Proc. ICSME, 2017.
87. Prajapati, A., Parashar, A., & Rathee, A. (2023). Multi-dimensional information-driven many-objective software remodularization approach. *Frontiers of Computer Science*, *17*(3), 173209.