Software Module Clustering: An In-Depth Literature Analysis

Qusay I. Sarhan, Bestoun S. Ahmed, Miroslav Bures, and Kamal Z. Zamli

Abstract—Software module clustering is an unsupervised learning method used to cluster software entities (e.g., classes, modules, or files) with similar features. The obtained clusters may be used to study, analyze, and understand the software entities' structure and behavior. Implementing software module clustering with optimal results is challenging. Accordingly, researchers have addressed many aspects of software module clustering in the past decade. Thus, it is essential to present the research evidence that has been published in this area. In this study, 143 research papers from well-known literature databases that examined software module clustering were reviewed to extract useful data. The obtained data were then used to answer several research questions regarding state-of-the-art clustering approaches, applications of clustering in software engineering, clustering processes, clustering algorithms, and evaluation methods. Several research gaps and challenges in software module clustering are discussed in this paper to provide a useful reference for researchers in this field.

Index Terms—Systematic literature study, software module clustering, clustering applications, clustering algorithms, clustering evaluation, clustering challenges

1 Introduction

Clustering (also called cluster analysis) is an unsupervised data mining technique that groups a set of data points into several clusters [1]. When several points fall inside a cluster, they are similar in some features. Measurement of the similarity and dissimilarity relies on the extent to which the data points share the same features. Clustering has been employed in many important fields of study and applications, including software engineering, information retrieval, machine learning, pattern recognition, and statistics [2].

In the context of software engineering, software clustering is being defined as the process of decomposing large software systems into smaller, manageable, meaningful (highly cohesive), independent (loosely coupled), and feature-oriented (share common features) subsystems [3], [4]. These subsystems may contain entities/artifacts (e.g., classes, modules, or files) of similar features. A software module clustering approach that accomplishes this task can have a substantial impact and practical benefits, especially for developers working on legacy systems with documentation that is outdated or nonexistent. Clustering in software engineering can be used in many applications, such as architecture recovery [5], code clone detection [6], and poor design detection [7].

With several algorithms and studies published in the literature, software module clustering has become an active research area. Although many module clustering approaches have been proposed and applied, they have difficulties meeting the current development and advancement needs in software and its various applications. For example, there is a lack of experimental studies on clustering a system developed using more than one programming language [8] or a system that performs some of its tasks by invoking ready-made web services available on the Internet/network [9], [10].

This paper presents a comprehensive systematic literature study to structure and categorize the state-of-the-art research evidence related to software module clustering during the past decade. The study defines several research questions (RQs) that cover many aspects of the field and then identifies relevant papers and their results. It concludes by discussing future research opportunities in the area. In this process, a systematic method is used to collect and analyze the related published research.

The remainder of this paper is structured as follows: Section 2 presents the motivation and an overview of related work. Section 3 describes in detail the research methodology used to conduct this study. Section 4 presents the results and outcomes. Section 5 discusses the issues of validity. Finally, Section 6 presents the conclusions of the study.

2 Motivation and Related Work

Software module clustering is an important topic of research in software engineering. Although it started in the 1990s, software module clustering research has experienced

- *Qusay I. Sarhan is with the Department of Computer Science, University of Duhok, Duhok, Kurdistan Region 01006, Iraq. E-mail: qusay.sarhan@uod.ac.*
- *Bestoun S. Ahmed is with the Department of Mathematics and Computer Science, Karlstad University, 651 88 Karlstad, Sweden, and the Department of Computer Science, Faculty of Electrical Engineering, Czech Technical University in Prague, Czech Republic. E-mail: bestoun@kau.se.*
- *Miroslav Bures is with the Department of Computer Science, Faculty of Electrical Engineering, Czech Technical University in PragueKarlovo nam. 13121 35 Praha 2, Czech Republic. E-mail: buresm3@fel.cvut.cz.*
- *Kamal Z. Zamli is with the Faculty of Computing, College of Computing and Applied Sciences, Universiti Malaysia Pahang, Pekan 26600, Malaysia. E-mail: kamalz@ump.edu.my.*

Manuscript received 29 Apr. 2020; revised 9 Nov. 2020; accepted 1 Dec. 2020. Date of publication 4 Dec. 2020; date of current version 14 June 2022. (Corresponding author: Bestoun S. Ahmed.)
Recommended for acceptance by L. Tan.
Digital Object Identifier no. 10.1109/TSE.2020.3042553

more momentum and attention in the past decade. This momentum and attention are reflected in the dramatic increase in the number of publication. Among the many factors leading to this increased attention, there have been two leading factors in the past decade. First is the dramatic increase in software application size due to the newly added functionalities and features they provide. This, in turn, has led to an increase in the number of modules for these applications. Here, software module clustering is a good approach to manage and maintain this kind of application. Second, the advancement of artificial intelligence (AI) methods (such as data mining, clustering, optimization, and machine learning methods) in the past decade has played a substantial role in increasing the research activity related to software module clustering.

Although it has been an active research area since the 1990s, a systematic literature study exploring research on software module clustering in the past decade is lacking. A few survey/review studies on software module clustering have been found in the literature. Shtern and Tzerpos [11] provided an overview of different software clustering methods and their applications in software engineering. The study highlighted some approaches for the evaluation of software clustering results. It also presented some research challenges to be addressed to improve software clustering results. In [12], the basic concepts and necessity of software module clustering were presented briefly. The authors also described different metaheuristic search techniques that have been applied to the software module clustering problem in the maintenance phase of the software development life cycle. In [13], the authors presented different search-based approaches to software clustering that have been classified into several categories (mono-objective, multiobjective, and many-objective) based on the number of clustering quality criteria. Furthermore, the advantages and disadvantages of each category are presented briefly. In [14] and its extended version [15], the authors described search-based optimization techniques and their applications in different software engineering domains. They briefly introduced software modularization and refactoring as clustering problems that can be addressed using several search algorithms. Additionally, they presented some research challenges with search algorithms, including determining suitable stopping criteria to terminate the search and issues related to visualizing the search results. The authors in [16] also dedicated parts of their study to performing software modularization and refactoring using search-based optimization techniques. They briefly introduced a number of algorithms in this respect, including NSGA-II and PCA-NSGA-II. Additionally, a number of evaluation metrics, such as coupling, cohesion, and modularization quality (MQ) have been mentioned.

The aforementioned studies were not focused on conducting a dedicated literature analysis of software module clustering: some aspects of software module clustering are covered as part of other related topics. By contrast, our paper presents an in-depth and systematic analysis with a detailed research methodology to examine different software module clustering aspects, such as applications, algorithms, tools, target systems, evaluation methods, and possible research gaps.

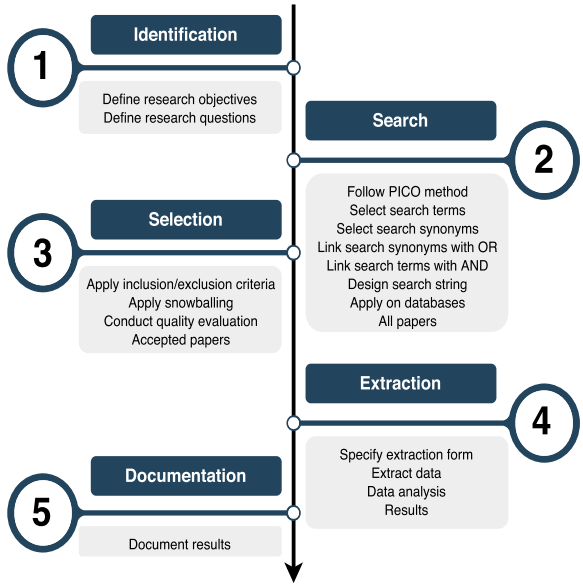

Fig. 1. Stages of the used literature analysis study.

3 Research Methodology

This study was based on the guidelines of conducting systematic mapping studies provided by [17], the guidelines for conducting literature review studies provided by [18], and other studies. These studies aimed to investigate software engineering in a particular context other than software module clustering (e.g., [19], [20], [21]). Fig. 1 shows the five-stage systematic process used in this research.

The first stage is defining the research scope and questions. The main research problem of the study was identified, and several questions were formed to address the problem. The second stage is conducting the search process in which a search strategy to select the primarily published papers was specified. The third stage is screening the published papers obtained from the previous stage via several filtering methods. The fourth stage is data extraction, where the selected papers are carefully analyzed, and useful data are extracted to answer the questions defined in this study. Finally, the fifth stage is the results reporting. The following subsections elaborate on the aforementioned stages in detail.

3.1 Identification of the Research Need and Scope

3.1.1 *Research Objectives*

This study aims to provide a comprehensive analysis of publications on software module clustering in the past decade by identifying, examining, and categorizing state-of-the-art contributions. Only papers from the past decade were considered to keep the review focused on recent works on the topic. Notable changes in software development have occurred over the past decade. Software systems migrated from simple architectures, such as monolithic and two-tiered architectures to multitier (also called n-tier or multilayered) architectures. Additionally, development approaches changed from structured to object-oriented to service-oriented. In other words, software development continues to change. Software developers use vast amounts of source code to understand large and complex systems before making changes, which is not feasible. Thus, software clustering has been employed to

TABLE 1
The Used Database Sources to Explore the Literature

Source	URL
IEEE Xplore	https://ieeexplore.ieee.org/
Elsevier ScienceDirect	https://sciencedirect.com/
ACM Digital Library	https://dl.acm.org/
Scopus	https://scopus.com/
SpringerLink	https://link.springer.com/

provide automated assistance to recover the abstract structure of systems and understand them.

3.1.2 Research Questions

To achieve the aim of this study and reflect its objectives, several RQs related to the topic were defined. Each defined RQ addresses a specific aspect of software module clustering.

- RQ1. What is the number and distribution of publications on software module clustering over the last decade?
- RQ2. Which researchers, organizations, and countries are active in software module clustering research?
- RQ3. What are software module clustering applications and how these applications are distributed?
- RQ4. What is the standard process of software module clustering?
- RQ5. What are the software systems used as targets for the experiments of software module clustering?
- RQ6. What are the current factbase sources, their types, their forms, and extraction tools used in software module clustering?
- RQ7. What are the most used similarity measures in the software module clustering?
- RQ8. What are the most used algorithms, their types, and standard stop conditions?
- RQ9. What are the most used tools for visualizing clustering results?
- RQ10. What are the metrics used to evaluate clustering results and the current approaches to obtaining the gold/expert decomposition?
- RQ11. What are the potential future research directions on software module clustering?

3.2 Search Strategy

3.2.1 Literature Sources

In this study, five standard online databases were selected as sources that index the literature of software engineering and computer science [17]. These sources are presented in Table 1.

3.2.2 Search String

The population, intervention, comparison, and outcomes (PICO) method [18] was employed to identify related studies. Here, population (P) refers to the applied area of clustering, intervention (I) refers to the process or procedure used to solve the clustering problem and outcomes (O) refers to the outcomes of the work and research of clustering in software engineering. Comparison (C) is not considered in the keywords, as this study is a general analysis of clustering in software engineering. Table 2 presents the keywords associated with each part of the method in the final search string. Notably, the search terms used in the final search string were linked with one another based on the steps described in [22]. Here, the "OR" Boolean operator was used to link synonyms or related search terms to the topic of this study and the "AND" Boolean operator was employed to link the main search terms.

Different combinations of search strings tried to construct the final one since the term "clustering" is used in other research areas such as data mining, image processing, and statistics. The final search string was the one that meets the following two criteria.

- The search string that returns the most relevant studies.
- The search string that returns the maximum number of the identified pilot set. For this study, a pilot set of 25 papers has been selected based on our experience and initial research review.

As an example, the search string Try 2 in Table 3 was excluded because the returned result is incomplete compared to the search string Try 5.

3.3 Paper Selection

3.3.1 Inclusion/Exclusion Criteria

To decide whether a paper is relevant to the scope of this research, a set of criteria, which are presented below, for inclusion and exclusion was considered.

- *Inclusion criteria are:*
  - Papers published online from 2008-2019. Papers related directly to software module clustering. This is ensured by reading the title of each obtained paper. However, the abstract or full-text reading has been also applied when the title

TABLE 2
PICO Related Keywords

PICO method	Keywords used
Population (P)	(software OR project OR system OR application OR program OR module OR component OR service OR source code OR package OR file OR function OR class)
Intervention (I)	(clustering OR cluster analysis OR partitioning OR grouping OR splitting OR structuring OR modularizing OR constructing OR composing OR categorizing OR classifying)
Comparison (C)	No comparisons considered in the keywords
Outcomes (O)	(methodology OR algorithm OR technique OR approach OR method OR tool OR improvement OR evaluation OR similarity measurement OR application OR metric OR problem OR challenge OR limitation)

TABLE 3
Search Strings Piloted on IEEE Xplore

# Try	Search string	# Results	# Missing studies
Try 1	((software) AND (clustering) AND (problem))	355	15
Try 2	((software OR module OR component) AND (clustering OR cluster analysis) AND (algorithm OR technique OR approach OR problem))	450	11
Try 3	((software OR system OR program OR module OR component OR source code OR package OR file) AND (clustering OR cluster analysis OR partitioning OR grouping OR splitting OR structuring OR categorizing OR classifying) AND (methodology OR algorithm OR technique OR approach OR method OR tool OR improvement OR problem OR challenge OR limitation))	631	8
Try 4	((software OR system OR program OR module OR component OR source code OR package OR file) AND (clustering OR cluster analysis OR partitioning OR grouping OR splitting OR structuring OR categorizing OR classifying) AND (methodology OR algorithm OR technique OR approach OR method OR tool OR improvement OR measurement OR application OR problem OR challenge OR limitation))	811	3
Try 5	((software OR project OR system OR application OR program OR module OR component OR service OR source code OR package OR file OR function OR class) AND (clustering OR cluster analysis OR partitioning OR grouping OR splitting OR structuring OR modularizing OR constructing OR composing OR categorizing OR classifying) AND (methodology OR algorithm OR technique OR approach OR method OR tool OR improvement OR evaluation OR similarity measurement OR application OR metric OR problem OR challenge OR limitation))	905	0

reading was not enough. This criterion filtered most of the papers out.

- *Exclusion criteria are:*
  - Papers not published in English are excluded since English is the prevalent language used in the scientific peer-reviewing global community.
  - Papers without accessible full text.
  - Papers not formally peer-reviewed (gray literature and books).
  - Papers not published electronically.
  - The duplicated papers were excluded from the list. Authors sometimes publish expanded versions of their conference papers to journal venues. Such papers share most of the material and considering them both would affect the quality of this study. To overcome this issue, duplicated papers are identified by comparing paper titles, abstracts, and contents. When the duplication is confirmed, the least recent publication is removed.
  - Master and Ph.D. dissertations are excluded because the content of such publications is eventually presented in peer-reviewed venues, which have already been considered in our study.
  - Papers that are published as surveys are filtered out because they do not actually bring new technical contributions to software module clustering.

3.3.2 Snowballing

The snowballing [23] search method was applied to the remaining papers to reduce the possibility of missing critical related papers. In this method, each research paper's list of references is examined in terms of the previously applied inclusion/exclusion criteria. The process was then recursively applied to newly identified papers.

3.3.3 Quality Evaluation

For quality evaluation, each paper must be evaluated to determine whether sufficient information can be extracted from it. Papers that did not provide answers to the following two questions were excluded:

- Is the process of software module clustering described in detail?
- Are there experimental results and evaluation for the software module clustering process?

The number of included and excluded papers at each stage of the paper selection process is shown in Fig. 2. Besides, all the identified papers and their references, full names, and publication years are listed in Table 12.

3.4 Data Extraction

In this phase, the data of the considered studies were extracted and analyzed to answer the defined RQs. The data

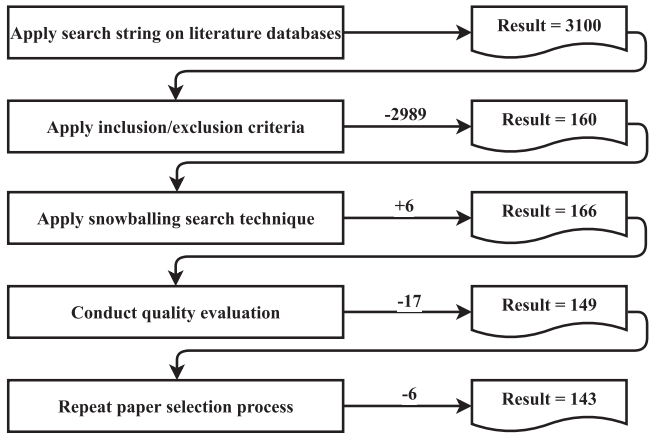

Fig. 2. Results of the paper selection process.

TABLE 4
Data Extraction Form

Data Item	Value	Relevant RQ
ID	Integer paper ID number	None
Title	Paper title	None
Year	Paper publication year	RQ1
Type	Paper publication type	RQ1
Venue	Publication venue name	RQ1
Author Name	Name of the author(s)	RQ2
Author Organization	Name of the organization for each participated author/co-author	RQ2
Author Country	Name of the country for each participated author/co-author	RQ2
Clustering Applications	Applications of software module clustering	RQ3
Clustering Process	The standard steps of software module clustering	RQ4
Target Systems	The software systems used in the experimental testing	RQ5
Factbase Extraction	The sources, fact types, and tools used for factbase extraction	RQ6
Similarity Measures	The similarity measures used in the clustering process	RQ7
Clustering Algorithms	The algorithms used in clustering and their types	RQ8
Visualization Tools	The tools used for visualizing clustering results	RQ9
Evaluation Metrics	The metrics used for clustering results evaluation	RQ10
Future Research	The possible future research directions	RQ11

obtained in this phase were stored in an Excel file with different fields created for this purpose. Each data extraction field has a data item and a value, as presented in Table 4. Notably, a reliable data extraction method was followed in this stage: the data were extracted first by the first author and then double-checked by the other authors separately.

4 Results

The selected papers were carefully analyzed to answer the RQs. Here, a short title is used to represent each RQ. The following subsections present and discuss the results based on each RQ.

4.1 Distribution of Publications (RQ1)

4.1.1 Publication Frequency

As previously mentioned, 143 papers published over the past decade (2008-2019) were included in this study. The selected papers were analyzed to determine their frequency and evolution, as shown in Fig. 3.

The figure also shows that the average number of publications per year is approximately 12. Additionally, interest in the topic has increased in the past three years, which indicates the successful application of module clustering to solve problems in software engineering along with the exponential growth of software applications in number, size, and complexity. One reason for this growth is that software has no limited lifetime: software code is constantly changed to meet user needs. Thus, developers are always in need of tools and approaches to ease the process of software maintenance. Software module clustering helps considerably in this context. Another valid reason is that software development is constantly changing with the development of new technologies. In the past few years, AI and the Internet of Things (IoT) have become trending technologies [24] impacting software development. The complexity of software systems based on such technologies is increasing dramatically as advanced features are employed. As a result, many legacy systems have been transformed to cope with this change. Understanding those systems before integrating them with new technologies requires an in-depth analysis, which may be achieved by employing software clustering.

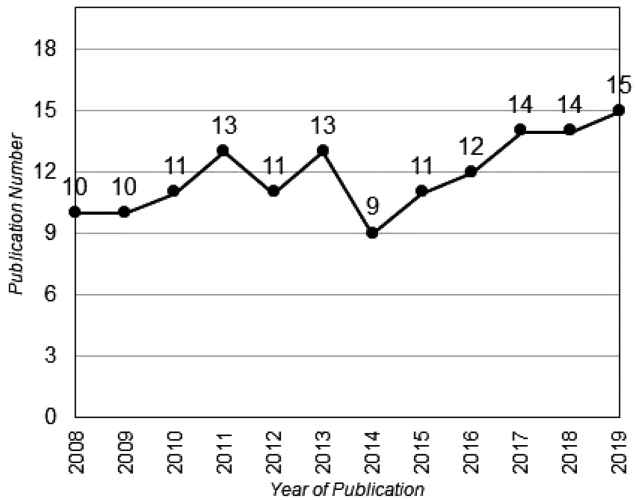

Fig. 3. Publication per year.

4.1.2 Publication Venue

Fig. 4 shows that the considered papers, namely, 92 are conference papers, 43 are journal papers, 4 are symposium papers, and 4 are workshop papers were published in

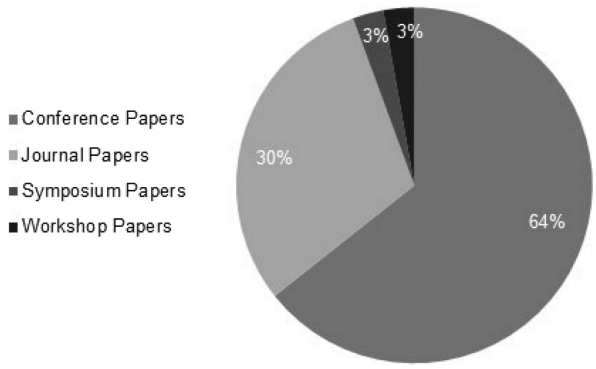

Fig. 4. Publication ratio per each venue.

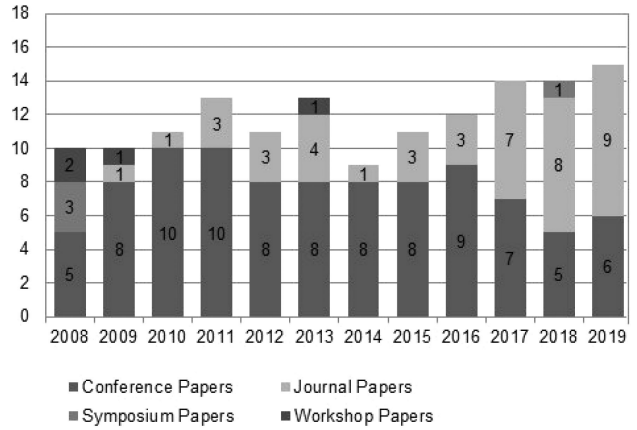

Fig. 5. Publication number per each venue.

various venues. This result also shows that only 30 percent of the considered papers have reached the maturity of a journal publication, indicating that software module clustering is a very young or even immature research area [25], [26]. Moreover, few conference papers were published as book chapters. For these papers, consideration was given to their original venues, that is conferences. Fig. 5 shows the publication number per year by venue type.

The most active and top journals, conferences, symposiums, and workshop venues that publish papers on software module clustering can be determined by analyzing the publications. Abbreviations are used in this paper instead of full names. Fig. 6 shows the active journals in which the considered studies were published. The full names of the journals are presented in Table 13. The figure also shows that the most active and top journals are "Inf. Softw. Technol.", "J. Syst. Softw.", "IEEE Trans. Softw. Eng.", "Procedia Comput. Sci.", "Soft Comput.", and "IET Softw.". Notably, 44 percent of the journal papers were published in the top six journals, whereas the other 56 percent were published in individual journals.

Fig. 7 shows the active conferences that published papers in software module clustering. The full names of the conferences can be found in Table 14. The most active and top conferences are the International Conference on Program Comprehension (ICPC), Conference on Software Maintenance and Reengineering (CSMR), and Working Conference on Reverse Engineering (WCRE). Notably, approximately 15 percent of the conference papers were published at these top three conferences. If other conferences that publish two or three papers are considered, then approximately 37 percent of the conference papers were published by annual conferences. The largest number, approximately 63 percent, of the published conference papers were at individual conferences, represented as "Others" in Fig. 7.

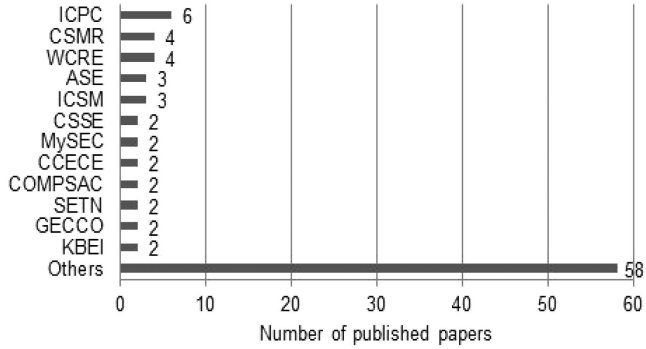

Fig. 7. Number of published papers versus conference name.

4.2 Active Researchers, Organizations, and Countries (RQ2)

Many researchers are interested and involved in software module clustering research. However, the most active researchers are those who have more than two published papers either as a main author or coauthor. The ranking of active researchers is shown in Fig. 8. The ranking shows that "Jitender Kumar Chhabra" and "Giuseppe Scanniello" are the top two researchers in this field, with nine and eight published papers, respectively. These authors participated in approximately 12 percent (17/143) of all publications.

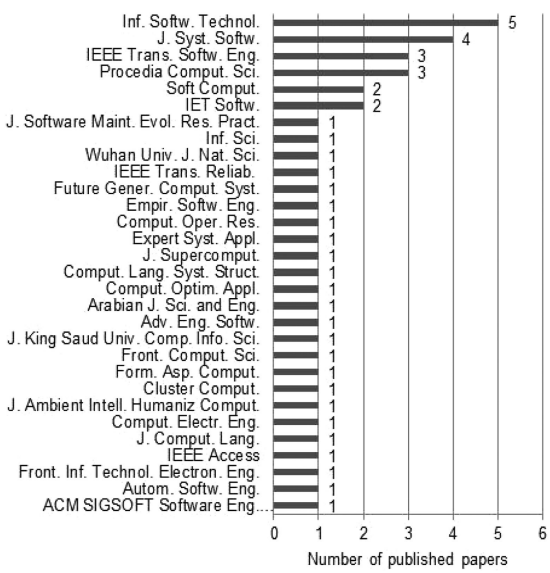

Fig. 6. Number of published papers versus journal name.

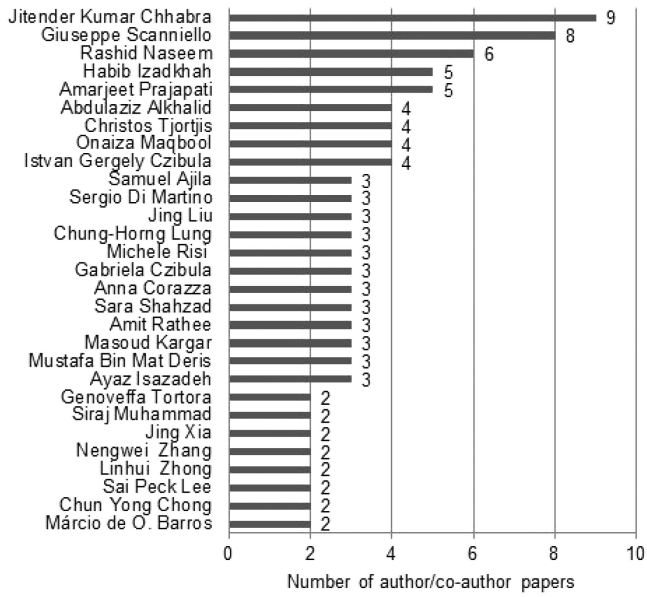

Fig. 8. Active researchers based on the published papers.

TABLE 5
Countries, Organizations, and Researchers Active in Researching Software Module Clustering

Country	Organization	Author(s)	Published Papers	Total
India	National Institute of Technology Kurukshetra	Jitender Kumar Chhabra, Amarjeet Prajapati, and Amit Rathee	[27], [28], [29], [30], [31], [32], [33], [34], [35], [36]	10
Italy	University of Basilicata	Giuseppe Scanniello	[37], [38], [39], [40], [41], [42], [43], [44]	8
Iran	University of Tabriz	Habib Izadkhah, Hamid Masoud, and Ayaz Isazadeh	[31], [45], [46], [47], [], [48], [49]	7
Pakistan	City University of Science and Information Technology	Rashid Naseem	[7], [50], [51], [52], [53]	5
Pakistan	Quaid-i-Azam University	Onaiza Maqbool, Adeel Ahmed, and A.Q. Abbasi	[7], [50], [52], [54], [55],	5
Romania	Babeș-Bolyai University	Istvan Gergely Czibula and Gabriela Czibula	[3], [56], [57], [58]	4
Canada	Carleton University	Samuel Ajila, Chung-Horng Lung, and Abdulaziz Alkhalid	[59], [60], [61]	3
Italy	University of Naples Federico II	Anna Corazza and Sergio Di Martino	[37], [41], [44]	3
Malaysia	Universiti Tun Hussein Onn Malaysia	Mustafa Mat Deris and Rashid Naseem	[51], [52] , [55]	3
China	Xidian University	Jing Liu	[62], [63], [64]	3
Italy	University of Salerno	Michele Risi and Genoveffa Tortora	[39], [42], [65]	3
Iran	Islamic Azad University	Masoud Kargar	[31], [], [49]	3
Pakistan	University of Peshawar	Sara Shahzad	[52] , [53], [55]	3
Saudi Arabia	King Abdullah University of Science and Technology	Abdulaziz Alkhalid	[66], [67]	2
Pakistan	Shaheed Benazir Bhutto University	Siraj Muhammad	[50], [54]	2
Brazil	UNIRIO - Universidade Federal do Estado do Rio de Janeiro	Márcio de O. Barros	[68], [69]	2
England	University of Manchester	Christos Tjortjis	[2], [70],	2
Greece	International Hellenic University	Christos Tjortjis	[71], [72]	2
China	Jiangxi Normal Universit	Linhui Zhong, Nengwei Zhang, and Jing Xia	[73], [74]	2
Malaysia	University of Malaya	Chun Yong Chong and Sai Peck Lee	[75], [76]	2

Table 5 shows the ranking of the active countries and organizations, including the name of country, organization, participating researchers, reference to the published papers, and the total number of papers.

The active countries in the published papers can also be extracted from the information presented in Table 5. Such data can be obtained based on the organizational affiliation of the authors/coauthors. Fig. 9 shows the countries that are most active in publishing papers on software module clustering and the share of papers published by each country with respect to the total number of publications.

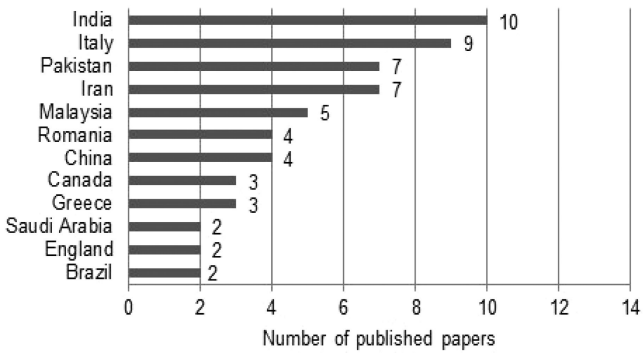

Fig. 9. Active countries.

The most active countries were those that published more than one research paper. Such countries produced approximately 41 percent (58/143) of the total publications. India and Italy were the two most active countries, with 10 and 9 published papers, respectively. Those papers published from India were from a collaboration between the authors "Jitender Kumar Chhabra," "AmarjeetPrajapati," and "Amit Rathee."

4.3 Applications of Software Module Clustering (RQ3)

An in-depth analysis of the selected papers revealed that software module clustering applications can be classified into 14 areas (A1-A14). This result is presented in Table 6 which shows the papers in each application area. From the identified application areas, it is clear that "information recovery (A1)" and "restructuring (A2)" are the top two applications of software module clustering with 59 and 44 published papers, respectively.

4.4 Software Module Clustering Process (RQ4)

Answering RQ4 helps to determine the detailed steps of the software module clustering process. Moreover, it helps to identify the algorithms, techniques, and tools used in each

TABLE 6
Application Areas of Software Module Clustering

ID	Application Areas	Published Papers	Total
A1	To support information recovery such as abstraction levels (e.g., modules or design patterns as low-level abstraction and architectures as high-level abstraction). Information recovery is essential for better software understanding, maintaining, and updating.	[77], [5], [29], [30], [31], [35], [38], [39], [42], [44], [45], [], [48], [49], [51], [52], [53], [59], [60], [61], [63], [64], [65], [69], [73], [75], [76], [78], [79], [80], [81], [82], [83], [84], [85], [86], [87], [88], [89], [90], [91], [92], [93], [94], [95], [96], [97], [98], [99], [100], [101], [102], [103], [104], [105], [106], [107], [108], [109]	59
A2	To increase software maintainability by reducing the complexity of its modules via decoupling, restructuring, or refactoring.	[3], [7], [27], [32], [33], [34], [37], [47], [50], [55], [56], [57], [58], [66], [67], [68], [70], [72], [74], [110], [111], [112], [113], [114], [115], [116], [117], [118], [119], [120], [54], [121], [122], [123], [124], [125], [126], [127], [128], [129], [130], [131], [132], [133]	44
A3	To identify software components that implement related functionality (a component is a strongly related set of software entities such as files, classes, and modules). Software component identification is important to promote the reuse of recovered components from existing software systems as building blocks for newly developed systems.	[4], [28], [36], [41], [46], [62], [134], [135], [136], [137], [138], [139], [140], [141], [142], [143]	16
A4	To identify the fault/defect proneness of software modules that require further examination and maintenance.	[43], [71], [144], [145], [146], [147], [148]	7
A5	To find concept locations (also called change propagation or change impact analysis) in source code. Finding concept locations in source code helps to identify where changes are to be made in response to new requirements.	[40], [149], [150]	3
A6	To evaluate a software system's maintainability via determining the entities that are more difficult to maintain than others.	[2], [151], [152]	3
A7	To map components found in the source code onto machines found in a distributed environment. Here, the number of clusters is created based on the number of machines in the used distributed environment.	[153], [154]	2
A8	To identify duplicate code (also called code clone). A code clone is a piece of code that appears at least twice in a software system due to copy-paste activity or reusability of existing code. Code clone detection helps improve the performance of software and reduce its maintenance cost and effort.	[6], [155], [156]	3
A9	To identify groups of similar code changes. A code change is a sequence of edit operations that turns an original code (old version) into its modified version (new version). The identified code changes can serve as inputs for some software tools such as recommendation and bug fixing tools.	[157]	1
A10	To identify different types of software architectures that are used in heterogeneous systems.	[158]	1
A11	To understand locations within an AOP system that could cause or exhibit aspect interference problems. Thus, it is essential to develop an interference-free AOP system.	[159]	1
A12	To discover program topoi (which are summaries of the main capabilities of a program).	[160]	1
A13	To generate microservice candidates from a monolithic software system.	[161]	1
A14	To assess the initial partition of a software system (assess the extent to which a certain partition allows its parts to evolve independently).	[162]	1

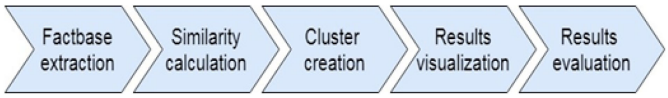

Fig. 10. Software module clustering process.

step of the process. The ideal software module clustering process includes five main steps, as shown in Fig. 10. Here, the process ends when reaching a stopping criterion such as a maximum number of iterations or a desired number of clusters. The following subsections describe the steps of the process.

4.4.1 Factbase Extraction

The input that a software clustering algorithm expects is often called a factbase as it contains facts (e.g., relationships between software entities) extracted from the target software system [163]. In this step, the extracted factbase should consist of sufficient information about the target software system to ensure meaningful clustering [70]. Factbase extraction includes target software system selection, factbase source selection, filtering and preprocessing, entity selection, and feature selection.

A. Target Software System Selection (RQ5)

Before starting the software clustering process, target software systems must be specified. The analysis of the selected studies reveals that open-source software systems written in Java and C/C++ have been the focus in the literature. Fig. 11 shows the most commonly used target software systems.

The figure clearly shows that "Junit," "jEdit," and "JHotDraw" are the most commonly used systems in the experimental tests, accounting for approximately 16 percent (23/143), 13 percent (18/143), and 11 percent (16/143) of the published papers, respectively. In addition, the top three systems are written in Java. The reason for this fact could be that Java is still the most widely used programming language according to the TIOBE Index [164]. For the experimental clustering tests, we recommend selecting the target systems that are well-known to the developer and researcher communities. They are free and open-source, written in widely used programming languages, and updated frequently.

B. Factbase Source Selection (RQ6)

Any software clustering process starts with the construction of a factbase. As mentioned, the factbase contains information on the target software system, such as software entities (e.g., classes and variables) and their relationships (e.g., inheritance and method calls). Such information can be extracted from various sources. Table 7 presents in detail the different types of factbase sources. After constructing the factbase, a software clustering algorithm can be applied to group entities from the factbase into useful subsystems. Many clustering methods combine facts extracted from different sources to obtain reasonable results at the cost of more complex data processing [82].

Fig. 12 shows that "source code", "documentation", "dynamic information", and "bytecode" are the most commonly used sources for factbase extraction, constituting approximately 82 percent (117/143), 3 percent (4/143), 2 percent (3/143), and 3 percent (2/143) of the total published

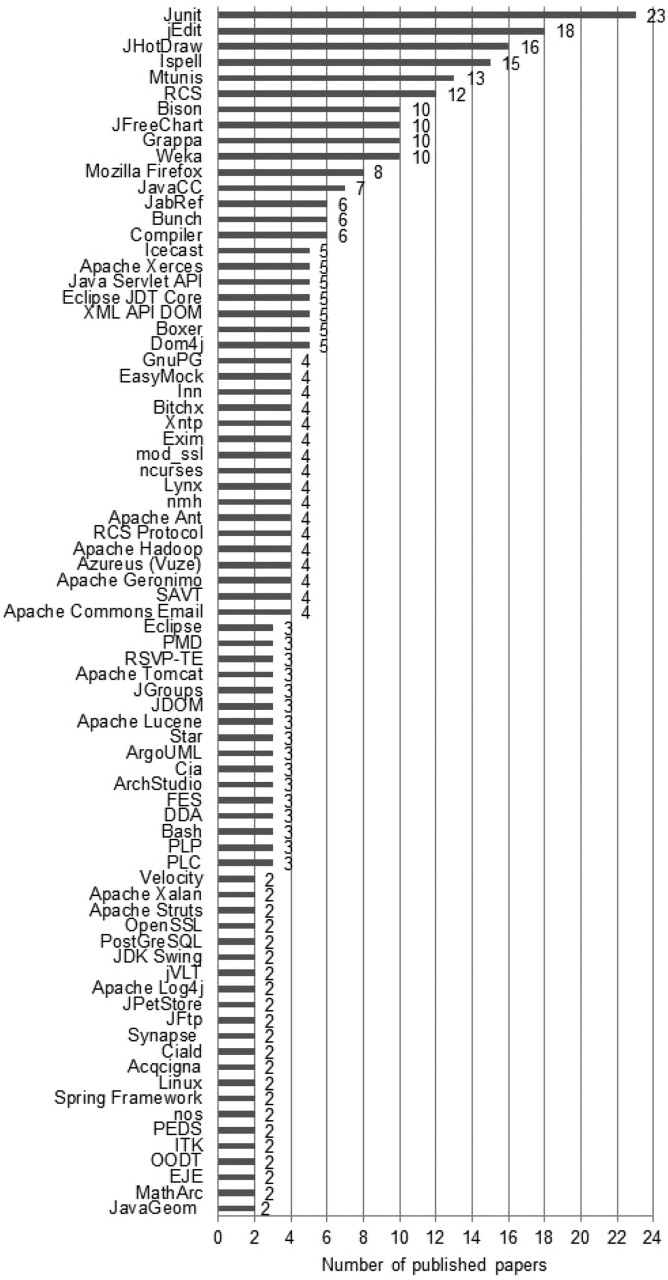

Fig. 11. Number of published papers versus the targeted software systems.

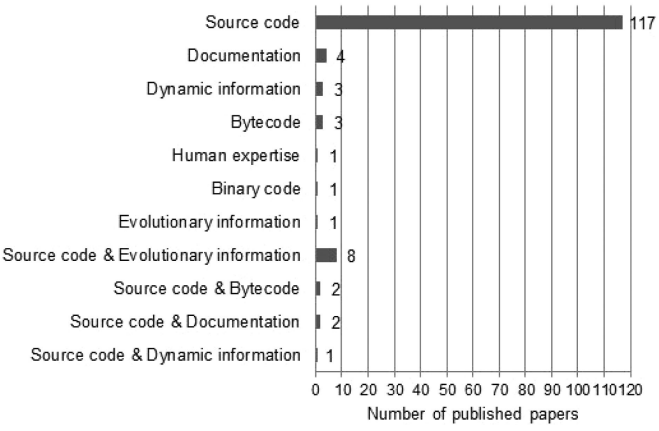

Fig. 12. Number of published papers versus the factbase source.

TABLE 7
Factbase Sources

Factbase Sources	Factbase Extracted	Factbase Type	Major Drawbacks
Source code	• Structural dependencies (also called syntactic information) between software entities (e.g., function calls, class references, class inheritance, file inclusions, package usages, and global variable access). Often represented as directed graphs or matrices. • Domain or naming information (also called lexical information or semantic information) found in comments, variable names, object names, method names, class names, file names, and source code statements	Static	• It is language-dependent. • May be mismatched to a product version of binary modules because of errors, and updates • Comments and naming conventions may not be used or followed by software engineers. • Extracting domain knowledge is known to be a difficult task due to the existence of noise in the data. • It deals with structural relationships rather than behavioral relationships (dynamic interactions among system entities). • Not available with some software systems. • If the software system is extremely large. The number of entities involved in clustering would be so large that it could be unmanageable to track. • Natural language processing techniques (stemming) to extract naming information is very time-consuming
Binary code (executable files)	• Compilation parameters, linkage parameters, and symbol table that contain relationships between components and modules of a software system.	Static	• It is language-dependent. • It is a compiler dependent. Thus, compiling a source code via two different compilers may result in two different binary codes. This is because each compiler has its building parameters and nature of work.
Bytecode	• Class dependencies. • Names of variables and classes	Static	It is language-dependent. • Not available with some software systems.
Documentation	• System's structure or design. • System's functional and non-functional requirements.	Static	• Facts are often incomplete, outdated, unsynchronized, vague, or even missing due to long term maintenance or personnel turnover. Thus, they may not reflect the current state of the system. Also, documentation could be completely missing.
Human expertise	• Knowledge of requirement documents. • High-level design. • Knowledge of architectural styles.	Static	• On the same software system, different experts may construct different software decomposition.
Configuration files	• Declaration of used resources. • Security permissions.	Static	• Unavailable because they are stored in middleware configuration files that are not part of the the software systems' source or binary code.
Dynamic information	• Object construction and destruction. • Sources of exceptions and errors. • Method entry and exit. • Method invocations/calls. • Execution logs (structured execution outputs that capture the process of building, deploying and testing various scenarios) • Performance counters and statistics such as number of threads, size of buffers, number of network connections, CPU and memory usage, number of component instances, average, maximum, and minimum response time.	Dynamic	• It is language-dependent. • Often, the source code does not remain untouched.
Data files	• Such as database, text, and reports files used as input or output that contain information about which classes or modules perform file operations (e.g., create, read, and write).	Dynamic	• Some software systems do not deal with input and output files.
Files organization	• The organization of software classes, files, folders, packages and their physical locations may provide information about their responsibilities.	Organizational	• Often, software files and folders are not organized well.
Human organization	• The structure of the system. Usually, a developer is responsible for associated components.	Organizational	• In medium and large software systems, it is not identified well due to developers turn out.
Evolutionary (historical) information	• File ownership, file timestamps, file versions, commit logs, persons making changes, maintenance activities, and bug report information extracted from software repositories, version control/management systems, bug/issue tracking systems, or release documents. • Also, it tells us how, when, and why a given software entity was modified.	Evolutionary	• It does not cover the essential parts of the software system unless huge evolutionary data has to be dealt with. Thus, it is usually difficult to deal with due to the issue of size. • The lack of evolutionary data formatting.

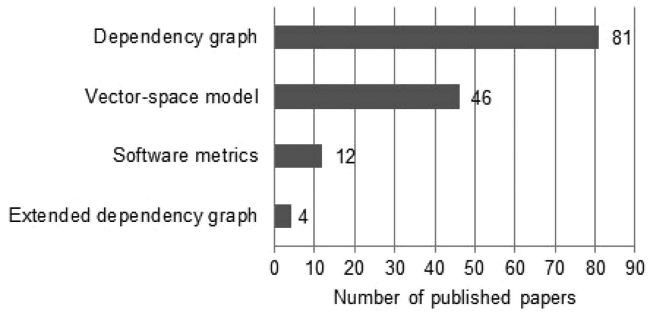

Fig. 13. Number of published papers versus the factbase extracted.

papers, respectively. In addition, many papers combined two sources for factbase extraction. In this respect, "source code and evolutionary information" is the most commonly used combination, accounting for approximately 6 percent (8/143) of the total publications.

The factbases are extracted from the sources in different forms. Analysis of the selected papers reveals that the "dependency graph," "vector-space model," "software metrics," and "extended dependency graph" are the most commonly used forms of the factbase. Fig. 13 shows the analysis results of the published papers. The figure also shows that approximately 57 percent (81/143), 32 percent (46/143), 8 percent (12/143), and 3 percent (4/143) of the total published papers considered "dependency graph", "vector-space model", "software metrics", and "extended dependency graph", respectively. The following points describe these factbase forms in detail:

- *Dependency graph:* It is a graph representation of the target software system. The nodes in the dependency graph represent software entities, whereas the edges represent the logical/static relationships between entities. In some cases, edges are weighted to denote the degree of dependency. Once the dependency graph is extracted, many characteristics of the target software system can be discovered, such as the independence degree of the software entities based on their relationships.
- *Vector-space model:* It is used to capture the relative importance of terms (e.g., class name, function name, object name, and variable name) in a document, e.g., class file, and program file. In the vector-space model, a document is represented by a vector of terms extracted from the document with associated weights (which can often be computed using the term frequency-inverse document frequency (TF-IDF) method [40]) representing the importance of the terms in the document and within the whole document collection (the target software system).
- *Software metrics:* They are quantitative measures that enable software engineers and managers to understand the target software system. The number of code lines per class, number of methods per class, and depth of inheritance level are possible metrics.
- *Extended dependency graph:* It is a graph that combines logical/static relationships and evolutionary relationships among software entities. The evolutionary relationships of the targeted software system represent the changes applied to its source files over time. Currently, many version control systems, such as the concurrent versions system (CVS) and Git, store these changes.

Notably, researchers typically use tools, which are mostly open-source Java programs, to perform factbase extraction. Table 8 presents all the tools that have been used in the selected studies, along with their links.

C. Filtering and Preprocessing

Filtering is a useful preprocessing phase in any clustering process to identify and remove unnecessary textual and non-textual information that has been extracted from comments and source codes. Textual information can be meaningless words, such as words with less than three characters, language keywords, or common English words that are not usually useful for a search [39]. Textual information can also be library classes or header files used in multiple modules and

TABLE 8
Factbase Extraction Tools

Tool	URL
Dependency Finder	http://depfind.sourceforge.net/
Class Dependency Analyzer (CDA)	http://www.dependency-analyzer.org/
ASM	http://asm.ow2.org
CScout	https://www.spinellis.gr/cscout/
Bunch	https://wiki.eecs.yorku.ca/project/cluster/tools
Sotograph	https://www.hello2morrow.com/products/sotograph
Structure101	https://structure101.com/
Understand 2.0	https://scitools.com/
JRipples	https://marketplace.eclipse.org/content/jripples
E-Quality	http://smart.cs.itu.edu.tr/tools/equality/
Javassist	http://www.javassist.org/
Visual Paradigm	https://www.visual-paradigm.com/
Doxygen	http://www.doxygen.nl/
Jdeps	https://docs.oracle.com/javase/8/docs/technotes/tools/unix/jdeps.html
Roslyn	https://github.com/dotnet/roslyn
Stan4J	http://www.stan4j.com/
PomWalker	https://github.com/raux/PomWalker
SrcML.NET	https://github.com/abb-iss/SrcML.NET

TABLE 9
Entities Selection

Clustering aim	Input entities	Entities abstraction level
Software comprehension	Functions and their call statements	Low-level (also called detailed level)
Architecture recovery	Software classes, packages, modules, and files	High-level (also called architectural level)

made available across the implementations of a programming language. These classes have to be eliminated; otherwise, they tend to group many classes in a single large cluster around them [38]. Nontextual information can be operators, symbols, special characters, and punctuations. Preprocessing can be implemented in the form of a normalization procedure. The attributes of the software entities can be a mixture of numerical and categorical data. Here, with the help of normalization, all the attributes are treated equally [100]. Preprocessing can also be conducted in the form of dimensional reduction, which is used to reduce the size of the input vectors [148]. Filtering is an essential phase to facilitate further processing and avoid the risk of decreasing the clustering quality [82].

D. Entity Selection

Software entities are the input to any software clustering technique. Therefore, entities to be clustered must be identified beforehand. The selection of entities depends mainly on the aim of the clustering technique [66]. Table 9 presents two examples in this respect.

Low-level entities represent the functionality of the target software system much more clearly than high-level entities. However, some software systems are large and contain enormous numbers of functions that make the use of functions in the clustering process inappropriate. In such cases, the clustering of high-level entities is preferable.

E. Feature Selection

Each software entity has a set of features. The features of a class entity, for example, can be broadly divided into two types [4], [93]:

- Nonformal features: These include file naming convention, class creation date, number of functions, number of variables, number of lines of codes, and comments. Nonformal features do not directly affect system behaviors. Also, they can be easily extracted, interpreted, and understood by humans.
- Formal features: These include class invocations, method invocations, and entity relationships. Relationships can be categorized into two types [66]:
  - Direct relationships: Represent an immediate connection between two entities e.g., if function f_1 calls another function f_2, then f_1 and f_2 are directly related.
  - Indirect relationships: Represent the proportion of common features that two entities share e.g., if functions f_1 and f_2 both call function f_3, then f_1 and f_2 are indirectly related to each other.

The formal features have a direct impact on system behaviors. For example, if a class changes its invocation from one class to another, then changes in the system's behavior should be expected. Extracting formal features, however, can be more complicated, as parsing rules need to be applied. This process becomes even more complicated when only partial information is available.

Feature selection aims to prepare the features of all software entities for the next step as a feature matrix (also called the entity-feature matrix) [83]. A feature matrix is a two-dimensional matrix where the rows represent software entities and the columns represent their features. The value of each cell in the matrix is either 0, which indicates the absence of a feature, or 1, which indicates the presence of a feature [137]. Table 10 presents a feature matrix of software containing 5 entities and 4 features. Feature F1 is present in all the entities, while feature F3 is absent in all the entities. Features F2 and F4 are present or absent in the entities.

Some clustering approaches apply weighting schemes (e.g., binary weighting, absolute weighting, and relative weighting) to the features to represent the significance of each one. Thus, the connection strength between a pair of entities can be calculated [4].

4.4.2 Similarity Calculation (RQ7)

Similarity measures are used with software module clustering to determine the most similar or dissimilar entities based on their features. Generally, entities are considered more similar if they share more common features [38]. After the computation of the similarity among all pairs of software entities, a similarity matrix can be generated for the next step. The most commonly used measures in the considered papers are shown in Fig. 14. Clearly, "Jaccard distance," "Cosine distance," and "euclidean distance" are the most common, accounting for approximately 12 percent (17/143), 8 percent (12/143), and 6 percent (8/143) of the total publications, respectively.

4.4.3 Cluster Creation (RQ8)

Here, a clustering algorithm must be applied to similar group entities of the target system based on specific features. The selection of a suitable algorithm to apply to a task is difficult. However, the authors in [163] introduced a method to help in this respect.

TABLE 10
A Simple Feature Matrix

	F1	F2	F3	F4
E1	1	0	0	1
E2	1	0	0	0
E3	1	1	0	1
E4	1	1	0	0
E5	1	1	0	1

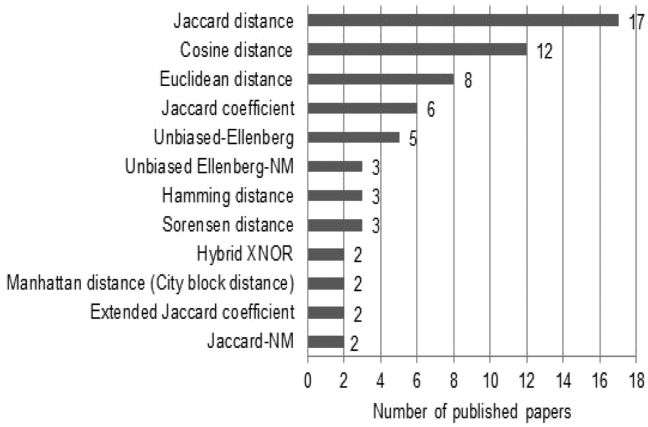

Fig. 14. Number of published papers versus similarity/dissimilarity measure.

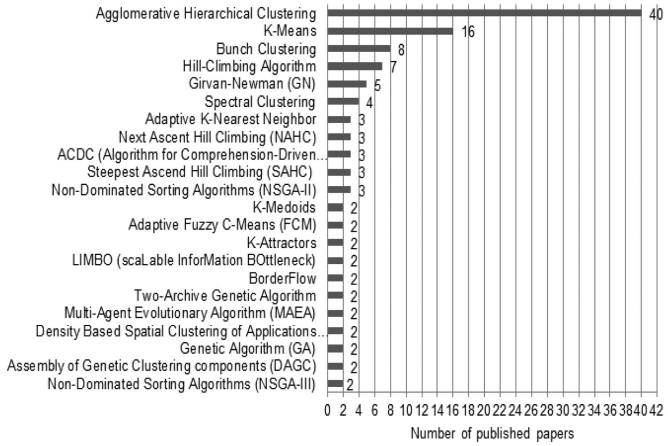

Fig. 15. Number of published papers versus clustering algorithm.

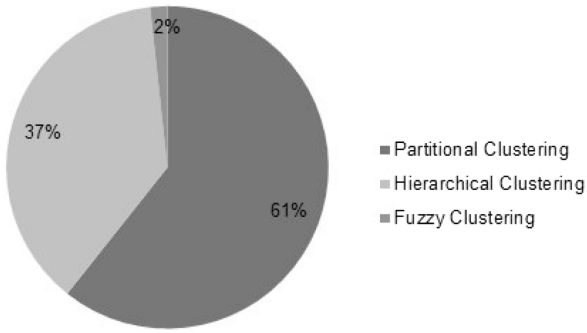

Fig. 16. Number of published papers versus clustering type.

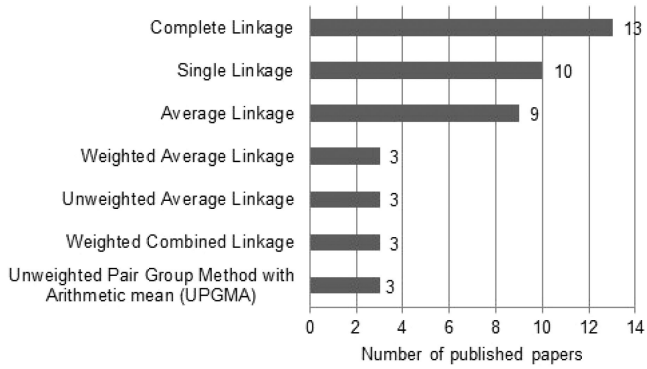

Fig. 17. Number of published papers versus distance computation methods in hierarchical clustering.

The analysis of the considered published papers in the study reveals the following general categories of clustering algorithms used in the literature:

- *Hard clustering:* In this type, a software entity can be in only one cluster and can be divided into two types:
  - Hierarchical clustering [3]: For a given set of n entities, hierarchical clustering algorithms are divided into two types:
    * Agglomerative (bottom-up) algorithms: The algorithms begin with n singletons (each of one entity) and merge them until a single cluster is obtained. The two most similar clusters are merged in each step of the process. Agglomerative algorithms provide different perspectives of software clustering: the earlier clustering iterations present a detailed view of the software architecture and the later ones present a high-level view [4].
    * Divisive (top-down) algorithms: These algorithms start with one cluster (containing all n entities) and split it until n clusters are obtained.
  - Partitional clustering: For a given set of n entities, this approach simply divides the set of entities into nonoverlapping clusters such that each entity is in exactly one cluster.

Notably, some studies have combined more than one clustering algorithm. For example, hierarchical and partitional clustering can be combined to achieve a common goal. This type of clustering is called cooperative clustering [84].

- *Soft clustering* (also called fuzzy clustering): This approach is based on fuzzy logic, where a software entity can be in one or more clusters. Thus, a probability or a membership degree of that entity in those clusters is assigned [39].

A deep analysis of the selected papers reveals that twenty-two clustering algorithms are the most commonly used, as shown in Fig. 15. Notably, "agglomerative hierarchical clustering" and "k-means" are the most commonly used clustering algorithms in the context of software module clustering, with percentages of approximately 28 percent (40/143) and 11 percent (16/143) of the total publications, respectively. Fig. 15 clearly shows this result.

Fig. 16 shows that "Partitional Clustering" is the most commonly used type of clustering, accounting for approximately 50 percent (71/143) followed by "Hierarchical Clustering", with approximately 31 percent (44/143), and "Fuzzy Clustering", with approximately 1 percent (2/143) of the total publications.

In hierarchical clustering, several methods can be used to compute the distance between clusters. An examination of the selected studies reveals the seven most commonly used methods. Fig. 17 shows this result. Clearly, the top three methods

TABLE 11
Results Visualization Tools

Tool	URL	Supported Languages
Prefuse	http://vis.stanford.edu/papers/prefuse	Java
Sotograph	http://www.hello2morrow.com/products/sotograph	C/C++, C#, Java, PHP, and Typescript
Graphviz	http://www.graphviz.org	C/C++, Java, PHP, Python, Ruby, Perl, Guile, and TCL

are "complete linkage", with approximately 9 percent (13/143), "single linkage", with approximately 7 percent (10/143), and "average linkage", with approximately 6 percent (9/143) of the total publications.

The analysis of the selected studies indicated the use of several different termination conditions. The clustering process can be terminated based on one of the following cutoff conditions [4], [116]:

- A good solution has been reached (e.g., when satisfying results are obtained by applying one or more clustering evaluation metrics).
- A maximum number of iterations has been reached.
- Improvement has not been found for a long time.
- All software entities are combined into a single cluster.
- A pre-specified termination condition has been met (e.g., a specific number of clusters has been formed).

4.4.4 Results Visualization of the Clustering (RQ9)

Visualization is used after the software module clustering process to view the clustering results as graphs, dendrograms, or distribution maps. Visualization of the results is also used to enable software engineers to efficiently and conveniently examine the clustering results. To achieve this goal, the visualization tools used in the literature ease and automate the visualization process. For example, graph visualization tools compile a graph description language and generate an image file outlining the subsystems as the output of software clustering [77]. Analysis of the selected studies reveals that researchers have not focused on the visualization of the clustering results. A potential reason for this situation is related to the software module clustering systems' end-users, which are the developers themselves or domain experts. However, Table 11 shows the visualization tools used in three of the studies.

4.4.5 Results Evaluation Metrics of the Clustering (RQ10)

The software clustering results should be evaluated in accordance with specific criteria to assess their quality [77]. Several methods in the literature can be used to assess the quality of software clustering algorithms. Although crucial, validation of the results produced by these algorithms is difficult. Fig. 18 shows the most commonly used evaluation methods in the considered papers. The following points summarize these evaluation methods:

- Results Compared with Other Results: The results obtained from the applied clustering algorithm are compared to those of previously published studies.
- Modularization Quality: MQ measures the cohesion and coupling of modules and is used to ensure that the clusters of a system are the cohesive modules in the clusters and the loose connections between clusters.
- MoJo Similarity Metrics: They measure the similarity between the partition produced by the software clustering process and the partition created by an expert (also called the expert decomposition, authoritative decomposition, benchmark decomposition, reference decomposition, gold decomposition, ground truth decomposition, or baseline decomposition) using MoJo metric and its family based on the number of Move and Join operations required to transform one decomposition into the other [77]. The similarity between the two partitions should be as high as possible.
- Execution Time: The clustering process should not require a very long time [77].
- Extremity: Also called Nonextremity cluster distribution (NED), it measures whether the clusters of the partition produced by the software clustering process have extreme values or not. A good clustering process should not produce a partition that includes (a) Enormous clusters – clusters with too many software entities would reduce the cohesion. (b) Singleton clusters (also called isolated clusters) or small clusters – clusters with one or too few software entities would increase the coupling of software [128].

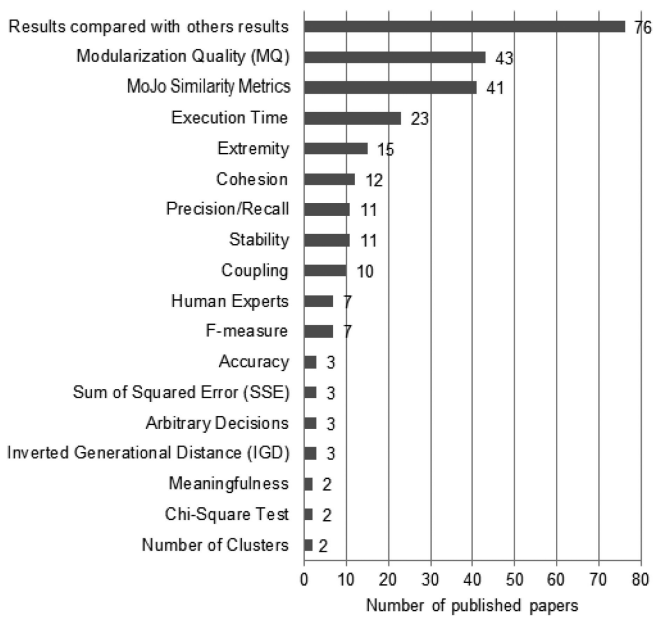

Fig. 18. Number of published papers versus clustering evaluation metric.

- Cohesion (intraconnectivity): It measures the density of connections among the software entities in a single cluster. High cohesion indicates good clustering, as highly dependent modules are grouped into the same cluster [140].
- Precision/Recall: Precision measures the percentage of entity pairs (in the same cluster) produced by the clustering algorithm that are also in different clusters in the expert decomposition. High precision indicates good clustering. Recall measures the percentage of entity pairs (in different clusters) in the expert decomposition that were also found by the clustering algorithm. High recall indicates good clustering [116].
- Stability: It measures whether the clustering process produces similar partitions in the case of small changes between successive versions of an evolving software system [81].
- Coupling (interconnectivity): It measures the density of connections among software entities in different clusters. Low coupling indicates good clustering, as the clusters are highly independent of each other [140].
- Human Experts: One or more domain experts manually evaluate the results obtained from the applied clustering algorithm.
- F-Measure: It measures the goodness or accuracy of the clustering methods by calculating the weighted average of recall and precision [145].
- Accuracy: The percentage of software entities that are correctly classified.
- Sum of Squared Error (SSE): The sum of the squared differences between entities in each cluster generated by a clustering algorithm and the entities of each cluster in the expert decomposition. Thus, SSE can be used as a measure of variation between clusters. For example, if the SSE between entities of two clusters is equal to 0, then they are matched perfectly and have no error. Smaller SSE is better, and obtaining clusters that minimize the SSE is always desirable [139].
- Arbitrary Decisions: An arbitrary decision is required when two or more entities hold the same similarity. Thus, the percentage of arbitrary decisions is employed to evaluate the clustering [83].
- Inverted Generational Distance (IGD): It is calculated as the average euclidean distance from each reference point (true Pareto front) to the nearest solution (Pareto front obtained by the algorithm) in the solution set. Here, the set of Pareto optimal solutions produced by all algorithms overall runs is used as the true Pareto front [30].
- Meaningfulness: Generated clusters should resemble the subsystems of the original system [77].
- Chi-Square Test: It determines if the entities of two different clusters are related in terms of some features.
- Number of Clusters: It measures the compactness of the clusters created during the software clustering process. A high number of clusters indicates that they are highly compact (cohesive), while a low number of clusters indicates that they are noncohesive [55].

Fig. 18 shows that "Results Compared with Others Results" is the most commonly used clustering evaluation method, accounting for approximately 53 percent (76/143) of all publications, followed by "Modularization Quality", 30 percent (43/143), and "MoJo Similarity Metrics", 16 percent (23/143). Many studies use more than one clustering evaluation method; thus, the results overlap.

Some clustering evaluation methods (e.g., MoJo Similarity Metrics and precision/recall) work only when expert decomposition is available as a comparison standard. Examination of the total publications reveals that a number of approaches have been presented for obtaining expert decomposition. A summary of these approaches is presented as follows:

- Domain Expert-Based Decomposition: The decomposition is performed by software domain experts. Domain experts are personnel with experience in software design and development [55]. The experts either evaluate the results produced by a clustering algorithm (e.g., if the results have a positive impact on the system's understandability) or they provide a clustering benchmark that can be compared with the results produced by a clustering algorithm [38]. Well-known drawbacks of this approach are as follows: (a) Experts may provide many valid ways to decompose a software system into meaningful subsystems [84]. (b) They might lead to poor decomposition if they did not fully understand the purpose of the clustering approach. (c) They may lead to poor decomposition if their experience and knowledge are not sufficient [114]. In addition, finding domain experts with suitable experience, especially on open-source software systems [37] and legacy systems [92], is difficult. (d) Software systems are constantly evolving and maintaining an up-to-date expert decomposition can be a tedious, error-prone, and time-consuming task [93].
- Factual Information-Based Decomposition: The decomposition is obtained using the current factual information of the targeted software system, e.g., the folder structure, the package structure, or the file structure.

  The advantage of using this approach is that its decompositions have good quality because they have been created by the original developers and domain experts [81].
- Original Developer-Based Decomposition: The decomposition is obtained by approaching the original developers of the target software system. The drawback of this approach is that the original developers are typically not available.
- Documentation-Based Decomposition: The decomposition is obtained using the key functional concepts extracted from the software architecture documentation.
- Maintenance Log-Based Decomposition: The decomposition is obtained by extracting information embedded in maintenance logs, which can be utilized to produce multiple decomposition stages of the target system.

Many clustering approaches use expert decomposition to evaluate the clustering results. Fig. 19 shows the distribution

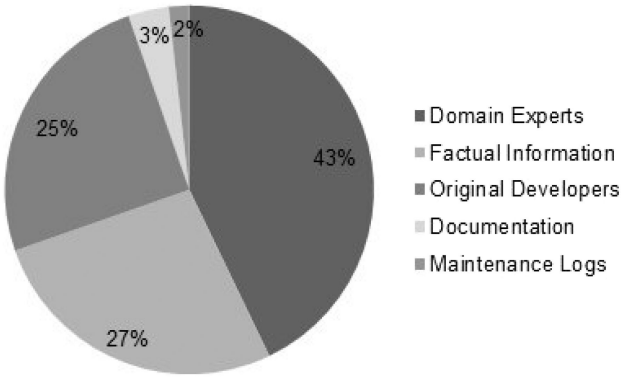

Fig. 19. Publication ratio by source of expert decomposition.

of the expert decomposition, where 24 approaches are based on domain experts, 15 are based on factual information, 14 are based on the original developers, two are based on the documentation, and one is based on maintenance logs.

Many factors affect the quality and efficacy of a clustering process. The following points summarize those factors [65]:

- The domain, type, size, and architecture of the targeted software system. For example, a clustering algorithm that is successful for a procedural program or a small software system might be unsuccessful for a large system developed in an object-oriented paradigm [87].
- The choice of factbase sources, preprocessing of data extracted from them and finding an appropriate representation of them [100].
- The entities, features, and relationships between them in terms of how well they have been selected.
- The type of similarity measures and algorithms used for the clustering process [100].
- Arbitrary decisions during the clustering process influence the quality and performance of clustering [82].

4.5 Potential Future Directions of Research (RQ11)

Answering RQ11 will help to identify possible research areas that may require further investigation. Based on the analysis of the considered papers, several potential directions of research were identified. The following points summarize and categorize these future research directions:

- Scalability: The clustering approach should handle a growing quantity of input without decreasing the clustering results' quality. This process can be achieved by performing clustering in parallel using multithreading programming techniques and hardware systems that have multicore processors. The consistency of clustering must be ensured, i.e., performing the clustering multiple times on the same dataset should produce the same results.
- Visualization: The display of the clustering results should be improved in cases where large outputs are presented on the screen simultaneously. This improvement can be achieved by applying filters to separate the results into different abstraction layers. As a result, clusters within a specific layer only can be viewed instead of going through all the displayed results. Thus, the user can better understand the results. The visualization tools should also automatically associate labels with the generated clusters.
- CASE tool: It is essential to consider making the whole clustering process a working tool (e.g., third-party libraries, plugins, and standalone applications) available for researchers, software engineers, and practitioners to perform further experimentation and collect feedback for future improvements.
- Targeted systems: The targeted or subject software systems used in the experiments are mostly Java-based open-source systems. It would be interesting to examine the use of non-open-source systems and systems written in other popular programming languages. Furthermore, considering different software application domains to keep the results generic and widely applicable requires further study. When the targeted systems are selected from various application domains, a specific clustering algorithm may exhibit various performance characteristics. Therefore, the various performance features of a particular clustering algorithm, when applied to multiple target systems, should be investigated.
- Entity features: Some software entities such as files and classes have a large number of different types of features (e.g., lines of code, executable statements, number of functions, and number of variables or objects) to be extracted. From a practical perspective, addressing this large number of various features is not preferable. Therefore, experimental studies to determine the number and type of features needed for better clustering results should be conducted. Further experiments may reveal a clustering approach that is more suitable for target systems that present specific characteristics (e.g., size or implemented functionality).
- Factbase sources: There are different sources for factbase extraction. Each has its own set of features and drawbacks. Thus, experimental studies that can determine the type of factbase sources that provide better clustering results should be performed. The impact of integrating different factbase sources on the overall clustering accuracy should be studied.
- Cooperative clustering: Only a few studies have combined more than one clustering algorithm to achieve a common goal. Currently, clusters that are produced by the first clustering algorithm are subdivided and reclustered by the second clustering algorithm, a situation that is often undesirable. Thus, more experiments should be performed, and tools to address the situation by not reclustering all the clusters that are part of the initial solution set should be developed.
- Selection of clustering algorithms: The selection of appropriate software clustering algorithms plays a significant role in producing meaningful clustering results. The authors in [163] proposed guidelines for selecting or rejecting a clustering algorithm for a given software system. However, there are no

TABLE 12
List of All Papers Included in the Study

ID#	Ref.	Paper Title	Year
1	[153]	A Double K-Clustering Approach for restructuring Distributed Object-Oriented software	2008
2	[114]	An algorithm of system decomposition based on laplace spectral graph partitioning technology	2008
3	[162]	Assessing software archives with evolutionary clusters	2008
4	[115]	Cluster analysis of Java dependency graphs	2008
5	[56]	Clustering based automatic refactorings identification	2008
6	[70]	Employing Clustering for Assisting Source Code Maintainability Evaluation according to ISO / IEC- 9126	2008
7	[116]	Evolution Strategy Based Automated Software Clustering Approach	2008
8	[3]	Object-Oriented Software Systems Restructuring through Clustering	2008
9	[110]	Refactoring module structure	2008
10	[117]	Using Cluster Analysis to Improve the Design of Component Interfaces	2008
11	[111]	An Approach for Software Architecture Refactoring Based on Clustering of Extended Component Dependency Graph	2009
12	[2]	Clustering for Monitoring Software Systems Maintainability Evolution	2009
13	[79]	Clustering of Software Systems Using New Hybrid Algorithms	2009
14	[5]	Comparison of Graph Clustering Algorithms for Recovering Software Architecture Module Views	2009
15	[112]	Decomposing object-oriented class modules using an agglomerative clustering technique	2009
16	[80]	Design pattern directed clustering for understanding open source code	2009
17	[154]	Restructuring Distributed Object-Oriented Software Using Hierarchical Clustering	2009
18	[137]	Software Clustering Using Dynamic Analysis and Static Dependencies	2009
19	[113]	Splitting a large software repository for easing future software evolution-an industrial experience report	2009
20	[118]	Towards automating class-splitting using betweenness clustering	2009
21	[146]	A Density Based Clustering approach for early detection of fault prone modules	2010
22	[37]	A Probabilistic Based Approach towards Software System Clustering	2010
23	[65]	Architecture Recovery Using Latent Semantic Indexing and K-Means: An Empirical Evaluation	2010
24	[81]	Evaluating the Impact of Software Evolution on Software Clustering	2010
25	[57]	Hierarchical clustering for adaptive refactorings identification	2010
26	[82]	Improved Hierarchical Clustering Algorithm for Software Architecture Recovery	2010
27	[83]	Object-oriented software architecture recovery using a new hybrid clustering algorithm	2010
28	[84]	On the Comparability of Software Clustering Algorithms	2010
29	[85]	Software architecture reconstruction: An approach based on combining graph clustering and partitioning	2010
30	[66]	Software refactoring at the function level using new Adaptive K-Nearest Neighbor algorithm	2010
31	[38]	Using the Kleinberg Algorithm and Vector Space Model for Software System Clustering	2010
32	[4]	Applying agglomerative hierarchical clustering algorithms to component identification for legacy systems	2011
33	[147]	Assessing Software Quality by Program Clustering and Defect Prediction	2011
34	[39]	Clustering and lexical information support for the recovery of design pattern in source code	2011
35	[27]	Clustering Dynamic Class Coupling Data to Measure Class Reusability Pattern	2011
36	[77]	Clustering software systems to identify subsystem structures using knowledgebase	2011
37	[40]	Clustering Support for Static Concept Location in Source Code	2011
38	[86]	Deriving High-level Abstractions from Legacy Software Using Example-driven Clustering	2011
39	[41]	Investigating the Use of Lexical Information for Software System Clustering	2011
40	[119]	JDeodorant: Identification and Application of Extract Class Refactorings	2011
41	[87]	Object oriented software clustering based on community structure	2011
42	[88]	Software Module Clustering as a Multi-Objective Search Problem	2011
43	[67]	Software refactoring at the package level using clustering techniques	2011
44	[89]	Solving software module clustering problem by evolutionary algorithms	2011
45	[68]	An analysis of the effects of composite objectives in multiobjective software module clustering	2012
46	[149]	Clustering Source Code Files to Predict Change Propagation during Software Maintenance	2012
47	[134]	Comparing and Combining Genetic and Clustering Algorithms for Software Component Identification from Object-Oriented Code	2012
48	[54]	Evaluating relationship categories for clustering object-oriented software systems	2012
49	[90]	Feature-gathering dependency-based software clustering using Dedication and Modularity	2012
50	[151]	Maintenance activities in object oriented software systems using K-means clustering technique: A review	2012
51	[7]	Program restructuring using agglomerative clustering technique based on binary features	2012
52	[92]	Reconstructing Architectural Views from Legacy Systems	2012
53	[91]	Software Clustering: Unifying Syntactic and Semantic Features	2012
54	[120]	Towards module-based automatic partitioning of Java applications	2012
55	[42]	Using fold-in and fold-out in the architecture recovery of software systems	2012
56	[121]	A new hierarchical clustering technique for restructuring software at the function level	2013
57	[43]	Class level fault prediction using software clustering	2013
58	[135]	Clustering Software Components for Component Reuse and Program Restructuring	2013
59	[50]	Cooperative clustering for software modularization	2013
60	[136]	Document Clustering Using Hybrid XOR Similarity Function for Efficient Software Component Reuse	2013
61	[75]	Efficient software clustering technique using an adaptive and preventive dendrogram cutting approach	2013
62	[93]	Evaluating software clustering algorithms in the context of program comprehension	2013
63	[94]	Mixed-Integer Linear Programming Formulations for the Software Clustering Problem	2013
64	[59]	Software architecture decomposition using adaptive K-nearest neighbor algorithm	2013
65	[60]	Software Architecture Decomposition Using Clustering Techniques	2013
66	[53]	Software Clustering Using Automated Feature Subset Selection	2013
67	[95]	Software module clustering using a hyper-heuristic based multi-objective genetic algorithm	2013
68	[96]	Using spectral clustering to automate identification and optimization of component structures	2013
69	[46]	A clustering-based model for class responsibility assignment problem in object-oriented analysis	2014
70	[144]	A Package Based Clustering for enhancing software defect prediction accuracy	2014
71	[28]	An empirical study of the sensitivity of quality indicator for software module clustering	2014
72	[124]	Assessing modularity using co-change clusters	2014
73	[72]	Combining Clustering and Classification for Software Quality Evaluation	2014
74	[97]	Cooperative based software clustering on dependency graphs	2014
75	[125]	High dimensional search-based software engineering: Finding Tradeoffs Among 15 Objectives for Automating Software Refactoring Using NSGA-III	2014

TABLE 12
(*Continued*)

ID#	Ref.	Paper Title	Year
76	[122]	Remodularization analysis using semantic clustering	2014
77	[123]	Software modularization using the modified firefly algorithm	2014
78	[138]	A clustering technique based on the specifications of software components	2015
79	[98]	A search-based approach to multi-view clustering of software systems	2015
80	[61]	Adaptive Clustering Techniques for Software Components and Architecture	2015
81	[139]	CCIC: Clustering analysis classes to identify software components	2015
82	[99]	Clustering Source Code Elements by Semantic Similarity Using Wikipedia	2015
83	[76]	Constrained agglomerative hierarchical software clustering with hard and soft constraints	2015
84	[126]	Multi-objective Module Clustering for Kate	2015
85	[145]	Object oriented based technique for software quality prediction through clustering and chi-square test	2015
86	[78]	Software Architecture Recovery using Genetic Black Hole Algorithm	2015
87	[6]	Software Clone Detection Using Clustering Approach	2015
88	[127]	Source Code Driven Enterprise Application Decomposition: Preliminary Evaluation	2015
89	[51]	A New Binary Similarity Measure Based on Integration of the Strengths of Existing Measures: Application to Software Clustering	2016
90	[62]	A similarity-based modularization quality measure for software module clustering problems	2016
91	[100]	A software component selection technique based on fuzzy clustering	2016
92	[73]	A tool to support software clustering using the software evolution information	2016
93	[157]	Automatic clustering of code changes	2016
94	[155]	Clones clustering using K-means	2016
95	[71]	Clustering Software Metric Values Extracted from C# Code for Maintainability Assessment	2016
96	[140]	Hyper-heuristic Approach for Multi-Objective Software Module Clustering	2016
97	[158]	Implementation and evaluation of optimized algorithm for software architectures analysis through unsupervised learning (clustering)	2016
98	[128]	Modularizing Software Systems using PSO optimized hierarchical clustering	2016
99	[74]	Software Evolution Information Driven Service-Oriented Software Clustering	2016
100	[44]	Weighing lexical information for software clustering in the context of architecture recovery	2016
101	[58]	A hierarchical clustering-based approach for software restructuring at the package level	2017
102	[63]	A multi-agent evolutionary algorithm for software module clustering problems	2017
103	[29]	A Particle Swarm Optimization-Based Heuristic for Software Module Clustering Problem	2017
104	[129]	Class Modularization Using Indirect Relationships	2017
105	[30]	FP-ABC: Fuzzy-Pareto dominance driven artificial bee colony algorithm for many-objective software module clustering	2017
106	[102]	Framework Information Based Java Software Architecture Recovery	2017
107	[55]	Improved binary similarity measures for software modularization	2017
108	[33]	Improving package structure of object-oriented software using multi-objective optimization and weighted class connections	2017
109	[69]	Large Neighborhood Search applied to the Software Module Clustering problem	2017
110	[101]	On the significance of relationship directions in clustering algorithms for reverse engineering	2017
111	[103]	Reconstructing and evolving software architectures using a coordinated clustering framework	2017
112	[31]	Semantic-based software clustering using hill climbing	2017
113	[32]	Software Remodularization by Estimating Structural and Conceptual Relations Among Classes and Using Hierarchical Clustering	2017
114	[159]	Using hierarchical agglomerative clustering to locate potential aspect interference	2017
115	[150]	A design structure matrix approach for measuring co-change-modularity of software products	2018
116	[64]	Analyzing the structure of Java software systems by weighted K-core decomposition	2018
117	[130]	Automatic Software Refactoring via Weighted Clustering in Method-Level Networks	2018
118	[141]	BCD: Decomposing Binary Code Into Components Using Graph-Based Clustering	2018
119	[160]	Discovering Program Topoi via Hierarchical Agglomerative Clustering	2018
120	[104]	Effectively incorporating expert knowledge in automated software remodularisation	2018
121	[161]	Functionality-Oriented Microservice Extraction Based on Execution Trace Clustering	2018
122	[34]	Improving Cohesion of a Software System by Performing Usage Pattern Based Clustering	2018
123	[148]	Improving Problem Identification via Automated Log Clustering using Dimensionality Reduction	2018
124	[142]	Improving reusability of software libraries through usage pattern mining	2018
125	[152]	In Object-Oriented Software Framework Improving Maintenance Exercises Through K-Means Clustering Approach	2018
126	[35]	Many-objective artificial bee colony algorithm for large-scale software module clustering problem	2018
127	[105]	Software Module Clustering Algorithm Using Probability Selection	2018
128	[106]	Software Module Clustering Based on the Fuzzy Adaptive Teaching Learning Based Optimization Algorithm	2018
129	[143]	A Mechanism for Automatically Summarizing Software Functionality from Source Code	2019
130	[36]	A multi-objective search based approach to identify reusable software components	2019
131	[45]	A new algorithm for software clustering considering the knowledge of dependency between artifacts in the source code	2019
132	[131]	A novel approach for automatic remodularization of software systems using extended ant colony optimization algorithm	2019
133	[132]	An efficient and stable method to cluster software modules using ant colony optimization algorithm	2019
134	[156]	Code similarity detection through control statement and program features	2019
135	[52]	euclidean space based hierarchical clusterers combinations: an application to software clustering	2019
136	[109]	Evaluating the effectiveness of multi-level greedy modularity clustering for software architecture recovery	2019
137	[133]	GUI-based software modularization through module clustering in edge computing based IoT environments	2019
138	[47]	Hybrid of genetic algorithm and krill herd for software clustering problem	2019
139	[8]	Multi-programming language software systems modularization	2019
140	[107]	Software Architecture Module-View Recovery Using Cluster Ensembles	2019
141	[108]	Software clusterings with vector semantics and the call graph	2019
142	[48]	Software Modularization by Combining Genetic and Hierarchical Algorithms	2019
143	[49]	Tarimliq: A new internal metric for software clustering analysis	2019

TABLE 13
List of Active Journals With Abbreviations

Acronym	Journal Full Name
Inf. Softw. Technol.	Information and Software Technology
J. Syst. Softw.	Journal of Systems and Software
IEEE Trans. Softw. Eng.	IEEE Transactions on Software Engineering
Procedia Comput. Sci.	Procedia Computer Science
Soft Comput.	Soft Computing
IET Softw.	IET Software
J. Software Maint. Evol. Res. Pract.	Journal of Software Maintenance and Evolution: Research and Practice
Inf. Sci.	Information Sciences
Wuhan Univ. J. Nat. Sci.	Wuhan University Journal of Natural Sciences
IEEE Trans. Reliab.	IEEE Transactions on Reliability
Future Gener. Comput. Syst.	Future Generation Computer Systems
Empir. Softw. Eng.	Empirical Software Engineering
Comput. Oper. Res.	Computers and Operations Research
Expert Syst. Appl.	Expert Systems with Applications
J. Supercomput.	Journal of Supercomputing
Comput. Lang. Syst. Struct.	Computer Languages, Systems, and Structures
Comput. Optim. Appl.	Computational Optimization and Applications
Arabian J. Sci. Eng.	Arabian Journal for Science and Engineering
Adv. Eng. Softw.	Advances in Engineering Software
J. King Saud Univ. Comp. Info. Sci.	Journal of King Saud University: Computer and information sciences
Front. Comput. Sci.	Frontiers of Computer Science
Form. Asp. Comput.	Formal Aspects of Computing
Cluster Comput.	Cluster Computing
J. Ambient Intell. Humaniz. Comput.	Journal of Ambient Intelligence and Humanized Computing
Comput. Electr. Eng.	Computers and Electrical Engineering
J. Comput. Lang.	Journal of Computer Languages
IEEE Access	IEEE Access
Front. Inf. Technol. Electron. Eng.	Frontiers of Information Technology and Electronic Engineering
Autom. Softw. Eng.	Automated Software Engineering
ACM SIGSOFT Software Eng. Notes	ACM SIGSOFT Software Engineering Notes

comprehensive methods for clustering algorithm selection. Thus, further research and experiments can be conducted to provide formal selection methods based on empirical evidence.

- Clustering with aesthetic aspects of the software design: The computational determination of the optimized cluster is often mechanistic, ignoring the fact that software is a creative artifact. Typically, cluster arrangement is determined via the grouping of nodes from dependency graphs. Having an optimized clustering of modules versus a meaningful set of clusters are two sides of the same coin with competing objectives. On the one hand, it is desirable to maximize cohesion and minimize coupling at any cost. On the other hand, one may also need to capture semantic as well as the essence and aesthetic aspects of the software design. Integrating natural language processing with deep learning and considering on other criteria (apart from simply grouping nodes from dependency graphs), such as utilizing the defined naming of modules and internal variables or other factbase sources as part of clustering arrangement criteria, would be a useful endeavor.
- Beyond clustering: The ultimate aim of software module clustering is to help software engineers apply the recommended clustering results to their software projects. However, do software engineers really adhere to the recommended clustering results? If yes, what will be the impact, cost, or effort of realizing the suggested results? No research has comprehensively addressed these issues. Thus, a thorough investigation in this respect may be a good step for further study.

5 Threats to Validity

Every literature mapping study has a number of threats that might affect its validity. In this study, several threats were eliminated by considering well-known recommendations and guidelines on conducting literature mapping studies as follows:

- Coverage of research questions: The threat here is that the research questions of this study may not cover all the aspects of the state-of-the-art research in software module clustering. To address this threat, all the authors of this study used brainstorming to define the desired set of research questions that cover the existing research in the area.
- Coverage of relevant studies: It cannot be guaranteed that all the relevant studies in software module clustering have been identified. Accordingly, different literature databases have been used, and a PICO method-based search string with various term synonyms (each author of the paper suggested different terms that lead to desired clustering concepts) has been applied to obtain the relevant research publications. However, some unidentified papers may

TABLE 14
List of Active Conferences With Abbreviations

Acronym	Conference Full Name
ICPC	International Conference on Program Comprehension (ICPC)
CSMR	European Conference on Software Maintenance and Reengineering (CSMR)
WCRE	Working Conference on Reverse Engineering (WCRE)
ASE	International Conference on Automated Software Engineering (ASE)
ICSM	International Conference on Software Maintenance (ICSM)
CSSE	International Conference on Computer Science and Software Engineering (CSSE)
MySEC	Malaysian Software Engineering Conference (MySEC)
CCECE	Canadian Conference on Electrical and Computer Engineering (CCECE)
COMPSAC	Annual Computer Software and Applications Conference (COMPSAC)
SETN	Hellenic Conference on Artificial Intelligence (SETN)
GECCO	International Conference on Genetic and Evolutionary Computation (GECCO)
KBEI	International Conference on Knowledge Based Engineering and Innovation (KBEI)

remain. To address this issue, the snowballing method was intensively applied to reduce the possibility of missing important related papers.

- Paper inclusion/exclusion criteria: Application of the criteria can suffer from single-author judgment and personal bias. To address this issue, each paper was included or excluded for this study only after the authors reached a consensus.
- Accuracy of data extraction: Data extraction can suffer from the single-author experience. Accordingly, each author individually performed the data extraction process, and the outcomes of all authors were compared in an online meeting. In the meeting, all authors discussed differences between the outcomes until a final and agreed consensus was reached. Automatic filtering provided by Microsoft Excel was also used to ensure the accuracy of the data extraction process.
- Reproducibility of the study: The issue here is whether other researchers can perform this study with similar results. Accordingly, all the steps followed and performed in this study were reported in the research methodology (see Section 3).

6 Conclusion

This paper systematically reports the state-of-the-art empirical contributions in software module clustering. Thus, to ascertain the recent clustering applications in software engineering, the algorithms and tools used to enable the software module clustering process were identified. A total of 143 papers from popular literature databases published in the area of software module clustering from 2008-2019 were selected for this study. The published papers were a combination of works from conferences, journals, symposiums, and workshops. However, most of the published papers were from conferences. From different perspectives and based on several identified RQs, the selected studies were thoroughly reviewed and analyzed. The findings were in different categories. For instance, statistics on the published studies, their publication venues, active authors, and countries were reported. Then, software module clustering applications were categorized. All the algorithms, tools, target software systems, evaluations, and metrics that enabled the clustering process were briefly discussed. Finally, as there are many research studies on software module clustering, novice researchers are likely to experience difficulties in addressing different aspects of the area. Therefore, we propose this analysis study as a primary reference to simplify the process of finding the most relevant information.

Acknowledgments

This work has been partially funded by the Knowledge Foundation of Sweden (KKS) through the Synergy Project AIDA - A Holistic AI-driven Networking and Processing Framework for Industrial IoT (Rek:20200067).

References

[1] A. Adolfsson, M. Ackerman, and N. C. Brownstein, "To cluster, or not to cluster: An analysis of clusterability methods," *Pattern Recognit.*, vol. 88, pp. 13–26, 2019.
[2] P. Antonellis *et al.*, "Clustering for monitoring software systems maintainability evolution," *Electron. Notes Theor. Comput. Sci.*, vol. 233, pp. 43–57, 2009.
[3] G. Şerban and I.-G. Czibula, "Object-oriented software systems restructuring through clustering," in *Proc. Int. Conf. Artif. Intell. Soft Comput.*, 2008, pp. 693–704.
[4] J. Feng and H. Seok, "Applying agglomerative hierarchical clustering algorithms to component identification for legacy systems," *Inf. Softw. Technol.*, vol. 53, no. 6, pp. 601–614, 2011.
[5] R. A. Bittencourt and D. D. S. Guerrero, "Comparison of graph clustering algorithms for recovering software architecture module views," in *Proc. 13th Eur. Conf. Softw. Maintenance Reeng.*, 2009, pp. 251–254.
[6] B. Joshi, P. Budhathoki, W. L. Woon, and D. Svetinovic, "Software clone detection using clustering approach," in *Proc. Int. Conf. Neural Inf. Process.*, 2015, pp. 520–527.
[7] R. Naseem, A. Ahmed, S. U. Khan, M. Saqib, and M. Habib, "Program restructuring using agglomerative clustering technique based on binary features," in *Proc. Int. Conf. Emerg. Technol.*, 2012, pp. 1–6.
[8] M. Kargar, A. Isazadeh, and H. Izadkhah, "Multi-programming language software systems modularization," *Comput. Electr. Eng.*, vol. 80, pp. 1–22, 2019.
[9] Y. Yu, J. Lu, J. Fernandez-Ramil, and P. Yuan, "Comparing web services with other software components," in *Proc. IEEE Int. Conf. Web Services*, 2007, pp. 388–397.
[10] N. Arunachalam, A. Amuthan, C. Kavya, M. Sharmilla, K. Ushanandhini, and M. Shanmughapriya, "A survey on web service clustering," in *Proc. Int. Conf. Comput. Power Energy Inf. Commun.*, 2017, pp. 247–252.
[11] M. Shtern and V. Tzerpos, "Clustering methodologies for software engineering," *Advances Softw. Eng.*, vol. 2012, pp. 1–18, 2012.

[12] V. Singh, "Software module clustering using metaheuristic search techniques: A survey," in *Proc. 3rd Int. Conf. Comput. Sustain. Global Develop.*, 2016, pp. 2764–2767.
[13] F. Morsali and M. R. Keyvanpour, "Search-based software module clustering techniques: A review article," in *Proc. IEEE 4th Int. Conf. Knowl.-Based Eng. Innov.*, 2017, pp. 0977–0983.
[14] M. Harman, "The current state and future of search based software engineering," in *Proc. Future Softw. Eng.*, 2007, pp. 342–357.
[15] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based software engineering: Trends, techniques and applications," *ACM Comput. Surv.*, vol. 45, no. 1, pp. 11:1–11:61, 2012.
[16] A. Ramírez, J. R. Romero, and S. Ventura, "A survey of many-objective optimisation in search-based software engineering," *J. Syst. Softw.*, vol. 149, pp. 382–395, 2019.
[17] K. Petersen, S. Vakkalanka, and L. Kuzniarz, "Guidelines for conducting systematic mapping studies in software engineering: An update," *Inf. Softw. Technol.*, vol. 64, pp. 1–18, 2015.
[18] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," version 2.3., EBSE Technical Report EBSE- 2007–01, Software Engineering Group, School of Computer Science and Mathematics, Keele University, U.K. and Department of Computer Science, University of Durham, 2007.
[19] O. Pedreira, F. García, N. Brisaboa, and M. Piattini, "Gamification in software engineering – A systematic mapping," *Inf. Softw. Technol.*, vol. 57, pp. 157–168, 2015.
[20] M. Usman, R. Britto, J. Börstler, and E. Mendes, "Taxonomies in software engineering: A Systematic mapping study and a revised taxonomy development method," *Inf. Softw. Technol.*, vol. 85, pp. 43–59, 2017.
[21] B. S. Ahmed, K. Z. Zamli, W. Afzal, and M. Bures, "Constrained interaction testing: A systematic literature study," *IEEE Access*, vol. 5, pp. 25 706–25 730, 2017.
[22] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil, "Lessons from applying the systematic literature review process within the software engineering domain," *J. Syst. Softw.*, vol. 80, no. 4, pp. 571–583, 2007.
[23] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proc. 18th Int. Conf. Eval. Assessment Softw. Eng.*, 2014, pp. 1–10.
[24] Google. Google trends, 2020. Accessed: Mar. 10, 2020. [Online]. Available: https://trends.google.com/trends/?geo=US
[25] O. Pedreira, F. García, N. Brisaboa, and M. Piattini, "Gamification in software engineering–A systematic mapping," *Inf. Softw. Technol.*, vol. 57, pp. 157–168, 2015.
[26] R. E. Lopez-herrejon, L. Linsbauer, and A. Egyed, "A systematic mapping study of search-based software engineering for software product lines," *Inf. Softw. Technol.*, vol. 61, pp. 33–51, 2015.
[27] A. Parashar and J. K. Chhabra, "Clustering dynamic class coupling data to measure class reusability pattern," in *Proc. Int. Conf. High Perform. Architecture Grid Comput.*, 2011, pp. 126–130.
[28] Amarjeet and J. K. Chhabra, "An empirical study of the sensitivity of quality indicator for software module clustering," in *Proc. 7th Int. Conf. Contemp. Comput.*, 2014, pp. 206–211.
[29] A. Prajapati and J. K. Chhabra, "A particle swarm optimization-based heuristic for software module clustering problem," *Arabian J. Sci. Eng.*, vol. 43, no. 12, pp. 7083–7094, 2017.
[30] Amarjeet and J. K. Chhabra, "FP-ABC: Fuzzy-Pareto dominance driven artificial bee colony algorithm for many-objective software module clustering," *Comput. Lang. Syst. Structures*, vol. 51, pp. 1–21, 2017.
[31] M. Kargar, A. Isazadeh, and H. Izadkhah, "Semantic-based software clustering using hill climbing," in *Proc. Int. Symp. Comput. Sci. Softw. Eng. Conf.*, 2017, pp. 55–60.
[32] A. Rathee and J. K. Chhabra, "Software remodularization by estimating structural and conceptual relations among classes and using hierarchical clustering," in *Proc. Int. Conf. Adv. Informat. Comput. Res.*, 2017, pp. 94–106.
[33] Amarjeet and J. K. Chhabra, "Improving package structure of object-oriented software using multi-objective optimization and weighted class connections," *J. King Saud Univ. - Comput. Inf. Sci.*, vol. 29, no. 3, pp. 349–364, 2017.
[34] A. Rathee and J. Kumar, "Improving cohesion of a software system by performing usage pattern based clustering," *Procedia Comput. Sci.*, vol. 125, pp. 740–746, 2018.
[35] Amarjeet and J. K. Chhabra, "Many-objective artificial bee colony algorithm for large-scale software module clustering problem," *Soft Comput.*, vol. 22, no. 19, pp. 6341–6361, 2018.
[36] A. Rathee and J. K. Chhabra, "A multi-objective search based approach to identify reusable software components," *J. Comput. Lang.*, vol. 52, pp. 26–43, 2019.
[37] A. Corazza, S. Di Martino, and G. Scanniello, "A probabilistic based approach towards software system clustering," in *Proc. 14th Eur. Conf. Softw. Maintenance Reeng.*, 2010, pp. 88–96.
[38] G. Scanniello, A. D'Amico, C. D'Amico, and T. D'Amico, "Using the kleinberg algorithm and vector space model for software system clustering," in *Proc. IEEE 18th Int. Conf. Program Comprehension*, 2010, pp. 180–189.
[39] S. Romano, G. Scanniello, M. Risi, and C. Gravino, "Clustering and lexical information support for the recovery of design pattern in source code," in *Proc. 27th IEEE Int. Conf. Softw. Maintenance*, 2011, pp. 500–503.
[40] G. Scanniello and A. Marcus, "Clustering support for static concept location in source code," in *Proc. IEEE 19th Int. Conf. Program Comprehension*, 2011, pp. 1–10.
[41] A. Corazza, S. D. Martino, V. Maggio, and G. Scanniello, "Investigating the use of lexical information for software system clustering," in *Proc. 15th Eur. Conf. Softw. Maintenance Reeng.*, 2011, pp. 35–44.
[42] M. Risi, G. Scanniello, and G. Tortora, "Using fold-in and fold-out in the architecture recovery of software systems," *Formal Aspects Comput.*, vol. 24, no. 3, pp. 307–330, 2012.
[43] G. Scanniello, C. Gravino, A. Marcus, and T. Menzies, "Class level fault prediction using software clustering," in *Proc. 28th IEEE/ACM Int. Conf. Autom. Softw. Eng.*, 2013, pp. 640–645.
[44] A. Corazza, S. Di Martino, V. Maggio, and G. Scanniello, "Weighing lexical information for software clustering in the context of architecture recovery," *Empir. Softw. Eng.*, vol. 21, no. 1, pp. 72–103, 2016.
[45] S. Mohammadi and H. Izadkhah, "A new algorithm for software clustering considering the knowledge of dependency between artifacts in the source code," *Inf. Softw. Technol.*, vol. 105, pp. 252–256, 2019.
[46] H. Masoud and S. Jalili, "A clustering-based model for class responsibility assignment problem in object-oriented analysis," *J. Syst. Softw.*, vol. 93, pp. 110–131, 2014.
[47] M. Akbari and H. Izadkhah, "Hybrid of genetic algorithm and krill herd for software clustering problem," in *Proc. IEEE 5th Conf. Knowl. Based Eng. Innov.*, 2019, pp. 565–570.
[48] A. H. Farajpour Tabrizi and H. Izadkhah, "Software modularization by combining genetic and hierarchical algorithms," in *Proc. 5th Conf. Knowl. Based Eng. Innov.*, 2019, pp. 454–459.
[49] M. Kargar, H. Izadkhah, and A. Isazadeh, "Tarimliq: A new internal metric for software clustering analysis," in *Proc. 27th Iranian Conf. Electr. Eng.*, 2019, pp. 1879–1883.
[50] R. Naseem, O. Maqbool, and S. Muhammad, "Cooperative clustering for software modularization," *J. Syst. Softw.*, vol. 86, no. 8, pp. 2045–2062, 2013.
[51] R. Naseem and M. M. Deris, "A new binary similarity measure based on integration of the strengths of existing measures: Application to software clustering," in *Proc. Int. Conf. Soft Comput. Data Mining*, 2016, pp. 304–315.
[52] R. Naseem, M. M. Deris, O. Maqbool, and S. Shahzad, "Euclidean space based hierarchical clusterers combinations: An application to software clustering," *Cluster Comput.*, vol. 22, pp. 7287–7311, 2019.
[53] Z. Shah, R. Naseem, M. A. Orgun, A. Mahmood, and S. Shahzad, "Software clustering using automated feature subset selection," in *Proc. 9th Int. Conf. Adv. Data Mining Appl.*, 2013, pp. 47–58.
[54] S. Muhammad, O. Maqbool, and A. Abbasi, "Evaluating relationship categories for clustering object-oriented software systems," *IET Softw.*, vol. 6, no. 3, pp. 260–274, 2012.
[55] R. Naseem, M. B. M. Deris, O. Maqbool, J. Peng Li, S. Shahzad, and H. Shah, "Improved binary similarity measures for software modularization," *Front. Inf. Technol. Electron. Eng.*, vol. 18, pp. 1082–1107, 2017.
[56] I. G. Czibula and G. Czibula, "Clustering based automatic refactorings identification," in *Proc. 10th Int. Symp. Symbolic Numeric Algorithms Sci. Comput.*, 2008, pp. 253–256.
[57] I. Czibula and G. Czibula, "Hierarchical clustering for adaptive refactorings identification," in *Proc. IEEE Int. Conf. Autom. Quality Testing Robot.*, 2010, pp. 1–6.

[58] Z. Marian, I. G. Czibula, and G. Czibula, "A hierarchical clustering-based approach for software restructuring at the package level," in *Proc. 19th Int. Symp. Symbolic Numeric Algorithms Sci. Comput.*, 2017, pp. 239–246.

[59] A. Alkhalid, C.-H. Lung, and S. Ajila, "Software architecture decomposition using adaptive K-nearest neighbor algorithm," in *Proc. 26th IEEE Canadian Conf. Electr. Comput. Eng.*, 2013, pp. 1–4.

[60] A. Alkhalid, C.-H. Lung, D. Liu, and S. Ajila, "Software architecture decomposition using clustering techniques," in *Proc. IEEE 37th Annu. Comput. Softw. Appl. Conf.*, 2013, pp. 806–811.

[61] D. Liu, C.-H. Lung, and S. A. Ajila, "Adaptive clustering techniques for software components and architecture," in *Proc. IEEE 39th Annu. Comput. Softw. Appl. Conf.*, 2015, pp. 460–465.

[62] J. Huang and J. Liu, "A similarity-based modularization quality measure for software module clustering problems," *Inf. Sci.*, vol. 342, pp. 96–110, 2016.

[63] J. Huang, J. Liu, and X. Yao, "A multi-agent evolutionary algorithm for software module clustering problems," *Soft Comput.*, vol. 21, no. 12, pp. 3415–3428, 2017.

[64] W. Pan, B. Li, J. Liu, Y. Ma, and B. Hu, "Analyzing the structure of Java software systems by weighted k-core decomposition," *Future Gener. Comput. Syst.*, vol. 83, pp. 431–444, 2018.

[65] M. Risi, G. Scanniello, and G. Tortora, "Architecture recovery using latent semantic indexing and k-means: An empirical evaluation," in *Proc. 8th IEEE Int. Conf. Softw. Eng. Formal Methods*, 2010, pp. 103–112.

[66] A. Alkhalid, M. Alshayeb, and S. Mahmoud, "Software refactoring at the function level using new Adaptive K-Nearest Neighbor algorithm," *Advances Eng. Softw.*, vol. 41, no. 10/11, pp. 1160–1178, 2010.

[67] A. Alkhalid, M. Alshayeb, and S. Mahmoud, "Software refactoring at the package level using clustering techniques," *IET Softw.*, vol. 5, no. 3, pp. 274–286, 2011.

[68] M. D. O. Barros, "An analysis of the effects of composite objectives in multiobjective software module clustering," in *Proc. 14th Int. Conf. Genetic Evol. Comput. Conf.*, 2012, pp. 1205–1212.

[69] M. C. Monçores, A. C. Alvim, and M. O. Barros, "Large neighborhood search applied to the software module clustering problem," *Comput. Operations Res.*, vol. 91, pp. 92–111, 2017.

[70] P. Antonellis *et al.*, "Employing clustering for assisting source code maintainability evaluation according to ISO / IEC- 9126," in *Proc. Artif. Intell. Techn. Softw. Eng. Workshop*, 2008, pp. 1–5.

[71] S. Arshad and C. Tjortjis, "Clustering software metric values extracted from C# code for maintainability assessment," in *Proc. 9th Hellenic Conf. Artif. Intell.*, 2016, pp. 1–4.

[72] D. Papas and C. Tjortjis, "Combining clustering and classification for software quality evaluation," in *Proc. Hellenic Conf. Artif. Intell.*, 2014, pp. 273–286.

[73] L. Zhong, L. Xue, N. Zhang, J. Xia, and J. Chen, "A tool to support software clustering using the software evolution information," in *Proc. 7th IEEE Int. Conf. Softw. Eng. Service Sci.*, 2016, pp. 304–307.

[74] L. Zhong, J. He, N. Zhang, P. Zhang, and J. Xia, "Software evolution information driven service-oriented software clustering," in *Proc. IEEE Int. Congress Big Data*, 2016, pp. 493–500.

[75] C. Y. Chong, S. P. Lee, and T. C. Ling, "Efficient software clustering technique using an adaptive and preventive dendrogram cutting approach," *Inf. Softw. Technol.*, vol. 55, no. 11, pp. 1994–2012, 2013.

[76] C. Y. Chong and S. P. Lee, "Constrained agglomerative hierarchical software clustering with hard and soft constraints," *Proc. 10th Int. Conf. Eval. Novel Approaches Softw. Eng.*, 2015, pp. 177–188.

[77] M. N. Adnan, M. R. Islam, and S. Hossain, "Clustering software systems to identify subsystem structures using knowledgebase," in *Proc. Malaysian Conf. Softw. Eng.*, 2011, pp. 445–450.

[78] K. Jeet and R. Dhir, "Software architecture recovery using genetic black hole algorithm," *ACM SIGSOFT Softw. Eng. Notes*, vol. 40, no. 1, pp. 1–5, 2015.

[79] A. S. Mamaghani and M. R. Meybodi, "Clustering of software systems using new hybrid algorithms," in *Proc. 9th IEEE Int. Conf. Comput. Inf. Technol.*, 2009, pp. 20–25.

[80] Z. Han, L. Wang, L. Yu, X. Chen, J. Zhao, and X. Li, "Design pattern directed clustering for understanding open source code," in *Proc. IEEE 17th Int. Conf. Program Comprehension*, 2009, pp. 295–296.

[81] F. Beck and S. Diehl, "Evaluating the impact of software evolution on software clustering," in *Proc. 17th Work. Conf. Reverse Eng.*, 2010, pp. 99–108.

[82] Y. Wang, P. Liu, H. Guo, H. Li, and X. Chen, "Improved hierarchical clustering algorithm for software architecture recovery," in *Proc. Int. Conf. Intell. Comput. Cognitive Informat.*, 2010, pp. 247–250.

[83] Q. Zhang, D. Qiu, Q. Tian, and L. Sun, "Object-oriented software architecture recovery using a new hybrid clustering algorithm," in *Proc. 7th Int. Conf. Fuzzy Syst. Knowl. Discov.*, 2010, pp. 2546–2550.

[84] M. Shtern and V. Tzerpos, "On the comparability of software clustering algorithms," in *Proc. IEEE 18th Int. Conf. Program Comprehension*, 2010, pp. 64–67.

[85] I. Sora, G. Glodean, and M. Gligor, "Software architecture reconstruction: An approach based on combining graph clustering and partitioning," in *Proc. Int. Joint Conf. Comput. Cybern. Tech. Informat.*, 2010, pp. 259–264.

[86] M. Faunes, M. Kessentini, and H. Sahraoui, "Deriving high-level abstractions from legacy software using example-driven clustering," in *Proc. Conf. Center Adv. Studies Collaborative Res.*, 2011, pp. 188–199.

[87] U. Erdemir, U. Tekin, and F. Buzluca, "Object oriented software clustering based on community structure," in *Proc. 18th Asia-Pacific Softw. Eng. Conf.*, 2011, pp. 315–321.

[88] K. Praditwong, 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, Mar./Apr. 2011.

[89] K. Praditwong, "Solving software module clustering problem by evolutionary algorithms," in *Proc. 8th Int. Joint Conf. Comput. Sci. Softw. Eng.*, 2011, pp. 154–159.

[90] K. Kobayashi, M. Kamimura, K. Kato, K. Yano, and A. Matsuo, "Feature-gathering dependency-based software clustering using dedication and modularity," in *Proc. 28th IEEE Int. Conf. Softw. Maintenance*, 2012, pp. 462–471.

[91] J. Misra, K. Annervaz, V. Kaulgud, S. Sengupta, and G. Titus, "Software clustering: Unifying syntactic and semantic features," in *Proc. 19th Work. Conf. Reverse Eng.*, 2012, pp. 113–122.

[92] G. E. Boussaidi, A. B. Belle, S. Vaucher, and H. Mili, "Reconstructing architectural views from legacy systems," in *Proc. 19th Work. Conf. Reverse Eng.*, 2012, pp. 345–354.

[93] A. Mahmoud and N. Niu, "Evaluating software clustering algorithms in the context of program comprehension," in *Proc. 21st Int. Conf. Program Comprehension*, 2013, pp. 162–171.

[94] V. Köhler, M. Fampa, and O. Araújo, "Mixed-integer linear programming formulations for the software clustering problem," *Comput. Optim. Appl.*, vol. 55, no. 1, pp. 113–135, 2013.

[95] A. C. Kumari, K. Srinivas, and M. P. Gupta, "Software module clustering using a hyper-heuristic based multi-objective genetic algorithm," in *Proc. 3rd IEEE Int. Advance Comput. Conf.*, 2013, pp. 813–818.

[96] C. Deiters, A. Rausch, and M. Schindler, "Using spectral clustering to automate identification and optimization of component structures," in *Proc. 2nd Int. Workshop Realizing Artif. Intell. Synergies Softw. Eng.*, 2013, pp. 14–20.

[97] A. Ibrahim, D. Rayside, and R. Kashef, "Cooperative based software clustering on dependency graphs," in *Proc. IEEE 27th Canadian Conf. Electr. Comput. Eng.*, 2014, pp. 1–6.

[98] A. M. Saeidi, J. Hage, R. Khadka, and S. Jansen, "A search-based approach to multi-view clustering of software systems," in *Proc. IEEE 22nd Int. Conf. Softw. Anal. Evol. Reeng.*, 2015, pp. 429–438.

[99] M. Schindler, O. Fox, and A. Rausch, "Clustering source code elements by semantic similarity using Wikipedia," in *Proc. IEEE/ACM 4th Int. Workshop Realizing Artif. Intell. Synergies Softw. Eng.*, 2015, pp. 13–18.

[100] J. Kaur and P. Tomar, "A software component selection technique based on fuzzy clustering," in *Proc. 1st India Int. Conf. Inf. Process.*, 2016, pp. 1–5.

[101] D. Jensen and A. Lundkvist, "On the significance of relationship directions in clustering algorithms for reverse engineering," in *Proc. Symp. Appl. Comput.*, 2017, pp. 1239–1244.

[102] X. Li, L. Zhang, and N. Ge, "Framework information based Java software architecture recovery," in *Proc. 24th Asia-Pacific Softw. Eng. Conf. Workshops*, 2017, pp. 114–120.

[103] S. M. Naim, K. Damevski, and M. S. Hossain, "Reconstructing and evolving software architectures using a coordinated clustering framework," *Autom. Softw. Eng.*, vol. 24, pp. 543–572, 2017.

[104] M. Hall, N. Walkinshaw, and P. McMinn, "Effectively incorporating expert knowledge in automated software remodularisation," *IEEE Trans. Softw. Eng.*, vol. 44, no. 7, pp. 613–630, Jul. 2018.

[105] J. Sun and B. Ling, "Software module clustering algorithm using probability selection," *Wuhan Univ. J. Natural Sci.*, vol. 23, no. 2, pp. 93–102, 2018.
[106] K. Z. Zamli, F. Din, N. Ramli, and B. S. Ahmed, "Software module clustering based on the fuzzy adaptive teaching learning based optimization algorithm," in *Proc. Intell. Interactive Comput.*, 2018, pp. 167–177.
[107] C. Cho, K.-S. Lee, M. Lee, and C.-G. Lee, "Software architecture module-view recovery using cluster ensembles," *IEEE Access*, vol. 7, pp. 72 872–72 884, 2019.
[108] M. Papachristou, "Software clusterings with vector semantics and the call graph," in *Proc. 27th ACM Joint Meeting Eur. Softw. Eng. Conf. and Symp. Found. Softw. Eng.*, 2019, pp. 1184–1186.
[109] H. Sözer, "Evaluating the effectiveness of multi-level greedy modularity clustering for software architecture recovery," in *Proc. Eur. Conf. Softw. Architecture*, 2019, pp. 71–87.
[110] L. Lövei *et al.*, "Refactoring module structure," in *Proc. 7th ACM SIGPLAN Workshop ERLANG*, 2008, pp. 83–89.
[111] L.-H. Zhong, L. Xu, M.-S. Ye, Y. Zheng, and B. Xie, "An approach for software architecture refactoring based on clustering of extended component dependency graph," in *Proc. Int. Conf. Comput. Intell. Softw. Eng.*, 2009, pp. 1–4.
[112] M. Fokaefs, N. Tsantalis, A. Chatzigeorgiou, and J. Sander, "Decomposing object-oriented class modules using an agglomerative clustering technique," in *Proc. IEEE Int. Conf. Softw. Maintenance*, 2009, pp. 93–101.
[113] M. Glorie, A. Zaidman, A. van Deursen, and L. Hofland, "Splitting a large software repository for easing future software evolution-an industrial experience report," *J. Softw. Maintenance Evol.: Res. Pract.*, vol. 21, no. 2, pp. 113–141, 2009.
[114] Y. Liu, G. Guo, and J. Qi, "An algorithm of system decomposition based on laplace spectral graph partitioning technology," in *Proc. Int. Conf. Comput. Sci. Softw. Eng.*, 2008, pp. 85–89.
[115] J. Dietrich, V. Yakovlev, C. McCartin, G. Jenson, and M. Duchrow, "Cluster analysis of Java dependency graphs," in *Proc. 4th ACM Symp. Softw. Visuallization*, 2008, pp. 91–94.
[116] B. Khan, S. Sohail, and M. Y. Javed, "Evolution strategy based automated software clustering approach," in *Proc. Adv. Softw. Eng. Appl.*, 2008, pp. 27–34.
[117] R. Adnan, B. Graaf, A. van Deursen, and J. Zonneveld, "Using cluster analysis to improve the design of component interfaces," in *Proc. 23rd IEEE/ACM Int. Conf. Autom. Softw. Eng.*, 2008, pp. 383–386.
[118] K. Cassell, P. Andreae, L. Groves, and J. Noble, "Towards automating class-splitting using betweenness clustering," in *Proc. 24th IEEE/ACM Int. Conf. Autom. Softw. Eng.*, 2009, pp. 595–599.
[119] M. Fokaefs, N. Tsantalis, E. Stroulia, and A. Chatzigeorgiou, "JDeodorant: Identification and application of extract class refactorings," in *Proc. 33rd Int. Conf. Softw. Eng.*, 2011, pp. 1037–1039.
[120] Y. Zhang, G. Huang, W. Zhang, X. Liu, and H. Mei, "Towards module-based automatic partitioning of Java applications," *Front. Comput. Sci.*, vol. 6, no. 6, pp. 725–740, 2012.
[121] A. Hussain and M. S. Rahman, "A new hierarchical clustering technique for restructuring software at the function level," in *Proc. 6th India Softw. Eng. Conf.*, 2013, pp. 45–54.
[122] G. Santos, M. T. Valente, and N. Anquetil, "Remodularization analysis using semantic clustering," in *Proc. Softw. Evol. Week - IEEE Conf. Softw. Maintenance Reengineering Reverse Eng.*, 2014, pp. 224–233.
[123] A. S. Mamaghani and M. Hajizadeh, "Software modularization using the modified firefly algorithm," in *Proc. 8th Malaysian Softw. Eng. Conf.*, 2014, pp. 321–324.
[124] L. L. Silva, M. T. Valente, and M. D. A. Maia, "Assessing modularity using co-change clusters," in *Proc. 13th Int. Conf. Modularity*, 2014, pp. 49–60.
[125] M. W. Mkaouer, M. Kessentini, S. Bechikh, K. Deb, and M. Ó Cinnéide, "High dimensional search-based software engineering: Finding tradeoffs among 15 objectives for automating software refactoring using NSGA-III," in *Proc. Conf. Genetic Evol. Comput.*, 2014, pp. 1263–1270.
[126] M. Paixao, M. Harman, and Y. Zhang, "Multi-objective module clustering for kate," in *Proc. Int. Symp. Search Based Softw. Eng.: Search-Based Softw. Eng.*, 2015, pp. 282–288.
[127] I. Šūpulniece *et al.*, "Source code driven enterprise application decomposition: Preliminary evaluation," *Procedia Comput. Sci.*, vol. 77, pp. 167–175, 2015.
[128] M. Bishnoi and P. Singh, "Modularizing software systems using PSO optimized hierarchical clustering," in *Proc. Int. Conf. Comput. Techn. Inf. Commun. Technol.*, 2016, pp. 659–664.
[129] J. Lee, D.-K. Kim, J. Park, and S. Park, "Class modularization using indirect relationships," in *Proc. 22nd Int. Conf. Eng. Complex Comput. Syst.*, 2017, pp. 110–119.
[130] Y. Wang, H. Yu, Z. Zhu, W. Zhang, and Y. Zhao, "Automatic software refactoring via weighted clustering in method-level networks," *IEEE Trans. Softw. Eng.*, vol. 44, no. 3, pp. 202–236, Mar. 2018.
[131] B. G. Varghese R, K. Raimond, and J. Lovesum, "A novel approach for automatic remodularization of software systems using extended ant colony optimization algorithm," *Inf. Softw. Technol.*, vol. 114, pp. 107–120, 2019.
[132] E. Hatami and B. Arasteh, "An efficient and stable method to cluster software modules using ant colony optimization algorithm," *The J. Supercomputing*, vol. 76, pp. 6786–6808, 2020.
[133] Y.-S. Seo and J.-H. Huh, "GUI-based software modularization through module clustering in edge computing based IoT environments," *J. Ambient Intell. Humanized Comput.*, vol. 22, pp. 7287–7311, 2019.
[134] S. Kebir, A.-D. Seriai, A. Chaoui, and S. Chardigny, "Comparing and combining genetic and clustering algorithms for software component identification from object-oriented code," in *Proc. 5th Int. Conf. Comput. Sci. Softw. Eng.*, 2012, pp. 1–8.
[135] C. Srinivas, V. Radhakrishna, and C. V. G. Rao, "Clustering software components for component reuse and program restructuring," in *Proc. 2nd Int. Conf. Innov. Comput. Cloud Comput.*, 2013, pp. 261–266.
[136] V. Radhakrishna, C. Srinivas, and C. Rao, "Document clustering using hybrid XOR similarity function for efficient software component reuse," *Procedia Comput. Sci.*, vol. 17, pp. 121–128, 2013.
[137] C. Patel, A. Hamou-Lhadj, and J. Rilling, "Software clustering using dynamic analysis and static dependencies," in *Proc. 13th Eur. Conf. Softw. Maintenance Reengineering*, 2009, pp. 27–36.
[138] S. Vodithala and S. Pabboju, "A clustering technique based on the specifications of software components," in *Proc. Int. Conf. Adv. Comput. Commun. Syst.*, 2015, pp. 1–6.
[139] S. Hasheminejad and S. Jalili, "CCIC: Clustering analysis classes to identify software components," *Inf. Softw. Technol.*, vol. 57, pp. 329–351, 2015.
[140] A. C. Kumari and K. Srinivas,"Hyper-heuristic approach for multi-objective software module clustering," *J. Syst. Softw.*, vol. 117, pp. 384–401, 2016.
[141] V. Karande, S. Chandra, Z. Lin, J. Caballero, L. Khan, and K. Hamlen, "BCD: Decomposing binary code into components using graph-based clustering," in *Proc. Asia Conf. Comput. Commun. Secur.*, 2018, pp. 393–398.
[142] M. A. Saied, A. Ouni, H. Sahraoui, R. G. Kula, K. Inoue, and D. Lo, "Improving reusability of software libraries through usage pattern mining," *J. Syst. Softw.*, vol. 145, pp. 164–179, 2018.
[143] C. Psarras, T. Diamantopoulos, and A. Symeonidis, "A mechanism for automatically summarizing software functionality from source code," in *Proc. IEEE 19th Int. Conf. Softw. Quality Rel. Secur.*, 2019, pp. 121–130.
[144] R. Islam and K. Sakib, "A package based clustering for enhancing software defect prediction accuracy," in *Proc. 17th Int. Conf. Comput. Inf. Technol.*, 2014, pp. 81–86.
[145] A. Ali, K. Choudhary, and A. Sharma, "Object oriented based technique for software quality prediction through clustering and chi-square test," in *Proc. Int. Conf. Appl. Theor. Comput. Commun. Technol.*, 2015, pp. 238–245.
[146] P. S. Sandhu, M. Kaur, and A. Kaur, "A density based clustering approach for early detection of fault prone modules," in *Proc. Int. Conf. Electron. Inf. Eng.*, 2010, pp. 525–530.
[147] X. Tan, X. Peng, S. Pan, and W. Zhao, "Assessing software quality by program clustering and defect prediction," in *Proc. 18th Work. Conf. Reverse Eng.*, 2011, pp. 244–248.
[148] C. M. Rosenberg and L. Moonen, "Improving problem identification via automated log clustering using dimensionality reduction," in *Proc. 12th ACM/IEEE Int. Symp. Empir. Softw. Eng. Meas.*, 2018, pp. 1–10.
[149] M. Bailey, K.-I. K.-I. K.-I. K.-I. Lin, and L. Sherrell, "Clustering source code files to predict change propagation during software maintenance," in *Proc. Annu. Southeast Conf.*, 2012, pp. 106–111.
[150] R. Benkoczi, D. Gaur, S. Hossain, and M. A. Khan, "A design structure matrix approach for measuring co-change-modularity of software products," in *Proc. 15th Int. Conf. Mining Softw. Repositories*, 2018, pp. 331–335.

[151] A. K. Malviya and V. K. Yadav, "Maintenance activities in object oriented software systems using K-means clustering technique: A review," in *Proc. CSI 6th Int. Conf. Softw. Eng.*, 2012, pp. 1–5.
[152] B. Mathur and M. Kaushik, "In object-oriented software framework improving maintenance exercises through k-means clustering approach," in *Proc. 3rd Int. Conf. Internet Things: Smart Innov. Usages*, 2018, pp. 1–7.
[153] S. H. Hamad, T. Fergany, R. A. Ammar, and A. A. Abd El-Raouf, "A double k-clustering approach for restructuring distributed object-oriented software," in *Proc. IEEE Symp. Comput. Commun.*, 2008, pp. 169–174.
[154] A. A. El-raouf, "Restructuring distributed object-oriented software using hierarchical clustering," in *Proc. WSEAES 13th Int. Conf. Comput.*, 2009, pp. 412–416.
[155] A. Ashish, "Clones clustering using K-means," in *Proc. 10th Int. Conf. Intell. Syst. Control*, 2016, pp. 1–6.
[156] M. Sudhamani and L. Rangarajan, "Code similarity detection through control statement and program features," *Expert Syst. Appl.*, vol. 132, pp. 63–75, 2019.
[157] P. Kreutzer, G. Dotzler, M. Ring, B. M. Eskofier, and M. Philippsen, "Automatic clustering of code changes," in *Proc. 13th Int. Workshop Mining Softw. Repositories*, 2016, pp. 61–72.
[158] Q. Khan, U. Akram, W. H. Butt, and S. Rehman, "Implementation and evaluation of optimized algorithm for software architectures analysis through unsupervised learning (clustering)," in *Proc. 17th Int. Conf. Sci. Techn. Autom. Control Comput. Eng.*, 2016, pp. 266–276.
[159] B. T. Bennett, "Using hierarchical agglomerative clustering to locate potential aspect interference," in *Proc. SoutheastCon*, 2017, pp. 1–8.
[160] C. Ieva, A. Gotlieb, S. Kaci, and N. Lazaar, "Discovering program topoi via hierarchical agglomerative clustering," *IEEE Trans. Rel.*, vol. 67, no. 3, pp. 758–770, Sep. 2018.
[161] W. Jin, T. Liu, Q. Zheng, D. Cui, and Y. Cai, "Functionality-oriented microservice extraction based on execution trace clustering," in *Proc. IEEE Int. Conf. Web Services*, 2018, pp. 211–218.
[162] A. Vanya, L. Holland, S. Klusener, P. van de Laar, and H. van Vliet, "Assessing software archives with evolutionary clusters," in *Proc. 16th IEEE Int. Conf. Program Comprehension*, 2008, pp. 192–201.
[163] M. Shtern and V. Tzerpos, "Methods for selecting and improving software clustering algorithms," in *Proc. IEEE 17th Int. Conf. Program Comprehension*, 2009, pp. 248–252.
[164] TIOBE. The TIOBE programming community index. Accessed: Mar. 01, 2020. [Online]. Available: https://www.tiobe.com/tiobe-index

Qusay I. Sarhan received the BSc degree in software engineering from the University of Mosul, Iraq, in 2007, and the MTech degree in software engineering from Jawaharlal Nehru Technological University, India, in 2011. Currently, he is a lecturer and the leader of Software Engineering and Embedded Systems (SEES) research group at the University of Duhok, Iraq. He has a couple of national and international publications and his research interests include software engineering, Internet of Things, and embedded systems.

Bestoun S. Ahmed received the MSc degree from the University Putra Malaysia, Malaysia, in 2009, and the PhD degree in software engineering from the University Sains Malaysia (USM), Malaysia, in 2012. He was a senior lecturer with Salahaddin University, Iraq. He spent one year doing his postdoctoral research at the Swiss AI Lab Istituto Dalle Molle di Studi sull'Intelligenza Artificiale, Switzerland. He was also an assistant professor with Czech Technical University, Prague. He is currently an associate professor with Karlstad University, Sweden. His primary research interests include combinatorial testing, search-based software testing, computational intelligence, and quality assurance for the IoT services.

Moroslav Bures received the PhD degree from the Faculty of Electrical Engineering, Czech Technical University in Prague, Czechia, where he is currently a researcher and a senior lecturer in software testing and quality assurance. His research interests include model-based testing, efficiency of test automation, and quality assurance methods for the Internet of Things solutions, reflecting specifics of this technology. In these areas, he also leads several R&D and experimental projects. He is a member of Czech chapter of the ACM, CaSTB, ISTQB Academia work group, and participates in broad activities in the professional testing community.

Kamal Z. Zamli received the degree in the field of electrical engineering from Worcester Polytechnic Institute, Worcester, Massachusetts, in 1992, the MSc (Real-Time Software Engineering) degree from Universiti Teknologi Malaysia, Malaysia, in 2000, and the PhD degree in software engineering from the University of Newcastle upon Tyne, United Kingdom, in 2003. His research interests include search based software engineering and computational intelligence.