Improving Software Modularity Using Software Remodularization: Challenges and Opportunities

Naveen[1], *Randeep* Singh[2], and *Amit* Rathee[3]

[1] Research Scholar, Department of Computer Science & Engineering, IEC University, Solan, India
[2] Professor, Department of Computer Science & Engineering, IEC University, Solan, India
[3] Assistant Professor, Department of Computer Science, GC Barota, Sonipat, Haryana, India

Abstract. Long-period maintenance of software often causes the original program modularization to decay, resulting in a degraded quality of the software. Placement of software artifacts, namely code files or classes in a suboptimal manner among software packages is one of the most common causes of this decaying modularization of software. Software re-modularization (remodularization) is an old reverse engineering approach that helps in alleviating this issue by improving the quality of software modularization. However, in literature, a multitude of software remodularization approaches exists leaving researchers and developers in a dilemma of choosing appropriate remodularization criteria. Therefore, this paper carries out a Systematic Literature Review (SLR) of the last two decades and presents concise information by answering different relevant research questions important to both developers and researchers of this field. This paper considers 54 research articles as primary studies.

1 Introduction

The modularity of a software system is just like a signboard on a street that generally guides a software maintenance team in the right direction. This is because it reflects the degree of low coupling and high cohesion software engineering principles among different modules of a software system. It helps to understand and navigate the current structure. This information is quite dispersed in the form of names for every module, and modules are subdivided into submodules. However, due to regular software maintenance activities, the original criterion behind making a module/ submodule is generally lost or in some cases, it is unavailable due to outdated software documentation. As a result, the modularity of the software becomes patchy. In this scenario, the modularity of software sometimes becomes misleading instead of guiding in the right direction normally just as the street signboard does. Thus, inadequate modularity results in the difficult and expensive maintenance of the software.

The improvement of a program's modular structure is a preventive maintenance step necessary when code base becomes infeasible and requires restructuring steps before applying any kind of maintenance task. This restructuring is commonly termed software re-modularization/ remodularization in the field of reverse engineering branch of software engineering. It aims at making software high complexity manageable. However, in the literature, a multitude of modularization criteria exists that can be applied to reduce architectural complexity. Thus, developers must understand which criteria have been used before they can extend, reuse, or restructure a system. Further, Software engineering researchers are interested in examining typical patterns of modularization across a variety of projects in order to understand how modularization criteria are applied in practice.

The work in this paper explains about the need for software remodularization, tools, and techniques available for developers by carrying out a systematic literature survey of the last two decades. Several trade-offs exist regarding software remodularization especially related to different perspectives of developers about modularity parameters such as cohesion, coupling, etc.; the level at which remodularization should be applied viz function, file/ class, package/ module, etc.; quality evaluation parameters viz encapsulation vs decomposition. Therefore, this paper considers reviewing software remodularization necessary and presents concise information for researchers and developers engaged with software remodularization and maintenance activities. The rest of this paper is organized as follows: section entitled "Literature Survey" summarizes recent relevant literature work related with the topic considered in this paper, section entitled "Research Methodology" describe details about the methodology adopted to carry out systematic investigation in this paper, section "Results and Interpretation" elaborates obtained investigation results, and finally section entitled "Conclusion and Future Work" summarizes research work of this paper and suggest possible future directions.

2 Literature Survey

Numerous research methods and tools have been developed to perform and automate software remodularization. Below are brief reviews of several prominent studies and explanations of their limitations.

2.1 Clustering Based Software Remodularization

Candela et al. carried out an extensive study to investigate the role of cohesion and coupling during software remodularization [1]. A large dataset consisting of 100 open-source projects is investigated together with survey reports of 29 developers is also considered. They conclude that cohesion and coupling are solely not enough to produce good remodularization solutions from the developers' perspective. Moreover, tools that produce "big-bang" remodularization have limited applicability. Rathee et al. [2] proposed a software remodularization approach based on structural and conceptual coupling measured at the class level. They used hierarchical agglomerate clustering to perform software remodularization. Bavota et al. proposed an approach to improve software modularization quality by exploiting structural and semantic relationships between classes [3]. The proposed approach makes use of a chain of strongly related classes to define new packages in the remodularized system. Authors in [4] proposed an automatic software remodularization approach based on the move refactoring recommendations. The proposed approach is based on Potts Model and considers software as a complex network represented at the class level. Santos et al. applied semantic clustering to perform software remodularization [5]. Authors in [6] proposed a top-down hierarchical clustering (TDHC) approach to represent the system as a tree-like structure. The nodes represent different artifacts and the path represents a possible software module (cluster). Hakik et al. applied the formal concept analysis (FCA) technique to obtain better remodularized architecture of a software system [7].

2.2 Search Based Software Remodularization

The authors in [8] studied the feasibility of search-based software remodularization at *Adyen's* code base. They conclude that it is possible to scale search-based approaches very well even to large code bases. Nonetheless, they further conclude that there is still room for improvement in recommending which module students should move to. Amarjeet et al. used a harmony search metaheuristic algorithm to handle software remodularization [9]. They created four different variations of their harmony search algorithm and compared them with other metaheuristic algorithms. Mahouachi et al. proposed a cost-effective search-based

approach using a multiobjective NSGA-II algorithm by identifying optimal move class refactoring opportunities [10]. Bavota et al. proposed an approach called IGA (Interactive Genetic Algorithm) that integrated the developer's expertise to perform better remodularization results [11]. They further conclude that IGA outperforms full-automated GA. The authors in [12] proposed four heterogeneous features based multi-factor remodularization approach. They conclude that their approach produces approx. 11% better results as compared to single-objective metaheuristic algorithms. The authors in [13] proposed a software remodularization approach using a many-objective NSGA-III algorithm using seven conflicting fitness functions. Mu et al. proposed a hybrid genetic algorithm (HGA) to obtain high quality remodularization results for a software system [14]. The proposed algorithm outperforms the hill-climbing algorithm. The authors in [15] perform software remodularization using a multi-objective hill-climbing algorithm. Similarly, authors in [16] proposed a many-objective based approach called MaABC by modifying the basic ABC algorithm.

The possible research gaps are as follows: (1) the existing literature is huge, diverse and expanding rapidly with the introduction of new tools/ techniques; (2) the systematic literature survey is helpful for a newbie by presenting diverse and vast literature in concise form.

3 Research Methodology

This research paper carefully follows the steps suggested by Petersen et al. [17] in order to systematically conduct and conclude the review process. Four different main steps followed are: (1) deciding relevant Research Questions (RQs); (2) deciding the search strategy and screening of articles; (3) classification and data extraction; and (4) mapping and conclusion. Each of these steps is explained in detail as follows:

3.1 Research Questions

The goal of this paper is to conduct SLR and present concise knowledge about software remodularization. This goal is achieved by considering the following research questions that are relevant for both researchers and practitioners:

RQ 1: What is the evolution pattern of the number of publications and their frequency over time? The answer to this RQ helps the practitioners and researchers in determining the relevance of software remodularization in the field of software quality and maintenance reduction. Moreover, the number and consistency of publications help in reflecting the maturity of the research being conducted.

RQ 2: Which research publication platform do researchers generally adopt? The field of software remodularization does not belong to a specific forum and is relevant to many software engineering fields such as quality, maintenance, reusability, etc. Therefore, the research relevant to software remodularization may be published in different fora of software engineering. This formulated RQ aims at identifying the most preferred research publication platforms and it helps the researchers to select the best source for publishing his/ her future research.

RQ 3: What are the main techniques used to perform software remodularization in literature? This formulated RQ aims at investigating the most suitable user preferences adopted by researchers to perform software remodularization.

RQ 4: What are the evaluation methods for the software remodularization approaches? It is particularly important to perform an evaluation phase in order to demonstrate that the approach is applicable in real-life situations. The purpose of this research question is to provide information on how other approaches are evaluated to help the researcher better plan the evaluation of their own. Any kind of tool support is also identified in this RQ.

3.2 Search Strategy and Screening of Research Article

In order to provide satisfactory SLR results, an accurate and comprehensive search strategy must be designed to explore a wide range of literature. To form the search terms, a set of keywords are identified from considered research questions and goals by following the guidelines of [18]. Finally, these keywords are combined together by using three Boolean operators, namely NOT, AND, and OR to form a search string used to determine the initial set of relevant research articles available in the literature. **Table 1** depicts the considered search string for carrying out SLR. The considered keywords are taken from three main domains viz general, search, and preference-based techniques.

Table 1. Considered Search String.

S. No.	Software Domain	Keywords	Search String
1.	General	Software- Modularization, Remodularization, Quality, Maintenance, Design Improvement; Reverse Engineering; Model, Method, Technique, Survey, Review, Framework	Software **AND** (Modularization **OR** Remodularization **OR** Quality **OR** Design **OR** Maintenance **OR** Reverse Engineering **OR** Search/Search-Based **OR** Multi/ Multi- **OR** Optimization **OR** Objective **OR** Genetic **OR** Heuristic **OR** IGA/ PSO/ ACO/ IGA/ NSGA/ Hill Climbing/ Simulated Annealing) **AND** (Interactive **OR** User/Human **OR** Weight/ Ranking- Based) **AND** (Method **OR** Model **OR** Approach **OR** Algorithm **OR** Technique **OR** Survey **OR** Review **OR** Framework)
2.	Search Techniques	Search/Search- Based, Optimization, Multi/Multi-Objective, Genetic Algorithm, Hill Climbing, IGA, NSGA, PSO, ACO, PBEA, Simulated Annealing, Meta/Meta-Heuristic	
3.	Preference-based Techniques	Interactive, User/ Human-Specified/ Provided, Weight/ Ranking- Based	

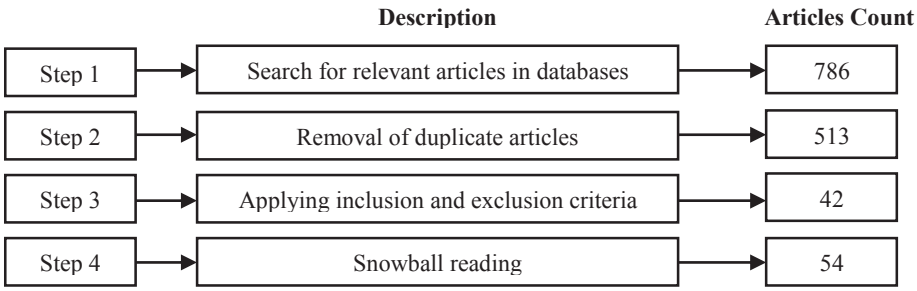

Fig. 1 Primary Studies Search and Selection Process.

After deciding search string, we conducted a four-step process to find and select relevant primary studies as depicted in **Fig. 1**. During the first step, different electronic databases (**Table 2)** are searched for a given search string and articles from the last two-decade duration are collected (total 786 articles). Repeated publications are further discarded during the second step. Next, in the third step, inclusion and exclusion criteria as specified in the **Table 3** are applied to further scrutinize collected articles from different electronic databases.

Table 2. Details of different considered electronic databases.

S.No.	Database Source Name	Web-link	# Articles
1.	IEEE Xplore	http://ieeexplore.ieee.org	117
2.	Science Direct	http://www.sciencedirect.com	202
3.	Scopus	http://www.scopus.com	168
4.	ACM Digital Library	http://dl.acm.org	62
5.	Springer	http://www.springerlink.com	75
6.	Wiley Online Library	https://onlinelibrary.wiley.com	98
7.	Google Scholar	https://scholar.google.com	64

Table 3. Considered Inclusion and Exclusion Criteria.

Inclusion Criteria	Exclusion Criteria
1. The article must be written in ENGLISH language only. 2. The article publication category must be one of the journal, technical report, conference, thesis, workshop, patent, or book chapter. 3. Studies such as literature reviews, mapping studies, or surveys are given preferences. 4. The article must be available as HTML or PDF. 5. The article must cover the scope considered in this paper in the form of various RQs. 6. The article must be at least 4 pages.	1. Articles are not available online. 2. Short papers such as editorial articles or abstracts are not considered. 3. Doctoral symposiums and poster papers are not considered. 4. Articles that do not cover the scope considered in this paper.

3.3 Classification and Data Extraction

Once the primary studies are selected, the next phase of the SLR process is to carry out their intensive manual analysis in order to answer the considered RQs. For this purpose, we created schemes and categories interactively as new primary studies are read and added. Here it is important to note that more than one category may be assigned to a paper. The first author categorized primary articles and the second and third authors validated the classifications. A few disagreements and doubts are resolved through discussion with appropriate changes to classification scheme. We used a data extraction form and created a spreadsheet to analyze and evaluate extracted data after the manual analysis of primary studies. The stored information in the spreadsheet includes title, authors detail, year, publication source, proposed technique details, evaluation method used, datasets used, developed/ used tool details, and scope and/ or application of proposed work about the considered primary studies.

3.4 Mapping and Conclusion

This is the last step of SLR and it involves mapping analysis results to different RQs and concluding them. These mapped results are presented in the subsequent section.

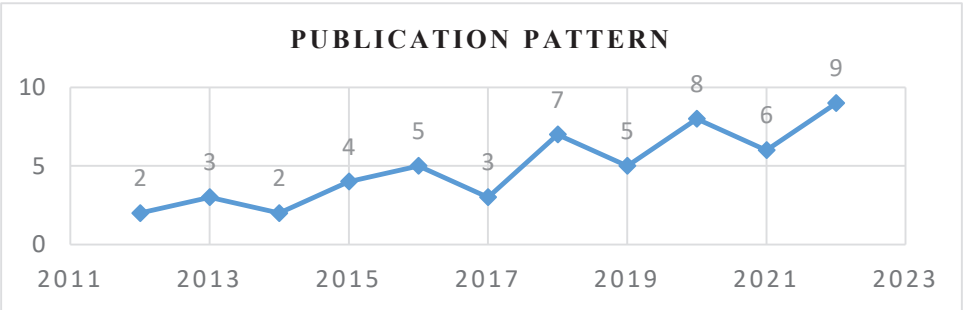

Fig. 2 Publication pattern over the years.

4 Results And Interpretation

According to the extracted information and classification schemes used above in this paper, we answer each research question in this section.

4.1 What is the evolution pattern of the number of publications and their frequency over time? (RQ1)

Fig. 2 shows the distribution pattern of different primary studies over the years considered in the last two decades. The publication pattern shows that there is considerable growth in the number of the paper published over years and software remodularization has started

recognizing its importance among the research community and/or industry. Furthermore, the results indicate that software remodularization continues to raise interest, though it has not yet reached maturity. Because of this mapping study, we hope that new research will be stimulated and that the field can be solidified and consolidated.

4.2 Which research publication platform do researchers generally adopt? (RQ2)

Similar to the trend that we observed for articles in RQ 1, we also observed preference patterns for conferences and journals chosen by researchers in past for software remodularization related activities. The publication preferences are studied for three categories viz (1) journal, (2) conference, and (3) others. We observed that journals are preferred in 30% (16 of the total 54 articles) of the considered primary research articles. Further, 65% of the research articles (35 of the total 54 articles) considered different conference sources for their publication. Finally, the rest of the 5% primary studies are available in the form of other resources such as the thesis, workshop, articles, etc. The main fora opted by different publishers (either journal or conference) are listed in **Fig. 3** and Error! Reference source not found.. Based on the publication counts, we clearly observe that there are no specific and preferred journals or conferences in the literature. Further, researchers are left with a wide variety of options to publish in near future.

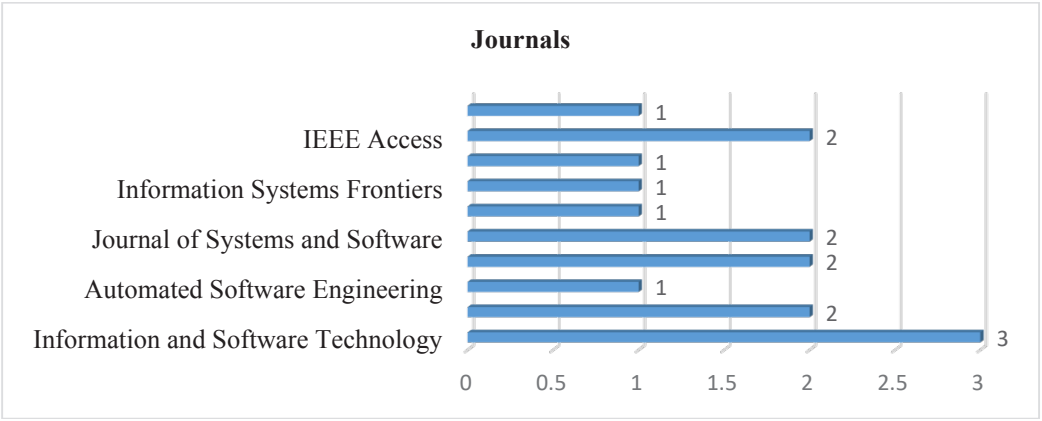

Fig. 3 Found Journal Venues.

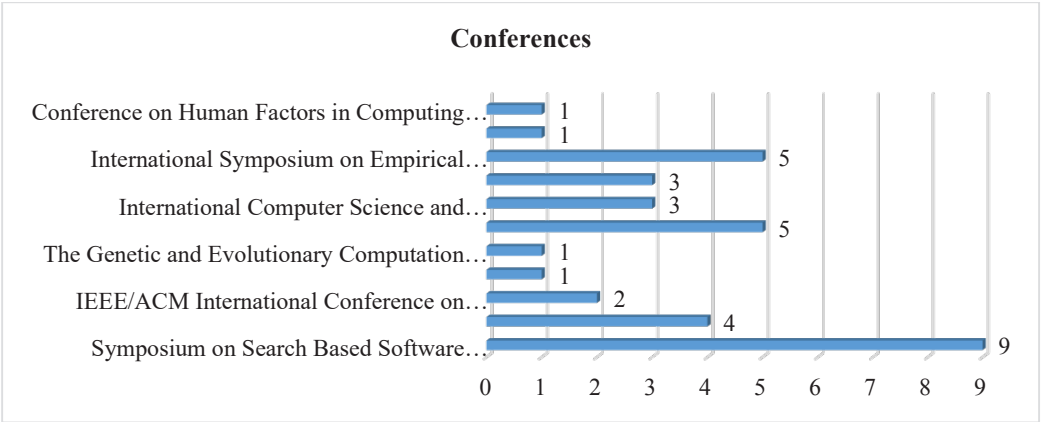

Fig. 4 Found Conference Venues.

4.3 What are the main techniques used to perform software remodularization in literature? (RQ3)

There is a need for periodic remodularization of software since its structure degrades over time due to prolonged maintenance activities. In the literature, the following two methods are adopted for remodularizing software systems.

4.3.1 Clustering Based Software Remodularization

Clustering is an old, highly sophisticated key activity in reverse engineering to discover a better design of the systems or to extract significant concepts from the code. It is mainly based on poor cohesion and coupling relations of a software system. It mainly aims at decomposing packages with poor cohesion into several meaningful packages with higher cohesion properties. Clustering is very sophisticated and a separate research domain in itself as it offers several methods that fulfill different needs such as size and type of data, apriori knowledge about expected results, etc. While applying clustering for software remodularization, we need to handle three main factors. (1) **Entities' Description**- is crucial and for a software, an entity can be a module/ package, file/ class, method/ process/ routine, or a valid statement. In literature, entity definition is subjective and language-specific. (2) **Entities' Coupling**- is another crucial aspect that decides when a pair of entities must be grouped together (form a cluster). It is generally defined using the following two approaches. The first approach called "direct link" considers direct dependency relations among entities and put entities together in a single cohesive unit only if they directly make use of each other. The second approach called "sibling link" is based on the indirect relationship among entities such as entities engaged with the same behavior. These two approaches are explained in **Fig. 5**. (3) **Clustering Algorithm**- wide varieties of algorithms are available in the literature to perform clustering. However, it is important to note here that some links are ignored (arbitrarily) and others are favored by these algorithms. Similarity metrics and the algorithm themselves contribute to this decision. Interested readers can find further explanation about it in [19]. Moreover, it is our observation that each clustering algorithm imposes a different structure. In some cases, they are definitely useless since they do not match a reasonable software engineer's view of the system. In addition, there may be other structures that are interesting from a different perspective.

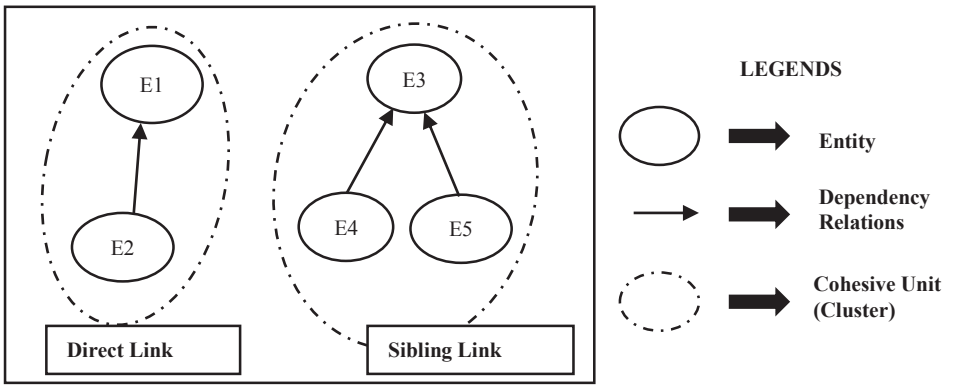

Fig. 5 Entities' Coupling Criteria.

In literature, dependency (coupling) among different entities is also measured based on two kinds of features belonging to an entity, namely (1) formal descriptive features, and (2) non-formal descriptive features. The formal descriptive features consist of information that has a direct impact and/ or direct consequence on the software's behavior. These are generally derived from the underlying source code of a software system and they are most commonly

used by researchers in literature. Some examples are variables declaration and usage pattern, method definition and calling pattern, files/ packages included by an entity, entity name, and its type, etc. (typically termed as structural information). Another type of such information is termed dynamic information and is related to the execution pattern of a software. The non-formal descriptive features include information that does not directly influence the underlying system's behavior. Typical examples include information extracted from comments declared within the source code of an entity, naming patterns assigned to different parts of an entity (typically called semantic information), changing patterns associated with ongoing maintenance activities of a software system (typically termed as change-history or evolutionary information).

Further, different clustering algorithms result in different remodularized solutions of the same software. Therefore, it is necessary to evaluate which of the obtained remodularized solution is optimal. In the literature, four types of quality criteria are mainly suggested for this purpose. The first criterion is termed "Design Criteria" and is related to optimal values of cohesion and coupling among entities. The second criterion is termed "Expert Criteria" and is related to matching obtained remodularized solutions against expert-suggested system decomposition. The third criterion is related to assessing redundancy among entities in the obtained remodularized solution. The fourth criterion is related to the clustering size and it suggests that the obtained cluster must be never too large or too small.

4.3.2 Search Based Software Remodularization

Due to various problems associated with clustering-based remodularization such as there is a poor distribution of classes within packages, semantic inconsistencies, big-bang remodularized solutions suggested many a time, and the growing size of the software systems, search-based metaheuristic techniques are also gaining popularity in literature for handling software remodularization. The search-based modularization approaches assume that we begin by building a dependence graph for the modules and a fitness function guides the modular improvement process. **Table 4** lists different search-based remodularization metaheuristic algorithms used in the literature. Moreover, this also lists various software remodularization centric characteristics (objectives) used by researchers for formulating fitness functions in their search-based approaches. From this table, it is clear that GA-based, HS-based, and NSGA-II are among the most preferred approaches in literature.

Table 4. Search-Based Software Remodularization Approaches.

S.No.	Algorithm Name	Reference	Search Objectives Used
1.	Hill Climbing (HC)	[15]	Cohesion, Coupling, Number of Modules, Package Size, Number of Required Changes, Number of broken Dependencies, Semantic Similarity, Efforts Required, Method Call and Field Access Pattern, Characteristics of Clusters
2.	Genetic Algorithm (GA) and its variations	[11, 14, 20]	
3.	Simulated Annealing (SA)	[21]	
4.	MOEA	[12]	
5.	NSGA-II	[8, 10, 22]	
6.	Harmony Search (HS) and its variants	[9, 23, 24]	
7.	NSGA-III	[13]	
8.	Ant Bea Colony (ABC)	[16]	

4.4 What are the evaluation methods for the software remodularization approaches? (RQ4)

An overview of approach evaluation is presented here. Evaluating the approach is especially important because it shows evidence that it can be applied in the field. Based on our investigation of the literature study, we observed that approx.. 70% of studies uses real-world

instances such as JFreeChart, Mozilla, Firefox, Apache Ant, etc. and 30% of total studies uses artificial instances of software system that are randomly or specifically selected for evaluation purposes. This counting also includes software that is small or software that is not properly maintained in the research community. This creates problems of reproducibility or verification of such approaches by researchers in near future. Moreover, an academic context is used for most evaluations and industrial evaluation is mostly carried out in a constrained environment resulting in problems related to replication of the carried out experimentation.

5 CONCLUSION AND FUTURE WORK

In this paper, an SLR of software remodularization is carried out. Our study reveals several challenges of conventional software remodularization techniques, which emphasize coupling and cohesion mainly and offer a few types of operations (move class, split package). The finding suggests that search-based remodularization solutions help in improving the quality of modularized solutions and they also help in incorporating user preferences better. Many objective metaheuristic algorithms are found to be useful to incorporate more than three conflicting objectives during software remodularization. Moreover few challenges identified during the study are as follows: (1) research gaps still exist for the efficient abstraction and modeling of the relationship between them exist, (2) robustness approximation is another challenge for the proposed remodularization algorithms, and (3) scalability along with quality assessment criteria are another future research areas in the field of software remodularization.

As future work, research should continue in the domains of analyzing the software remodularization with respect to its potential similarity to the legacy system's original structure, as well as analysis of the performance of the remodularized software (modularity).

References

1. I. Candela, G. Bavota, B. Russo, and R. Oliveto, "Using cohesion and coupling for software remodularization: Is it enough?," ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 25, no. 3, pp. 1-28, 2016.
2. A. Rathee and J. K. Chhabra, "Software remodularization by estimating structural and conceptual relations among classes and using hierarchical clustering," in International Conference on Advanced Informatics for Computing Research, 2017, pp. 94-106: Springer.
3. G. Bavota, A. De Lucia, A. Marcus, and R. Oliveto, "Software re-modularization based on structural and semantic metrics," in 2010 17th Working Conference on Reverse Engineering, 2010, pp. 195-204: IEEE.
4. M. S. Zanetti, C. J. Tessone, I. Scholtes, and F. Schweitzer, "Automated software remodularization based on move refactoring: a complex systems approach," in Proceedings of the 13th international conference on Modularity, 2014, pp. 73-84.
5. G. Santos, M. T. Valente, and N. Anquetil, "Remodularization analysis using semantic clustering," in 2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE), 2014, pp. 224-233: IEEE.
6. M. Aghdasifam, H. Izadkhah, and A. Isazadeh, "A new metaheuristic-based hierarchical clustering algorithm for software modularization," Complexity, vol. 2020, 2020.
7. L. M. Hakik and R. El Harti, "Measuring Coupling and Cohesion to Evaluate the Quality of a Remodularized Software Architecture Result of an Approach Based on Formal Concept Analysis," International Journal of Computer Science and Network Security (IJCSNS), vol. 14, no. 1, p. 11, 2014.

8. C. Schröder, A. van der Feltz, A. Panichella, and M. Aniche, "Search-based software re-modularization: a case study at Adyen," in 2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), 2021, pp. 81-90.
9. J. K. Chhabra, "Harmony search based remodularization for object-oriented software systems," Computer Languages, Systems & Structures, vol. 47, pp. 153-169, 2017.
10. R. Mahouachi, "Search-based cost-effective software remodularization," Journal of Computer Science and Technology, vol. 33, no. 6, pp. 1320-1336, 2018.
11. G. Bavota, F. Carnevale, A. D. Lucia, M. D. Penta, and R. Oliveto, "Putting the developer in-the-loop: an interactive GA for software re-modularization," in International Symposium on Search Based Software Engineering, 2012, pp. 75-89: Springer.
12. J. Hwa, S. Yoo, Y.-S. Seo, and D.-H. Bae, "Search-based approaches for software module clustering based on multiple relationship factors," International Journal of Software Engineering and Knowledge Engineering, vol. 27, no. 07, pp. 1033-1062, 2017.
13. W. Mkaouer et al., "Many-objective software remodularization using NSGA-III," ACM Transactions on Software Engg. and Methodology (TOSEM), vol. 24, no. 3, pp. 1-45, 2015.
14. L. Mu, V. Sugumaran, and F. Wang, "A hybrid genetic algorithm for software architecture re-modularization," Information Systems Frontiers, vol. 22, no. 5, pp. 1133-1161, 2020.
15. N. Sadat Jalali, H. Izadkhah, and S. Lotfi, "Multi-objective search-based software modularization: structural and non-structural features," Soft Computing, vol. 23, no. 21, pp. 11141-11165, 2019.
16. A. Prajapati, A. Parashar, and A. Rathee, "Multi-dimensional information-driven many-objective software remodularization approach," Frontiers of Computer Science, vol. 17, no. 3, pp. 1-18, 2023.
17. K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," in 12th International Conference on Evaluation and Assessment in Software Engineering (EASE) 12, 2008, pp. 1-10.
18. B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering–a systematic literature review," Information and software technology, vol. 51, no. 1, pp. 7-15, 2009.
19. V. Kumar, J. K. Chhabra, and D. Kumar, "Performance evaluation of distance metrics in the clustering algorithms," INFOCOMP Journal of Computer Sci, vol. 13, no. 1, pp. 38-52, 2014.
20. G. Bavota, M. Di Penta, and R. Oliveto, "Search based software maintenance: Methods and tools," in Evolving software systems: Springer, 2014, pp. 103-137.
21. H. Abdeen, S. Ducasse, H. Sahraoui, and I. Alloui, "Automatic package coupling and cycle minimization," in 2009 16th Working Conference on Reverse Engineering, 2009, pp. 103-112: IEEE.
22. A. Prajapati, A. Parashar, and J. K. Chhabra, "Restructuring Object-oriented software systems using various aspects of class information," Arabian Journal for Science and Engineering, vol. 45, no. 12, pp. 10433-10457, 2020.
23. A. Prajapati and J. K. Chhabra, "MaDHS: Many-objective discrete harmony search to improve existing package design," Computational Intelligence, vol. 35, no. 1, pp. 98-123, 2019.
24. A. Prajapati, "Software package restructuring with improved search-based optimization and objective functions," Arabian Journal for Science and Engineering, vol. 46, no. 9, pp. 9023-9043, 2021.