IJICTR
International Journal of Information & Communication Technology Research

Volume 16- Number 1 – 2024 (28 -41)

Software Re-Modularization Method Based on Many-Objective Function

Mohammad Reza Keyvanpour*
Department of Computer Engineering
Faculty of Engineering
Alzahra University
Tehran, Iran
Keyvanpour@alzahra.ac.ir

Zahra Karimi Zandian
Data Mining Lab
Department of Computer Engineering
Faculty of Engineering
Alzahra University
Tehran, Iran

Fatemeh Morsali
Data Mining Lab
Department of Computer Engineering
Faculty of Engineering
Alzahra University
Tehran, Iran

Received: 2 December 2022 – Revised: 7 February 2023 - Accepted: 15 March 2023

Abstract—**Software evolution and continuous changes make maintenance difficult, reducing the quality of software structure and architecture. To cope with this challenge, re-modularization is used to promote the modular structure of software system by the re-grouping of software elements. In this paper, the proposed method recognizes various dependencies in terms of an objective function unlike what has been stated in some other methods. In this method, a search-based many-objective fitness function is proposed to formulate re-modularization as an optimization problem. The results of the proposed method have been compared to the effects of four other methods based on MQ and NED. The results show the proposed method improved re-modularization remarkably compared to others in terms of both MQ and NED criteria especially for smaller software. Therefore, the proposed method can be effective in redefining real-world applications.**

Keywords: Software, Re-modularization, Many-objective function, Elements dependencies, Clustering, Search-based algorithm.

Article type: Research Article

I. INTRODUCTION

A good modular design of a large and complex software system is a desirable feature. Although most software systems are designed and developed modularly at first, modularity is degraded over time. This degradation makes future evolution hard. Software maintenance is a crucial process to cope with continuously changing software and ensure that software is preserved in its life cycle [1][2]. One of the important activities to better understand a software system for maintenance and development is modularization [3]. The purpose of modularization is to partition system elements in clusters, subsystems or modules automatically [4][3] so that in the obtained system the external connection (i.e., the relationship between the components of two separate clusters) is minimum, while the internal connection (i.e., the connection between the cluster components) is maximized. The modular structure helps to develop the software by replacing the necessary elements into the modules without significantly impacting the complete system [5].

* Corresponding Author

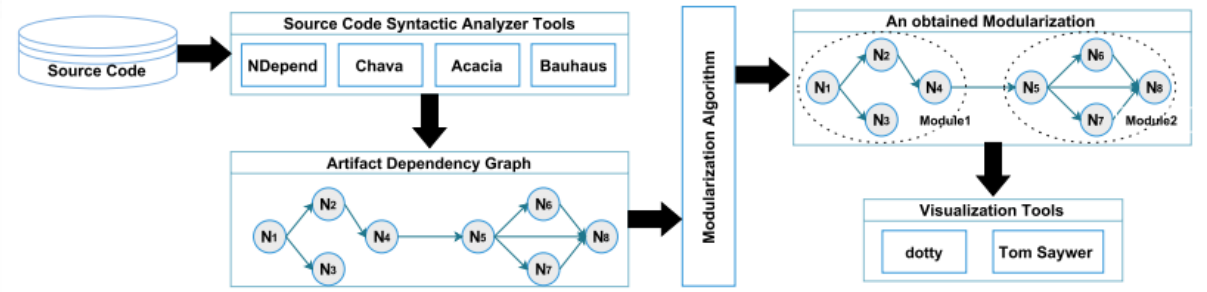

Figure 1. The general schema of software modularization process [7]

Software clustering is a modularization technique that modularizes different artifacts into the module with more similar artifacts than other modules [6]. Figure 1 shows the general schema of software modularization process.

As shown in this figure, modularization involves three steps: source code analysis using analyzer tools and dependency graph creation, software modularization based on the graph using modularization algorithms, and labeling and displaying the obtained modules [8].

During software evolution and continuous changes, its structure is often modified, moving it away from the original design and reducing its quality [9]. Therefore, re-modularization is a necessary procedure as a re-grouping process of software elements at the modular level to improve the modular structure of software system [10].

In this regard, different methods have been proposed in the field of software re-modularization. However, the main challenge in this field is the lack of methods aimed at recognizing various types of dependencies. Investigating one type of dependencies makes the method useful in an limited aspects of re-modularization. Therefore, in this paper, we propose a new method considering various types of structural dependencies. In addition, there is a need to model objective functions in terms of various dimensions of the structural relationships that can guide the optimization process towards an acceptable modularization solution for developers. Therefore, in this paper a new search-based method is proposed for re-modularization. In this method, a search-based many-objective fitness function is proposed to formulate re-modularization as an optimization problem where a modified harmony search algorithm is used to solve it. To evaluate the proposed method, three open source software systems are used: Junit, Java Servlet API, and DOM 4J. The results based on two popular criteria, MQ and NED show the efficiency of the proposed method in re-modularization in real-world software. The rest of the paper is organized as follows. In Section II, the problem is defined. In Section III, related works are discussed. In Section IV, the proposed method is introduced. Experiments and evaluation results are presented in Section V, followed by the concluding remarks in Section VI.

II. Problem Definition

If software maintainers do not have any insight into the system design, they may change the source code undesirably. This influences the software structure quality negatively [11]. On the other hand, it is difficult to understand the complexity of relationships between various source code components in a software system. One way to cope with structural complexity is to cluster the relevant processes and data in the same modules or classes automatically [4]. In search-based clustering, system modularization is considered as a search-based optimization problem, in which case it needs to introduce an objective function [12].

Due to software evolution and continuous changes, its structure is often modified, moving it away from the original design and reducing its quality [9]. Therefore, re-modularization is a necessary procedure as a re-grouping process of software elements at the modular level to improve the modular structure of the software system [10]. Therefore, modularization and re-modularization are based on clustering.

The lack of research aimed at recognizing various types of structural dependencies to introduce an objective function is the main challenge of re-modularization.

Definition (many-objective problem). Many objective optimization problems are mathematically defined as follows:

$$F(M^*) = \min\big(F_1(M), F_2(M), \dots, F_m(M)\big) \quad (1)$$

Where, m is the number of objective functions, F_i is an objective function, and M is a non-dominated re-modularization solution.

III. Related Works

Modularization and re-modularization are based on clustering which provides easier navigation and tracking among software parts [13]. Clustering also leads to increased comprehensiveness between software elements and software quality. Numerous studies have been conducted on search-based module clustering.

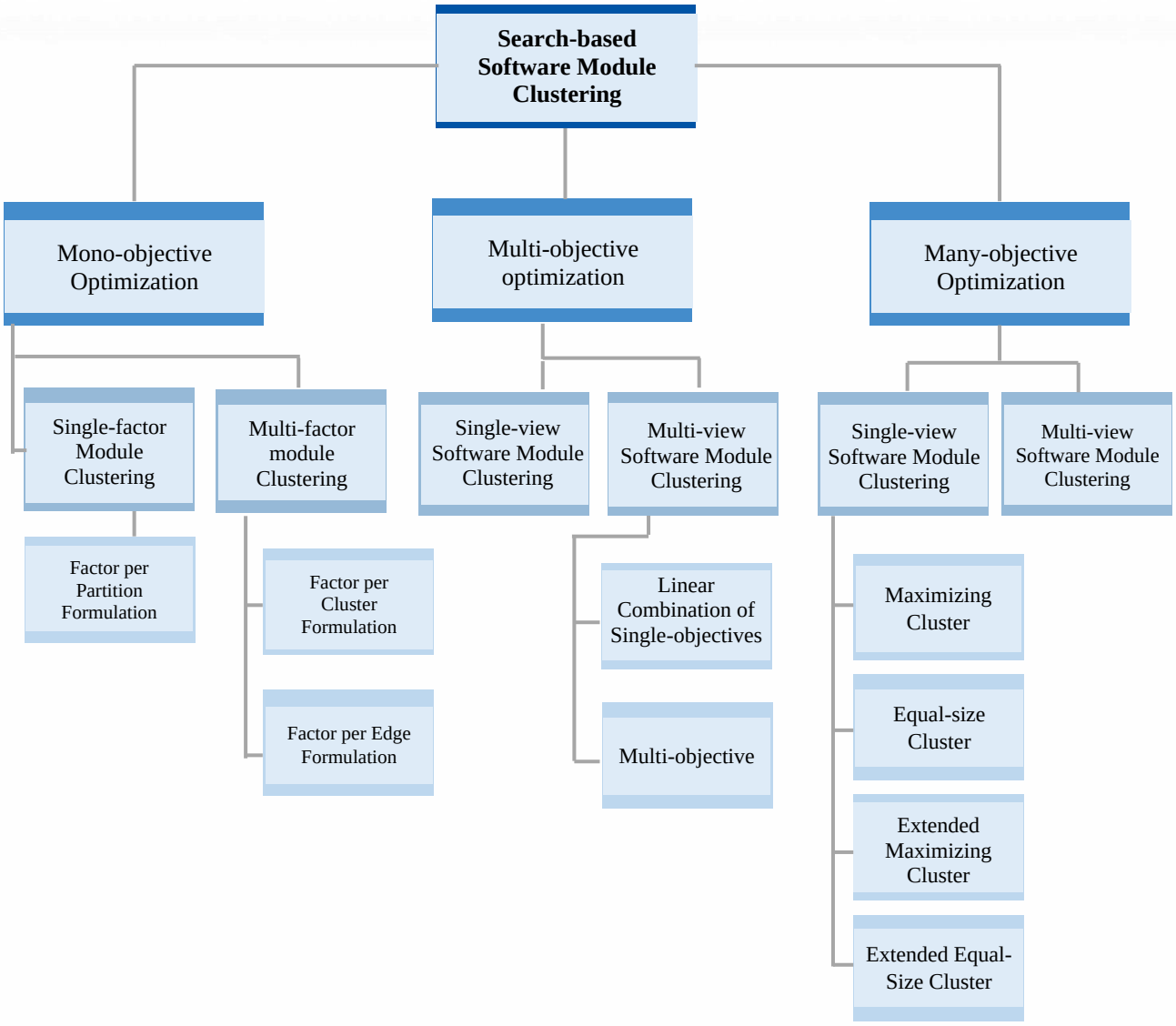

Figure 2. Classification of search-based software module clustering techniques

Reviewing the proposed methods in search-based software module clustering techniques, as mentioned in [13] and Figure 2, shows that these methods can be categorized into three groups: Mono-objective optimization, Multi-objective optimization, and Many-objective optimization. Mono-objective optimization can be addressed by Single-factor module clustering or Multi-factor module clustering [14]. In Mono-objective optimization, the aim is to find a solution for objective function. The advantage of this approach is its low processing time. However, the quality of solutions is lower than that of other approaches because of considering one aspect of the software system [13]

For a given cluster design vector (c), the optimal solution (c^*) is calculated according to the following equation [15].

$$f(c^*) = min/\max f(c) | \, c \in \psi \quad (2)$$

Where ψ is the set of all feasible clustering solutions.

Multi-objective optimization-based methods can be divided into Single-view software module clustering and Multi-view software module clustering approaches. The purpose of this type of optimization is to find several non-dominant solutions for objective functions [16]. To formulate and optimize clustering solutions for a given cluster design vector (c), the final and optimal solution (c^*) can be calculated according to the equation below [15].

$$f(c^*) = \begin{cases} \min(f_1(c), f_2(c), \dots, f_M(c)^T) & M > 1 \\ g_j(c) \geq 0 & j = 1, \dots, P \\ h_k(c) = 0 & k = 1, \dots, Q \\ c_i^L \leq c_i \leq c_i^U & i = 1, \dots, N \end{cases} \quad (3)$$

Where M is the number of objective functions, f_i is the ith objective function, P, Q, c_i^L, and c_i^U represent the number of inequality design constraints, the number of equality design constraints, the lower bound of the decision variable x_i, and the upper bound of the decision variable x_i.

Many-objective optimization approaches are classified into Single-view software module clustering and Multi-view software module clustering. The following equation shows how $f(c^*)$ is calculated in many-objective optimization approaches.

$$f(c^*) = \begin{cases} \min(f_1(c), f_2(c), \dots, f_M(c)^T) & M > 3 \\ g_j(c) \geq 0 & j = 1, \dots, P \\ h_k(c) = 0 & k = 1, \dots, Q \\ c_i^L \leq c_i \leq c_i^U & i = 1, \dots, N \end{cases} \quad (4)$$

In many-objective optimization, the aim is to optimize more than three design criteria as objective functions at the same time [17] while the multi-objective optimization methods find two or three criteria as objective functions, simultaneously.

The advantages of these approaches include producing more accurate solutions in comparison with the mono-objective optimization approach and improving the estimate of test cycles. In contrast, processing time in these approaches is longer than that in the first approach [13].

- **Mono-objective Optimization:** Scanniello et al. [18] have presented a phased clustering approach based on the combination of structural and lexical dimensions. Structural information is used to decompose the system into horizontal layers and lexical similarity is employed to group each layer. Patel et al. [19] have proposed a clustering method based on static and dynamic analysis to identify the elements in each cluster. In this paper, the researchers used a two-phase clustering technique to combine software features with structural information to refine these clusters.
- **Multi-objective Optimization:** Saeidi et al. [12] proposed a search-based method for software multi-view clustering. This method makes clusters by incorporating knowledge from different viewpoints of the system, such as the source code and structural dependencies within the system. In this research, two techniques were used to combine various views: a linear combination of objective functions and a multi-objective formulation. Praditwong et al. [16] proposed two multi-objective formulations called ECA and MCA to investigate several different objectives including cohesion and coupling for software module clustering. In this paper, a two-archive Pareto optimal genetic algorithm was used to solve the formulations. Barros [20] proposed a multi-objective search-based clustering for software modularization. In this method, ECA formulation [16] was used and simplified. In [21], the software module clustering problem was solved by a new modularization quality measure based on similarity. Huang et al. presented this measure to automatically navigate optimization algorithms to find a suitable partition of software systems by considering both global modules and edge directions. The optimization algorithms used in paper were hill-climbing algorithm, genetic algorithm, and multi-agent evolutionary algorithm. In [22], the researchers focused on space and time constraints of existing modularization and clustering algorithms. To solve these problems, Teymourian et al. proposed a new and fast technique performing operations on the dependency matrix and extracting other matrices based on a set of features. This algorithm is appropriate for both small and large-sized applications. Khalilipour et al. [23] proposed a new algorithm for object-oriented code re-modularization. In this paper, different types of invocations between classes and objects such as synchronous, one-way, or asynchronous were considered in the clustering. The researchers claimed the proposed method could decrease waiting times for service invocations by parallel services running and response times by transferring these services to new clusters. Prajapati [24] proposed a search-based modularization method to improve the software package structure from various perspectives. The researcher used different strategies to introduce harmony search algorithm and objective functions based on the nature of the software package.
- **Many-objective Optimization:** In [25], Bavota et al. proposed an automated clustering method that considered hidden information in the source code and structural dependencies to improve cohesion between the classes into a package. Naseem et al. [26] proposed a cooperative clustering technique for software modularization. This method used some similarity criteria during the hierarchical clustering process for both binary and non-binary features. Pourasghar e al. [6] proposed a new modularization technique called GMA based on the graph. One of the criteria used in this paper to compute similarity between software elements was the depth of relationships. Using this criterion led the algorithm to use graph-theoretic information. Bavota et al. [27] proposed a new technique for automatic re-modularization of packages using both structural and semantic measures to decompose a package into smaller and more cohesive ones.

As mentioned before, unlike previous methods in this paper, various types of structural dependencies are considered to propose a more accurate re-modularization method. Therefore, by investigating these three approaches and their advantages and shortcomings, this paper proposes a many-objective optimization based algorithm. In this method, 4 objective functions are proposed to optimize for re-modularization.

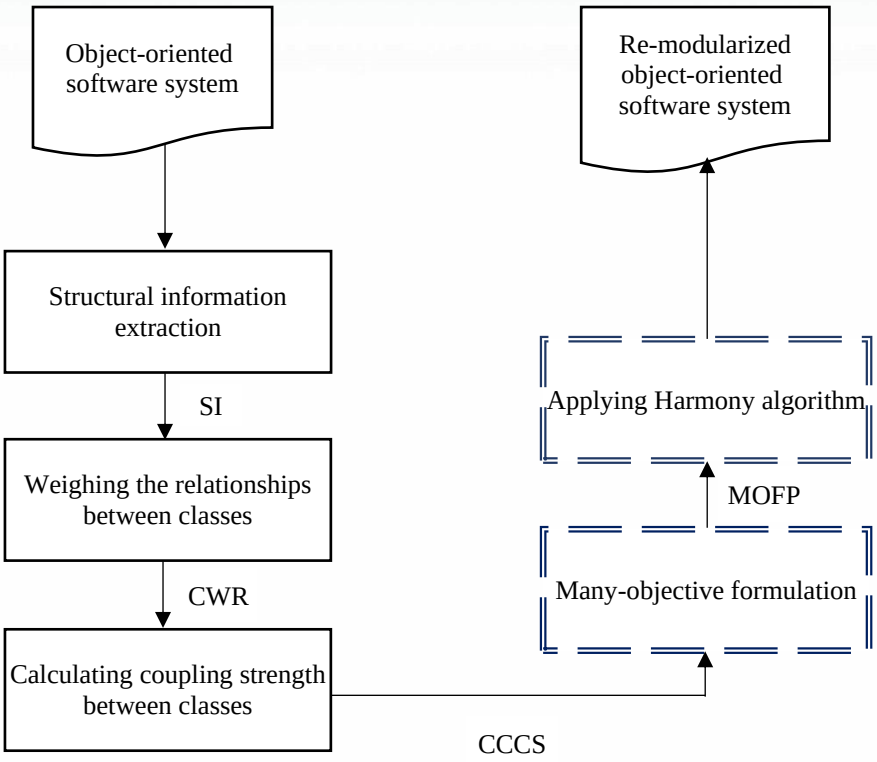

Figure 3. General structure of RMMOF method

IV. RMMOF: THE PROPOSED RE-MODULARIZATION METHOD BASED ON MANY-OBJECTIVE FUNCTION

The purpose of RMMOF method is to receive object-oriented software system and re-modularize it to improve the modular structure of the software system and maintain its quality. As mentioned before, to achieve this aim, various types of dependencies are used in the form of a search-based many-objective fitness function. The general structure of the proposed method is shown in Figure 3. As shown in this figure, RMMOF method includes 5 steps. In the first step, the structural information of the software system source code is extracted from the object-oriented software system. In the second step, using extracted structural information from the first step, each type of relationships between the classes is weighted, as different types of relationships are effective in the proposed method. In the third step, a structural coupling metric is calculated that shows the structural coupling strength between software system classes. The measured values between the classes are saved in an n×n matrix where n implies the number of classes [28]. In the fourth step, re-modularization metrics are formulated by 4 objective functions: (1) to maximize total intra coupling of the modules index, (2) to minimize the total coupling inter modules index, (3) to maximize the isolated packages index, and (4) to maximize the packages size index. In this step, the software re-modularization problem is formulated as an optimization one based on many-objective search. In the last step, this problem is solved by Harmony algorithm-based method called MHS and the re-modularized object-oriented software system is the output of the proposed method.

A. Structural information extraction

As shown in Figure 3, in this phase, the object-oriented software system is the input and the output is structural information (SI). In the software re-modularization process, source code entities are grouped and clustered into the set of modules based on their connecting properties. Modularity principles have a relationship with source code entities connections [28]. The software entities can connect with each other with zero or more types of relationships. Some researchers have considered one of them to re-modularize software systems, but when just one type of structural relationships is considered, re-modularization is often limited to especial aspects. Therefore, to propose a more accurate re-modularization method, improve the modular structure of software system and maintain its quality, there is a need to use various types of structural relationships in the software re-modularization approach. One of the characteristics of the proposed method in this paper (RMMOF) is to use a combination of different examples of various types of structural relationships with their relative strengths. In the RMMOF method, to re-modularize an object-oriented software system, structural connectivity inter packages must be minimized and intra cohesion of the packages must be maximized.

In this paper, we use 8 types of structural relationships as in [28], which are applied and considered in some software architecture modeling tools [29][30], to achieve a structural coupling metric in the third step. Therefore, in this step, these eight relationships are extracted as structural information.

- Extends (EX): an extend relationship implies that a specialized class extends another general class.

- Has parameter (HP): in this type of relationship, a class has a method with a parameter that is of another class type.
- Reference (RE): this relationship implies that a class makes an instance from another class and makes reference to the attributes of the second class using this instance.
- Calls (CA): in this relationship, a class makes an instance from another class and invokes the methods of that class using this instance.
- Implement (IM): this relationship implies that one of the classes realizes or implements one or more functions of another class.
- Is- of- Type (IT): if a class is the type of an instance attribute of another class, this relationship exists between two classes.
- Return (RT): this relationship exists between two classes if a class has a method that returns an object of another one.
- Throws (TH): when a method in a class throws an exception object to an exception handler method, and the handler method exists in another class.

B. *Weighting the relationship between classes*

As shown in Figure 3, SI is the input of this phase and the classes with weighted relationships between them (CWR) are the output. Most coupling metrics have been introduced based on one especial type of relationships, while the presence of more than one type of relationship can help to present a more accurate metric for re-modularization. On the other hand, some metrics that utilize multiple types of relationships do not consider the difference in their relative importance and effectivities [25][31][16][17][10][28].

As mentioned before, according to the proposed method, the objective functions are modeled and formulated based on different dimensions of structural relationships. This formulation can guide the optimization process to achieve an acceptable re-modularization solution. Therefore, to introduce a coupling metric, the relative weights are specified for each type of relationships. To reach this goal, an approach proposed in some studies like [28][32][33][34] is used in this paper.

In this approach, weighting depends on the numbers of inter-module and intra-module samples for each type of relationships. Equation 5 shows the weighting method for each of the relationships in one software system.

$$w_r^i = \begin{cases} 10 & N_{RIN} \neq 0 \wedge N_{ROUT} = 0 \\ 1 & N_{RIN} = 0 \wedge N_{ROUT} = 0 \\ Round(0.5 + 10 \times \dfrac{N_{RIN}}{N_{RIN} + N_{ROUT}}) & otherwise \end{cases} \quad (5)$$

Where w_r is the weight of the r th type of relationship, i is a specific software system, N_{RIN} is the number of inter-module samples for a specific type of relationships (r), and N_{ROUT} is the number of intra-module samples for a specific type of relationships.

As it is possible that there are more than one software system (n) with good quality, they need to be averaged, as shown in Equation 6.

$$w_r = \sum_{i=1}^{n} w_r^i / n \quad (6)$$

Figure 4 shows an example of 3 modules and five classes with different relationships (cohesion or connection).

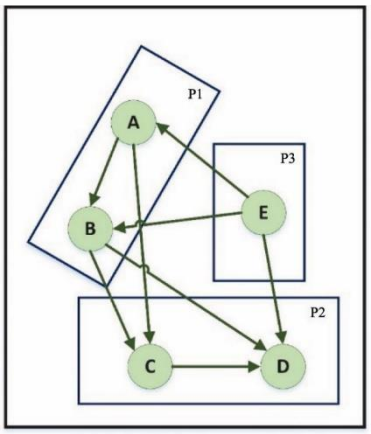

Figure 4. An example of 3 modules and five classes with different relationships [28]

C. *Calculating coupling strength between classes*

As indicated in Figure 3, the input of this phase is CWR and the outputs are the classes with the coupling strength calculated between them (CCCS). The strength of coupling between classes refers to the interdependence between them that affects software system testability and maintenance. Therefore, the strength of coupling influences software quality directly [10]. As mentioned in the previous section, different types of relationships have different weights. Consequently, in this paper we use the weighted term frequency inverse document frequency coupling scheme (WTFIDF) to calculate the structural coupling strength (SCS) between classes. In this method, we utilize calculated weights mentioned in the previous section for 8 types of structural relationships extracted in section 4.1. The results show that WTFIDF has better performance than other methods as the re-modulation solutions produced by this method are more reliable and cover more aspects than other schemes [28].

Equation 7 shows how to calculate the strength of coupling between two classes based on WTFIDF.

$$SCS_{WTFIDF}(c_i, c_j) = \frac{\sum_{r \in R} w_r \times n_r(c_i, c_j)}{\sum_{k=1}^{|C|} \sum_{r \in R} w_r \times n_r(c_k, c_j)} \times \log \frac{|C|}{n_{c_j}} + \frac{\sum_{r \in R} w_r \times n_r(c_j, c_i)}{\sum_{k=1}^{|C|} \sum_{r \in R} w_r \times n_r(c_k, c_i)} \times \log \frac{|C|}{n_{c_i}} \quad (7)$$

Where c_i and c_j are two classes, C is the set of all classes, $|C|$ is the number of all classes in C, R is the set of relationship types (r), n_{c_i} is the number of classes

related to c_i, w_r is the weight of relationship type r, and $n_r(c_i, c_j)$ is the number of samples of relationship type r from c_i to c_j [28].

D. Many-objective formulation

In this phase, the input is CCCS and the output is the many-objective formulated re-modularization problem (MOFP). To navigate the search-based re-modularization algorithm for better re-modularization, it needs to be formulated. As mentioned before, in this paper, four objectives are investigated to obtain a re-modularization method. Therefore, in this part, Pareto optimization is used to study multiple objectives. This method determines if a solution is better than another one or not. In the Pareto optimization method, each set of objectives lead to a different many-objective formula for the problem [16]. Equation 1 shows how to determine re-modularization solution (M^*).

It is important to have more than one non-dominated re-modularization solution. Therefore, to determine the best solution, the following equation needs to be established between the better solution (M_1) and another one (M_2) [16]. Pareto optimization search finds a set of non-dominated re-modularization solutions in this set.

$$F(M_1) > F(M_2) \iff \forall i. F_i(M_1) \geq F_i(M_2) \wedge \exists i. F_i(M_1) > F_i(M_2) \quad (8)$$

In this paper, as mentioned before, the structural aspects of software systems are considered. The purpose of the proposed method is to re-modularize object-oriented classes into the package automatically. The main feature of this re-modularization is software connection reduction and software cohesion increase. To achieve this goal, we use four objective functions: TotalIntraCoupling as in [10], TotalInterCoupling, IsolatedPackageIndex and MaxMinDifferent. These functions are explained in Equations 9-13, respectively.

$$TotalIntraCoupling = \sum_{k=1}^{n} IntraPackageCoupling(P_k) \quad (9)$$

Where,

$$IntraPackageCoupling(P_k) = \frac{\sum_{\forall c_i \in P_k} \sum_{\forall c_j \in P_k, c_i \neq c_j} SCS(c_i, c_j)}{\sum_{i=1}^{|c|} \sum_{j=1}^{|c|} SCS(c_i, c_j)} \quad (10)$$

Where k is the number of packages.

$$TotalInterCoupling = 1 - \left(\frac{TotalIntraCoupling}{\sum_{i=1}^{|c|} \sum_{j=1}^{|c|} SCS(c_i, c_j)} \right) \quad (11)$$

$$IsolatedPackageIndex = 1 - \left(\frac{isolatedPackageNumber}{allPackageNumber} \right) \quad (12)$$

$$MaxMinDifferent = \begin{cases} 0 & maxPackage = minPackage = 1 \\ 1 - \left(\frac{maxPackagesize - minPackagesize}{maxPackage} \right) & Otherwise \end{cases} \quad (13)$$

To evaluate a re-modularization solution, multiplicative aggregate fitness function is used based on the four-quality metrics:

$$Fitness = \text{TotalIntraCoupling} \times \text{TotalInterCoupling} \times \text{IsolatedPackageIndex} \times \text{MaxMinDifferent} \quad (14)$$

E. Applying Harmony Search Algorithm

As shown in Figure 3, MOFP is the input in this phase and the output is re-modularized object-oriented software system. Harmony search (HS) algorithm is a population-based meta heuristic method [35]. This algorithm uses random search where the decision variables do not need to be initialized [10]. Figure 5 shows the flowchart of HS algorithm.

HS algorithm is used in different fields. Some instances are presented in Figure 6. In this paper, this algorithm is applied for software re-modularization.HS algorithm involves 6 steps [37]: (1) encoding the optimization problem and initializing the algorithm parameters, (2) initializing the harmony memory, (3) improvising a new harmony, (4) updating the harmony memory, (5) recording the best solution in HM, and (6) investigating the termination criterion. In this paper, a modified HS algorithm called MHS that supports unusual discrete optimization problem is used based on HSBRA [10] as a re-modularization method. This is because the main algorithm is usually suitable for continuous optimization problems while the re-modularization problem is an unusual discrete optimization problem [10].

To apply the HS algorithm, in the first step of this algorithm (initializing the algorithm parameters), HMCR and PAR are introduced as the following Equations:

$$HMCR(gn) = \frac{(gn + 1)}{NI} \quad (15)$$

$$PAR(gn) = \frac{gn}{NI} \times e^{-\ln\left(\left(\frac{gn+1}{NI}\right) \times gn\right)} \quad (16)$$

Where gn and NI are generation number and maximum number of improvisations, respectively.

In improvising a new harmony step, to select the value of the decision variable from the values stored in the harmonic memory, the main idea in this paper is to use the best solutions in the harmony memory and improve them. Therefore, instead of randomly selecting the decision variables from the harmony memory, the vector with maximum fitness is first extracted from solutions vectors in the harmony memory. Then, dimension j is presented as dimension j in the new solution. This operation is applied to all decision variables of the new decision solution with probability HMCR.

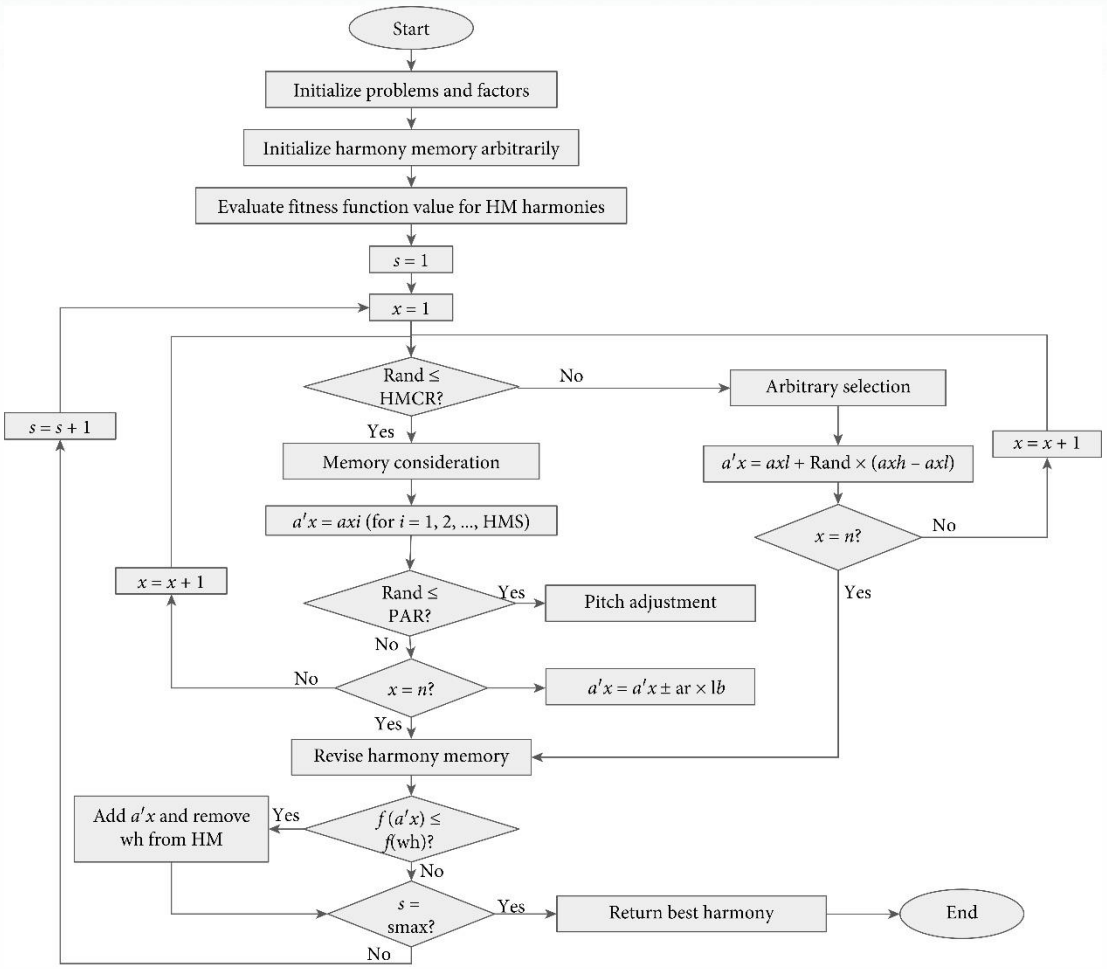

Figure 5. Flowchart of HS algorithm [36]

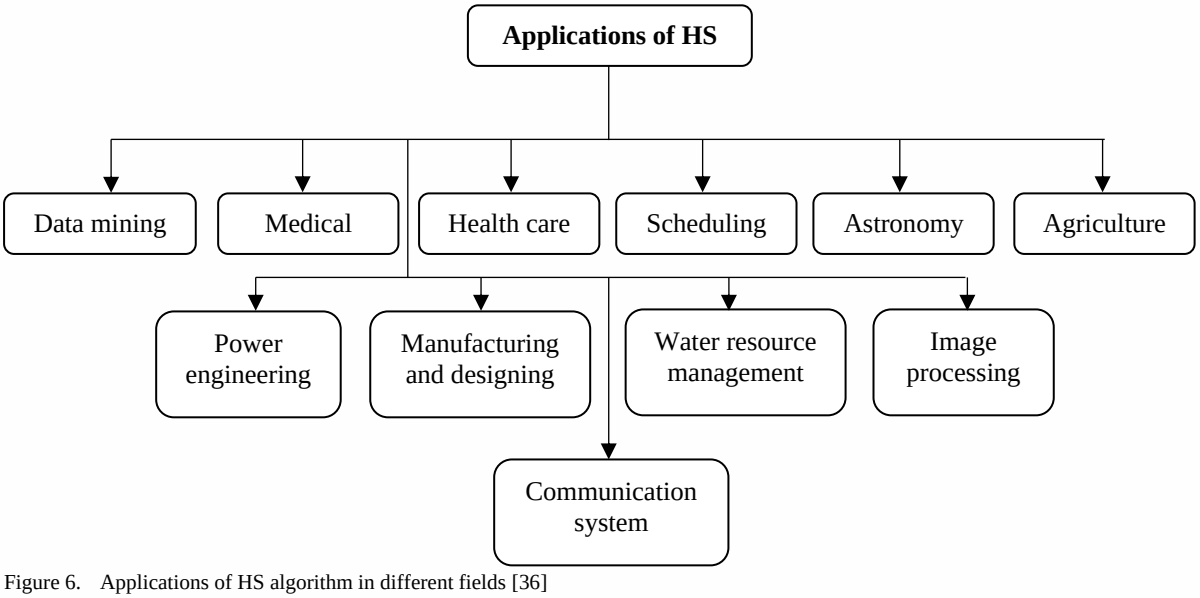

Figure 6. Applications of HS algorithm in different fields [36]

In the proposed method, in addition to the vectors in the harmony memory, the best vectors in the harmony memory can increase the performance of re-modularization.

To create a new value for the decision variable *j*, the basis of the work is to reduce the number of individual packages (the packages with one class) or packages with less than five classes. This increases evaluation criteria NED and MQ. According to the proposed method, after extracting the vector with maximum fitness, the number of individual packages or packages with less than five classes in the vector is calculated. If there are different individual packages (or with less than five classes), the maximum coupling strength between class *j* and the classes in individual

packages (or with less than five classes) is computed. If this number is zero, the proposed system chooses one of the values of the decision variable in the set of {1,2,…,p} (p is the number of packages with 10% classes of all classes in the software systems) as the value of the decision variable j in the new solution. Otherwise, class j in the new solution is placed in an individual package with which it has maximum connection. This operation is applied to all decision variables of the new solution with the probability of 1-HMCR.

Using this method reduces the number of individual packages and the connections between the packages, but increases cohesion. Consequently, MQ is increased, the number of individual packages is reduced, the number of classes in the package is increased, and NED is improved.

To replace the decision variable (c_j^{new}) with its neighbor, according to the MHS method, the first class with maximum SCS compared to c_j^{new} is extracted and replaced with c_j^{new}. If there is no class that is connected with c_j^{new}, the value of decision variable is set to one of the values in {1,2,…n}(n is total number of classes) randomly. These operations are applied to c_j^{new} with the probability PAR and $c_j^{new}$is not changed with the probability 1-PAR. The computational procedure of the proposed MHS is summarized in Figure 7 in the form of a pseudo code.

V. EXPERIMENTS

This section includes four main subsections. In the first part, the data set used in this paper for testing the proposed system is explained. In the second part, the evaluation criteria are introduced. In the third part, the parameters in the algorithms used and their values are expressed. In the last part, the results of the proposed system testing are provided. In this part, the results obtained are analyzed and compared with the results of other methods based on the evaluation criteria.

A. Dataset

In this research, a software set is used as dataset. The features of these software systems are expressed in Table 2. The software systems are java language source codes and open sources. These software systems have been widely used by some researchers to evaluate their proposed methods like [10][28][17][20][38]. As shown in Table 1, the super classes and packages number in each software is different from others and are in wide ranges. These features were extracted by STAN tool. The general schemes of the software systems used are shown in Figures 8-10.

TABLE I. THE CHARACTERISTICS OF THE SOFTWARE SYSTEMS USED

Software System	Version	Super classes number	Packages number
Junit	3.8.1	47	6
Java Servlet API	2.3	63	4
DOM 4J	1.5.2	170	16

Algorithm: MHS Algorithm

Step1: Set Parameters: HMS, HMCR, PAR
Step2: Initialize Harmony Memory (HM)
  $For\ (i = 1\ to\ HMS)\ do$ //HMS is the size of harmony memory
    $If\ (i <= HMS/2)\ then$
    $For\ (j = 1\ to\ n)\ do$ //n is the total number of classes
      $c_j^i \leftarrow Randint(UB^i - LB^i)$/*select random integer value between $UB^i = n\ \ and\ \ LB^i = 0$*/
    $End\ for$
  $Else$
    $For\ (j = 1\ to\ n)\ do$ // n is the total number of classes
      $c_j^i \leftarrow Randint(UB^i - LB^i)$ /*select random integer value between $UB^i = (n * 10)/100\ \ and\ \ LB^i = 0$ */
    $End\ for$
  $End\ if$
  $f(c^i) = Evaluate(c^i)$
$End\ for$
Step3: Improvisation harmony c^{new}
  $For\ (j = 1\ to\ n)\ do$ //n is the total number of classes
    $If\ (r_1 < HMCR)\ then$ //r_1 is a uniform random number in the range of [0,1]
      $c_j^{new} = c_j^{BestFitness},\ BestFitness \in (1, \dots, HMS)$
      $If\ (r_2 < PAR)\ then$ //r_2 is a uniform random number in the range of [0,1]
      $c_j^{new} = GetNeighbour(BestFitness, j)$ //c_j^{new} is replaced by its neighbor
    $End\ if$
  $Else$
      $c_j^{new} = GetSolution(BestFitness)$
  $End\ if$
  $End\ for$
  $f(c^{new}) = Evaluate(c^{new})$ //Perform fitness computation
Step4: Update the Harmony memory
  $If\ (f(c^{new}) > f\ (c^{worst}))\ then\ update\ the\ HM\ as\ c^{worst} = c^{new}$
Step5: $Record\ the\ best\ solution\ in\ HM$
Step6: $If\ (satisfy\ stopping\ criteria)\ then$
  $end\ the\ procedure\ and\ return\ best\ solution\ in\ HM$
$else$
  $return$ **step3**

Figure 7. The computational procedure of the proposed MHS

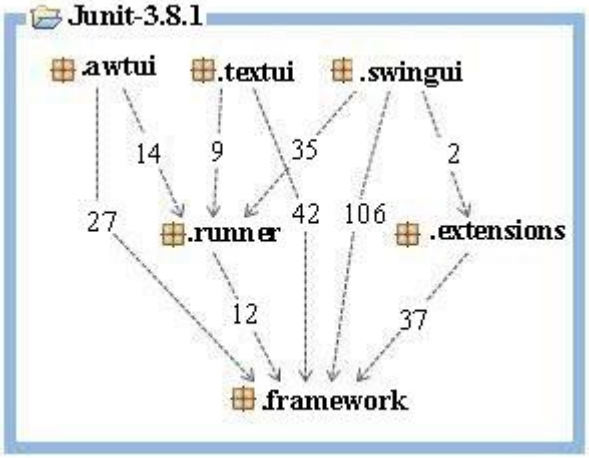

Figure 8. The general scheme of Junit software

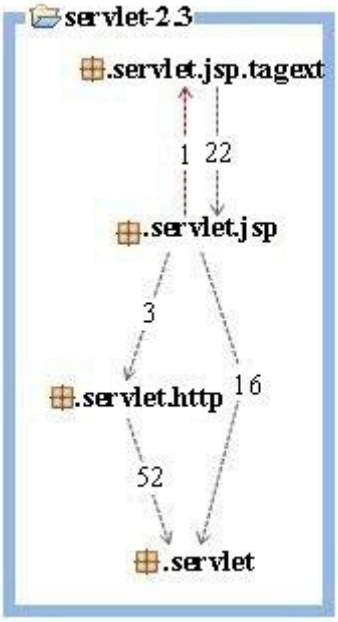

Figure 9. The general scheme of Java Servlet API software

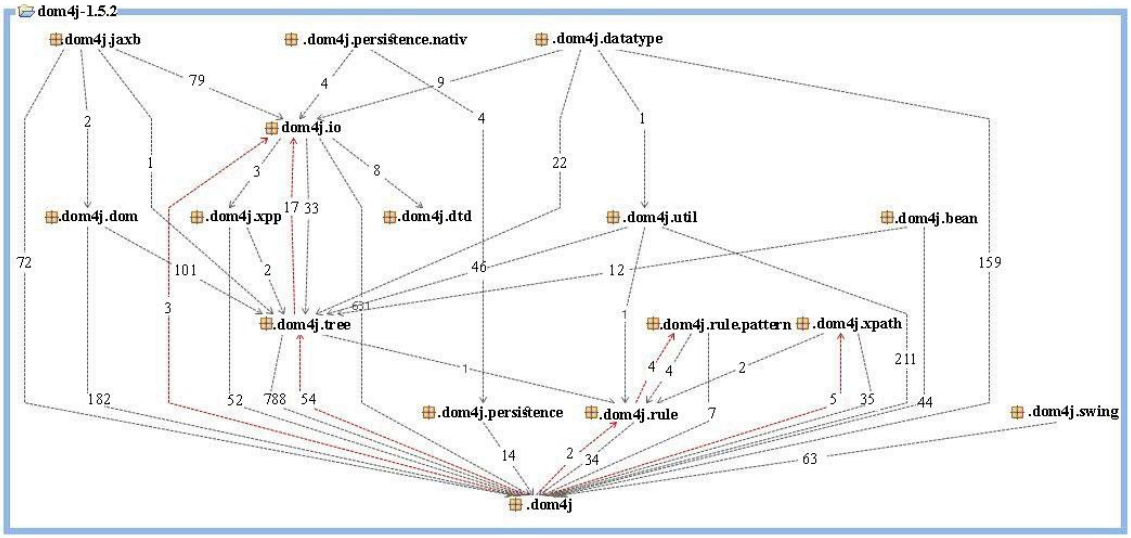

Figure 10. The general scheme of DOM 4J software

B. Evaluation Criteria

In the field of re-modularization, there are standard criteria to test different methods. Therefore, to investigate the performance of the proposed method and evaluate it, in addition to the multiplicative aggregate fitness function obtained from four objective functions, two evaluation criteria MQ [4] and NED [39] are used in the present paper.

Modularization quality (MQ) tradeoffs between inter-connectivity and intra-connectivity [4]. To calculate MQ, Cluster Factor (CF) needs to be obtained for each cluster (i). Equation 17 shows how CF is calculated.

$$CF_i = \begin{cases} 0 & \mu_i = 0 \\ \dfrac{2\mu_i}{2\mu_i + \sum_{\substack{j=1 \\ i \neq j}}^{k} (\varepsilon_{i,j} + \varepsilon_{j,i})} & Otherwise \end{cases} \quad (17)$$

Where i and j are two different clusters and k is the clusters number, μ_i is the intra-cluster coupling strength of i cluster (it means intra edges of cluster i), $\varepsilon_{i,j}$ and $\varepsilon_{j,i}$ are coupling strengths for the relation that originates in cluster i and terminates in cluster j, and coupling strength for the relation that originates in cluster j and terminates in cluster i, respectively.

MQ is obtained from the sum of CF_i for all clusters in the software (Equation 18).

$$MQ = Sum_{i=1}^{k} CF_i \quad (18)$$

More MQ shows better re-modularization.

Software re-modularization must not create very small or large modules, as they are not normal and lead to increased connections between packages and reduced package cohesion. Therefore, Non-Extreme Distribution (NED) criteria measure the extremity of module distribution. NED is introduced in Equation 19.

$$NED = \frac{\sum_{i=1, M_i\ not\ extreme}^{k} |M_i|}{n}$$

$$M_i\ is\ not\ extreme\ if\ \ 5 < |M_i| < 1.5 \times |M_{max}| \quad (19)$$

Where $|M_i|$ is the size of module i, k is the modules number, n is the number of all classes in the system, $|M_{max}|$ is the size of the largest module in the current organization of software modules.

C. Parameters Settings

To evaluate our method, we have compared it with three popular methods, HSBRA [10], GA and HC. But before comparison, it needs to justify the parameters of each algorithm. To set them, we have followed previous research like [10][16][40][41][42]. Table 2 shows the parameters and their values used in this paper. In addition, we choose the number of fitness evaluations (N_{FE}) for fair comparison amongst meta-heuristic algorithms. For all the three considered algorithms, we set $N_{FE} = 20000$ as stopping criteria. To achieve this purpose, HMS is set to 50 and the number of improvisations is set to 400. In RMMOF, HMS is set to 20 and the number of improvisations is set to 1000 for HSBRA. For the GA algorithm, population size and the number of generations are set to 20 and 1000, respectively. Finally, in the HC algorithm, the number of iterations is set to 20000.

TABLE II. PARAMETERS VALUES OF DIFFERENT ALGORITHMS

TABLE III. LGORITHM	TABLE IV. ARAMETER	TABLE V. YMBOL	TABLE VI. ALUE
RMMOF	Harmony Memory Size	HMS	50
	Harmony Memory Consideration Rate	$HMCR_{min}$	0.7
		$HMCR_{max}$	0.99
	Pitch Adjustment Rate	PAR_{min}	0.01
		PAR_{max}	0.99
	Number of Improvisation	NI	400
HSBRA	Harmony Memory Size	HMS	20
	Harmony Memory Consideration Rate	$HMCR_{min}$	0.7
		$HMCR_{max}$	0.99
	Pitch Adjustment Rate	PAR_{min}	0.01
		PAR_{max}	0.99
	Number of Improvisation	NI	1000
GA	Crossover Probability	P_{cr}	0.80
	Mutation Probability	P_{mt}	0.15
	Population Size	PS	20
	Number of Generation	NG	1000
HC	Number of Iteration	NI	20000

D. Experiments Results

To evaluate the proposed method, two tests have been designed and run. In Test 1, the effect of different meta-heuristic algorithms on re-modularization in RMMOF method is investigated based on MQ and NED. In this test, in addition to comparing various meta-heuristic algorithms, MHS algorithm is run on 3 different datasets. In Test 2, the proposed method RMMOF is compared with the methods proposed in [10] based on Fitness, MQ and NED.

1) Test 1: the effect of different meta-heuristic algorithms in RMMOF on re-modularization based on Fitness, MQ and NED

In this part, the effects of the proposed meta-heuristic algorithm MHS, GA and HC on re-modularization are investigated. The purpose of this test is to investigate the performance of re-modularization using different meta-heuristic algorithms based on the criteria introduced in the previous subsection. Figures 11-13 show the result of this test in different systems. To examine the proposed method in different systems and find the application in which this method has the highest efficiency, the experimental results of the MHS

are also reported based on different software systems, as shown in Figure 14.

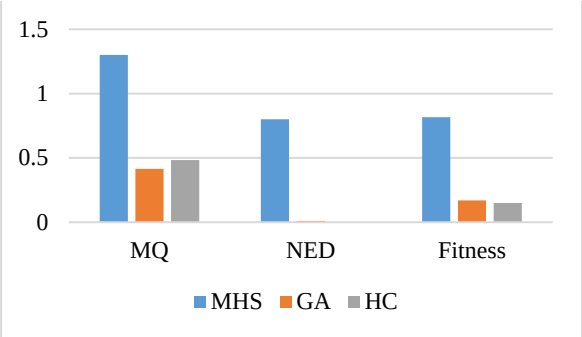

Figure 11. The result of Test1 on Junit system

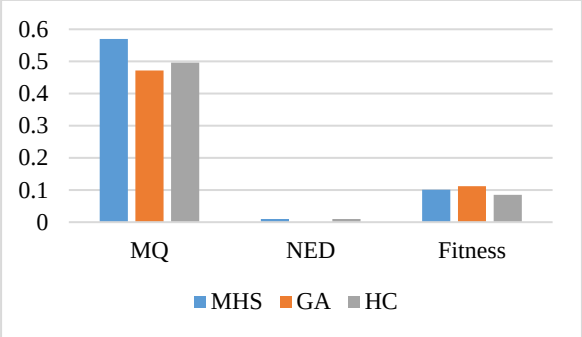

Figure 12. The result of Test1 on Java Servlet API system

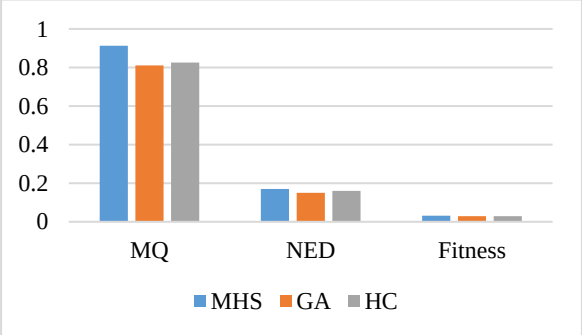

Figure 13. The result of Test1 on DOM 4J system

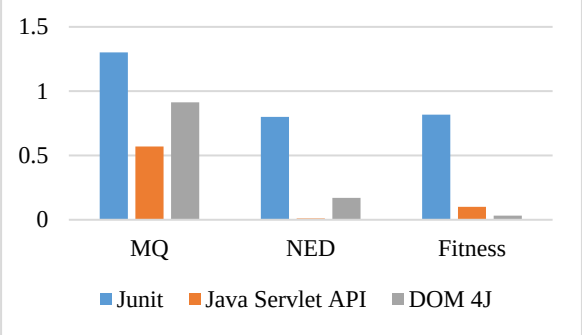

Figure 14. The result of Test1 on MHS

Discussion

As shown in Figure 11, the proposed method MHS has better performance than others based on MQ, NED and Fitness on Junit system. This means that using Harmony search-based algorithm with the proposed changes enhances the efficiency of re-modularization and decreases extreme modules. Therefore, in this algorithm, the fitness obtained from the four proposed objective functions is also remarkably higher. In contrast, using GA and HC reduces the NED to a great extent. This means that the size of the modules created is very heterogeneous and different from each other. As for the other criteria, the results obtained from using these two algorithms are approximately similar. By investigating the formulations of the three criteria and their values for each algorithm, we observed that if both MQ and NED are high, Fitness criteria are enhanced. However, if the algorithm has less efficiency based on MQ and NED, Fitness is reduced.

As shown in Figure 12, the value of MQ in the proposed method is higher compared to others, while the values of NED and Fitness in the three algorithms on Java Servlet API system are approximately similar. In addition, these results show that the proposed method has a much poorer performance on the Java Servlet API system than Junit. However, the performances of GA and HC on Java Servlet API system and Junit are almost the same.

According to Figures 11 and 12, it seems that HC and GA are not able to distribute the classes with a homogenous size based on Junit and Java Servlet API which are small applications in terms of size.

As shown in Figure 13, the performance of the proposed method based on the three criteria is better compared to others on DOM 4J system. The performance of GA and HC is approximately similar. It is clear that the proposed method has been able to re-modularize the system (MQ) better than reducing the size of modules (NED).

By considering all three figures, it can be concluded that the proposed method has a better performance on all three datasets than others and MHS algorithm is more suitable on the first dataset (Junit system) among these three datasets. The results are shown in Figure 14.

As explained before, to compare the results of Test 1 in MHS based on different systems, the results in Figure 14 show that the proposed algorithm has the best performance on Junit based on all the proposed criteria. This indicates that the proposed method is suitable for the small applications. The reason for obtaining a higher MQ and NED on DOM 4J compared to Java Servlet API, despite the smaller size of Java Servlet API, is the type of dependencies in Java Servlet API software.

2) Test 2: the comparison between the proposed re-modularization method RMMOF and the methods proposed in [10] *based on MQ and NED*

To evaluate the proposed method, we should compare it with other similar methods. Therefore, in this part, the proposed method (RMMOF) is compared with the 4 methods proposed in [10]. In [10], four Harmony search-based re-modularizations have been proposed based on linear and exponential changes in Harmony Memory Consideration Rate (HMCR) and Pitch Adjusting Rate (PAR). Figures 15 and 16 show the results of these comparisons based on MQ and NED.

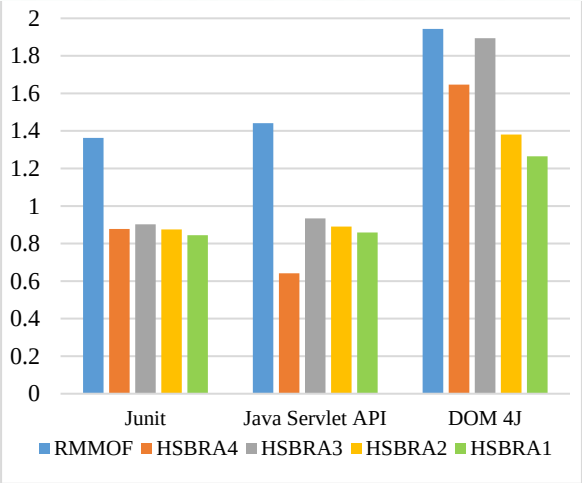

Figure 15. The result of Test2 based on MQ

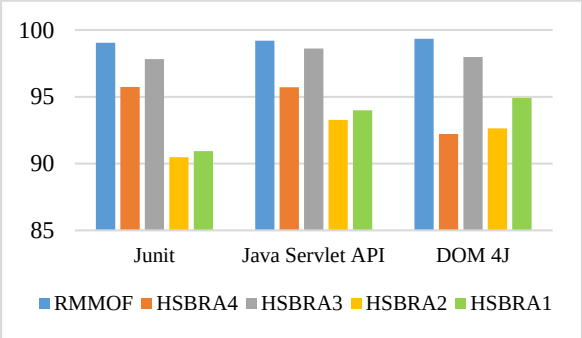

Figure 16. The result of Test2 based on NED

Discussion

As shown in Figure 15, the proposed method has the best MQ on Junit, Java Servlet API and DOM 4J systems, while RMMOF has the best result on DOM 4J among the three data sets. As shown in this figure, the improvement of the proposed method in comparison to other methods is more obvious on Junit system. Among the proposed methods in [10], HSBRA3 has the best MQ on all the three systems. In HSBRA3, HMCR is changed linearly and PAR is changed exponentially during the improvisation. The important difference between RMMOF and HSBRA is the use of modified harmony algorithm (MHS) and using the proposed fitness function with four different and effective objective functions. These improvements seem to lead to better results in terms of MQ. What is clear in this figure is that the improvement rate of the proposed method on small applications is more evident as compared to the methods proposed in [10].

As indicated in Figure 16, the results of applying RMMOF on re-modularization are better compared to others. Applying the proposed improvements also has positive impacts on NED. In this test, HSBRA3 has better performance compared to the other methods proposed in [10].

It seems using various structural relationships and suitable relative weights for them helps to maximize the total intra coupling of the modules and minimize the total coupling inter modules. In addition, using MHS algorithm as a meta-heuristic algorithm and new fitness function is effective to obtain the best MQ and NED.

VI. Conclusion

Although most software systems are designed and developed modularly at first, modularity is degraded over time. Re-modularization is used to improve the modular structure of software system. In this paper, the proposed method recognizes various dependencies in terms of an objective function. In this method, a search-based many-objective fitness function is proposed to formulate re-modularization as an optimization problem. To solve these objective functions, a helpful harmony-based algorithm called MHS has been used. The experiments and comparison results have shown the efficiency of the proposed method in re-modularization compared to other methods.

VII. References

[1] V. Lenarduzzi, A. Sillitti, and D. Taibi, “Analyzing forty years of software maintenance models,” in 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), 2017, pp. 146–148.

[2] M. Gupta, A. Serebrenik, and P. Jalote, “Improving software maintenance using process mining and predictive analytics,” in 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2017, pp. 681–686.

[3] A. H. F. Tabrizi and H. Izadkhah, “Software modularization by combining genetic and hierarchical algorithms,” in 2019 5th Conference on Knowledge Based Engineering and Innovation (KBEI), 2019, pp. 454–459.

[4] S. Mancoridis, B. S. Mitchell, C. Rorres, Y. Chen, and E. R. Gansner, “Using automatic clustering to produce high-level system organizations of source code,” in Proceedings. 6th International Workshop on Program Comprehension. IWPC’98 (Cat. No. 98TB100242), 1998, pp. 45–52.

[5] L. Mu, V. Sugumaran, and F. Wang, “A hybrid genetic algorithm for software architecture re-modularization,” Inf. Syst. Front., vol. 22, no. 5, pp. 1133–1161, 2020.

[6] [6] B. Pourasghar, H. Izadkhah, A. Isazadeh, and S. Lotfi, “A graph-based clustering algorithm for software systems modularization,” Inf. Softw. Technol., vol. 133, p. 106469, 2021.

[7] M. Kargar, A. Isazadeh, and H. Izadkhah, “Improving the modularization quality of heterogeneous multi-programming software systems by unifying structural and semantic concepts,” J. Supercomput., vol. 76, no. 1, pp. 87–121, 2020, doi: 10.1007/s11227-019-02995-3.

[8] A. Isazadeh, H. Izadkhah, and I. Elgedawy, Source code modularization: theory and techniques. springer, 2017.

[9] 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.

[10] J. K. Chhabra and others, “Harmony search based remodularization for object-oriented software systems,” Comput. Lang. Syst. \& Struct., vol. 47, pp. 153–169, 2017.

[11] B. S. Mitchell and S. Mancoridis, “On the automatic modularization of software systems using the bunch tool,” IEEE Trans. Softw. Eng., vol. 32, no. 3, pp. 193–208, 2006.

[12] A. M. Saeidi, J. Hage, R. Khadka, and S. Jansen, “A search-based approach to multi-view clustering of software systems,” in 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), 2015, pp. 429–438.

[13] F. Morsali and M. R. Keyvanpour, “Search-based software module clustering techniques: A review article,” in 2017 IEEE 4th International Conference on Knowledge-Based Engineering and Innovation (KBEI), 2017, pp. 977–983.

[14] J. Hwa, S. Yoo, Y.-S. Seo, and D.-H. Bae, “Search-based approaches for software module clustering based on multiple relationship factors,” Int. J. Softw. Eng. Knowl. Eng., vol. 27, no. 07, pp. 1033–1062, 2017.

[15] J. K. Chhabra and others, “Many-objective artificial bee colony algorithm for large-scale software module clustering problem,” Soft Comput., vol. 22, no. 19, pp. 6341–6361, 2018.

[16] 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, 2010.

[17] W. Mkaouer et al., “Many-objective software remodularization using NSGA-III,” ACM Trans. Softw. Eng. Methodol., vol. 24, no. 3, pp. 1–45, 2015.

[18] 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 2010 IEEE 18th International Conference on Program Comprehension, 2010, pp. 180–189.

[19] C. Patel, A. Hamou-Lhadj, and J. Rilling, “Software clustering using dynamic analysis and static dependencies,” in 2009 13th European Conference on Software Maintenance and Reengineering, 2009, pp. 27–36.

[20] M. de O. Barros, “An analysis of the effects of composite objectives in multiobjective software module clustering,” in Proceedings of the 14th annual conference on Genetic and evolutionary computation, 2012, pp. 1205–1212.

[21] J. Huang and J. Liu, “A similarity-based modularization quality measure for software module clustering problems,” Inf. Sci. (Ny)., vol. 342, pp. 96–110, 2016.

[22] N. Teymourian, H. Izadkhah, and A. Isazadeh, “A fast clustering algorithm for modularization of large-scale software systems,” IEEE Trans. Softw. Eng., 2020.

[23] A. Khalilipour and M. Challenger, “Automatic Re-modularization of Clustered Codes Considering Invocation Types,” in 2021 7th International Conference on Web Research (ICWR), 2021, pp. 109–113.

[24] A. Prajapati, “Software Package Restructuring with Improved Search-based Optimization and Objective Functions,” Arab. J. Sci. Eng., pp. 1–21, 2021.

[25] G. Bavota, M. Gethers, R. Oliveto, D. Poshyvanyk, and A. de Lucia, “Improving software modularization via automated analysis of latent topics and dependencies,” ACM Trans. Softw. Eng. Methodol., vol. 23, no. 1, pp. 1–33, 2014.

[26] R. Naseem, O. Maqbool, and S. Muhammad, “Cooperative clustering for software modularization,” J. Syst. Softw., vol. 86, no. 8, pp. 2045–2062, 2013.

[27] G. Bavota, A. De Lucia, A. Marcus, and R. Oliveto, “Using structural and semantic measures to improve software modularization,” Empir. Softw. Eng., vol. 18, no. 5, pp. 901–932, 2013.

[28] J. K. Chhabra and others, “Improving modular structure of software system using structural and lexical dependency,” Inf. Softw. Technol., vol. 82, pp. 96–120, 2017.

[29] “http://www.stan4j.com/.”

[30] “http://www.structure101.com/.”.

[31] H. Abdeen, S. Ducasse, and H. Sahraoui, “Modularization metrics: Assessing package organization in legacy large object-oriented software,” in 2011 18th Working Conference on Reverse Engineering, 2011, pp. 394–398.

[32] F. B. e Abreu, G. Pereira, and P. Sousa, “A coupling-guided cluster analysis approach to reengineer the modularity of object-oriented systems,” in Proceedings of the fourth european conference on software maintenance and reengineering, 2000, pp. 13–22.

[33] F. B. e Abreu and M. Goulao, “Coupling and cohesion as modularization drivers: Are we being over-persuaded?,” in Proceedings Fifth European Conference on Software Maintenance and Reengineering, 2001, pp. 47–57.

[34] C. Y. Chong and S. P. Lee, “Analyzing maintainability and reliability of object-oriented software using weighted complex network,” J. Syst. Softw., vol. 110, pp. 28–53, 2015.

[35] X. Wang, X.-Z. Gao, and K. Zenger, “The overview of harmony search,” in An introduction to harmony search optimization method, Springer, 2015, pp. 5–11.

[36] M. Dubey, V. Kumar, M. Kaur, and T.-P. Dao, “A systematic review on harmony search algorithm: theory, literature, and applications,” Math. Probl. Eng., vol. 2021, 2021.

[37] M. Mahdavi, M. Fesanghary, and E. Damangir, “An improved harmony search algorithm for solving optimization problems,” Appl. Math. Comput., vol. 188, no. 2, pp. 1567–1579, 2007.

[38] U. Erdemir and F. Buzluca, “A learning-based module extraction method for object-oriented systems,” J. Syst. Softw., vol. 97, pp. 156–177, 2014.

[39] J. Wu, A. E. Hassan, and R. C. Holt, “Comparison of clustering algorithms in the context of software evolution,” in 21st IEEE International Conference on Software Maintenance (ICSM’05), 2005, pp. 525–535.

[40] V. Kumar, J. K. Chhabra, and D. Kumar, “Parameter adaptive harmony search algorithm for unimodal and multimodal optimization problems,” J. Comput. Sci., vol. 5, no. 2, pp. 144–155, 2014.

[41] J. Brest, S. Greiner, B. Boskovic, M. Mernik, and V. Zumer, “Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems,” IEEE Trans. Evol. Comput., vol. 10, no. 6, pp. 646–657, 2006.

[42] A. Farrugia, “Vertex-partitioning into fixed additive induced-hereditary properties is NP-hard,” arXiv Prepr. math/0306158, 2003.

Mohammad Reza Keyvanpour is a Professor at Alzahra University, Tehran, Iran. He received his B.Sc. degree in Software Engineering from Iran University of Science & Technology, Tehran, Iran. He received his M.Sc. and Ph.D. degrees in Software Engineering from Tarbiat Modares University, Tehran, Iran. His research interests include Software Engineering and Data Mining.

Zahra Karimi Zandian received her B.Sc. degree in Software Engineering from Islamic Azad University, South Tehran Branch, Tehran, Iran. She also received her M.Sc. degree in Software Engineering from Alzahra University, Tehran, Iran. Her research interests include Data Mining, Machine Learning, Software Engineering, Fraud Detection and Social Network Analysis.

Email: z.karimizandian@yahoo.com

Fatemeh Morsali received her M.Sc. in Software Engineering from Alzahra University, Tehran, Iran. Her research interests include Software Engineering, Machine Learning and Data Mining.