Information Systems Frontiers (2020) 22:863–880
https://doi.org/10.1007/s10796-019-09897-y

Information-Theoretic Remodularization of Object-Oriented Software Systems

Amarjeet Prajapati [1] · Jitender Kumar Chhabra [2]

Published online: 25 January 2019

Abstract
Software remodularization consists in reorganizing software entities into modules such that pairs of entities belonging to the same modules are more similar than those belonging to different modules. In recent years, Search-Based Software Engineering (SBSE) approach has gained unprecedented growth for solving software remodularization problem. Most of the previous studies remodularize the software system by optimizing the structural coupling and cohesion metrics as objective functions. These metrics are defined in terms of the number of structural relationships counts, rather than taking patterns of relationships. It has been observed that the computation of coupling and cohesion based on patterns of relationships (i.e., information-theory based) are more accurate than the number of relationships. This paper proposes an information-theoretic software remodularization where an entropy-based similarity measure is introduced as an objective function along with other objective functions i.e., inter-module class change coupling, intra-module class change coupling, module size index (MSI), and module count index (MCI) and is further optimized using many-objective meta-heuristic algorithm. To evaluate the effectiveness of the proposed approach, seven object-oriented software systems have been remodularized using NSGA-III, MOEA/D, IBEA, and TAA algorithms. The results are compared with existing multi-objective formulation of remodularization problem in terms of authoritative software remodularization, non-extreme distribution, and stability. The experimentation results suggest that the proposed approach can be a good alternative to improve the quality of software systems. The findings suggest that the approach is more suitable for generating remodularization solution good from both quality metrics and developers perspective.

Keywords Search- based software engineering · Software remodularization · Software entropy · Information theoretic technique

1 Introduction

In order to keep pace with ever-changing user, business and technological requirements, the source code of a software system often needs to be changed. It has been observed that the short deadlines of project delivery, budget constraints, and unfamiliarity of existing source code generally forces developers' to focus on functionality rather than design structure (Fowler et al. 1999; Mancoridis et al. 1998). Such maintenance practices increase the complexity of design structure and degrade software quality. The software system with poor design quality is difficult to understand and evolve. The problem becomes more difficult in case of highly convoluted software design with the unavailability of their up-to-date documentation as well as original developers (Mkaouer et al. 2015).

There are many source code anomalies that contribute in degradation of design quality of a software system. In an object-oriented software system, the suboptimal placement of source code classes into packages is one of the crucial anomalies that cause the degradation of design quality (Bavota et al. 2014). To improve the design quality of software system various software remodularization approaches based on deterministic and search-based techniques have been proposed (e.g., Praditwong et al. 2011; Barros 2012; Prajapati and Chhabra 2017, 2017a; Parashar et al. 2016; Corazza et al. 2016; Bavota et al. 2010, 2013; Prajapati and Chhabra 2017b; Mkaouer et al. 2015a; Kumari et al. 2013; Doval et al. 1999; Prajapati and Chhabra 2014). The deterministic based software remodularization approaches perform well

✉ Amarjeet Prajapati
amarjeetnitkkr@gmail.com

Jitender Kumar Chhabra
jitenderchhabra@gmail.com

[1] Department of Computer Engineering & IT, JIIT Noida, Noida, UP, India

[2] Department of Computer Engineering, NIT Kurukshetra, Kurukshetra, Haryana, India

for small size software, however, for large and complex software they become impractical and sometimes infeasible (Mancoridis et al. 1998; Harman et al. 2012).

In case of a large and complex software, search-based software engineering (SBSE) (Harman et al. 2012) approach has been found as a good alternative for solving a software remodularization problem (Prajapati and Chhabra 2017; Bavota et al. 2013; Prajapati and Chhabra 2017b; Mkaouer et al. 2015a). The major advantage of using SBSE approach is that it guarantees in the generation of near-optimal solution within a reasonable amount of time. The effectiveness of SBSE based software remodularization approaches depends on many factors such as search algorithms, fitness and objective function formulations, etc. However, fitness and objective function formulation are the two most important factors that help in driving the remodularization process towards good solutions (Bavota et al. 2014; Anquetil and Lethbridge 1999). Hence, in order to achieve a good quality remodularization solution appropriate formulation of fitness and objective functions need to be incorporated in search algorithms.

In last two decades, unprecedented efforts have been put forward in designing fitness and objective functions along with search algorithms to solve the different aspects of single and multi-objective software remodularization problems (e.g., Kumari et al. 2013; Prajapati and Chhabra 2014, 2018, 2018a). Majority of the fitness and objective functions for software remodularization are designed in terms of direct link coupling of software artefacts, such as method calls or inheritance (Mancoridis et al. 1998; Praditwong et al. 2011; Barros 2012; Prajapati and Chhabra 2017, 2017a). However, some researchers (Parashar and Chhabra 2016; Corazza et al. 2016) have attempted to design the objective functions by analyzing the sibling link similarity based on lexical and changed information. Some other researchers have tried to combine the structural based direct link similarity with lexical based sibling link similarity in their remodularization techniques (Bavota et al. 2010, 2013, 2014; Prajapati and Chhabra 2017a). Other researchers have also attempted to design the objective functions by combining the changed history information with structural information and lexical information (Mkaouer et al. 2015a).

Although the existing fitness and objective functions designed for search-based software remodularization approaches have been reported to be quite effective, the major limitation of such approaches is that they are not able to generate the remodularization solution that is meaningful from developers' perspective (Mkaouer et al. 2015a) The main reason is that such remodularization fitness and objective functions do not comply with the perspective of developers. Therefore, to generate a remodularization solution that is meaningful from developers' perspective, we need to improve our remodularization fitness and objective functions that must consider factors that convey the developers' perception along with optimization of design principles.

Further, the existing software remodularization approaches commonly treat all sources of information of entities to be modularized equally (i.e., presence or absence of a feature) to determine the fitness and objective functions. However, software developers usually give different importance to different types of features while fitness and objective functions (Bavota et al. 2013a). Therefore, the fitness and objective functions used for remodularization evaluation should consider different dimensions of information with their relative importance. Most of the search-based software remodularization approaches except (Mkaouer et al. 2015a) ignore the changed history dependency information. However, the changed history information may reveal many dependencies among software components which cannot be observed by the structural or lexical based information. The study (Bavota et al. 2013a) showed that the changed history information is also one of the factors that some extent reflects the developers' perception of coupling. Therefore, remodularization fitness and objective functions should also consider the changed-history information along with the structural and lexical information.

To address the above-discussed issues, this paper introduces a multi-objective formulation of object-oriented remodularization problem where entropy-based similarity measure along with inter-module class change coupling, intra-module class change coupling, module size index (MSI), and module count index (MCI) have been used as objective functions. In this contribution, the remodularization objective functions use different types of structural as well as lexical information that captures the developers' aspects of coupling in the computation of similarity measure. Moreover, the approach uses different types of structural and lexical information with their relative importance. However, relative weights of different dimensions of information are subjective in nature and depend on many factors (e.g., quality measurement goal). To deal with this, this paper uses term frequency-inverse document frequency (TFIDF) (Yates and Neto 1999; Corazza et al. 2016) to compute the weight. Using the different dimensions of structural and lexical information, an information theoretic similarity measure (i.e., entropy-based similarity measure) has been used. The information theoretic concepts have been successfully applied to other unsupervised machine learning approaches (e.g., data clustering) (Gokcay and Principe 2002; Sugiyama et al. 2014; Andritsos and Tzerpos 2005). The entropy measures uncertainty about a random event, which can be used to design a remodularization criterion for restructuring the packages of software systems. In fact, when we assign a class to one of the different modules we incur an entropy cost. Minimizing this incremental entropy cost could be an effective evaluation criterion for software remodularization. The model also exploits change-history information stored in the version repository. In particular, the approach extracts the changed dependencies between the classes and uses them to make the remodularization solution consistent with changed history. The major idea of

using such dependencies in remodularization is to force the remodularization process towards a solution where classes changing together should be grouped together.

The organization of the rest of this paper is as follows: Section 2 presents related works. Section 3 provides background from information theory and structural/lexical based coupling computation. Section 4 discusses the proposed approach. Section 5 presents an experimental setup. Section 6 presents results and discussion. Section 7 concludes with future directions.

2 Related Works

Automatic remodularization of software systems has become an interesting application for SBSE, where a different aspect of software remodularization problems are simulated as search-based optimization problems (e.g., such as single, multi or many-objective optimization) and are solved using search-based meta-heuristics (Harman et al. 2012). The main attraction of SBSE towards software remodularization is that the combinatorial and NP-hard nature of software remodularization problem makes SBSE approaches best alternative (Praditwong et al. 2011). Recently, many researchers have applied various SBSE approaches by adopting different metaheuristics and single/multi-objective formulations to address the different aspects of software remodularization problems (Praditwong et al. 2011; Ouni et al. 2013, 2014, 2015; Prajapati and Chhabra 2017a; Kumari et al. 2013; Ouni et al. 2016, 2016a, b , 2017; Mancoridis et al. 1998).

In previous literature, the software remodularization problem has been defined in different ways according to different aspect of software restructuring. For example, 1) number of objectives: single-objective remodularization (Mancoridis et al. 1998, 1999; Doval et al. 1999) multi-objective remodularization (Praditwong et al. 2011; Barros 2012; Kumari et al. 2013; Prajapati and Chhabra 2014), and many-objective remodularization (Mkaouer et al. 2015, 2015a; Prajapati and Chhabra 2018, 2018a), 2) type of information: structural-based remodularization (Praditwong et al. 2011; Mancoridis et al. 1998, 1999; Mahdavi et al. 2003), lexical-based remodularization (Corazza et al. 2016), and combined structural + lexical based remodularization (Mancoridis et al. 1998; Prajapati and Chhabra 2017a; Bavota et al. 2010) level of modifications: moderate remodularization (Bavota et al. 2010; Prajapati and Chhabra 2017; Abdeen et al. 2009) software clustering (Praditwong et al. 2011; Kumari et al. 2013; Ouni et al. 2016).

The application of SBSE technique to the software remodularization is approximately two-decade-old. In the formulation of search-based remodularization, the first credit goes to the authors Mancoridis et al. (1998) who first applied the SBSE concepts to cluster the software entities into more cohesive form. In their contribution, they also introduced modularization quality (MQ) measure to evaluate the clustering quality which is defined in terms of two software quality attributes (i.e., inter-connectivity and intra-connectivity). The MQ, a structural information based software modularity quality criterion, has been widely used as the fitness function to guide the remodularization process (Doval et al. 1999; Harman et al. 2002; Mitchell and Mancoridis 2002; Mamaghani and Meybodi 2009). The authors Mancoridis et al. (1999) customized different meta-heuristic search techniques such as Genetic Algorithm (GA), Simulated Annealing (SA), and Hill-Climbing (HC) algorithm to address the software module clustering problem.

Recently, authors Praditwong et al. (2011) used the MQ measure along with other software clustering criteria to formulate the software clustering problem as a multi-objective optimization problem. They also introduced two new multi-objective clustering formulations namely maximize cluster approach (MCA) and equal cluster size approach (ECA). Each of the MCA and ECA formulations contains five partially conflicting objectives and is based on the structural information. The researchers (Barros 2012; Kumari et al. 2013; Prajapati and Chhabra 2014) have also used the MCA and ECA formulation to evaluate different meta-heuristic algorithms.

The above discussed MQ measure is based on the direct link coupling. Recently, the authors Jinhuang and Jing (2016) defined a new MQ measure which is determined on the basis of similarity coupling. Their experimentation results demonstrated that the similarity based MQ outperformed compared to the direct link based MQ. The authors (Prajapati and Chhabra 2017a) have also used the similarity based coupling measure to remodularize the software system. The results demonstrated that the similarity based coupling measure is able to generate good quality software modularization.

Even though structural based software modularity measure able to drive search algorithms towards remodularization solution which is good from the structural point of view but not good from a semantic perspective or developers view. To remodularize the software system which is good from the semantic point of view the researchers Corazza et al. (2016) used the lexical information to compute the similarity between the software entities. Their results showed that the lexical based software remodularization is able to generate remodularization solution which is better from the semantic perspective. Some researchers (e.g., Prajapati and Chhabra 2017a; Bavota et al. 2010, 2013, 2014) used combined structural and lexical information to remodularize the software system.

3 Basic Concepts

This section presents a brief description of information theory which is used in our proposed many-objective remodularization approach. The information theory concepts

are very wide; here it is not possible to describe them in detail. The interested reader may find details about the information theory concepts in any information theory textbook (e.g., Cover and Thomas 1991). Apart from the basic concepts of information theory, in this section, a brief description about the various types of structural (e.g., calls, inheritance, contains, etc.) and lexical (e.g., method name, class name, parameter name, etc.) are also provided.

3.1 Minimum Entropy Concept

In this section, we explain the concepts of software entropy corresponding to an object-oriented software remodularization. Here the term feature is used to refer to different types of coupling information (e.g., structural and lexical) of a class. The different values that each feature takes are referred as a feature values. We assume that the object-oriented software to be remodularized contains a set of N number of source code classes, i.e., $C = \{c_1, c_2, \ldots, c_N\}$ and each class have a set of M features, i.e., $F = \{f_1, f_2, .., f_M\}$ with feature values w_i of i-th feature. Our approach starts by representing software system into matrix M as given in Table 1.

The rows of the above matrix represent the source code classes to be remodularized while the columns shows the values of the features that describe these source code classes. Each entry of the matrix Mij is filled with the coupling value of jth feature in ith class. Let X represents a discrete random variable taking its values from a set of classes C.

If p (x_i) is the probability distribution function (pdf) of the values x_i that X takes ($x_i \in C$), the entropy H(X) of variable X is defined as follows:

$$H(X) = -\sum_{x_i \in C} p(x_i)\log\Big((p(x_i))\Big) \tag{1}$$

Intuitively, entropy quantifies the disorder of a system. The higher the uncertainty leads higher the entropy. Since entropy determines the amount of "disorder" of a system, many approaches utilizes some form of such a measure as a quality criterion (i.e., fitness function) for clustering different types of data (Cover and Thomas 1991; Gokcay and Principe 2002; Hino and Murata 2014). In clustering, the cluster containing elements with high similarity showed the low entropy.

Table 1 Matrix M representing software system

	f_1	f_2	.	.	.	f_N
c_1	w_{11}	w_{12}	.	.	.	w_{1N}
c_2	w_{21}	w_{22}	.	.	.	
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
c_N	w_{N1}	w_{N2}	.	.	.	w_{NN}

In the context of the remodularization of object-oriented package structure, we assume that a module/package is the group of classes, which has minimum entropy. It means that a module is a partition of set of classes with minimum degree of "disorder". The entropy of a module is directly related to its classes. In terms of probability, the entropy of the module depends of the probability distribution function of its classes. For object-oriented software system, let N number of source code classes are distributed among the M number of modules and c_i is an i^{th} class of N. The pdf of c_i is defined as follows:

$$p(c_i) = \sum_{t=1}^{M} p(c_i|t)p(t) \tag{2}$$

where $p(t)$ is the prior probability for the t^{th} module and $p(c_i|t)$ is the prior probability of c_i given the t^{th} module. However we would like to know the dependence of pdf of the t^{th} module with respect to c_i. This dependency can be computed using Bayes Theorem (Joyce 2008):

$$p(t|c_i) = \frac{p(c_i|t)p(t)}{p(c_i)} \tag{3}$$

when $p(t|c_i)$ is uniformly distributed for all t, we can say that the class c_i belongs to any module and thus the uncertainty is maximum. On the other hand if all $p(t|c_i)$ but one are zero (one having the value unity) then we are certain about the module to which the class c_i belongs to. Now, let C be a random variable whose possible values are *1, 2,..,K* which represent the modules of M. Let *X* be a random variable whose possible values are all elements c_i that belong to N. Then the entropy of *C* given *X* is:

$$H(C|X) = -\sum_{t=1}^{K} p(t|c_i)\log(p(t|c_i)) \tag{4}$$

where $p(t|c_i)$ is a posteriori pmf. Thus, our goal is to find this function such that $H(C|X)$ is minimum. The entropy given by Eq.4 is called Conditional Entropy.

The information theoretic similarity measure (i.e., $H(C|X)$) as discussed above is a representative of collective similarity measure rather than traditional direct link similarity measure. The direct link similarity measures (e.g., inter-module class coupling and intra-module class coupling) may leads optimization process towards remodularization solution better from the coupling and cohesion perspective and may not be meaningful from the developers' perspective. However, information theoretic similarity measure encompasses many dimensions of similarity; hence it is supposed that the incorporation of such similarity measure into remodularization process may lead generation of meaningful solution. To compute the conditional entropy more accurately, we need to compute the feature value of classes more accurately. In this paper, we

consider structural as well as lexical features of the source code classes.

3.2 Structural and Lexical Features

In this paper, we utilize various types of class information to determine the class features. In particular, the proposed approach considers 8 different types of structural (e.g., calls, inheritance, contains, etc.) and 6 different types of lexical (e.g., method name, class name, parameter name, etc.) information as class features.

3.2.1 Structural-Based Features

The classes in an object-oriented software system may be connected with other classes by zero or more structural coupling relationships (e.g., method calls, inheritance, references, etc). Hence, structural features of a source code class can be defined in terms of individual classes which are connected with that class. In this paper, the structural features of a class are the different structural-relationships by which it is connected with another class. Combination of all structural relations among a pair of classes determines the connection strength among these two classes and is considered as the feature value for this pair. To determine the coupling strength between the classes, we use eight different types of structural relationships between the classes (Prajapati and Chhabra 2017a). The brief descriptions of these relationships are given in Table 2.

The value of connection strength between class c_i and class c_j is computed by aggregating the number of instances of each relationship with their relative weights. The connection strength (CS) from class c_i to class c_j is defined as follows:

$$CS(c_i, c_j) = \sum_{r \in R} w_r(c_i, c_j) \times n_r(c_i, c_j) \quad (5)$$

where n_r and w_r represent the number of instances and weights of r-type relationship respectively. To compute the relative weights w_r for r-type relationships this paper uses term frequency-inverse document frequency (TFIDF) weighting scheme. This is the most widely used technique in data mining to assign the weights to document terms (Yates and Neto 1999). In this study, the documents are the source code classes and terms are their relationships. The weight w_r of r-type relationship from class c_i to class c_j is given as follows:

$$w_r(c_i, c_j) = tf(r).\log(idf(r)) \quad (6)$$

Where tf(r) (term frequency) is the frequency of r-type relationship from class c_i to class c_j and idf(r) (inverse document frequency) is the fraction $n/n_r(c_j)$, where n is the number of classes, and $n_r(c_j)$ is the number of classes connected with class c_j with r-type relationships.

3.2.2 Lexical-Based Features

In addition to structural-based features of source code classes, the proposed remodularization approach also uses the lexical-based features of classes. Different types of unique terms present in the classes are considered as the feature of the classes. Our approach uses six major categories of lexical features similar to the approach reported in (Corazza et al. 2016). A brief description of the different categories is given in Table 3.

Similar to the structural-based features, here first each lexical-based feature (i.e., the terms with their occurrence) is computed, then the relative importance (weights) of each term is determined, and finally, the corresponding weight is multiplied with the actual terms occurrences. Given a class c_i, the weight of the term t_i of a particular zone is computed as follows:

$$w(t, c, z) = tf(t).\log(idf(t)) \quad (7)$$

where tf(t) (term frequency) is the frequency of term t of zone z for class c and idf(t) (inverse document frequency) is the fraction n/df(t, z), where n is the number of classes, and df(t, z)is the number of classes in which the term t occurs, within the zone z.

4 Proposed Approach

This section presents the detailed descriptions of major steps used in proposed information-theoretic software

Table 2 Structural relationships existing between two classes

#	Relations	Abbr.	Description
1.	Inheritance	EX	class c_i extends class c_j
2.	Has Parameter	HP	class c_i has a method with a parameter of class c_j
3.	Reference	RE	instance of class c_i invokes to the attribute or method of class c_j.
4.	Calls	CA	method of class c_i calls the method of class c_j.
5.	Implement	IM	class c_i implement or realize, the behavior specified by the class c_j.
6.	Is of Type	IT	class c_i has an attribute that is type of class c_j.
7.	Return	RN	class c_i has a method which returns an object of class c_j.
8.	Throws	TH	class c_i, throws an exception to the class c_j.

Table 3 Types of lexical class relationships

#	Zone	Abbr.	Description
1	Class Name	CN	Name of the class, super class or implemented interfaces
2	Attribute Name	AN	Name of attributes and corresponding types
3	Method Name	MN	Names of methods and their return types
4	Parameter Name	PN	Names of parameters and their types
5	Comment	CO	Lexemes extracted from all comments of classes
6	Source Code Statement	SCS	Lexemes occurring in the body of methods.

remodularization. The main objective of the proposed approach is to re-organize the source code classes of object-oriented software system among packages/modules such that the re-grouping is good from the quality metrics point of view as well as meaningful from developers' perspectives. To achieve the goal, proposed work defines five package quality evaluation criteria (i.e., package entropy (to minimize), inter-module class change-coupling (to minimize), intra-module class change-coupling (to maximize), Module Count Index (to minimize), and Module Size Index (to minimize)) that help in guiding the remodularization process towards more promising search region.

The complete framework of overall working process of the proposed software remodularization approach is presented in Fig. 1. The activities of whole framework are divided into three main phases. In first phase, the required information regarding the structural, lexical, and changed history of software are extracted. In second phase, structural and lexical coupling is computed independently and then combined together. In third phase, package entropy, inter-module class change-coupling, intra-module class change-coupling, Module Count Index, and Module Size Index are computed. Finally, the search-based algorithm is applied to generate the remodularization solution.

4.1 Extraction of Software Information

In this phase the software information such as structural information (e.g., classes, packages, and class relationships) lexical information (e.g., class name, attribute name, method name, parameter name, comments, and source code statements), and changed history information are collected. As the proposed software remodularization approach is specially designed for the object-oriented software implemented in the Java programming language. Hence, various terminologies and information used in this work is based on the Java programming language.

4.2 Remodularization Objectives

The automated software remodularization driven by a search-based meta-heuristic algorithm requires fitness functions that can force the optimization process towards expected solution. To achieve a meaningful remodularization solution from the developers' perspective, it is necessary to incorporate a better software evaluation model and metrics during remodularization process that can reflect developer's perspective. The developers' perspective can be inference from the source code information where developers' knowledge is embedded. Based on the

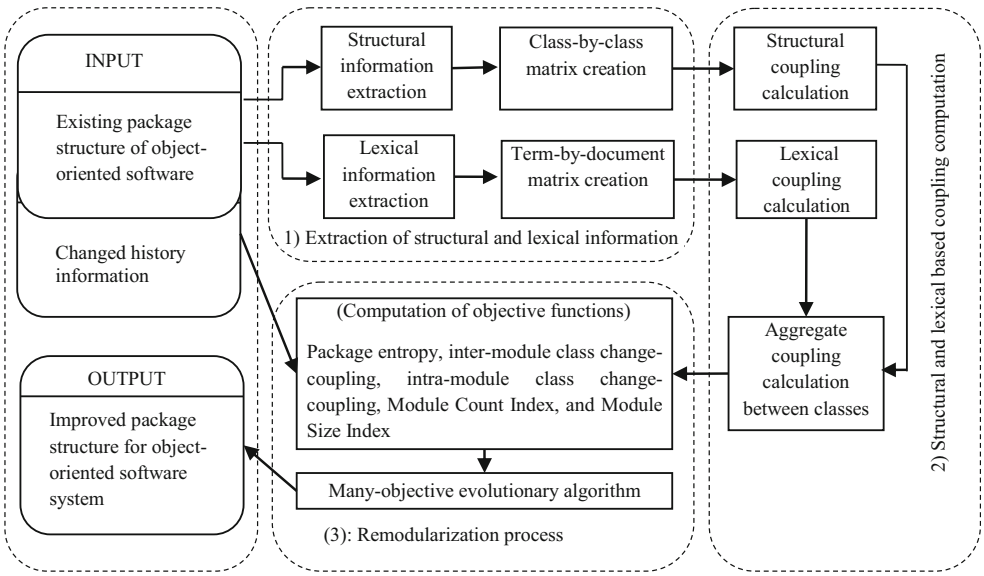

Fig. 1 Framework of proposed remodularization approach

various dimensions of structural and lexical information, change-history information, and module dispersion information this paper designs the following objective functions:

- **Software Entropy:** Entropy is the measure of disorder; higher the entropy, lower is the certainty. For a good software module, it is necessary that it should contain highly correlated classes. The entropy of the software system is defined as follows (Aldana-Bobadilla and Kuri-Morales 2011):

$$H(C|X) = \sum_{t=1}^{K} H(t|X) \tag{8}$$

where $H(t|X)$ is the entropy of module t. The objective is to minimize the entropy for each module.

- **Inter and Intra-module Class Change Coupling:** The classes changed together should be grouped together is one of the core design principle for software systems. Here, we use the change-history information from the version repository and compute the change coupling between classes in terms of change-coupling at the class level by mining their co-change pattern. The inter-module class change coupling refers to the total class change-strength within modules and intra-module class change coupling refers to the total class change-strength between modules (Parashar and Chhabra 2016). The change-strength between two classes is defined as follows:

$$Change{-}strength\left(C_i, C_j\right) = \frac{|C_i \cap C_j|}{|C_i|} + \frac{|C_i \cap C_j|}{|C_j|} \tag{9}$$

where $|C_i \cap C_j|$ is count of change-commits in which both C_i and C_j are changed together. $|C_i|$ is count of change-commits consisting of C_i.

- ***Module Count Index (MCI):*** In search-based software remodularization, the optimization process driven by the only similarity criteria may lead to generation of singleton module or modules containing single entities. To avoid such situation, this work uses two other conflicting namely module count index (MCI) and module size index (MSI) introduced by the authors (Prajapati and Chhabra 2017a). The MCI determine the deviation between the number of modules produced during optimization and number of modules set by the by developers. The MCI is defined as follows:

$$MCI = \exp\left[-\frac{1}{2}\left(\frac{\ln(m) - \ln(m^*)}{w\ln(n)}\right)^2\right] \tag{10}$$

Where m, m*, and n represent the number of modules in produced solution, number of module defined by the developers, and number of classes respectively. The parameter 'w' represents the penalty factor that penalizes the remodularization which is far away from m*.

- ***Module Size Index (MSI):*** Then main goal of the MSI objective function is to prevent generation of very large size of modules (i.e., containing large number of entities). In other words, the MSI evaluate the deviation between the size of generated modules and size of developer's modules. To determine the ideal module size, the researchers used the method of Component Packaging Density (CPD) (Abdeen et al. 2009). The MSI is defined as follows:

$$MSI = \exp\left[-\frac{1}{2}\left(\frac{\ln\left(s_{avg}\right) - \ln\left(s^*\right)}{w\ln(n)}\right)^2\right] \quad where \quad s_{avg} = \sum_{i=1}^{n} \frac{s_i}{n} \tag{11}$$

where s_i is the size of the module in which class i locate in. The ideal module size is defined as $s^* = n/m^*$. Similar to the MCI, the parameter 'w' represents the penalty factor.

4.3 Remodularization Problem Encoding

To apply a search-based meta-heuristic, the problems need to be encoded in a suitable form such that the various manipulator operator of meta-heuristics can be performed effectively. To encode the software package organization of existing software system, an n-sized (i.e., equal to the number of lasses) integer array is used, where the value v $0 < v \leq p$ of i^{th} element indicates the package number v, to which the i^{th} source code class is assigned. The symbol p represents the number of packages/modules. For example, in Fig. 2, array index 5 represents the class 5 and the value at index 5, i.e., 2 shows that the class 5 is assigned to the module 2.

To initialize the population, the solutions of the population are generated randomly in the range of lower and upper bounds of the each decision variables.

4.4 Many-Objective Evolutionary Algorithm

There has been an important progress in formulating the real-world optimization problem as search-based multi-objective optimization problem and solving them using multi-objective evolutionary algorithms MOEAs (e.g. NSGA-II (Deb et al. 2002), SPEA2 (Zitzler et al. 2002), and PESA-II (Corne et al. 2001). These algorithms used the Pareto-dominance

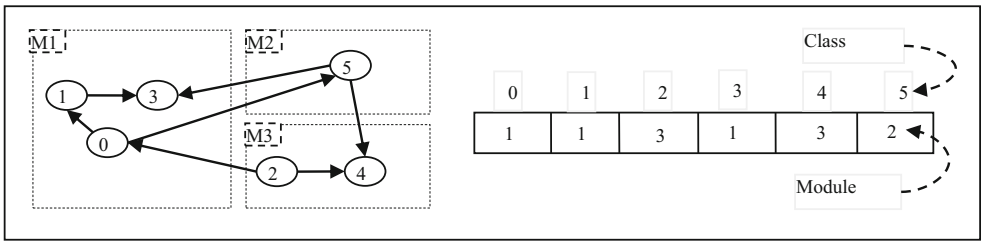

Fig. 2 An example of remodularization encoding

concept to rank the solutions in the population for the purpose of selection. However, the studies (Bingdong et al. 2015; Jaimes et al. 2009; Wang et al. 2015) demonstrated that the Pareto-dominance based MOEAs approaches are not able to perform well when the number of objective functions in the optimization problem gets large (specifically more than three).

In our software remodularization, there are five objective functions that have to be optimized simultaneously in order to improve the existing package structure. Hence, in this case the Pareto-dominance based MOEA may not work effectively. To optimize these objective functions simultaneously and effectively, we adapt NSGA-III (Deb and Jain 2014), MOEA/D (Zhang and Li 2007), IBEA (Zitzler and Kunzli 2004), and TAA (Praditwong and Yao 2006) the most popular many-objective evolutionary which are designed to work effectively in case of large number of objective functions**.** These algorithms have been applied successfully to solve the different many-objective optimization problems (Jain and Deb 2014).

5 Experimental Setup

To evaluate the ability of our entropy-based software remodularization to generate good modularization solution, we conducted a set of experiments on seven open-source software systems. The details of experimental setup includes 1) description of software systems on which proposed approach is evaluated, 2) results collecting method, 3) the results evaluation criteria, 4) existing remodularization algorithms, and 5) used Statistical tests.

5.1 Studied Software Projects

We chose seven object-oriented software systems each one is characterizing a real-world software system with diverse complexity in terms of number of connections, number of classes, number of modules, and lines of code (LOC). These software systems are open-source and written in Java programming language. The main reason of considering these software systems is that they are different sizes and complexities and these software systems have also been used by the previous researchers (Prajapati and Chhabra 2018; Erdemir and Buzluca 2014; Prajapati and Chhabra 2018a; Bavota et al. 2013) to evaluate the similar approach. Table 4 provides the complete description of their respective characteristics.

5.2 Collecting Results

Since search-based software remodularization approaches are stochastic optimizers, they can generate different results for the same software instance from one run to another. For this reason, we collect results from our proposed software remodularization approach by applying it for each test software system on 31 independent runs. In each execution, the many-objective optimization techniques generate a set of non-dominated solutions. To select a single solution that gives maximum trade-off to all the considered objective functions, we use the trade-off worthiness metric defined in the works (Rachmawati and Srinivasan 2009).

Table 4 Characteristics of the used software projects

Systems	Version	#Classes	#Connections	#Modules	Lines of code (KLOC)
Weka	3.3.6	455	1385	42	111
GEF	3.8	756	2349	45	63
Lucene	4.3.1	2090	11,959	205	285
ArgoUML	0.3.4	1686	5586	93	156
JFreeChart	0.9.21	401	1420	50	72
Solr	4.3.1	777	2423	64	102
Tomcat	7.0.42	972	3567	116	184

Table 5 Many-objective remodularization approaches

Existing multi-objective remodularization approaches			Proposed approach
Structural-based	Lexical-based	Structural + Lexical based	
1. Maximize structural-based MQ	1. Maximize lexical-based MQ	1. Maximize structural + lexical based MQ	1. Minimize software entropy
2. Minimize inter-module class change coupling	2. Minimize inter-module class change coupling	2. Minimize inter-module class change coupling	2. Minimize inter-module class change coupling
3. Maximize intra-module class change coupling	3. Maximize intra-module class change coupling	3. Maximize intra-module class change coupling	3. Maximize intra-module class change coupling
4. Minimize module count index	4. Minimize module count index	4. Minimize module count index	4. Minimize module count index
5. Minimize module size index	5. Minimize module size index	5. Minimize module size index	5. Minimize module size index

5.3 Result Evaluation Criteria

Evaluations of the search-based remodularization results are generally performed by two approaches: internal and external assessment. Internal assessment approach is to evaluate the quality of the internal characteristics of the modules in produced remodularization. There exist a number of quality metrics to assess the remodularization internally, for example, coupling and cohesion (Cui and Chae 2011), modularization quality (Praditwong et al. 2011), size of clusters (extremity) (Glorie et al. 2009; Erdemir and Buzluca 2014), and the number of clusters (Wang et al. 2010). In this study, we intend to use the size of clusters (extremity).

The aim of the external assessment is to find the association between obtained remodularization and the authoritative remodularization suggested by a human expert (e.g., original developer). The approach is also known as Authoritativeness. The produced remodularization solution should resemble the authoritative remodularization as much as possible (Wu et al. 2005). To find the authoritativeness, different measures may be used, MoJo and MoJoFM (Wu et al. 2005; Tzerpos and Holt 1999; Andritsos and Tzerpos 2005; Bittencourt and Guerrero 2009) and precision, recall (Sartipi and Kontogiannis 2003). In this study, we used MoJoFM a most widely used measure for authoritativeness. The brief descriptions of these metrics are given in following sub-sections.

5.3.1 Non-Extreme Distribution (NED)

For a well-modularized software system, it is necessary that the size of any individual module should not be extremely small or large (Erdemir and Buzluca 2014). Hence, an automatic software remodularization approach should generate a modularization solution that has a better distribution of classes into the modules. To evaluate the extremity of module size, Wu et al. (2005) defined a non-extreme distribution (NED) as follows:

$$NED = \frac{\sum_{i=1, M_i\, IS \quad NOT \quad EXTREME}^{k} |M_i|}{n}, M_i \; is \quad not \quad extreme \quad if \quad 5 < |M_i| < 1.5 \times |MA_{\max}| \tag{12}$$

Where k is the number of modules, n is the total number of classes, |Mi| is the size of module i, and $|MA_{max}|$ is the size of the largest module.

5.3.2 Authoritativeness

Authoritativeness is a measurement used for determining the similarity between the remodularization solution generated by the automatic remodularization method and the remodularization solution suggested by the experts (Erdemir and Buzluca 2014). To find the authoritativeness, different measures may be used. In this study, we use the MoJoFM measure. Let M and Ma be two remodularization solutions,

Table 6 Authoritativeness using NSGA-III

Systems	Structural-based	Lexical-based	Structural + Lexical	Entropy-based	Wilcoxon test
Weka	73.542	74.914	80.389	83.452	+ + +
GEF	73.845	71.515	77.236	86.265	+ + +
Lucene	65.351	67.253	73.654	73.254	++ ≈
ArgoUML	51.874	53.065	67.345	68.345	++ ≈
JFreeChart	62.378	61.258	70.385	73.361	++ ≈
Solr	61.287	65.245	71.284	76.289	+ + +
Tomcat	71.256	74.325	77.367	81.647	+ + +

Table 7 Authoritativeness using MOEA/D

Systems	Structural-based	Lexical-based	Structural + Lexical	Entropy-based	Wilcoxon test
Weka	74.285	73.724	83.614	80.035	+ + −
GEF	71.249	73.345	75.243	81.154	+ + +
Lucene	63.845	66.468	71.265	70.124	++ ≈
ArgoUML	56.195	53.065	61.256	66.231	+ + +
JFreeChart	62.378	61.258	68.124	70.235	++ ≈
Solr	64.235	63.547	69.621	73.854	+ + +
Tomcat	72.365	74.325	75.325	80.256	+ + +

mno(M,Ma) be the minimum number of join and move operations to transform remodularization M to remodularization Ma where join operation combines two modules into single module and move operation moves a component from one module to another module. MoJoFM (M,Ma) is defined as follows:

$$MoJoFM(M, Ma) = 100 - \left(\frac{mno(M, Ma)}{\max\left(mno(\forall M, Ma)\right)} \times 100 \right) \quad (13)$$

where MoJoFM(M,Ma) represents authoritativeness, M and Ma represent the modular architecture generated by the approach and authoritative modular structure suggested by the experts respectively, and max($mno(\forall M, Ma)$is the maximum possible distance of any remodularization M from the remodularization Ma.

In practice, for academicians it is a very difficult task to find the original developers of the software systems being evaluated, who were engaged in developing the corresponding original software system. To overcome this problem, we use the same method to obtain authoritative remodularization as used by the previous researchers (Erdemir and Buzluca 2014; Wu et al. 2005; Corazza et al. 2016). The process of obtaining authoritative remodularization for a test software system is summarized as follows: 1) Find the packages and classes associated with that package, 2) Validate the existing package organization with the comments are written in the source code class in these software systems, 3) Merge a package with its closed package in case it contains a number of classes that are less or equal to five, 4) Final, authoritative remodularization is developed from the preliminary authoritative remodularization by involving external expert developers.

5.3.3 Stability

An automatic software remodularization approach should not generate dramatically different modular structure for a similar version of the software with minor changes. Stability is formulated as follows:$Stability(M^n) = MoJoFM(M^n, M^{n-1})$, where M^n and $M^{n-1}$are the software remodularization results generated for two consecutive versions of a software system.

5.4 Rival Remodularization Approaches

Most of the existing software remodularization approaches used modularization quality (MQ) (Mancoridis et al. 1998) metric as remodularization objective function to drive the optimization process. In the multi-objective formulation of software remodularization problem, the MQ is used as core objective along with other supportive objective functions. The authors Praditwong et al. (2011) redesigned the MQ and used it as core objective function along with other four supportive objective functions (e.g., inter-cluster coupling, intra-cluster coupling, number of clusters, etc) to remodularize the software systems. Similarly, the authors (Barros 2012; Prajapati and Chhabra 2017, 2017a; Kumari et al. 2013; Prajapati and

Table 8 Authoritativeness using IBEA

Systems	Structural-based	Lexical-based	Structural + Lexical	Entropy-based	Wilcoxon test
Weka	59.678	66.356	75.689	76.235	++ ≈
GEF	63.257	71.351	76.254	77.475	++ ≈
Lucene	52.356	55.645	63.125	68.647	+ + +
ArgoUML	43.568	48.326	58.654	68.365	+ + +
JFreeChart	48.261	52.374	63.642	69.258	+ + +
Solr	57.921	56.823	62.654	72.925	+ + +
Tomcat	62.875	68.645	73.025	78.562	+ + +

Chhabra 2018, 2018a) used the MQ as the core objective function in their multi-objective formulation of software remodularization problem. The MQ is defined as follows:

$$MQ = \sum_{k=1}^{n} MF_k \tag{14}$$

Where n is the number of packages/modules and MF is the modularization factor. The Modularization Factor (MF_k) for module k is defined as follows:

$$MF_k = \begin{cases} 0, & if\ i = 0 \\ \dfrac{i}{i + \frac{1}{2}j}, & if\ i > 0 \end{cases} \tag{15}$$

where i is the coupling of the classes within the packages k and j is the coupling between the classes of package k and the classes exist in rest of the packages of the system. The coupling between the classes can be determined based on the different types of information (e.g., structural information, lexical information, and combined structural and lexical information). To make a fair comparison between our entropy based software remodularization (entropy as core objective) and existing MQ based software remodularization (MQ as core objective), we have taken the same supportive objective functions (i.e., minimize inter-module class change coupling, maximize intra-module class change coupling, minimize module count index, minimize module size index). Table 5 summarizes the used objective functions in the proposed approach and existing search-based multi-objective remodularization approaches.

5.5 Statistical Tests

The meta-heuristic optimization algorithms are random optimizers (i.e., algorithms can generate different results over the same test problem from one run to another Mkaouer et al. (2015). The result obtained through a single run cannot be used to reveal any conclusion about the algorithms. Hence, it becomes necessary to obtain a set of results for the same problem instance over many runs. In this study, we collect the results by executing each algorithm 31 times on the same problem instance. The sample with 31solutions are statistically analyzed by using the Wilcoxon rank sum test (Arcuri and Briand 2011) with a 95% confidence level ($\alpha = 5\%$).

6 Results and Analysis

In this section, we present the authoritativeness, NED, and stability results achieved through proposed and existing search-based software remodularization approaches.

6.1 Authoritativeness

Tables 6, 7, 8, and 9 present the authoritativeness results achieved through the proposed entropy-based approach and existing remodularization approaches (i.e., structural similarity, lexical similarity, structural + lexical based) on seven software systems for each considered many-objective meta-heuristic algorithms (i.e., NSGA-III, MOEA/D, IBEA, and TAA). The columns of each table labelled with Wilcoxon test shows the statistical results of comparison between the proposed approach and existing remodularization approaches. The symbol'+' denotes that there is a statistically significant difference between the proposed approach and the existing approach and it is in favour of the proposed approach. The symbol'–' denotes that there is a statistically significant difference between the proposed approach and the existing approach and it is in favour of the existing approach. The symbol '≈' denotes that there is no significant difference between the proposed approach and the existing approach. In the sequence three symbols of the Wilcoxon test columns, the first symbol is the result of the statistical test between structural-based approach and proposed approach, the second symbol is the result of a statistical test between lexical-based approach and proposed approach, and the third symbol is the result of a statistical test between structural + lexical based approach and the proposed approach. For example, the first symbol (i.e., '+') in sequence "+ + –" for Weka software system of Table 7, represents that there is a significant difference between the proposed approach and the structural-based approach and the difference

Table 9 Authoritativeness using TAA

Systems	Structural-based	Lexical-based	Structural + Lexical	Entropy-based	Wilcoxon test
Weka	52.345	62.378	71.642	74.378	+ + +
GEF	48.345	68.468	72.345	75.346	+ + +
Lucene	45.785	52.465	61.358	78.238	+ + +
ArgoUML	48.238	44.235	62.354	67.398	+ + +
JFreeChart	41.823	63.245	64.023	63.254	++ ≈
Solr	56.824	53.012	61.875	71.397	+ + +
Tomcat	56.231	67.321	70.231	79.564	++ ≈

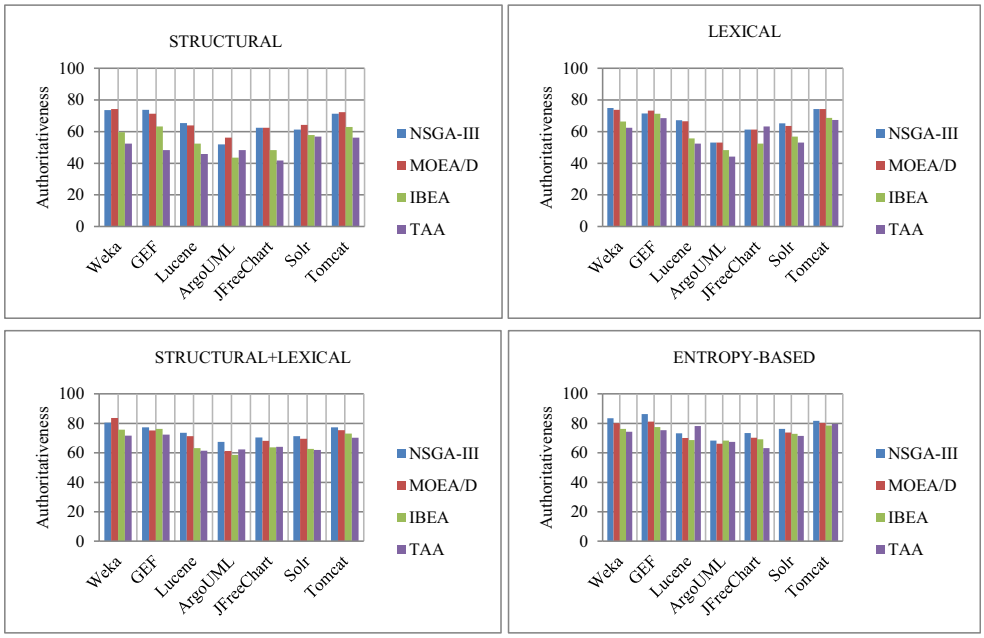

Fig. 3 The impact of different many-objective meta-heuristic algorithms on remodularization

is in favour of the proposed approach. The second symbol (i.e., '+') in sequence "+ + −" for Weka software system of Table 7, represents that there is a significant difference between the proposed approach and the lexical-based approach and the difference is in favour of the proposed approach. Similarly, the third symbol (i.e., '−') in "+ + −" for Weka software system represents that there is a significant difference between the proposed approach and the structural + lexical based approach and the difference is in favour of the structural + lexical based approach.

Now if we see the authoritativeness results presented in Tables 6, 7, 8, and 9 achieved through proposed and existing approaches on seven software systems with many-objective meta-heuristic algorithms (i.e., NSGA-III, MOEA/D, IBEA, and TAA), it is clearly indicates that the proposed approach outperforms the existing approaches by producing significant better authoritativeness values in most of the cases. In particular, the proposed approach outperforms the structural and lexical model significantly better in all test software systems. However, in some cases, authoritativeness of proposed approach is competitive with the structural + lexical approach.

Apart from the comparison between the proposed approach and existing approaches, we have also compared the authoritativeness of each many-objective meta-heuristic algorithms (i.e., NSGA-III, MOEA/D, IBEA, and TAA) over each remodularization approach (structural, lexical, structural + lexical, and entropy-based). These comparative results demonstrated by Fig. 3. Now if we observe the results of Fig. 3, it is clearly indicated that the NSGA-III algorithms performs better compared to the MOEA/D, IBEA, and TAA. However, the results of MOEA/D, IBEA, and TAA algorithms are competitive.

6.2 Non-Extreme Distribution (NED)

Tables 10, 11, 12, and 13 present the NED results achieved through the proposed entropy-based approach and existing remodularization approaches (i.e., structural similarity, lexical

Table 10 Non-extreme distribution (NED) using NSGA-III

Systems	Structural-based	Lexical-based	Structural + Lexical	Entropy-based	Wilcoxon test
Weka	99.325	100.000	100.000	100.000	≈≈≈
GEF	100.000	99.235	100.000	100.000	≈≈≈
Lucene	100.000	100.000	98.647	100.000	≈≈≈
ArgoUML	100.000	100.000	100.000	100.000	≈≈≈
JFreeChart	100.000	98.789	100.000	100.000	≈≈≈
Solr	99.357	100.000	98.567	100.000	≈≈≈
Tomcat	100.000	100.000	100.000	100.000	≈≈≈

Table 11 Non-extreme distribution (NED) using MOEA/D

Systems	Structural-based	Lexical-based	Structural + Lexical	Entropy-based	Wilcoxon test
Weka	100.000	100.000	99.754	100.000	≈≈≈
GEF	98.265	100.000	100.000	100.000	≈≈≈
Lucene	100.000	100.000	100.000	100.000	≈≈≈
ArgoUML	100.000	99.812	100.000	100.000	≈≈≈
JFreeChart	99.687	100.000	99.631	100.000	≈≈≈
Solr	100.000	100.000	100.000	100.000	≈≈≈
Tomcat	100.000	100.000	100.000	100.000	≈≈≈

similarity, structural + lexical based) on seven software systems for each considered many-objective meta-heuristic algorithms (i.e., NSGA-III, MOEA/D, IBEA, and TAA). The meanings of symbols used in Wilcoxon test columns of each table are same as described in section 6.1. For each proposed and exiting approaches the 100% NED value denotes the most stable remodularization solution. Now if we see the results presented in Tables 10, 11, 12, and 13, it clearly shows that the proposed entropy-based approach is able to achieve 100% NED for each many-objective meta-heuristic algorithms on each of the test software systems. However, existing remodularization approaches results are competitive and produce only slightly lower NED values in some cases. The Wilcoxon test results also show that there are no significant difference between the proposed entropy-based approach and the existing approaches in all cases.

6.3 Stability

To assess the stability of the proposed approach, we analyzed 21 successive versions of the JFreeChart. The stability results of proposed entropy-based approach and existing structural, lexical, structural + lexical based approaches with NSGA-III, MOEA/D, IBEA, and TAA algorithms are given in Tables 14, 15, 16, and 17. For each remodularization approaches, the bold value indicates the best stability result. The meanings of symbols used in Wilcoxon test columns of each table are same as described in section 6.1.

Now if we see the stability results presented in Table 14, it clearly shows that the structural, lexical, structural + lexical, and entropy-based approach with NSGA-III algorithm, the stability values range between 51.35–76.53%, 68.51–92.75%, 74.65–98.65%, and 81.25–99.35%, respectively. These stability values obtained show that the entropy-based evaluation model achieves higher stability compared to the existing approaches. Similar to the NSGA-III, the results of Tables 15, 16, and 17 shows that the entropy-based approach achieves higher stability in case of MOEA/D, IBEA, and TAA algorithms compared to the existing algorithms.

In summary, the results show that the information theoretic similarity measure has a significant impact on the generation of software modularization with better authoritativeness, NED, and stability values compared to other approaches. Therefore, we think our information theoretic many-objective approach can be useful for remodularizing object-oriented software systems.

6.4 Discussion

Even though the information theoretic similarity measure concept is not a wide-spread concept in SBSE. We believe that it is very useful for remodularizing software systems. The fact that remodularization of a software system using information theoretic similarity measure concept is not used by many SBSE practitioners. In the previous years, the software

Table 12 Non-extreme distribution (NED) using IBEA

Systems	Structural-based	Lexical-based	Structural + Lexical	Entropy-based	Wilcoxon test
Weka	100.000	100.000	100.000	100.000	≈≈≈
GEF	100.000	99.825	100.000	100.000	≈≈≈
Lucene	100.000	100.000	100.000	100.000	≈≈≈
ArgoUML	99.575	100.000	98.364	100.000	≈≈≈
JFreeChart	100.000	100.000	100.000	100.000	≈≈≈
Solr	98.754	100.000	100.000	100.000	≈≈≈
Tomcat	100.000	100.000	100.000	100.000	≈≈≈

Table 13 Non-extreme distribution (NED) using TAA

Systems	Structural-based	Lexical-based	Structural + Lexical	Entropy-based	Wilcoxon test
Weka	100.000	100.000	100.000	100.000	≈≈≈
GEF	100.000	99.347	100.000	100.000	≈≈≈
Lucene	100.000	100.000	100.000	100.000	≈≈≈
ArgoUML	98.164	100.000	100.000	100.000	≈≈≈
JFreeChart	100.000	99.286	100.000	100.000	≈≈≈
Solr	100.000	100.000	100.000	100.000	≈≈≈
Tomcat	98.374	100.000	100.000	100.000	≈≈≈

remodularization process was based on the structural and lexical similarity measure; the proper use of information theoretic similarity measure concept for software remodularization is a novel concept and has been used for the first time in this paper to remodularize software systems. The usefulness of this concept is supported by the experimental results of our approach, which clearly show the advantages of information theoretic similarity measure for software remodularization. Further, as there are five objective functions (i.e., more than three objective functions) to be optimized simultaneously in our remodularization approach, hence considering a many-objective meta-heuristic algorithm is a good alternative.

7 Threats to Validity

In this section, we explore the factors that can influence the validity of the results reported in this paper. For software engineering experimentation, the work reported in literature (Wohlin et al. 2000) has divided threats to validity into four categories: conclusion, internal, construct, and external threats.

Conclusion threats to validity: These threats are concerned with the relationship between treatment and outcome. The meta-heuristic algorithms use many random

Table 14 Stability results of NSGA-III over JFreeChart project

Version	MoJoFM (Mn, Mn − 1) (%)				
	Structural-based	Lexical-based	Structural + Lexical	Entropy-based	Wilcoxon test
0.9.0	–	–	–	–	–
0.9.1	66.325	82.023	87.284	**99.348**	+ + +
0.9.2	76.534	**89.348**	76.287	81.356	+ − +
0.9.3	56.285	81.524	81.593	**93.278**	+ + +
0.9.4	59.274	76.524	**98.645**	93.452	+ + −
0.9.5	61.945	79.521	**84.624**	84.012	+ + ≈
0.9.6	53.125	68.514	74.651	**81.254**	+ + +
0.9.7	58.654	**92.735**	85.021	86.457	+ − ≈
0.9.8	68.235	78.954	82.456	**90.412**	+ + +
0.9.9	65.621	85.475	86.379	**98.754**	+ + +
0.9.10	61.832	71.287	**83.254**	82.985	+ + ≈
0.9.11	53.652	78.953	76.154	**89.763**	+ + +
0.9.12	58.623	69.482	80.517	**86.425**	+ + +
0.9.13	67.628	79.257	78.614	**85.286**	+ + +
0.9.14	69.542	**92.341**	88.342	91.362	+ ≈ +
0.9.15	62.485	85.173	**97.624**	96.754	+ + ≈
0.9.16	53.295	82.314	89.652	**98.627**	+ + +
0.9.17	51.354	79.542	76.245	**85.364**	+ + +
0.9.18	54.392	89.645	92.456	**96.235**	+ + +
0.9.19	60.821	86.168	**93.254**	92.451	+ + ≈
0.9.20	63.637	81.254	86.746	**94.267**	+ + +

Table 15 Stability results of MOEA/D over JFreeChart project

Version	MoJoFM (Mn, Mn − 1) (%)				
	Structural-based	Lexical-based	Structural + Lexical	Entropy-based	Wilcoxon test
0.9.0	–	–	–	–	–
0.9.1	63.975	79.757	**96.754**	91.325	+ + −
0.9.2	74.954	77.334	73.973	**89.042**	+ + +
0.9.3	53.895	78.215	79.279	**89.964**	+ + +
0.9.4	56.145	**89.218**	88.331	89.138	+ ≈ ≈
0.9.5	59.556	76.287	79.310	**81.698**	+ + ≈
0.9.6	50.734	66.208	72.337	**78.862**	+ + +
0.9.7	56.854	80.121	**89.707**	83.143	+ + −
0.9.8	65.849	76.643	**88.142**	87.098	+ + ≈
0.9.9	63.626	**89.161**	83.065	88.443	+ ≈ +
0.9.10	59.467	67.573	80.940	**81.671**	+ + ≈
0.9.11	51.857	73.139	71.840	**85.449**	+ + +
0.9.12	56.328	67.165	**87.203**	81.385	+ + ≈
0.9.13	65.263	76.143	**82.308**	82.972	+ + ≈
0.9.14	67.747	81.027	**91.028**	86.048	+ + −
0.9.15	60.190	82.859	95.310	**96.416**	+ + ≈
0.9.16	50.906	80.065	87.338	**96.313**	+ + +
0.9.17	48.957	77.178	73.931	**83.552**	+ + +
0.9.18	51.197	87.561	90.142	93.921	+ + +
0.9.19	58.422	**83.854**	82.940	80.137	+ ≈ ≈
0.9.20	61.742	78.941	84.432	**91.253**	+ + +

operators (e.g., random initial population generation) and they may produce different results for the same problem instance on a different run. To mitigate this threat, we executed each algorithm on the same problem instance 31 times and collected the sample of results. To compare the results of meta-heuristic algorithms, we obtained results by executing each algorithm on each problem instance 31 times and it is statistically analyzed by using Wilcoxon rank sum test with a 95% confidence level (α =5%).

Internal threats to validity: In this category of threats, the effects of experimental design choices, algorithm's parameter settings, and data collection have been considered. The parameter settings of algorithms are based on the similar previous remodularization studies (Mkaouer et al. 2015, 2015a), while for others many-objective algorithms we used a trial-and-error calibration method.

Construct threats to validity: These threats are concerned with the relations between theory and observation.

Table 16 Stability results of IBEA over JFreeChart project

Version	MoJoFM (Mn, Mn − 1) (%)				
	Structural-based	Lexical-based	Structural + Lexical	Entropy-based	Wilcoxon test
0.9.0	–	–	–	–	–
0.9.1	51.661	**88.654**	87.321	87.011	+ ≈ ≈
0.9.2	72.640	75.020	71.659	**86.728**	+ + +
0.9.3	61.581	**86.245**	75.965	81.650	+ − +
0.9.4	63.831	76.904	**86.017**	82.824	+ + −
0.9.5	57.242	**76.973**	76.196	76.384	+ ≈ ≈
0.9.6	48.420	62.891	70.523	**76.548**	+ + +
0.9.7	54.540	77.876	81.345	**87.829**	+ + +
0.9.8	53.535	**85.329**	85.328	84.784	+ ≈ ≈
0.9.9	51.312	**86.747**	79.752	86.229	+ ≈ +
0.9.10	67.153	65.259	**79.638**	72.357	+ + −
0.9.11	49.543	70.825	69.526	**83.135**	+ + +
0.9.12	54.014	64.851	**84.819**	83.171	+ + ≈
0.9.13	62.949	73.829	77.994	**87.658**	+ + +
0.9.14	55.433	72.713	**88.714**	81.734	+ + −
0.9.15	57.876	**88.545**	82.996	82.102	+ − ≈
0.9.16	58.592	77.352	81.024	**95.935**	+ + +
0.9.17	66.643	71.864	75.617	**93.238**	+ + +
0.9.18	58.883	**95.247**	87.828	95.107	+ ≈ +
0.9.19	56.108	78.540	72.626	**87.823**	+ + +
0.9.20	52.428	76.627	82.118	**88.932**	+ + +

Table 17 Stability results of TAA over JFreeChart project

Version	MoJoFM (Mn, Mn − 1) (%)				
	Structural-based	Lexical-based	Structural + Lexical	Entropy-based	Wilcoxon test
0.9.0	–	–	–	–	–
0.9.1	59.347	86.340	75.007	**86.697**	+ ≈ +
0.9.2	71.326	72.706	79.341	**85.414**	+ + +
0.9.3	59.263	73.931	**83.651**	82.336	+ + ≈
0.9.4	62.517	81.590	82.703	**83.510**	+ ≈ ≈
0.9.5	56.923	64.659	63.882	**74.070**	+ + ≈
0.9.6	46.136	61.577	63.209	**74.231**	+ + +
0.9.7	52.226	**85.562**	80.031	84.515	+ ≈ +
0.9.8	52.224	**83.615**	83.014	83.470	+ ≈ ≈
0.9.9	58.513	71.433	**84.438**	83.915	+ + ≈
0.9.10	64.239	62.945	**78.324**	71.043	+ + −
0.9.11	47.229	68.511	67.212	**81.821**	+ + +
0.9.12	51.703	62.537	**82.505**	80.857	+ + ≈
0.9.13	61.231	71.515	**85.680**	85.344	+ + ≈
0.9.14	53.519	70.399	76.400	**89.420**	+ + +
0.9.15	55.562	**86.231**	81.682	79.788	+ − ≈
0.9.16	56.278	75.038	78.710	**93.621**	+ + +
0.9.17	64.329	69.550	73.303	**91.924**	+ + +
0.9.18	56.569	72.933	85.514	**92.793**	+ + +
0.9.19	53.794	76.226	70.312	**85.509**	+ + +
0.9.20	51.124	74.313	79.834	**86.638**	+ + +

The design of fitness functions are based on previous and widely used software remodularization works (Prajapati and Chhabra 2017a; Corazza et al. 2016; Parashar and Chhabra 2016). For a proper comparison between the two algorithms, we assigned an equal number of fitness evaluation.

External threats to validity: The external threats to validity are concerned with the generalization of the results achieved by the proposed approach. The approach has been carried out over medium to large real-world object-oriented software systems with different complexity in terms of number of connections, number of classes, number of modules, and lines of code. The correctness of the authoritativeness remodularization might also affect the results. To obtain the authoritativeness remodularization, we follow the same approach as reported in literature (Prajapati and Chhabra 2017a; Corazza et al. 2016; Erdemir and Buzluca 2014; Wu et al. 2005).

8 Conclusion and Future Directions

A new many-objective software remodularization approach for object-oriented software system has been proposed in this paper. The approach proposes the use of information theoretic proximity measure as a new objective function along with four other objective measures (i.e., inter-module class change coupling, intra-module class change coupling, module count index, and module size index). In addition, our approach utilized different aspects of structural and lexical with their relative weights. Information present in the change-history of the software has also been integrated into the approach for identifying consistent modularization. The proposed approach has been compared with other variants of remodularization evaluation models over seven test software using different search based meta-heuristics (NSGA-III, MOEA/D, IBEA, and TAA). The obtained results have been assessed in terms of authoritative software remodularization, non-extreme distribution, and stability. The results of the evaluation clearly suggest that the proposed approach can be a good alternative to improve the quality of software systems whose quality is not up to the mark. As part of the future work, we perform an empirical study on more problem instance with different configuration settings.

References

Abdeen, H., Ducasse, S., Sahraoui, H.A., Alloui, I. (2009). Automatic package coupling and cycle minimization, in: Proceedings of the 16th working conference on re- verse engineering, 103–112.

Aldana-Bobadilla, E., & Kuri-Morales, A. (2011). A methodology to find clusters in the data based on Shannon's entropy and genetic algorithms. In *Proceedings of the 10th WSEAS international conference on communications, electrical & computer engineering, world scientific and engineering academy and society (WSEAS)* (pp. 272–280). Wisconsin, USA: Stevens Point.

Andritsos, P., & Tzerpos, V. (2005). Information-theoretic software clustering. *IEEE Transaction on software engineering., 31*(2), 150–165.

Anquetil, N., Lethbridge, T. (1999). Experiments with clustering as a software remodularization method. In *Proceedings of 6th Working Conference on Reverse Engineering,* Atlanta, GA, USA 235-255.

Arcuri, A., Briand, L. (2011). A practical guide for using statistical tests to assess randomized algorithms in software engineering, *2011 33rd International Conference on Software Engineering (ICSE)*, Honolulu, HI, 1–10.

Barros, M. (2012). An analysis of the effects of composite objectives in multi-objective software module clustering. *in: Proceedings of the fourteenth international conference on Genetic and evolutionary computation*, 1205-1212.

Bavota, G., Lucia, A. D., Marcus, A., & Oliveto, R. (2010). Software remodularization based on structural and semantic metrics. *In Proceedings of WCRE, 2010*, 195–204.

Bavota, G., Lucia, A. D., Marcus, A., & Oliveto, R. (2013). Using structural and semantic measures to improve software modularization. *Empirical Software Engineering, 18*, 901–932.

Bavota, G., Dit, B., Oliveto, R., Penta, M. D., Poshyvanyk, D., Lucia, A.D. (2013a). An empirical study on the developers' perception of software coupling, *2013 35th International Conference on Software Engineering (ICSE),* San Francisco, CA, 692–701.

Bavota, G., Gethers, M., Oliveto, R., Poshyvanyk, D., & Lucia, A. D. (2014). Improving software modularization via automated analysis of latent topics and dependencies. *ACM Transaction on Software Engineering and Methodology, 4*(1), 1–33.

Bingdong, L., Jinlong, L., Tang, K., & Xin, Y. (2015). Many-objective evolutionary algorithms: A survey. *ACM Computing Survey, 48*(1), 1–37.

Bittencourt, R. A., & Guerrero, D. D. S. (2009). *Comparison of graph clustering algorithms for recovering software architecture module views* (pp. 251–254). In: Proceedings of the European Conference on Software Maintenance and Reengineering, IEEE CS Press.

Corazza, A., Martino, S. D., Maggio, V., & Scanniello, G. (2016). Weighing lexical information for software clustering in the context of architecture recovery. *Empirical Software Engineering, 21*(1), 72–103.

Corne, D.W., Jerram, N.R., Knowles, J.D., Oates, M. J. (2001). PESA-II: Region-based selection in evolutionary multiobjective optimization. *In Proc. 3rd Annual Conference on Genetic Evolutionary Computation*. 283–290.

Cover, T. M., & Thomas, J. A. (1991). Elements of information theory. *Wiley and Sons, 1991*.

Cui, J. F., & Chae, H. S. (2011). Applying agglomerative hierarchical clustering algorithms to component identification for legacy systems. *Information and Software Technology (IST), 53*(6), 601–614.

Deb, K., & Jain, H. (2014). An evolutionary many-objective optimization algorithm using reference-point based non-dominated sorting approach, part I: Solving problems with box constraints. *IEEE Transaction on Evolutionary Computing, 18*(4), 577–599.

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computing, 6*(2), 182–197.

Doval, D., Mancoridis, S., & Mitchell, B. S. (1999). *Automatic clustering of software systems using a genetic algorithm* (pp. 73–81). In: Proceedings of IEEE conference on software technology and engineering practice.

Erdemir, U., & Buzluca, F. (2014). A learning-based module extraction method for object-oriented systems. *Journal of Systems and Software, 97*, 156–177.

Fowler, M., Beck, K., Brant, J., Opdyke, Q., & Roberts, D. (1999). *Refactoring – Improving the Design of Existing Code* (1st ed.). Addison-Wesley.

Glorie, M., Zaidman, A., Deursen, A., & Hofland, L. (2009). Splitting a large software repository for easing future software evolution-an industrial experience report. *Journal of Software Maintenance and Evolution: Research and Practice, 21*(2), 113–141.

Gokcay, E., & Principe, J. C. (2002). Information theoretic clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 24*, 158–171.

Harman, M., Hierons, R., & Proctor, M. (2002). *A new representation and crossover operator for search-based optimization of software modularization* (pp. 1351–1358). In: Proc. genetic and evolutionary computation conference.

Harman, M., Mansouri, S. A., & Zhang, Y. (2012). Search-based software engineering: Trends, techniques and applications. *ACM Computing Survey, 45*(1), 1–61.

Hino, H., & Murata, N. (2014). A nonparametric clustering algorithm with a quintile-based likelihood estimator. *Neural Computing., 26*, 2074–2101.

Jaimes, A.L., Coello Coello, C.A., Barrientos, J.E.U. (2009). Online objective reduction to Deal with many-objective problems. In *the 5th international conference on Evolutionary Multicriterion Optimization*. 423–437.

Jain, H., & Deb, K. (2014). An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach, part II: Handling constraints and extending to an adaptive approach. *IEEE Transactions on Evolutionary Computation, 18*(4), 602–622.

Jinhuang, H., & Jing, L. (2016). A similarity-based modularization quality measure for software module clustering problems. *Information Sciences, 342*(10), 96–110.

Joyce, J. (2008). Bayes theorem. The Stanford encyclopedia of philosophy, fall 2008 edition, Eds: Zalta, Edward N.

Kumari, A. C., Srinivas, K., Gupta, M. P. (2013). Software module clustering using a hyper-heuristic based multi-objective genetic algorithm. 2013 IEEE 3rd international advance computing conference (IACC), Ghaziabad, 813-818.

Mahdavi, K., Harman, M., & Hierons, R. M. (2003). *A multiple hill climbing approach to software module clustering* (pp. 315–324). In: Proceedings of the international conference on software maintenance.

Mamaghani, A. S., & Meybodi, M. R. (2009). *Clustering of software systems using new hybrid algorithms* (pp. 20–25). In: Proceedings of the ninth IEEE international conference on computer and information technology.

Mancoridis, S., Mitchell, B. S., Chen, Y. F., Rorres, C., & Gansner, E. R. (1998). *Using automatic clustering to produce high-level system organizations of source code* (pp. 45–53). In: Proc. int'l workshop program comprehension.

Mancoridis, S., Mitchell, B. S., Chen, Y. F., & Gansner, E. R. (1999). Bunch: A clustering tool for the recovery and maintenance of software system structures. *Proc. IEEE Int'l Conf. Software Maintenance*, 50–59.

Mitchell, B. S., & Mancoridis, S. (2002). Using heuristic search techniques to extract design abstractions from source code. *Proc. Genetic and Evolutionary Computation Conf.*, 1375–1382.

Mkaouer, M. W., Kessentini, M., & Bechikh, S. (2015). On the use of many quality attributes for software refactoring: A many-objective search-based software engineering approach. *Empirical Software Engineering, 21*(6), 2503–2545.

Mkaouer, M. W., Kessentini, M., Shaout, A., Koligheu, P., Bechikh, S., Deb, K., & Ouni, A. (2015a). Many objective software remodularization using NSGA-III. *ACM Transaction on software engineering and methodology, 24*(3), 1–17.

Ouni, A., Kessentini, M., Sahraoui, H., & Boukadoum, M. (2013). Maintainability defects detection and correction: A multi-objective approach. *Journal of Automated Software Engineering (ASE), 20*(1), 47–79.

Ouni, A., Kessentini, M., & Sahraoui, H. (2014). Multiobjective optimization for software refactoring and evolution. *Advances in Computers, 94*, 103–167.

Ouni, A., Kessentini, M., Sahraoui, H., Inoue, K., & Hamdi, M. S. (2015). Improving multi-objective code-smells correction using development history. *Journal of Systems and Software, 105*, 18–39.

Ouni, A., Kula, R. G., Kessentini, M., Ishio, T., Germán, D. M., & Inoue, K. (2016). Search-based software library recommendation using multi-objective optimization. *Journal of Information and Software Technology, Elsevier, 83*, 2016.

Ouni, A., Kessentini, M., Sahraoui, H., Cinneide, M.O., Deb, K., Inoue, K. (2016a). MORE: A multi-objective refactoring recommendation approach to introducing design patterns and fixing code smells. Journal of Software*: Evolution and Process* (JSEP), John Wiley & Sons.

Ouni, A., Kessentini, M., Sahraoui, H., Inoue, K., & Deb, K. (2016b). Multi-criteria code refactoring using search-based software engineering: An industrial case study. *ACM Transactions on Software Engineering and Methodology, 25*(3), 1–23.

Ouni, A., Kessentini, M., Inoue, K., & Cinnéide, M. (2017). Search-based web service anti patterns detection. *IEEE Transactions on Services Computing, 10*(4), 603–617.

Parashar, A., & Chhabra, J. K. (2016). Mining software change data stream to predict changeability of classes of object-oriented software system. *Evolving Systems, 7*(2), 117–128.

Praditwong, K., Yao, X. (2006). A new multi-objective evolutionary optimization algorithm: The two-archive algorithm. *In: Cheung Y-M, Wang Y,* Liu H (eds) Proceedings of the international conference computational intelligence and security, vol 1, 286–291.

Praditwong, K., Harman, M., & Yao, X. (2011). Software module clustering as a multi-objective search problem. *IEEE Transaction on Software Engineering, 37*(2), 264–282.

Prajapati, A., & Chhabra, J. K. (2014). An empirical study of the sensitivity of quality indicator for software module clustering. In *2014 Seventh International Conference on Contemporary Computing (IC3), Noida, (2014)* (pp. 206–211).

Prajapati, A., & Chhabra, J. K. (2017). Improving package structure of object-oriented software using multi-objective optimization and weighted class connections. *Journal of King Saud University - Computer and Information Sciences, 29*(3), 349–364.

Prajapati, A., & Chhabra, J. K. (2017a). Improving modular structure of software system using structural and lexical dependency. *Information and Software Technology, 82*, 96–120.

Prajapati, A., & Chhabra, J. K. (2017b). Harmony search based remodularization for object-oriented software systems. *Computer Languages, Systems & Structures, 47*, 153–169.

Prajapati, A., & Chhabra, J. K. (2018). Many-objective artificial bee colony algorithm for large-scale software module clustering problem, soft computing., *22*(19), 6341–6361.

Prajapati, A., & Chhabra, J. K. (2018a). FP-ABC: Fuzzy-Pareto dominance driven artificial bee colony algorithm for many-objective software module clustering. *Computer Languages, Systems & Structures, 51*, 1–21.

Rachmawati, L., & Srinivasan, D. (2009). Multiobjective evolutionary algorithm with controllable focus on the knees of the Pareto front. *IEEE Transaction on Evolutionary Computation., 13*(4), 810–824.

Sartipi, K., & Kontogiannis, K. (2003). On modeling software architecture recovery as graph matching. In *International Conference on Software Maintenance, ICSM 2003. Proceedings., Amsterdam, the Netherlands* (pp. 224–234).

Sugiyama, M., Niu, G., Yamada, M., Kimura, M., & Hachiya, H. (2014). Information-maximization clustering based on squared-loss mutual information. *Neural Computing, 26*, 84–131.

Tzerpos, V., & Holt, R. C. (1999). MoJo: A distance metric for software clustering. In *Proceedings of the 6th working conference on reverse engineering* (pp. 187–193). GA, USA, October: Atlanta.

Wang, Y., Liu, P., Guo, H., Li, H., & Chen, X. (2010). Improved hierarchical clustering algorithm for software architecture recovery. In *International conference on intelligent computing and cognitive informatics* (pp. 247–250).

Wang, H., Jiao, L., & Yao, X. (2015). Two_Arch2: An improved Two-archive algorithm for many-objective optimization. *IEEE Transactions on Evolutionary Computation, 19*(4), 524–541.

Wohlin, C., Runeson, P., Host, M., Ohlsson, M. C., Regnell, B., & Wesslen, A. (2000). *Experimentation in software engineering: An introduction*. Kluwer Academic Publishers.

Wu, J., Hassan, A. E., & Holt, R. C. (2005). Comparison of clustering algorithms in the context of software evolution. In *In: Proceedings of the 21st IEEE International Conference on Software Maintenance* (pp. 525–535).

Yates, R.B., Neto, B.R. (1999). Modern information retrieval. *Addison-Wesley-Longman*.

Zhang, Q., & Li, H. (2007). MOEA/D: A multi-objective evolutionary algorithm based on decomposition. *IEEE Transaction on Evolutionary Computing, 11*(6), 712–731.

Zitzler, E., & Kunzli, S. (2004). Indicator-based selection in multi-objective search. In *In Proceedings of the 8thInternational Conference on Parallel Problem Solving from Nature. Springer* (pp. 832–842).

Zitzler, E., Laumanns, M., & Thiele, L. (2002). SPEA2: Improving the strength Pareto evolutionary algorithm. *In Proc. Evolutionary. Methods Design Optimization. Control Application*, 95–100.

Dr Amarjeet Prajapati is working as Assistant Professor in the Department of Computer Science & Engineering at Jaypee Institute of Information Technology (JIIT) Noida-62, Uttar Pradesh-India since January 2017. He has completed his PhD degree from the Department of Computer Engineering, NIT Kurukshetra, Haryana in 2017. He obtained his M. Tech degree in Computer Engineering from the Department of Computer Engineering, NIT Kurukshetra, Haryana in 2011. He obtained his B. Tech degree in Computer Science and Engineering from UPTU, Lucknow in 2004. He has presented and published many research papers in reputed journals and various national and international conferences. His important research contribution includes developing search-based software engineering methods that help in improving the modular structure of software systems. His research interests include Software Engineering, Soft Computing, and Metaheuristic Algorithms, etc.

Dr Jitender Kumar Chhabra Professor, Dept of Computer Engg has been always topper throughout his career. He did his B Tech as well as M Tech from N.I.T. Kurukshetra as Gold Medalist. He did his PhD in the area of Software Engineering. He has 25 years of teaching and research experience. He is author of three books from International publisher McGraw Hill including the one Schaum Series International book. He is Reviewer of IEEE Transactions, ACM Transactions, Elsevier, Springer, Wiley, T & F and many other Journals. He has published more than 120 papers in reputed International and National Journals and conferences including more than 50 publications from IEEE, ACM, Elsevier and Springer etc, which are SCI and Scopus indexed. He has visited many countries and presented his research work in USA, UK, Canada, Spain, France, Singapore, Turkey, UAE and Thailand. He has worked in collaboration with multinational IT companies Hewelett Packard and Tata Consultancy Services in the area of Software Engineering and is privileged to be invited as Judge for IBM National Contests. He has also received Sir Isaac Newton Scientific Award, Best Teacher award etc His area of interest are software engineering, soft computing, machine learning and data mining.