Journal of Network Intelligence
Taiwan Ubiquitous Information

Volume **7**, Number **3**, August **2022**

Bad Smells Identification using Community Detected based on Feature Envy Metric

Wan-Chang Jiang

School of Computer Science
Northeast Electric Power University
No.169 Changchun Road, Jilin, Jilin 132012, China
jwchang84@163.com

Jie-Tao Wu

School of Computer Science
Northeast Electric Power University
No.169 Changchun Road, Jilin, Jilin 132012, China
wchu@npu.edu.tw

Xiao-Xi Zhang

School of Computer Science
Northeast Electric Power University
No.169 Changchun Road, Jilin, Jilin 132012, China
zxxyx0711@163.com

Wei-Hua Zhu*

Department of Information Technology
Jilin Technology College of Electronic Information
No.65 Hanyang Road, Jilin, Jilin 132021, China
317010530@qq.com

*Corresponding author: Wei-Hua Zhu

Received November 21, 2021, revised January 11, 2022, accepted March 30, 2022.

Abstract. *Identifying feature envy bad smells plays a critical role in the software evolution, which can help the developer to refactor. In view of the complex scenes of identifying feature envy bad smells, the bad smells identification algorithm using community detection based on feature envy metric is proposed(BSICD). Different from other identification algorithms, we design a directed weighted feature dependency software network. Based on the software network, by considering the direction, weight and class which feature originally belong to, Feature-Class Envy Factor and Class-Class Envy Factor are defined to represent the envy degree of feature with class and the class cohesion. And then, feature membership parameters are calculated to measure the closeness of feature with its source class and target class. Furthermore, feature envy metric is designed based on the variation of feature membership parameters, which will be used to control community detection by a series of feature-moving operations. By comparing the obtained communities with nature ones, the feature envy bad smells can be identified. In order to evaluate the effectiveness of BSICD, we design two group experiments on the open software Colt. BSICD can identify the type of bad smells that the method more interested in classes than the one it actually is in, which is better than FEED algorithm. In comparison to JDeodorant, the criteria of false positive is used to show that BSICD can provide a better identification accuracy.*
Keywords: Bad smells, Software network, Feature envy metric, Community detection

1. **Introduction.** The quality and longevity of a software is largely determined by its internal structures. Then a good design of software internal structures indicates that low coupling and high cohesion [1]. Software should constantly be corrected faults, improved performance or adapted the changed environment in evaluation [2].

Bad smell is a typical symptom in the source code of an objected oriented software [3], which indicates a violation of fundamental design principles that may possibly slow down development in the future. Fowler et al. define 22 recurring bad smells by the elimination of which one may increase the software maintainability [4].

The bad smells can affect the codes of one software on different levels, such as packages [5], classes [6], and methods [7]. And feature envy is one method level bad smell, and the feature envy bad smell is expressed as a method that seems more interested in a class other than the one it actually is in. In other words, when a method of a particular class overly uses the attributes or methods of another class, the method is a feature envy bad smell.

Several approaches have been proposed to detect the feature envy bad smells. A famous move-method-based approach [8] is proposed, and the distance metric between methods and classes is used to identify the bad smells. In paper [9], an approach is introduced for the detection of the Feature Envy design flaw at the level of blocks of code. In essence, a method body is split into a hierarchical structure of blocks. The corresponding detection strategy is applied to identify envious leaf blocks, and these envious blocks are extended in order to localize the Feature Envy issue in the body of the analyzed method. However, the method do not have a very high accuracy regarding the boundaries of these areas and can not detect entire method body level. In almost all these situations, the involved methods looked as feature envy instances at the entire method body level. Woei-Kae Chen et al. propose a dataflow analysis approach for feature envy bad smells [10]. The approach considers the level of blocks of code inside the method bodies. However, this approach is difficult to detect the bad smells in open-source software projects for a large scale. In the paper [11], it proposed a deep learning based feature envy detection approach. The key insight is that deep neural networks. And deep neural networks have a large number of parameters. Consequently, they often require a large number of training data to adjust such parameters, and the training data generated based on such methods may be noisy, and some of them are labeled incorrectly. As a result, the feature envy bad smells are incorrect.

In order to represent one software as the dependency software network at the method and attribute level, two approaches [12, 13] have been proposed. The following observation have been made: methods belonging to the same class tend to be cohesive, while methods from different classes tend to be sparser. The feature envy bad smells increase the coupling between classes and violate the characteristic of communities. Therefore, the community detection approach can be applied to recondition class structures and detect feature envy bad smells. The approach in [6] refactors class structures by using the community detection, however, it neglects the call direction and call times between methods and attributes. Owe to the call direction and call times can better reflect the dependency relationship, our paper will propose a directed weighted feature dependency software network, design a novel feature envy metric and propose a bad smell identification approach using community detection for identifying feature envy bad smells.

In this paper, we analyze the limitation of bad smells identification methods. In order to improve the accuracy of feature envy bad smell identification, we design an algorithm of $BSICD$. $BSICD$ considers that the correlation between feature envy bad smells and the high cohesion and low coupling characteristics of community structures in software, which affects the improvement of $BSICD$ on the accuracy of feature envy bad smells. Previous researches [12, 13] have proved that feature envy bad smells can be identified using community detection, but the influence of weight on the accuracy of identifying bad smells is not considered. Section 2 of this paper builds the directed weighted feature dependency software network. Reference to the research [14] discover useful patterns in an uncertain database, Section 3 designs an uncertain feature envy bad smells candidate database and designs the feature envy metrics based on the feature membership to identify the certain feature envy bad smells. The bad smells identification algorithm using community detection is proposed in Section 4. In Section 5, the experiment was conducted on the Colt to evaluate effectiveness of our approach. Section 6 gives the final conclusion and looks forward to the future work.

2. **Directed Weighted Feature Dependency Software Network.** In order to accurately identify feature envy bad smells in one objected oriented software, two kinds of dependency relationship are considered. Method-method and method-attribute will be extracted for building software network. Apart from presence of dependency relationship, multiplicity of dependency relationship is also considered. Thus, the directed weighted feature dependency software network $G(V,E,W)$ is constructed to characterize internal structures and feature dependency relationships of the software.

As features of class, both methods and attributes are extracted as nodes. And each feature of every class is represented by one node. The feature k of class I is represented as v_{Ik}, k=1,2,...,n and n is the number of features in the software, I = 1,2,...,N and N is the number of classes in the software. Hence, V can be represented as the set of feature level of nodes and V=$\{v_{Ik}|k = 1,2,...,n \, and \, I = 1,2,...,N\}$. As well, V also can be represented as the set of class level of nodes and V=$\{V_I|I = 1,2,...,N\}$, where V_I is a subset to represent the class I.

Further, when one method calls another feature or one method references another attribute, the method and the feature have one dependency relationship, which is extracted as one edge. The edge reflects the presence of dependency relationship. And direction and weight of the edge reflects multiplicity of dependency relationship. That is to say, if method k directly calls feature l, or method k references attribute l, there will be a directed edge e_{IkJl} from node v_{Ik} to v_{Jl}. And E=$\{e_{IkJl}\}$ is the directed edge set in the software network. Moreover, the calling or referenced times of the dependency relationship can be extracted as the weight of the corresponding edge. And $W = \{w_{IkJl}\}$ is the set of weight of each edge, where w_{IkJl} denotes the weight of edge e_{IkJl}.

3. **Feature Envy Metric based on Feature Membership.** In order to identify feature envy bad smells, this section designs a bad smells identification using community detection approach based on feature envy matric. Firstly, In the directed weighted feature dependency software network $G(V,E,W)$, the XOR formula is used to determine a feature envy candidate nodes set by considering whether two features belong to the same source class or not. Secondly, considering that the feature belongs to the source class and the target class, the weights is used to design Feature-Class Envy Factor, SourceClass-Class Envy Factor, TargetClass-Class Envy Factor, and a feature envy metric based on the three factors is defined. Finally, a bad smells identification approach using community detection based on the feature envy metric is design.

3.1. **Feature envy candidate nodes set analysis.** In the G, owing to most features are not the feature envy bad smells, and in order to narrow down the calculation and save time, a feature envy candidate nodes set V^{Can} is defined to improve identification efficiency.

Definition 3.1. *(feature envy candidate nodes set): Two feature nodes that have an edge but are defined in two different classes, the set of all nodes meet the characteristics in the G is defined as the feature envy candidate nodes set V^{Can}. The V^{Can} is calculated by the following steps.*

Firstly, if feature k is defined in class I, it is represented as $f_c(v_{Ik}) = I$, where v_{Ik} is the node of feature k. Secondly, a method can directly call features of one class that it belongs to and also features of other classes through references. As a result, there are two types edges in the directed edge set E. The first type is that two nodes have an edge and belong to the same class. The second type is that two nodes have an edge but belong to different classes.

By using the XOR operation, which type of an edge of E is can be denoted as follows:

$$f(e_{IkJl}) = f_c(v_{Ik}) \oplus f_c(v_{Jl}) = I \oplus J \tag{1}$$

Where $f_c(v_{Ik})$ represents the class I that feature v_{Ik} originally belongs to. And $f_c(v_{Jl})$ represents the class J that feature k originally belongs to.

As a result, edges of E can be divided into two types by Formula (1). When $f(e_{IkJl})$ is equal to 0, the edge e_{IkJl} belongs to the first type. When $f(e_{IkJl})$ is equal to 1, the edge e_{IkJl} belongs to the second type.

We define an edge set E^{Can} to represent the second type of edges in E. And E^{Can} can be denoted as follows:

$$E^{Can} = \{e_{IkJl} | (e_{IkJl} \in E) \cap f(e_{IkJl}) = 1\} \tag{2}$$

For all edges in E^{Can}, the corresponding nodes constitute a candidate node set V^{Can}. And by using the candidate edge set E^{Can}, the set V^{Can} can be obtained as follows:

$$V^{Can} = \{v_{Ik}, v_{Jl} | ((v_{Ik}, v_{Jl}) \in e_{IkJl})\, and\, e_{IkJl} \in E^{Can}\} \tag{3}$$

Once feature v_{Ik} in V^{Can} is defined in class I, class I is named the source class of v_{Ik}. When $e_{IkJl} \in E^{Can}$, once node v_{Jl} is defined in class J, class J is named the target class of v_{Ik}.

For all feature nodes in V^{Can}, their source classes and target classes constitute a candidate class set C^{Can}. And the set C^{Can} can be denoted as follows:

$$C^{Can} = \{V_I, V_J | (e(v_{Ik}, v_{Jl}) \in E^{Can}) \cap (I = f_c(v_{Ik}), J = f_c(v_{Jl}))\} \tag{4}$$

3.2. **Feature envy metric.** To identify whether each feature node v_{Ik} of V^{Can} is the feature envy bad smell, the feature envy metric is designed, and the feature envy metric value is calculated to control community detection.

For each node v_{Ik} of V^{Can}, we traverse all the edges of G by XOR formula, and find that there are two types of edges. The first type is that an edge between two nodes but the two nodes are defined in the same class. We define an edge set E^{in}_{Ik} contains this type edges. The second type is that an edge between two nodes but the two nodes are defined two different classes. We define an edge set E^{out}_{Ik} contains this type edges. E^{in}_{Ik} and E^{out}_{Ik} are denoted as follows:

$$E_{Ik}^{in} = \{e_{IkJl} | (e_{IkJl} \in E) \cap (f_c(v_{Ik}) \oplus f_c(v_{Jl}) = 0)\} \tag{5}$$
$$E_{Ik}^{out} = \{e_{IkJl} | (e_{IkJl} \in E) \cap (f_c(v_{Ik}) \oplus f_c(v_{Jl}) = 1)\} \tag{6}$$

By using the two kinds of edges and the corresponding weights, the Feature-Class Envy Factor is defined and calculated.

Definition 3.2. *(Feature-Class Envy Factor): For each node v_{Ik} of V^{Can}, owing to different types and different strengths of dependency relationships, the envy degree of v_{Ik} and its source class can be affected. By utilizing the information that v_{Ik} calls features belonging to the source class I or other classes, a Feature-Class Envy Factor ($F\_CEF(v_{Ik})$) is defined to assess the calling ratio of features,and $F\_CEF(v_{Ik})$ can be calculated as follows:*

$$F\_CEF(v_{Ik}) = \frac{\sum_{(w_{IkJl} \in W_{Ik}^{in})}(w_{IkJl})}{\sum_{(w_{IkJl} \in W_{Ik}^{in})}(w_{IkJl}) + \sum_{(w_{IkJl} \in W_{Ik}^{out})}(w_{IkJl})} \tag{7}$$

where W_{Ik}^{in} and W_{Ik}^{out} are weight sets corresponding to edge sets E_{Ik}^{in} and E_{Ik}^{out}. W_{Ik}^{in} and W_{Ik}^{out} are expressed as follows:

$$W_{Ik}^{in} = \{w_{IkJl} | (f_c(v_{Ik}) \oplus f_c(v_{Jl}) = 0)\} \tag{8}$$
$$W_{Ik}^{out} = \{w_{IkJl} | (f_c(v_{Ik}) \oplus f_c(v_{Jl}) = 1)\} \tag{9}$$

The smaller value of $F\_CEF(v_{Ik})$ is, the more likely the feature k is to be the feature envy bad smell. However, only consider the feature envy ratio to detect bad smells is not comprehensive enough. The class structures play a role in detecting bad smells, the cohesion of one class can be calculated to measure envy degree of the class. By considering the node information contained in the class V_I of C^{Can}, $V_I = \{v_{Ik}, k = 1, 2, ..., n\}$, and the different types of dependency relationship of class I, the cohesion of class I can be defined. We define an edge set E_I of class I and the set contains follows two types edges.

When $f(e_{IkJl}) = 0$, e_{IkJl} includes two nodes v_{Ik} and v_{Jl} that belong to the same class I. We define an edge set E_I^{in} that contains e_{IkJl}. When $f(e_{IkJl}) = 1$, we use e_{IkJl} for this type edge, and e_{IkJl} includes two nodes v_{Ik} and v_{Jl}, however, v_{Ik} and v_{Jl} do not belong to the same class, and only one node v_{Ik} belong to class I. We define an edge set E_I^{out} contains e_{IkJl}. E_I^{in} and E_I^{out} are denoted as follows:

$$E_I^{in} = \{e_{IkJl} | f_c(v_{Ik}) \oplus f_c(v_{Jl}) = 0\} \tag{10}$$
$$E_I^{out} = \{e_{IkJl} | f_c(v_{Ik}) \oplus f_c(v_{Jl}) = 1\} \tag{11}$$

And the edge set E_I is expressed as follows:

$$E_I = E_I^{in} \cup E_I^{out} \tag{12}$$

Definition 3.3. *(SourceClass-Class Envy Factor, TargetClass-Class Envy Factor): In order to measure the cohesion of class I, by using weights that corresponds to edge sets E_I^{in} and E_I^{out}, SourceClass-Class Envy Factor ($SC\_CEF(V_I)$) is defined to calculate the envy ratio that methods of source class I calls features belonging to source class I and the whole software, $SC\_CEF(V_I)$ can measure the cohesion of class I. Thus, for each feature node v_{Ik} of V^{Can}, the target class J is corresponding to v_{Ik}, TargetClass-Class Envy Factor ($TC\_CEF(V_J)$) is defined to calculate the envy ratio that methods of target class J call features belonging to target class J and the whole software, and $SC\_CEF(V_I)$*

can measure the cohesion of class J. $SC\_CEF(V_I)$ and $(TC\_CEF(V_J))$ are calculated as follows:

$$SC\_CEF(V_I) = \frac{\sum_{v_{Ik}\in V_I}\sum_{w_{IkJl}\in W_{Ik}^{in}}(w_{IkJl})}{\sum_{v_{Ik}\in V_I}\sum_{w_{IkJl}\in W_{Ik}^{in}}(w_{IkJl}) + \sum_{v_{Ik}\in V_I}\sum_{w_{IkJl}\in W_{Ik}^{out}}(w_{IkJl})} \quad (13)$$

$$TC\_CEF(V_J) = \frac{\sum_{v_{Jl}\in V_J}\sum_{w_{IkJl}\in W_{Jl}^{in}}(w_{IkJl})}{\sum_{v_{Jl}\in V_J}\sum_{w_{IkJl}\in W_{Jl}^{in}}(w_{IkJl}) + \sum_{v_{Jl}\in V_J}\sum_{w_{IkJl}\in W_{Jl}^{out}}(w_{IkJl})} \quad (14)$$

where W_{Jl}^{in} and W_{Jl}^{out} are weight sets corresponding to edge sets E_{Jl}^{in} and E_{Jl}^{out}, and E_{Jl}^{in} and E_{Jl}^{out} represent the two types edge sets of E_{Ik} respectively.

Let node v_{Ik} belong to one target class J, the nodes of the source class I, nodes of the target class J and dependency relationships between these nodes will change. That is, the cohesion of class I and class J also will change. Thus, we will recalculate the Feature-Class Envy Factor $F\_CEF(v_{Ik})'$, SourceClass-Class Envy Factor $SC\_CEF(V_J)'$ and TargetClass-Class Envy Factor $TC\_CEF(V_J)'$ by using the following formulas:

$$F\_CEF(v_{Ik})' = \frac{\sum_{(w_{IkJl}\in W_{Ik}^{out})}(w_{IkJl})}{\sum_{(w_{IkJl}\in W_{Ik}^{in})}(w_{IkJl}) + \sum_{(w_{IkJl}\in W_{Ik}^{out})}(w_{IkJl})} \quad (15)$$

$$SC\_CEF(V_I)' = \frac{\sum_{v_{Ik}\in V_I}\sum_{w_{IkJl}\in W_{Ik}^{in}}(w_{IkJl}) - \sum_{(w_{IkJl}\in W_{Ik}^{in})}(w_{IkJl})}{\sum_{v_{Ik}\in V_I}\sum_{w_{IkJl}\in W_{Ik}^{in}}(w_{IkJl}) + \sum_{v_{Ik}\in V_I}\sum_{w_{IkJl}\in W_{Ik}^{out}}(w_{IkJl})} \quad (16)$$

$$TC\_CEF(V_J)' = \frac{\sum_{v_{Jl}\in V_J}\sum_{w_{IkJl}\in W_{Jl}^{in}}(w_{IkJl}) + \sum_{(w_{IkJl}\in W_{Jl}^{in})}(w_{IkJl})}{\sum_{v_{Jl}\in V_J}\sum_{w_{IkJl}\in W_{Jl}^{in}}(w_{IkJl}) + \sum_{v_{Jl}\in V_J}\sum_{w_{IkJl}\in W_{Jl}^{out}}(w_{IkJl})} \quad (17)$$

The change of Feature-Class Envy Factor can reflect the difference of node v_{Ik} envy source class and target class, the change of SourceClass-Class Envy Factor and TargetClass-Class Envy Factor can reflect the changes of cohesion of source class and target class before and after node v_{Ik} movement. Then the change of Feature-Class Envy Factor $\Delta F\_CEF(v_{Ik})$, SourceClass-Class Envy Factor $\Delta SC\_CEF(V_I)$, and TargetClass-Class Envy Factor $\Delta TC\_CEF(V_J)$ are also recalculated as follows:

$$\Delta F\_CEF(v_{Ik}) = F\_CEF(v_{Ik})' - F\_CEF(v_{Ik}) \quad (18)$$

$$\Delta SC\_CEF(V_I) = SC\_CEF(V_I)' - SC\_CEF(V_I) \quad (19)$$

$$\Delta TC\_CEF(V_J) = TC\_CEF(V_J)' - TC\_CEF(V_J) \quad (20)$$

Definition 3.4. *(Feature Envy metric): By calculating that the feature node belongs to different classes, the information of the class containing the feature in G, resulting in the change of the edge information. By calculating the change amount of the envy degree between the feature itself and the source class and the change amount of cohesion degree between the source class and the target class, a Feature Envy metric $FE(v_{Ik})$ of v_{Ik} is defined. The Feature Envy metric $FE(v_{Ik})$ can be used to control whether v_{Ik} accepts movement operations during the process of community detection. By using the dependency relationship that feature node v_{Ik} belongs to source class I, and the dependency relationship that v_{Ik} belongs to target class J, $FE(v_{Ik})$ is calculated as follows:*

$$FE(v_{Ik}) = \Delta F\_CEF(v_{Ik}) + \Delta SC\_CEF(V_I) + \Delta TC\_CEF(V_J) \quad (21)$$

4. **Bad Smells Identification Algorithm using Community Detection.** By using the feature envy metric, the bad smells identification algorithm using community detection ($BSICD$) is proposed. The algorithm can identify the feature envy bad smells in the feature level of one software, which can be divided into four stages. First of all, we extract a software as a directed weighted feature dependency software network G. Then, the candidate feature node set V^{Can} and the candidate class set C^{Can} are obtained. Third, the community detection starts from a state that each feature belongs to a specific community, in which the feature is defined, and it is not a random community as that in [12]. For each node v_{Ik} of V^{Can}, the feature envy metric $FE(v_{Ik})$ is designed. And community detection is proceeded by feature-moving operations based on the feature envy metric. In this stage, the moving operation of v_{Ik} can be controlled by using $FE(v_{Ik})$. And $FE(v_{Ik})$ decides whether accept or reject this feature movement. When $FE(v_{Ik})$ is greater than 0, v_{Ik} accepts the feature movement, otherwise, v_{Ik} rejects the movement. Finally, the new communities can be obtained after moving operations. Compared with nature communities, if a feature is not in nature community, it will be identified as the feature envy bad smell. The algorithm is described as follows:

Algorithm

Input: an objected oriented software
Output: feature envy bad smells
(1) For each class I of software do 2-6
(2) For each feature k (method k or attribute k) of class I do 3-6
(3) Extract feature k as node v_{Ik} and put it into node set v_I, put v_I into node set v;
(4) If (v_{Ik} calls other v_{Jl}) do 5-6
(5) Extract the dependency relationship as a directed edge e_{IkJl} and put it into edge set E;
(6) Extract call times as weight w_{IkJl} and put it into edge set W;
(7) For each $e(v_{Ik}, v_{Jl})$ in E do 8-10
(8) If ($f(e_{IkJl})$=1) do 9-10
(9) Put e_{IkJl} into edge set E^{Can}, put v_{Ik} and v_{Jl} into node set V^{Can};
(10) Put $f_c(v_{Ik})$ and $f_c(v_{Jl})$ into class set C^{Can};
(11) For each node v_{Ik} of V^{Can} with 12-16
(12) Feature-Class Envy Factor $F\_CEF(v_{Ik})$ is calculated by Formula (7);
(13) SourceClass-Class Envy Factor $SC\_CEF(V_I)$ and TargetClass-Class Envy Factor $TC\_CEF(V_J)$ are calculated by Formulas (13) and (14);
(14) When v_{Ik} belong to target class J, $F\_CEF(v_{Ik})'$, $SC\_CEF(V_I)'$ and $TC\_CEF(V_J)'$ are calculated by Formulas (15), (16) and (17);
(15) $\Delta F\_CEF(v_{Ik})$, $\Delta SC\_CEF(V_I)$, $\Delta TC\_CEF(V_J)$ are calculated by Formulas (18), (19) and (20);
(16) Feature envy metric is calculated with 12-16;
(17) For each node v_{Ik} in G do 19-20;
(18) Every node belongs to a specific community;
(19) When the feature envy metric $FE(v_{Ik})$ is greater than 0, we move node v_{Ik} of V^{Can} from source class to target class;
(20) With 19-20, new communities can be obtained;
(21) Compare with nature communities, the feature envy bad smell can be identified.

---

5. **Experiments.** Firstly, modularity [15] is used to evaluate effectiveness of the proposed approach $BSICD$. Then, to evaluate accuracy of $BSICD$, we compare the identified result with that of two other approaches $FEED$ [10] and JDeodorant [8, 16]. $FEED$ is a feature envy detection algorithm based on dataflow analysis. JDeodorant (version 5.0.0.201611112330) is a refactoring tool, which adapts the moving-method refactoring to identify bad smells.

5.1. **Experiment Object.** The experiment has been conducted on the objected oriented software Colt [17, 18]. As written in Java, Colt provides an open source library for high-performance scientific and technical computing with strong callability and reusability. The average LCOM value in CK metrics suite indicates that Colt system has poor cohesion. The CK metrics can measure cohesion and coupling of software and reflect the viewpoints of experienced of software. From the perspective of modularity, Colt has obvious community structures, which has advantages in using community detection to identify feature envy bad smells. Colt is abstracted as a directed weighted feature dependency software network, and the basic information is shown as Table 1 and the software network is shown as Figure 1. There are 125 classes in Colt, which indicates 125 communities in the start state during the community detection of $BSICD$, and each colour in the Figure 1 denotes one community. In section 5.3, the three community structures represented by the three colours in Figure 1 will be studied and analysed.

TABLE 1. Basic information of the directed weighted feature dependency software network of Colt

Software	Version	Number of nodes	Number of edges	Number of classes	Number of edges in E^{Can}	Number of nodes in C^{Can}
Colt	1.0.1	4375	6861	125	1101	1260

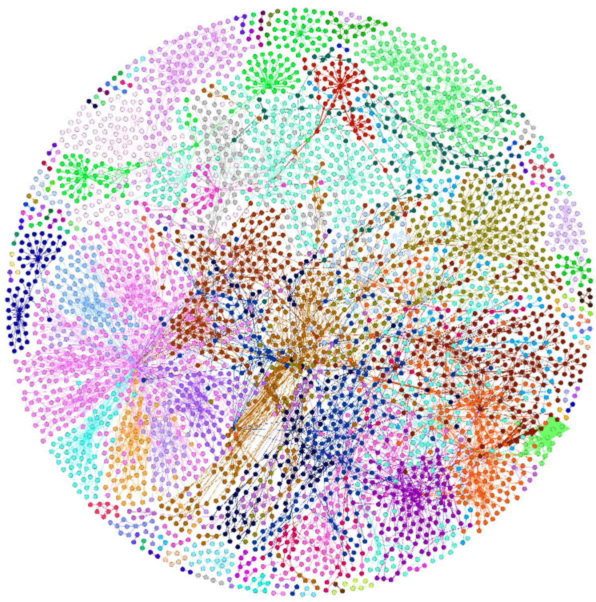


FIGURE 1. The directed weighted feature dependency software network of Colt

5.2. **Modularity.** The feature envy bad smells of $BSICD$ identified and its source class are shown as Table 2. In order to evaluate the effectiveness of $BSICD$, we calculate the modularity value of results of $BSICD$. And the feature envy nodes in software network are evaluated in the Table 3 from modularity perspective.

TABLE 2. Feature envy bad smells of $BSICD$ identified

Number	Feature envy nodes	Features	Classes
1	1139	$Keys$	$AbstractDoubleIntMap$
2	196	$Sort$	$Sorting$
3	1081	$Beta$	$Random$
4	1688	$zdemo1$	$DenseObjectMatrix3D$
5	1519	$toList$	$DoubleFactory1D$
6	1227	$SparseDoubleMatrix3D$	$SparseDoubleMatrix3D$
7	3445	$random$	$DoubleFactory2D$
8	4347	$composeDiagonal$	$DoubleFactory1D$
9	2446	$doubleTest31()$	$TestMatrix2D$
10	136	$like2D$	$DenseObjectMatrix3D$

TABLE 3. Modularity of the identified feature envy bad smells of $BSICD$

Feature node	The source class				The target class				The whole software
	number	Q_S	Q'_S	ΔQ_S	number	Q_T	Q'_T	ΔQ_T	ΔQ
1139	107	0.56	0.578	0.018	111	0.658	0.658	0.000	0.018
196	197	0.530	0.578	0.048	205	0.537	0.549	0.012	0.06
1081	197	0.530	0.537	0.007	201	0.607	0.633	0.026	0.033
1688	125	0.460	0.444	-0.016	131	0.643	0.674	0.031	0.015
1519	128	0.724	0.746	0.022	126	0.704	0.708	0.004	0.026
1227	171	0.858	0.857	-0.001	141	0.626	0.628	0.002	0.001
3445	129	0.527	0.528	0.001	221	0.760	0.763	0.003	0.004
4374	128	0.686	0.724	0.038	221	0.761	0.759	-0.002	0.036
2446	175	0.628	0.617	-0.011	177	0.550	0.587	0.037	0.026
136	150	0.750	0.777	0.027	149	0.722	0.703	-0.019	0.008

Modularity can be used to calculate the class cohesion of Colt as shown in Table 2. Before taking a feature-moving operation, we calculate the modularity Q_S of the source class of one feature with Gephi tool [19]. And after taking the feature-moving operation, the modularity Q'_S of the source class of one feature is calculated. Then, $\Delta Q_S = Q'_S - Q_S$, which can reflect the variation of the source class in modularity Q_S.

Similarity with modularity calculation of the source class, Q_T, Q'_T and ΔQ_T are calculated to represent the modularity of target class of the feature. Q_T represents the modularity before taking the feature-moving operation, Q'_T represents the modularity after taking moving operation, ΔQ_T represents the change of modularity.

For a feature envy bad smell, $\Delta Q = \Delta Q_S + \Delta Q_T$, which can reflect changing in modularity of the whole software of Colt. For each row in Table 3, when both ΔQ_S and ΔQ_T of a feature node are greater than 0, moreover, ΔQ is greater than 0, as a result, the cohesion of the source class I and the target class J of the feature are increased. For every feature node in Table 3, although not all ΔQ_S and ΔQ_T are greater than 0, ΔQ is greater than 0. Once ΔQ of one feature is greater than 0, the cohesion of the whole software of Colt is increased, which means that the feature moving operation is meaningful. That is, the feature should be defined in its target class and it is identified a feature envy bad smell.

When feature envy nodes in Table 3 move to the target class in turn, the Figure 2 is used to represented the variation trend of modularity of the software network. The abscissa represents the feature envy nodes of Table 2, the ordinate represents the variation of Colt in modularity. As shown in Figure 2, after the feature envy node moved into the target class one by one, the modularity value of the whole software network gradually increases. The modularity value increase and the whole system Colt shows an upward trend, which means the whole system Colt more consistent with the software design rule of high cohesion and low coupling.

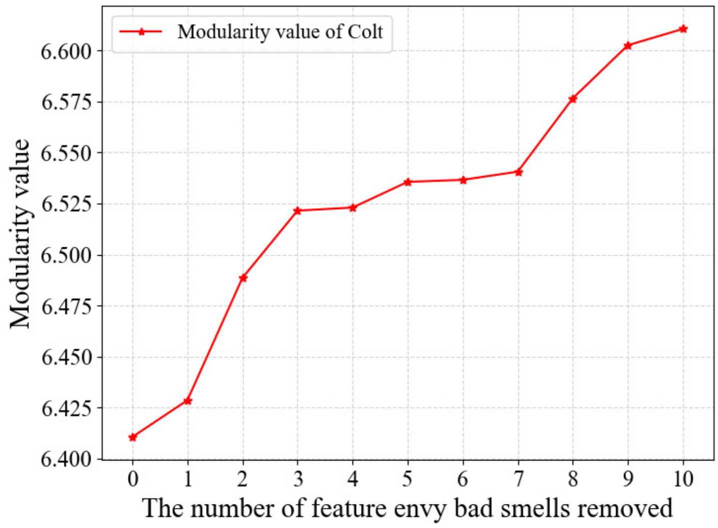


FIGURE 2. Variation of the Colt in modularity

Based on the above analysis of the Table 3 and Figure 2, the feature envy nodes in Table 2 can be evaluated effectively by $BSICD$.

5.3. **Comparison with *FEED*.** The feature envy bad smells will be divided into two types. The two types of feature envy bad smells are shown as Figure 3. The method of the first type is more interested in another class than the one it actually in. When feature k calls features $k1$, $k2$, $k3$ of class J, and feature k does call any features of class I. Feature k is called as the first type of feature envy bad smells. The method of the second type is interested in the other classes than the one it actually is in. That is, when feature k calls features in $J1$ and $J2$, the sum number of calling features is greater than that of source class I, feature k is the second feature envy bad smell.

The proposed approach $BSICD$ can identify the both two types of bad smells, but the $FEED$ algorithm is not feasible for the second type bad smells identification. We give one case to evaluate the advantage of $BSICD$ and disadvantage of $FEED$ on the second type feature envy bad smells. Method $sample()$ of class $DoubleFactory2D$ is a feature envy bad smell, which is detected by $BSICD$. The method $sample()$ is defined in class $DoubleFactory2D$. It calls two methods that belong to class $DoubleFactory2D$ and is called by four methods that belong to two other different classes $BenchmarkMatrix$ and $TestMatrix2D$. From the call relationships of $sample()$, $sample()$ fits the second type of feature envy bad smells.

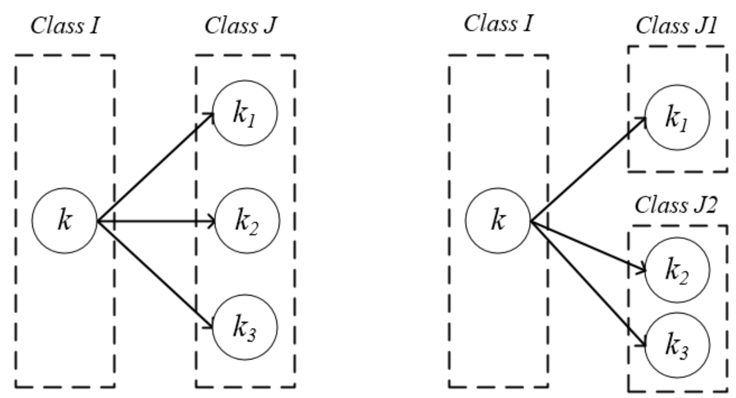
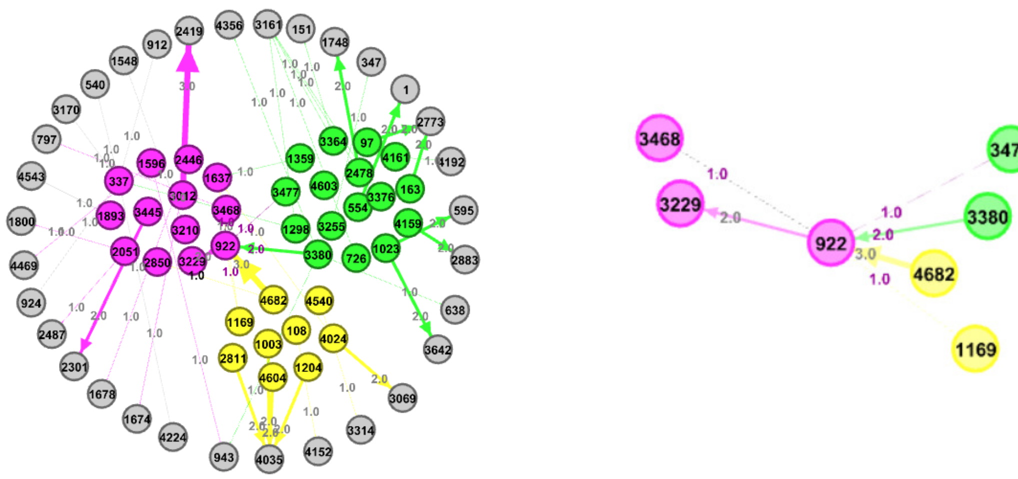


FIGURE 3. Variation of the Colt in modularity

$BSICD$ algorithm can identify the second type bad smells. In the start state of $BSICD$ algorithm, the community that $sample()$ belongs to and the communities that have dependency relationships with $sample()$ are shown in Figure 4. The purple circle shows the community 1 that class $DoubleFactory2D$ belongs to. The green circle shows the community 2 that class $BenchmarkMatrix$ belongs to. The yellow circle shows the community 3 that class $TestMatrix2D$ belongs to. The class $DoubleFactory2D$ is the source class, the class $BenchmarkMatrix$ and $TestMatrix2D$ are target classes. From the Figure 4(A), the feature node 922, 3229, 3468, 3477, 3380, 4682, 1169 are placed into the V^{Can} using the formula (1). And the edges of feature nodes as Figure 4(B).



(A) The dependency relationships of $sample()$ with three communities

(B) The dependency relationships of $sample()$

FIGURE 4. Local software network of $sample()$

From the Figure 4(B), the relationships between feature node 922 and the feature nodes of source class are not closeness. The $sample()$ is more interested in the target classes than class. $F\_CEF(v_{Ik})$ can be calculated by using the formula (8), which considers the dependency relationship between $sample()$ and the community 2, the dependency relationship between $sample()$ and the community 3 as a whole, we can calculate the $FE(v_{Ik})$ of feature nodes of V^{Can}.

From Table 4, only the $FE(v_{Ik})$ of feature node 922 greater than 0. The feature node 922 accepts feature movement during community detection. The $sample()$ is identified

TABLE 4. $FE(v_{Ik})$ of feature nodes of V^{Can}

Feature node	$F\_CEF(v_{Ik})$	$F\_CEF(v_{Ik})'$	$SC\_CEF(V_I)$	$SC\_CEF(V_I)'$	$TC\_CEF(V_J)$	$TC\_CEF(V_J)'$	$FE(v_{Ik})$
922	2/9	4/9	5/18	2/18	3/19	7/19	0.266
3229	2/3	1/3	5/18	4/18	4/19	4/21	-0.408
3468	1/2	1/2	5/18	5/18	3/19	3/19	0
3477	1/2	1/2	5/29	5/29	5/18	5/18	0
3380	2/3	1/3	5/29	4/27	5/18	7/19	-0.266
4682	2/5	3/5	4/19	2/16	5/18	2/21	-0.068
1169	1/2	1/2	4/19	4/19	5/18	5/18	0

as the feature envy bad smell. Because the $sample()$ call features of different classes, the $sample()$ is the second type of feature envy bad smells.

Furthermore, we analyse the reason of $FEED$ algorithm can not identify the second type of feature envy bad smell. As the statement granularity is considered as the smell detection unit in $FEED$ algorithm, when different methods calling features of different classes envies the same method, the parameter of calling will be calculated for different classes. Method $run()$ of class $BenchmarkMatrix$ calls $sample()$, method $runSpecial()$ of class $BenchmarkMatrix$ calls $sample()$, method $doubleTest23()$ of $TestMatrix2D$ calls $sample()$, and method $doubleTest29()$ of $TestMatrix2D$ calls $sample()$. For the method $sample()$, the parameter of calling of $Benchmark$ is equal to 2, and the parameter of calling of $TestMatrix2D$ is equal to 2. Since $sample()$ calls two features that its source class, so the $FEED$ algorithm can not detect the sample() as the feature envy bad smell.

As a result, feature envy bad smells of the second type can not be identify by $FEED$ algorithm, However, the feature envy bad smells of the second type can be identify by $BSICD$ algorithm. Compare with $FEED$ algorithm, $BSICD$ algorithm can identify one more type of bad smells than $FEED$ algorithm.

5.4. **Comparison with JDeodorant.** As an Eclipse plug-in tool, JDeodorant identifies feature envy bad smells by applying the moving-method refactoring. In order to illustrate the accuracy of $BSICD$, we compare the identified results with that of JDeodorant under two evaluation criteria.

(1)True Positive: Feature envy bad smells that can both be identified by and JDeodorant;

(2)False Positive: Feature envy bad smells that can not be identified by or JDeodorant.

$BSICD$ can identify 25 feature envy bad smells of Colt, however JDeodorant can identify only 23 bad smells. As shown in the Table 5, these 23 bad smells can be represented under the criteria of true positive. There are two methods $DenseObjectMatrix3D$. $like2D()$ and $TestMatrix2D$. $doubleTest31()$ that are identified by $BSICD$ and not identified by JDeodorant. The advantage of $BSICD$ can be shown under the criteria of false positive.

TABLE 5. Feature envy smells of Colt under $BSICD$ and JDeodorant.

	$BSICD$	JDeodorant
True Positive	25	23
False Positive	0	2

In order to prove these two methods $DenseObjectMatrix3D.\ like2D()$ and $TestMatrix2D.\ doubleTest31()$ are feature envy bad smells, we use the Coupling between object classes (CBO) [20] from the CK suite to evaluate the advantage of $BSICD$. CBO for a class is a count of the features number of other classes to which it is coupled. Good software design practice calls for minimizing coupling. We calculate the CBO of classes of $like2D()$ and $doubleTest31()$ belonging to.

Firstly, the feature calling graph of $like2D()$ can be extracted by using Doxygen [21] tool. From feature calling graph, we can calculate the CBO of $like2D()$. As we can observe from the feature calling graph of Figure 5, because the method $like2D()$ heavy uses methods and attributes of class $AbstracMatrix3D$, and $like2D()$ does not use any methods and attributes of class $DenseObjectMatrix3D$, and the coupling between class $DenseObjectMatrix3D$ and class $AbstractMatrix3D$ is too high. The CBO of $DenseObjectMatrix3D$ is 2. If the $like2D()$ belongs to $AbstracMatrix3D$, the CBO of class $DenseObjectMatrix3D$ is 0. According to the good software design practice and the definition of feature envy bad smells, the method $like2D()$ is a feature envy bad smell.

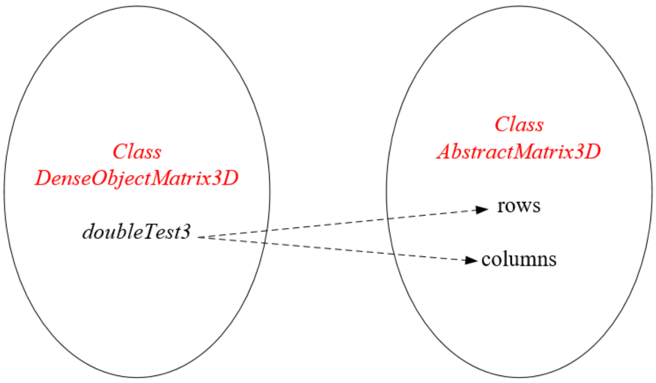


FIGURE 5. Feature calling graph of method $like2D()$

Secondly, from the feature calling graph of the method $doubleTest31()$ in Figure 6, because $doubleTest31()$ calls features of class $DoubleMatrix1D$ and $DoubleFactory1D$, the CBO of class $TestMatrix2D$ is 5. If $doubleTest31()$ belong to class $DoubleFactory1D$, the CBO of $TestMatrix2D$ is 0, and the CBO of $DoubleFactory1D$ is 3, which the coupling of the class $TestMatrix2D$ is decreased. If $doubleTest31()$ belong to class $DoubleMatrix1D$, the CBO of $DoubleMatrix1D$ is 2, which the coupling of the class $DoubleFactory1D$ is decreased. When $doubleTest31()$ belongs to different classes, the CBO changes of the whole system are shown in Table 6.

TABLE 6. CBO of classes under $doubleTest31()$ belongs to different classes.

	CBO of $TestMatrix2D$	CBO of $DoubleFactory1D$	CBO of $DoubleMatrix1D$	CBO of Software
$doubleTest31()$ belongs to $TestMatrix2D$	5	0	0	5
$doubleTest31()$ belongs to $DoubleFactory1D$	0	3	0	3
$doubleTest31()$ belongs to $DoubleMatrix1D$	0	0	2	2

From Table 6, we know that the CBO of the whole software is decreased, which is consistent with design rules of low coupling. That is to say, when the method $doubleTest31()$

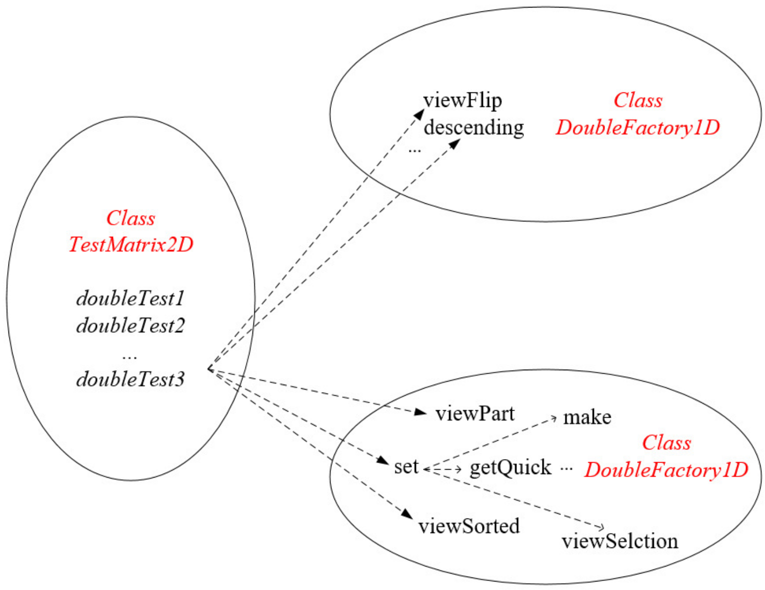


Figure 6. Feature calling graph of method $doubleTest31()$

belongs to class $DoubleMatrix1D$, the coupling of whole system is the lowest. And the CBO of classes of whole software can be shown as Figure 7, the different colours represent that the $doubleTest31()$ belongs to different classes. From the Figure 7, we can know that the CBO of whole software is the lowest, when the method $doubleTest31()$ belongs to class $DoubleMatrix1D$. As a result, the method $doubleTest31()$ should not be defined in class $TestMatrix2D$, and the method $doubleTest31()$ should be defined in class $DoubleMatrix1D$. It can prove that $doubleTest31()$ is a feature envy bad smell.

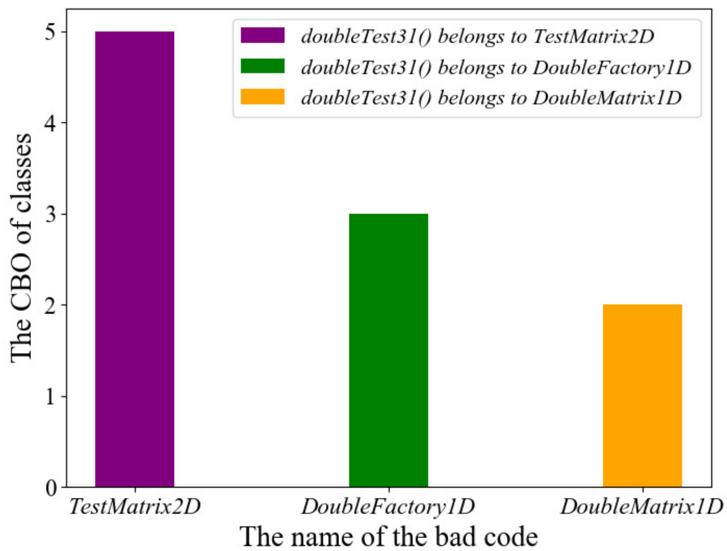


Figure 7. The CBO of classes under $doubleTest31()$ belongs to different classes.

As the feature envy bad smells, methods $like2D()$ and $doubleTest31()$ are only detected by our bad smells identification algorithm $BSICD$, in comparison with JDeodorant, $BSICD$ is more accurate in identifying feature envy bad smells.

6. **Conclusion and future work.** This paper proposes a feature envy metric based on feature membership parameters and the metric can be used to control community detection by a series of feature-moving operations. A bad smells identification algorithm using community detection is designed to identify the feature envy bad smells. In addition, for two types of feature envy bad smells we can both identify. The first type is a feature envies methods or attributes that belong to one class, the second type is that a feature envies features that belong to different classes. Compare with JDeodorant, our approach can provide a more accurate identification. Compare with $FEED$, our approach can identify the second type feature envy bad smells.

In the future, we would like to improve the technology of abstracting object oriented software systems into the software networks of levels of feature and class. We plan to consider identify bad smells of over coupling in class level.

Acknowledgment. This work was supported in part by the Research Project of the Education Department of Jilin Province under Grant JJKH20190706KJ and in part by Science and technology innovation development program of Jilin under Grant 20190104140.

REFERENCES

[1] R. Oliveto, M. Gethers, G. Bavota, D. Poshyvanyk and A. De Lucia, "Identifying method friendships to remove the feature envy bad smell: NIRE track," *2011 33rd International Conference on Software Engineering(ICSE)*, pp. 820–823, 2011.

[2] H. Xiao, M. H. Cao and R. Peng, "Artificial neural network based software fault detection and correction prediction models considering testing effort," *Applied Soft Computing*, vol. 94, 106491, 2020.

[3] B. L. Sousa, P. P. Souza, E. M. Fernandes, K. A.M. Ferreira and M. A.S. Bigonha, "FindSmells: flexible composition of bad smell detection strategies," *2017 IEEE/ACM 25th International Conference on Program Comprehension(ICPC)*, pp. 360–363, 2017.

[4] M. Fowler, "Refactoring: improving the design of existing code," *Xp Universe and First Agile Universe Conference on Extreme Programming and Agile Methods-xp/agile Universe.Springer-Verlag*, 2013.

[5] W. F. Pan, B. Jiang and Y. Xu, "Refactoring packages of object-oriented software using genetic algorithm based community detection technique," *International journal of computer applications in technology*, vol. 48, no. 3, pp. 185–194, 2013.

[6] W. F. Pan, B. Li, Y. T. Ma, J. Liu and Y. Y. Qin, "Class structure refactoring of object oriented softwares using community detection in dependency networks," *Frontiers of Computer Science*, vol. 3, no. 3, pp. 396–404, 2009.

[7] G. Saranya, H. K. Nehemiah, A. Kannan and V. Nithya, "Model level code smell detection using egapso based on similarity measures," *Alexandria engineering journal*, vol. 57, no. 3, pp. 1631–1642, 2018.

[8] N. Tsantalis, A. Chatzigeorgiou, "Identifification of move method refactoring opportunities," *IEEE Transactions on Software Engineering*, vol. 35, no. 3, pp. 347–367, 2009.

[9] Á. Kiss, P. F. Mihancea, "Towards feature envy design flaw detection at block level," *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 2576–3148, 2018.

[10] W. K. Chen, C. H. Liu and B. H. Li, "A feature envy detection method based on dataflow analysis," *2018 IEEE 42nd Annual Computer Software and Applications conference*, pp. 93–102, 2018.

[11] H. Liu, Z. Xu and Y. Zhou, "Deep learning based feature envy detection," *2018 33rd IEEE/ACM International Conference on Automated Software Engineering*, pp. 385–396, 2018.

[12] P. He, P. Wang, B. Li and S. W. Hu, "An evolution analysis of software system based on multi-granularity software network," *ACTA ELECTONICA SINICA*, vol. 46, no. 2, pp. 257–267, 2015.

[13] M. E. Newman, M. Girvan, "Finding and evaluating community structure in networks," *Physical review E*, vol. 69, no. 2, pp. 12–220, 2004.

[14] C. M. Chen, L. Chen, W. Gan, L. Qiu and W. Ding, "Discovering high utility-occupancy patterns from uncertain data," *Information Sciences*, vol. 546, pp. 1208–1229, 2021.

[15] M. S. Zanetti, F. Schweitzer, "A network perspective on software modularity," *Computer Science Software Engineering*, pp. 175–186, 2013.

[16] M. Fokaefs, N. Tsantalis and A. Chatzigeorgiou, "Jdeodorant: Identification and removal of feature envy bad smells," *IEEE International Conference on Software Maintenance*, vol. 42, pp. 14–19, 2018.
[17] J. Liu, B. Liu and D. Li, "Discovering protein complexes from protein-protein interaction data by local cluster detecting algorithm," *Fourth International Conference on Fuzzy Systems and Knowledge Discovery*, pp. 280–284, 2007.
[18] J. Liu, K. Q. He, R. Peng and Y. T. Ma, "A study on the weight and topology corrlation of object oriented software coupling network," *International Conference on Complex Systems and Applications*, vol. 13, pp. 955–959, 2006.
[19] V. D. Blondel, J. L. Guillaume, R. Lambiotte and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 10, pp. 1–7, 2008.
[20] A. Henderson-Sellers, A. J. Pitman, B. Henderson-Sellers, D. Pollard and J. M. Verner, "Applying Software Engineering Metrics to Land Surface Parameterization Schemes," *Journal of Climate*, vol. 8, no. 5, pp. 1043–1059, 2009.
[21] D. V. Heesch, "Doxygen: Source code documentation generator tool," http://www.doxygen.org, 2008.