Correlation Feature Mining Model Based on Dual Attention for Feature Envy Detection

1st Shuxin Zhao
Beijing Institute of Technology
School of Computer Science
Beijing, China
zhaosx@bit.edu.cn

2nd Chongyang Shi*
Beijing Institute of Technology
School of Computer Science
Beijing, China
cy_shi@bit.edu.cn

3rd Shaojun Ren
Beijing Institute of Technology
School of Computer Science
Beijing, China
3120191036@bit.edu.cn

4th Hufsa Mohsin
Beijing Institute of Technology
School of Computer Science
Beijing, China
hufsa.bit@yahoo.com

***Abstract*—Feature Envy is a code smell due to the abnormal calling relationships between methods and classes, which adversely affects software scalability and maintainability. Existing methods mainly use various technologies to model abnormal relationships to detect feature envy. However, these methods only rely on local features such as entity names, which is not robust enough. Moreover, the mining depth of correlation features between entities involved in feature envy is limited. In this paper, we propose a correlation feature mining model based on dual attention to detect feature envy. Firstly, we propose a multi-view-based entity representation strategy, which enhanced the robustness of the model while improving the suitability of the correlation feature and model. Secondly, we add attention mechanism to the channel dimension and spatial dimension of CNN to control the flow of information and capture the correlation features between entities more accurately. Finally, the evaluation results on projects both with and without feature envy injected show that our proposed approach outperforms the state-of-the-art methods.**

***Index Terms*—Code Smell, Feature Envy, Software Refactoring, Attention Mechanism, Deep Learning**

I. INTRODUCTION

A code smell is a potential problem in code caused by non-standard programming [1], [2]. Feature envy is a common code smell that has a significant impact on the degree of coupling and cohesion of software [3]–[5]. An accurate and widely accepted definition, first proposed by Beck and Fowler [6], is *more interested in a class other than the one it actually is in*. Based on this definition, many methods have been proposed to complete the critical step of the refactoring operation, that is, the detection of feature envy [7], [8].

The core of the existing feature envy detection methods is to model the abnormal calling relationship between methods and classes, which can be divided into the traditional method based on structural information (code metrics) [8] and the deep learning method based on text information [9]–[11]. The code metrics represent the element overlap degree between the method and the class, but this method relies heavily on artificial design features and selection threshold. In the case that the coding specification is met, there will be a certain correlation between the name of the method and the name of the class. Therefore, many deep learning methods are proposed to automate the end-to-end complex feature mapping [11].

Although deep learning methods based on text information have achieved good performance in feature envy detection, there are two key problems with such methods: 1) the selected local feature of entity name cannot fully represent the entity [11], [12], and the robustness of the model is also affected by the singleness of the feature, for example, when the method name conforms to the specification but the called variable is completely contained in another class. 2) Existing methods usually use CNN or RNN to extract correlation features [11], [13], but they cannot accurately capture effective information and filter other information, resulting in limited expression the ability of the model.

To solve the above problems, we first propose a multi-view based entity representation strategy, which selected name, context and content to represent the entity [14], [15]. The above entity representation strategy was mainly based on the following three observations:

- **Name** is usually an accurate summary of the entity's function, and the method name in the right place should have some correlation with the class name.
- **Context** refers to the external inputs of a method and its outputs to the external. For a method, the inputs are its parameters and the outputs are its return values. In a class with good cohesion, the methods contained in it must be similar in function or goal, which means that the context of multiple methods is similar.
- **Content** of a method includes the external properties and methods called by the method, and two methods that are similar in function are also similar in content. And for method names, which are intuitive generalizations of method functions, we can find deeper correlations between method functions due to the fine-grained nature of the content.

To sum up, we should select information from the three views of method, contain class and target class to get a comprehensive representation. In addition, in order to more accurately capture the correlation features in the selected text

DOI reference number:10.18293/SEKE2022-009

This work is supported by the National Key Research and Development Program of China(No. 2018YFB1003903), National Natural Science Foundation of China (No. 61502033) and the Fundamental Research Funds for the Central Universities.

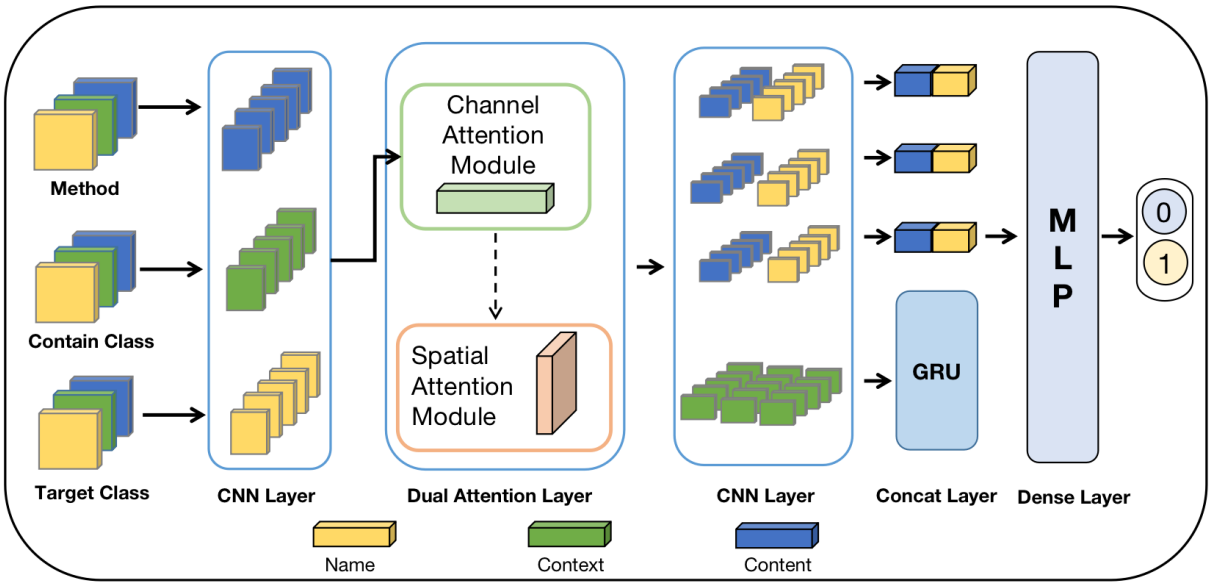

Fig. 1. Data generation process.

information, we propose a correlation feature mining model based on dual attention. The framework is illustrated in Fig.1. Inspired by CBAM [16], this model adds attention mechanism to channel dimension and spatial dimension of feature map obtained by CNN. It can assign more weight to the important feature and the level of the important feature, so the local correlation feature can be accurately captured and filtered. Finally, we use GRU to combine the context among the three entities to obtain the overall correlation from the global view.

The evaluation of the proposed method consists of two parts. 1) On an existing large-scale data-set, which was obtained by manually injecting feature envy through the operation of move methods on seven high-quality open-source Java projects [11]. On this data-set, our method can reach F-measure 55%, which is higher than the state-of-the-art. 2) We tested the proposed method on three open-source Java projects without feature envy injection, and the results showed that the performance of our method is still higher than the existing tools and technologies. The paper makes the following contributions:

- We propose a multi-view based entity representation strategy, which extracts text information from name, context and content to comprehensively represent the entity, so that the entity integrates more correlation features and improves the robustness of the model.
- We propose a dual attention based correlation feature mining model for feature envy detection, based on the comprehensive representation of entities. Dual-channel attention mechanism can expand the depth of text information and accurately filter and capture correlation features.
- The evaluation results on open source projects with and without feature envy injection show that our approach achieves better performance than state-of-the-art approaches.

The remainder of this paper is organized as follows. Section II introduces the work related to feature envy detection. Section III explains the proposed approach, after which Section IV presents the results of the proposed approach. Finally, the conclusions are drawn in Section V.

II. RELATED WORK

Feature envy is a common code smell characterized by being *more interested in a class other than the one it actually is in* [9]. Many methods have been proposed to detect this code smell, including the traditional method based on structural information (code metrics) and the deep learning method based on text information.

Existing feature envy detection methods rely primarily on a metrics that can measure the relationship between the method entity and the class entity [17], [18], which was first proposed by Simon et al. [19] in 2001 . They propose a metric based on the set operation to indicate the distance between entities, as follows:

$$distance(e_1, e_2) = 1 - \frac{|p(e_1) \cap p(e_2)|}{|p(e_1) \cup p(e_2)|} \tag{1}$$

The change rules of $p(e)$ with the entity types of e are as follows:

$$p(e) = \begin{cases} \{e, entities_{Called}\}, & if\ e\ is\ method \\ \{e, entities_{Access}\}, & if\ e\ is\ attribute \end{cases} \tag{2}$$

Entities in code are divided into method entities and attribute entities, where e represents an entity. Here, $P(e)$ represents the set of properties possessed by e. If e is a method, the set contains the entity itself, along with the attribute and method entities that are called by e. If e is an attribute, the set contains the entity itself and all methods that directly access e. Based on the filtered entity set and calculation formula (1)(2),

the distance between entities can be obtained. If the distance between a method entity and the entities in its class is greater than the distance between the entity and the entities in other classes, this method is associated with feature envy.

Tsantalis et al. [20] propose a new metric to define the distance between entities, that differs from that proposed by Simon et al. Although they divide entities into methods and attributes, the final result is the distance between method entities and classes; by contrast, Simon et al. focuses on the distance between method entities and attribute entities [19]. If the method entity m being detected belongs to the class entity C, the formula for calculating the distance is as follows :

$$distance(m, C) = 1 - \frac{S_m \cap S_C}{S_m \cup S_C}, where\ S_C = \bigcup_{e_i \epsilon C} \{e_i\} \quad (3)$$

Otherwise, the distance is computed as follows:

$$distance(m, C) = 1 - \frac{S_m \cap S_C^{'}}{S_m \cup S_C^{'}}, where\ S_C^{'} = S_C \backslash \{m\} \quad (4)$$

In formula (3)(4), m represents a method entity, S_m represents the collection of entities called by the method entity, and S_C represents the method and attribute entities contained in the class; moreover, the measured method entities should be excluded from the collection of their classes. If the final result shows that the distance between a method and the containing class exceeds the distance between the method and the target class, then the method is deemed to be associated with feature envy. This method is implemented by $JDeodorant$, a well-known code smell detection tool, has become the most commonly used benchmark in the code smell detection research field [11].

To make better use of metric information and text information, Liu et al. propose feature envy detection based on deep learning [11], [13]. This method can automatically extract the text information and metrics required by the training classifier from the open-source applications; here the metrics information is the distance metrics proposed by Tsantalis et al. [20], [21], [22]. The text information mainly includes the identifier of the method and the corresponding identifiers of the containing class and the target class [23], [24]. The classifier primarily uses the CNN neural network structure to extract the internal features of the input information. Finally, the extracted features are spliced into the linear layer to predict whether the method and target class is "smelly" or "non-smelly". They are the first to apply deep learning techniques to feature envy detection, and the detection effect is much higher than other methods.

III. METHODOLOGY

A. Data Generation

As we employ deep learning technology to build a mapping between input information containing comprehensive features and feature envy judgment, we need large-scale data to train the model. However, due to the characteristics of code smell, it is difficult to compile relevant training data on a large scale, making it necessary to artificially inject feature envy into the code to generate such large-scale data. We here utilize the method of automatic large-scale data generation proposed by Liu et al. [11].

Finally, we can obtain any number of positive and negative items, as shown in formula (5).

$$Item = < Input, Output > \quad (5)$$

$$Input = < input_m, input_C, input_T > \quad (6)$$

$$Output = < 0/1 > \quad (7)$$

Respectively, the $input_m$, $input_C$, and $input_T$ elements of $input$ represent the information extracted from the movable method m, the containing class C, and the target class T. This information comprises three main parts: name, context, and content, as shown in the following formula (8)(9)(10). The $Output$ information 0 / 1 respectively represents whether this item is negative or positive, as shown in formula (7).

$$input_m = < name(m), context(m), content(m) > \quad (8)$$

$$input_C = < name(C), context(C), content(C) > \quad (9)$$

$$input_T = < name(T), context(T), content(T) > \quad (10)$$

B. Information Processing

Based on the sample generation method, we can get text information that contains correlation features. However, in order to meet the input specifications of the neural network and make better use of data, we still need to process the data.

1) Text Processing: The information we obtain is composed of many identifiers, each of which is generally a combination of one or more words. Thus, to change the input into a form acceptable to the neural network, we need to do the following [25]:

- Divide the identifier into a sequence of words according to the camel case method based on lower-line change, uppercase letters, and numbers.
- Change all words to lowercase.
- Remove programming keywords, special characters and English stop words.

2) Information Combination: Among the three views emphasized by representation strategy, name and content indicate performance of different levels of function, for its part, the context focuses on the input and output of a method, which are the most intuitive representation of a method's interaction with the external environment. Therefore, we combine the name and content modules to represent the internal functional features of a method or class, while the context modules of the method and class are combined to explore the features of the interaction among the method, containing class and target class. By using this combination method, we get a comprehensive representation of entities, and obtain the combination information which is beneficial to correlation mining. We verify the effectiveness of this combination method for feature envy detection in Section IV.

C. Correlation Feature Mining

After the above processing, we have nine unordered sets of words from name, context, and content of three entities. Since the structure is the same, we chose the method name as an example to illustrate our model flow.

First, we obtain a dense embedding matrix $X \in R^{N \times d}$ by embedding the sequence of words forming method name through the embedding space, where N is the number of words and d is the dimension. Most of the text information extracted was short text, so we chose CNN (convolutional neural network) with good local feature extraction ability to process X.

$$F = Conv(Emb(X)) \tag{11}$$

As shown in formula (11), the size of three convolution kernels are 2 $\times d$, 3$\times d$ and 4$\times d$ respectively, F is the set of feature maps obtained after convolution.

In order to accurately capture features, inspired by CBAM, we add attention mechanisms in channel and spatial dimensions respectively, where channel attention focuses on the difference in importance of features and spatial attention focuses on the difference in location of features.

$$F' = F \times \sigma(W_1(W_0(F_{avg})) + W_1(W_0(F_{max})) \tag{12}$$

$$F'' = F' \times \sigma(f(F'_{avg}; F'_{max})) \tag{13}$$

The operation is shown in formula (12)(13), and the feature map set F'' containing weights is obtained. We then perform another convolution operation on F'' to capture the deep features.

As mentioned in the above chapter, context interaction is the most direct expression of the correlation of the three entities. Therefore, context information S_m, S_c and S_t of the three entities are input into GRU as state flow. The update process of GRU is as formula (14). The hidden state h_i in formula X contains the correlation features of three contexts.

$$z_i = \sigma(W_z x_i + U_z h_{i-1}) \tag{14}$$

$$r_i = \sigma(W_r x_i + U_r h_{i-1}) \tag{15}$$

$$\tilde{h_i} = tanh(W x_i + U(r_i \odot h_{i-1})) \tag{16}$$

$$h_i = (1 - z_i) h_{i-1} + z_i \tilde{h_i} \tag{17}$$

Finally, to extract the global correlation features, we concatenate the name and content of each entity, that is, F'' obtained in formula (13), the concatenate results and h_i from formula (14) are entered into the fully connected neural network, whose output represents the final classification result: smelly or no-smelly. In addition, we select $binary\_crossentropy$ as loss function, which is defined as follows:

$$L = \sum_{i=1}^{N} y^{(i)} \log \hat{y^{(i)}} + (1 - y^{(i)}) \log(1 - \hat{y^{(i)}}) \tag{18}$$

where $\hat{y^{(i)}}$ is the true type of the method, and $y_{(i)}$ is the prediction of our proposed model.

IV. EXPERIMENTS

A. Research Questions

We evaluate our proposed approach by answering the following research questions.

- **RQ1**: Does our proposed approach outperform the state-of-the-art approaches in detecting feature envy?
- **RQ2**: How does our proposed method perform on real projects without feature envy injected?
- **RQ3**: Are the proposed representation strategy and dual attention mechanism helpful for FE detection?

Both RQ1 and RQ2 focus on the performance difference between our proposed method and other technologies in feature envy detection, so we choose the deep learning-based method proposed by Liu et al. [11] and the two popular code smell detection tools $JDeodorant$ and $JMove$ as comparison methods. RQ3 is mainly to verify the validity of the model. We verify the combination of name, context and content. At the same time, we also conduct an ablation experiment on the dual-attention mechanism.

B. Dataset and Experimental Design

The data used for experimental evaluation can be divided into two parts; (i) Large-scale data with feature envy automatically injected for training and verification of the classifier. (ii) Small-scale data without feature envy injected that is used to evaluate the effectiveness of the proposed approach on real projects.

To avoid over-fitting and reduce the impact of insufficient data size on the detection results, we choose to use the k-fold cross-validation method. Moreover, we selected three commonly used indicators, F_1, $Recall$, and $Precision$, to evaluate the effect of each method (19) (20) (21).

$$precision = \frac{true\ positives}{true\ positives + false\ positives} \tag{19}$$

$$recall = \frac{true\ positives}{true\ positives + false\ negatives} \tag{20}$$

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall} \tag{21}$$

C. RQ1: Detection on Injected Projects

To effectively verify that our proposed approach performs better than the best comparison approach, we select the highest-performance deep learning-based method and two popular code smell detection tools for comparison. The evaluation results are presented in Table I. From the above data, it can be determined that our method outperforms the best existing technology. Specifically, the method we proposed is achieves significantly better results than the traditional tool $JDeodorant$ and $JMove$ on the three indicators. Moreover, compared to the method proposed by Liu et al., which also uses deep learning technology, our method has higher precision but slightly lower recall, and finally, our method performs better in terms of comprehensive indicators.

The improvement of precision indicates that the prediction results of our method are more reliable, which in turn suggests

TABLE I
EVALUATION RESULT ON FEATURE ENVY DETECTION

Applications	Proposed Approach			Approach of Liu			JDeodorant			JMove		
	precision	recall	F1	precision	recall	F1	precision	recall	F1	precision	recall	F1
JUnit	51.90%	100.00%	68.33%	40.59%	91.11%	56.16%	30.76%	14.82%	20%	22.72%	18.52%	20.41%
PMD	67.44%	78.38%	72.50%	41.27%	68.42%	51.49%	15.79%	5.36%	8%	30%	26.79%	28.3%
JExcelAPI	25.52%	68.52%	37.19%	31.9%	92.85%	47.49%	60%	10.7%	18.18%	27.27%	16.07%	20.22%
Areca	38.42%	75.26%	50.87%	46.05%	72.16%	56.23%	32.14%	9.28%	14.4%	26.76%	39.18%	31.8%
Freeplane	63.16%	75.29%	68.69%	38.09%	68.58%	48.97%	21.62%	8.94%	12.65%	24.83%	13.79%	17.73%
jEdit	34.66%	72.73%	46.94%	42.63%	78.57%	55.28%	22.73%	4.55%	7.58%	17.43%	13.57%	15.26%
Weka	35.19%	66.25%	45.97%	40.05%	86%	54.65%	58.33%	17.5%	26.92%	11.22%	11.75%	11.48%
Average	**45.18%**	**76.63%**	**55.79%**	39.79%	79.27%	52.98%	39.51%	12.22%	18.66%	18.37%	16.3%	17.27%

The data of approach of Liu, JDeodorant and Jmove are cited from Deep Learning Based Feature Envy Detection [11].

that the complementary relationship between the three views of representation strategy that we propose has indeed corrected the erroneous preference in prediction. Since a certain contradiction exists between the two indicators of precision and recall, the inconsistency of the two indicators is also within the acceptable range: specifically, precision and recall for our method is 31.45% (76.63%-45.18%), which is much smaller than the 39.48% (79.27%-39.79%) of the method proposed by Liu et al. This further proves that our extraction of feature envy features is highly comprehensive, allowing us to obtain a more reliable and stable detector.

D. RQ2: Detection on Projects without Injection

As shown in Table I, our proposed method outperforms the best technology on java projects that automatically inject feature envy. However, the automatically injected feature envy characteristics must be based on the assumption that there is no misplaced method in this project, which is difficult to fully guarantee. Moreover, the feature envy created by the moving method may have certain key differences when compared to real feature envy, which leads to deviations in the extracted features and affects the effect of the detector. We accordingly choose to verify our method on real projects without injected feature envy to its effectiveness in real-world scenarios. Four graduate students with rich Java development experience will review the test results and provide their opinions on whether the findings can be accepted as true feature envy, and take more than half of the opinions will be taken as the final result. Finally, we conducted feature envy testing on three real projects according to the process. The results are shown in Table II.

From the table, we can construct our method is still superior to other methods on real projects without feature envy injection. Compared with Liu's method, the detection accuracy of our method is 16.97% (58.20%-41.23%) higher. Moreover, our method also achieves performance improvements of 30.61% (58.20%-27.59%) and 42.85% (58.20%-15.35%) relative to $JDeodorant$ and $JMove$ respectively.

E. RQ3: The Effectiveness of Model

To verify the effectiveness of each of the three perspectives, we set up four sets of experiments. (i) The information contains three modules. (ii) Name module deleted; (iii) Context module deleted; (iv) Content module deleted. The results of the experiments are shown in Fig. 2. From the figure, it can

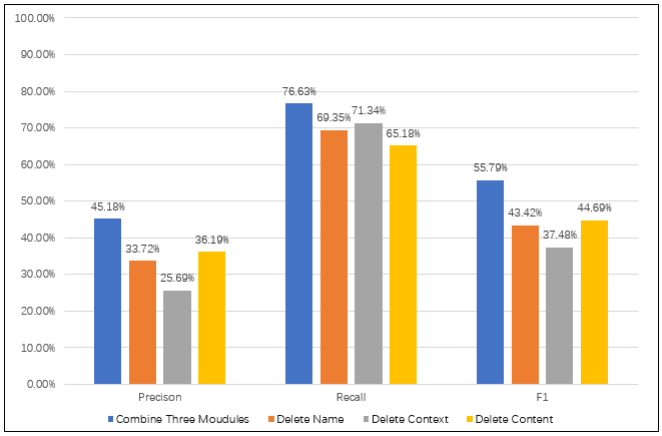

Fig. 2. Results of three perspectives.

be observed that deleting any module will lead to a decline in the detection result, which proves that all three perspectives complement each other and lead to more comprehensive feature representation. After deleting the context module, f1 dropped by 18.31% (55.79%-37.48%), which is the largest drop; this indicates that the context module is most important for feature envy detection.

Inside the neural network, we added a dual attention mechanism on CNN. To prove the effectiveness of this operation, we deleted the operation and got the results as shown in the following Table III:

As can be seen from the table, after deleting the dual attention mechanism, the three indicators of precision, recall, and F_1 were reduced by 4.01% (45.18%-41.17%), 3.67% (76.63%-72.96%), and 3.55% (55.79-52.24%). Therefore, the

TABLE II
EVALUATION RESULT ON PROJECTS WITHOUT INJECTING FEATURE ENVY

Metrics	Proposed Approach				Approach of Liu				JDeodorant				JMove			
	XMD	**JSmooth**	**Neuroph**	**Total**	**XMD**	**JSmooth**	**Neuroph**	**Total**	**XMD**	**JSmooth**	**Neuroph**	**Total**	**XMD**	**JSmooth**	**Neurpoh**	**Total**
Reported	40	22	72	134	32	26	56	114	8	3	18	29	106	27	82	215
Accepted	28	10	40	78	15	11	21	47	3	1	4	8	12	5	16	33
Precision	**70.00%**	**45.45%**	**55.56%**	**58.20%**	46.88%	42.31%	37.5%	41.23%	37.5%	33.33%	22.22%	27.59%	11.32%	18.52%	19.51%	15.35%

The data of approach of Liu, JDeodorant and Jmove are cited from Deep Learning Based Feature Envy Detection [11].

TABLE III
DETECTION RESULT WITHOUT JOINT FEATURES EXTRACTION

Applications	Precision	Recall	F1
JUnit	55.93%	80.49%	66.00%
PMD	52.54%	83.78%	64.58%
JExcelAPI	22.00%	61.11%	32.35%
Areca	41.72%	70.10%	52.31%
Freeplane	51.90%	85.88%	64.70%
jEdit	31.01%	60.61%	41.03%
Weka	33.09%	68.75%	44.68%
Average	**41.17%**	**72.96%**	**52.24%**

dual attention mechanism does enhance the expressive ability of the model.

V. CONCLUSION

In this paper, a correlation feature mining model based on dual attention to detect feature envy is proposed. At first, a new representation strategy is proposed, which extracts the text information from the three views of name, content and context to comprehensively express entities. Secondly, a dual attention mechanism is used to accurately capture local and global correlation features to detect Feature envy. Lastly, We respectively evaluated the method on seven open-source Java projects that automatically injected feature envy and three open-source Java projects that did not inject feature envy. The results show that the feature envy detection effect of the proposed method is indeed superior to the state-of-the-art. Recommending suitable target class for methods with feature envy will be considered as a potential future work.

REFERENCES

[1] A. April and A. Abran, *Software maintenance management: evaluation and continuous improvement*, vol. 67. John Wiley & Sons, 2012.

[2] W. J. Brown, R. C. Malveau, H. W. McCormick III, and T. J. Mowbray, "Refactoring software, architectures, and projects in crisis," 1998.

[3] A. K. Das, S. Yadav, and S. Dhal, "Detecting code smells using deep learning," in *TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON)*, pp. 2081–2086, Oct 2019.

[4] T. Sharma and D. Spinellis, "A survey on software smells," *Journal of Systems and Software*, vol. 138, pp. 158–173, 2018.

[5] P. Kruchten, R. L. Nord, and I. Ozkaya, "Technical debt: From metaphor to theory and practice," *Ieee software*, vol. 29, no. 6, pp. 18–21, 2012.

[6] M. Fowler, *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Signature Series (Fowler), Pearson Education, 2018.

[7] N. Moha, Y.-G. Guéhéneuc, L. Duchien, and A.-F. Le Meur, "Decor: A method for the specification and detection of code and design smells," *IEEE Transactions on Software Engineering*, vol. 36, no. 1, pp. 20–36, 2009.

[8] R. Marinescu, "Detection strategies: Metrics-based rules for detecting design flaws," in *20th IEEE International Conference on Software Maintenance, 2004. Proceedings.*, pp. 350–359, IEEE, 2004.

[9] M. I. Azeem, F. Palomba, L. Shi, and Q. Wang, "Machine learning techniques for code smell detection: A systematic literature review and meta-analysis," *Information and Software Technology*, vol. 108, pp. 115–138, 2019.

[10] F. Palomba, A. Panichella, A. De Lucia, R. Oliveto, and A. Zaidman, "A textual-based technique for smell detection," in *2016 IEEE 24th international conference on program comprehension (ICPC)*, pp. 1–10, IEEE, 2016.

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

[12] X. Guo, C. Shi, and H. Jiang, "Deep semantic-based feature envy identification," in *Proceedings of the 11th Asia-Pacific Symposium on Internetware*, pp. 1–6, 2019.

[13] H. Liu, J. Jin, Z. Xu, Y. Bu, Y. Zou, and L. Zhang, "Deep learning based code smell detection," *IEEE transactions on Software Engineering*, 2019.

[14] N. Tsantalis and A. Chatzigeorgiou, "Identification of extract method refactoring opportunities for the decomposition of methods," *Journal of Systems and Software*, vol. 84, no. 10, pp. 1757–1782, 2011.

[15] J. Chang and D. M. Blei, "Hierarchical relational models for document networks," *The Annals of Applied Statistics*, pp. 124–150, 2010.

[16] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon, "Cbam: Convolutional block attention module," in *Proceedings of the European conference on computer vision (ECCV)*, pp. 3–19, 2018.

[17] N. Anquetil and T. C. Lethbridge, "Experiments with clustering as a software remodularization method," in *Sixth Working Conference on Reverse Engineering (Cat. No. PR00303)*, pp. 235–255, IEEE, 1999.

[18] T. Mens, N. Van Eetvelde, S. Demeyer, and D. Janssens, "Formalizing refactorings with graph transformations," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 17, no. 4, pp. 247–276, 2005.

[19] F. Simon, F. Steinbruckner, and C. Lewerentz, "Metrics based refactoring," in *Proceedings fifth european conference on software maintenance and reengineering*, pp. 30–38, IEEE, 2001.

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

[21] S. Wang, L. L. Minku, and X. Yao, "Resampling-based ensemble methods for online class imbalance learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 5, pp. 1356–1368, 2014.

[22] O. Sagi and L. Rokach, "Ensemble learning: A survey," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, no. 4, p. e1249, 2018.

[23] H. Liu, M. Shen, J. Zhu, N. Niu, G. Li, and L. Zhang, "Deep learning based program generation from requirements text: Are we there yet?," *IEEE Transactions on Software Engineering*, pp. 1–1, 2020.

[24] Y. Jiang, H. Liu, and L. Zhang, "Semantic relation based expansion of abbreviations," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 131–141, 2019.

[25] M. F. Porter, "An algorithm for suffix stripping," *Program*, 1980.