ZKG INTERNATIONAL ISSN: 2366-1313

SOFTWARE CLONE REFACTORING ASSISTANCE USING MACHINE LEARNING

[1]Mrs. SHIVA SREE, [2]S. SHIVA TEJA, [3]K. SANDEEP SAGAR, [4]K. MADHU SUDHAN REDDY

(Assistant Professor) ,**CSE.** Teegala Krishna Reddy Engineering College Hyderabad

B,tech scholar ,**CSE.** Teegala Krishna Reddy Engineering College Hyderabad

ABSTRACT

Advising developers on refactoring code clones is designed to provide a thorough understanding of the process and its application. It starts with an introduction that underlines the importance of code clone refactoring and software quality improvement, providing context for the proposed approach. The background section provides an overview of code clones and existing approaches to code clone detection and refactoring, laying the groundwork for the suggested methodology. The methodology section digs into the intricacies of the proposed solution, outlining the process of automatically extracting features from discovered code clones and training machine learning models to categorize clones based on the type of refactoring required. It also covers the unusual approach of transforming clone type outliers into a "Unknown" clone set. This improves classification results. The section emphasizes the use of cutting-edge classification models for training, highlighting the technical sophistication of the technique. The documentation in the implementation section includes practical insights into how to implement the proposed approach, including the necessary software libraries and tools. It may include code snippets or illustrations to help developers comprehend important concepts and implement the method in their applications. The evaluation methodology section describes how the efficacy of the suggested strategy was assessed, including the metrics utilized for evaluation and comparisons to existing methods. This serves as a foundation for evaluating the performance and effectiveness of the

proposed technique in real-world circumstances. The case studies section provides real-world examples or case studies. The case studies section includes real-world case studies or examples that demonstrate the practical application and effectiveness of the suggested approach for guiding developers on reworking code clones. This helps to validate the methodology and its possible impact on software development methods. The future work section provides prospective areas for more research and enhancements to the suggested approach, setting the path for continued progress in the field of code clone refactoring and software quality enhancement. Finally, the conclusion outlines the important findings and contributions of the proposed approach, emphasizing its usefulness in improving software quality through effective code clone refactoring.

1. INTRODUCTION

In software development, code clones—pairs of code fragments with a high degree of similarity or identical content—occur frequently. Code cloning has drawbacks in addition to advantages like faster coding and code reuse. Clones can hinder code comprehension and make software maintenance more challenging, which emphasizes the necessity to solve clone-related concerns. Different methods have been created to find code clones in the source code of a system by finding code fragments that are identical to each other. Furthermore, clone structures can be changed without affecting their behaviour with the use of refactoring tools, which lowers the possibility of adding bugs. Dealing with code clones requires refactoring, an important technique for enhancing the quality and maintainability of code. However, clone refactoring's efficacy may be constrained by elements like clones' brief life spans and the difficulty of refactoring modified clones. The project's main goal is to help developers refactor code clones in order to improve software quality and reduce problems. In order to do this, the project suggests a novel method that includes automatically identifying attributes in identified code clones and using that information to train machine learning models that would advise developers on the kind of refactoring that should be done.

The project's method generates a unified model for recognizing different sorts of refactored clones and anonymous clones, in contrast to current approaches that consider refactored clone types as discrete classes. The project also presents a technique to improve classification accuracy by restructuring clone type outliers into a "Unknown" clone set. By discovering and modifying

duplicate code, the initiative hopes to reduce problems in software systems and enhance clone maintenance. The project's unique approach to clone refactoring and its ability to greatly enhance software quality and maintenance procedures are its key contributions.

1.1 PROBLEM STATEMENT "In software development, identifying and refactoring code clones is crucial for maintaining code quality and reducing redundancy. However, existing methods for clone detection and refactoring often struggle with accurately categorizing and refactoring clones, especially when dealing with unknown or abnormal clone types. This project aims to address these challenges by proposing a novel approach that combines outlier detection methods with supervised learning classifiers to enhance the detection and classification of code clones, particularly unknown clone sets. The goal is to improve the reliability and effectiveness of clone detection and refactoring in software projects."

1.2 PROBLEM STATEMENT Detecting and refactoring code clones is crucial for improving software quality and reducing redundancy. However, existing methods face challenges in accurately identifying and refactoring clones, especially when dealing with unknown or abnormal clone types. To address these issues, this project proposes a novel approach that integrates outlier detection techniques with supervised learning classifiers. By combining these methods, the project aims to enhance the detection and classification of code clones, particularly unknown clone sets. This innovative approach is designed to improve the reliability and effectiveness of clone detection and refactoring in software projects, ultimately leading to higher quality software products.

2. LITERATURE SURVEY

The literature survey on code clone detection and refactoring provides a thorough overview of the issues and techniques for handling code clones, which are identical or similar code fragments within a software system. One of the main conclusions of a Microsoft study is the difficulties in proactively preserving consistency and deleting unneeded clones, particularly in large-scale commercial software. This emphasizes the necessity of effective clone detection technologies and techniques in reducing the possible detrimental effects of code clones on program maintenance and comprehension. Another key technique, known as CREC, focuses on advising code clone refactoring using a combination of current and historical data. CREC creates a

training set by automatically extracting clone groups that have been historically refactored from those that have not. When compared to other techniques, this method shows promise in producing high F-scores for clones that need to be refactored. It emphasizes how crucial it is to use historical data to improve the efficacy of clone refactoring suggestions. Additionally, studies on clone maintenance and change through refactoring approaches highlight the contribution of refactoring to better software maintainability. Through the application of refactoring techniques like the "Extract Method" and "Pull Up Method," developers are able to eliminate code clones and enhance the software's complexity and understandability. This demonstrates how refactoring may be used in practice to address code clones and improve the quality of software.

3. SYSTEM DESIGN

3.1 SYSTEM ARCHITECTURE

Our code clone detection and refactoring guidance system's architecture is designed to streamline the process of identifying and improving code clones. It begins with the User Interface (UI), where users can upload code files for analysis. The Natural Language Toolkit (NLTK) is then used to process these files, extracting features and normalizing them to a range of 0 to 1 by averaging word counts. The next step involves passing the normalized feature vectors through the Local Outlier Factor (LOF) algorithm to identify significant attributes. The LOF algorithm assigns a score of 1 to attributes considered anomalies and -1 to outliers, helping to pinpoint important characteristics for further analysis. Once the essential features are extracted, they are used to train a machine learning model. This model is utilized by the Code Advisor module to predict which code files need to be refactored. Developers are then provided with recommendations on which sections of the codebase require attention and potential refactoring, aiding in improving the overall quality and maintainability of the codebase. This system architecture ensures accurate and efficient code clone detection and refactoring recommendations, ultimately enhancing the development process and software quality.

ISSN: 2366-1313

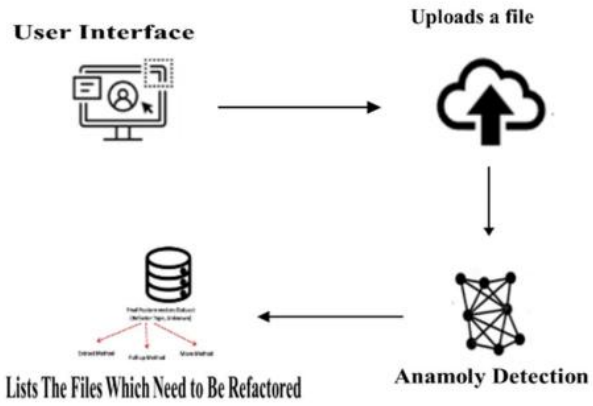

Fig 3.1 System Architecture

3.2 ACTIVITY DIAGRAM

An activity diagram in UML serves as a behavioral diagram that depicts the flow of control within a system or process. It provides a visual representation of the dynamic behavior of the system, showcasing the sequence of activities and actions carried out by actors or components. Below is a detailed explanation of the essential components and their functions in an activity diagram:

1. Activities: Activities shown as rounded rectangles, embody the tasks or actions executed within the system. They can vary from straightforward operations to intricate processes.

2. Transitions: Transitions in an activity diagram indicate the sequence of activities and how control flows from one activity to another. They are represented by arrows connecting activities and may include conditions or guards that determine when they are activated.

3. Decisions (Branches): Decisions in an activity diagram symbolize moments in the process where the flow of control can branch out depending on specific conditions. Shaped like diamonds, they feature multiple outgoing transitions, each corresponding to a potential outcome.

4. Initial and Final Nodes: The initial node signifies the beginning of the activity diagram and is denoted by a filled circle with an outward-pointing arrow. Conversely, the final node denotes the conclusion of the process and is depicted by a circle with a solid border.

5. Forks and Joins: Forks are used to indicate points in the process where the flow of control divides into multiple concurrent paths, enabling activities to be executed simultaneously. Conversely, joins indicate points where concurrent paths merge back into a single flow of control.

6. Swimlanes: Swimlanes, whether vertical or horizontal, are partitions used to group activities based on the entity or component responsible for them. They play a crucial role in organizing activities and elucidating the roles of various actors or system components.

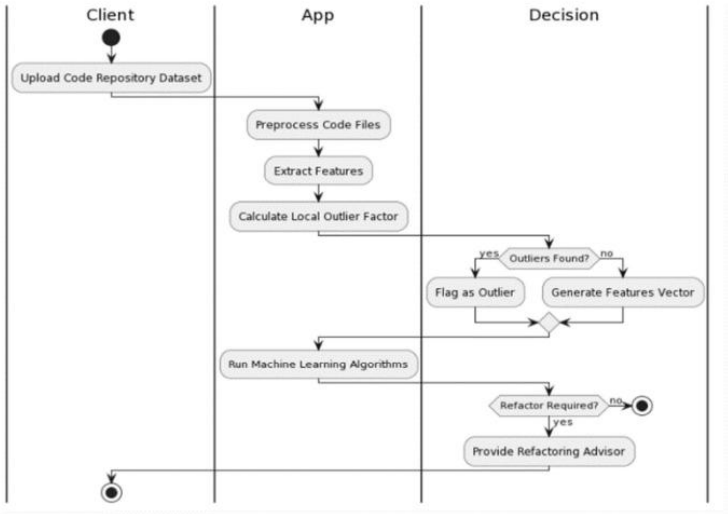

Fig 3.2 Represents Activity Diagram

The first step in preparing the code and data is to clean the code files and extract pertinent features from them. This stage makes sure the data is prepared in a way that makes it easy to analyze later. After that, any anomalies or odd patterns in the data are found using an outlier detection technique. Outliers are marked for additional analysis if they are found. The prepared data is then used to create a feature vector.

The primary characteristics that the machine learning model will employ for training and analysis are captured by this feature vector, which acts as a numerical representation of the data. The feature vector plays a vital role in converting the data into a format that the model can process efficiently.

The prepared data and feature vector are used to train the machine learning model in the following stage. Based on the input data, the model is trained to identify patterns and generate

predictions. The model is evaluated to determine its efficacy and performance once it has been trained. Should the model fail to satisfy the intended standards, an advisor may be made available to offer modifications and enhancements to the model.

4. OUTPUT SCREENS

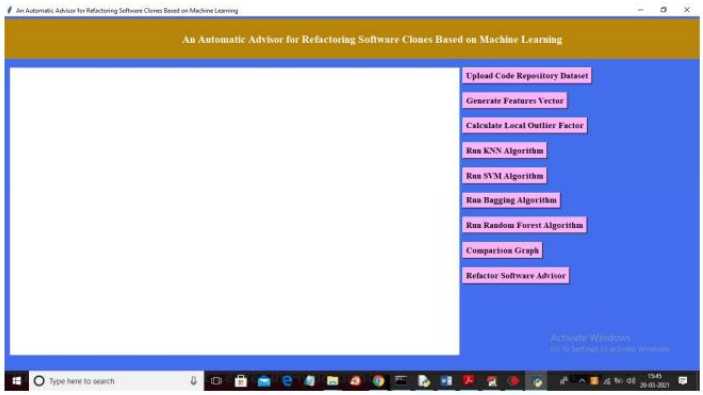

Fig 4.1 Represents Initial User Interface

The output screen represents the basic initial user interface shows all Input box and Buttons.

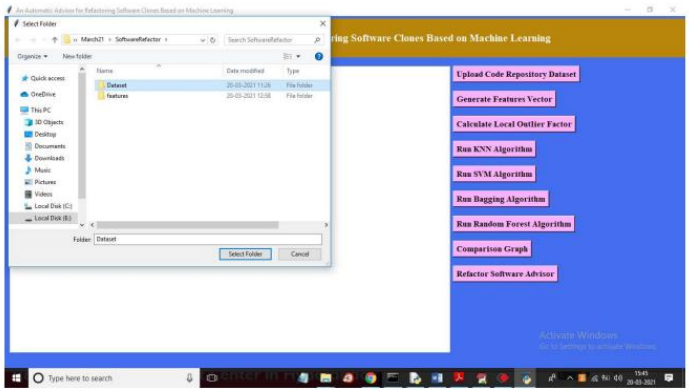

Fig 4.2 Represents Uploading Code Repository

The output screen shows how the user interface is accessed and input is taken.

ISSN: 2366-1313

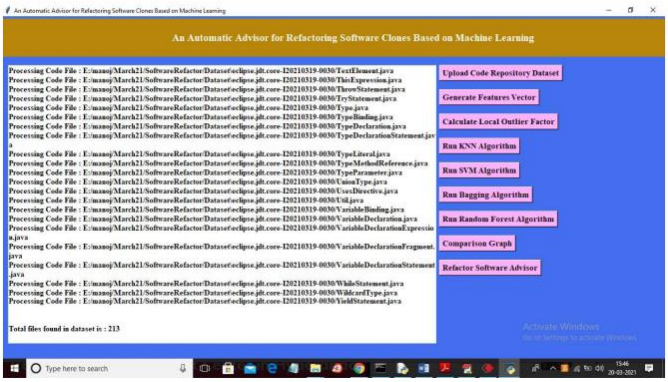

Fig 4.3 Represents List Of Files Present In Code Repository

In above screen application read each code file and then process it and total java files found in dataset is 213.

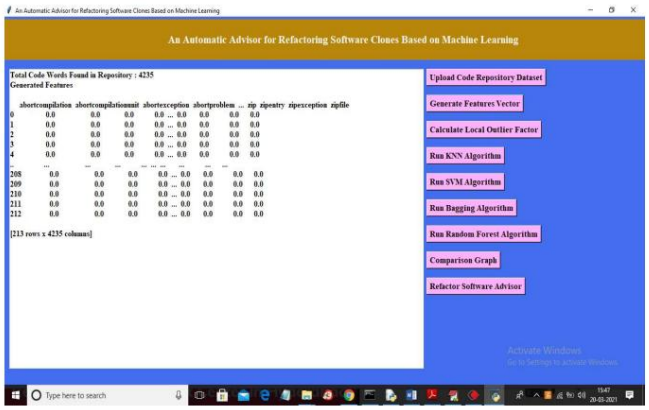

Fig 4.4 Represents Generating Feature Vector.

In above screen we can see all codes converted into vector where all words in codes will put as column header and the count of each word and its average values will put in rows and now vector is ready and now click on 'Calculate Local Outlier Factor' button to remove irrelevant columns/attributes.

ISSN: 2366-1313

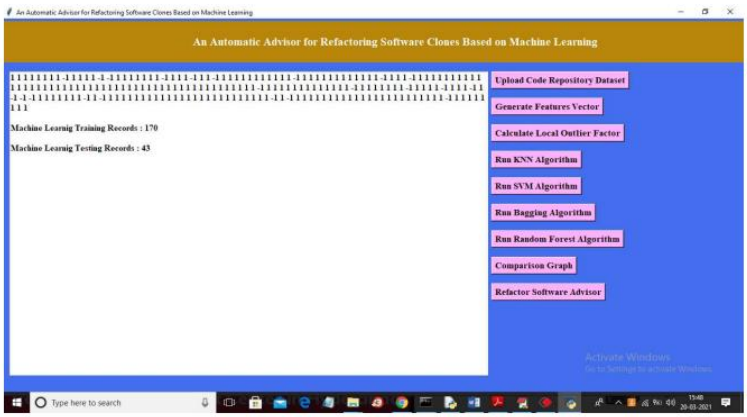

Fig 4.5 Represents Calculation Local Outlier Factor.

In above screen wherever we are seeing 1 that column in feature vector is important and where we are seeing -1 that column contains irrelevant attributes.

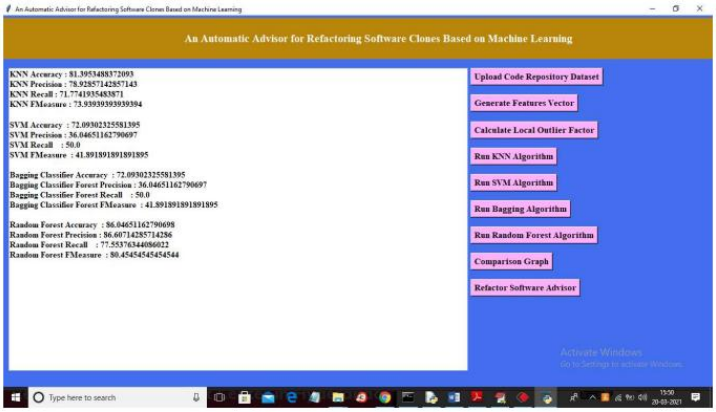

Fig 4.6 Represents Running Machine Learning Algorithms

The output screen shows running machine learning algorithms Accuracy, Precision, Recall and FMeasure.

ISSN: 2366-1313

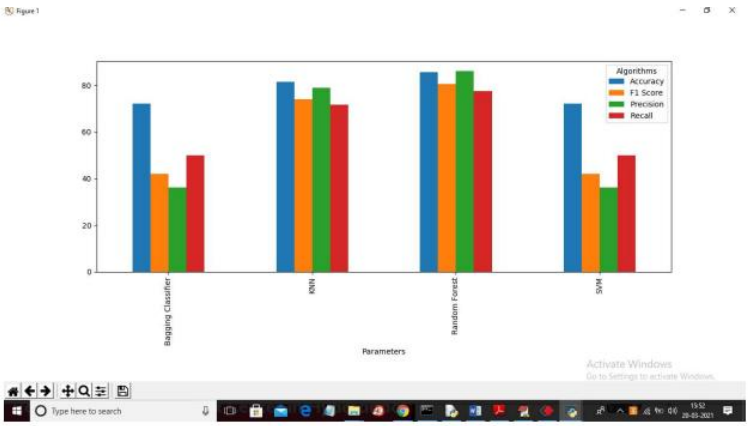

Fig 4.7 Represents Comparison Graph.

In above graph we can see performance of each algorithm and in all algorithm random forest giving better result and now machine learning models are ready and now click on 'Refactor Software Advisor' button to get all code names which require refactor.

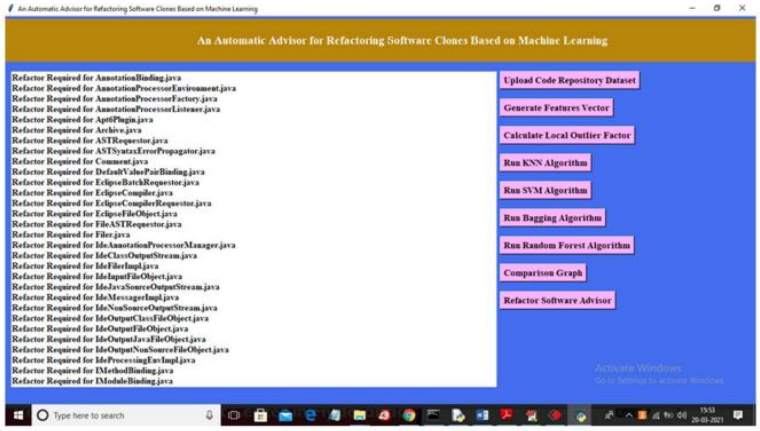

Fig 4.8 Represents giving list of class names which refactor

In above screen we got all class names which require refactor and now open first file called AnnotationBinding.java and see is there any duplicate code.

ISSN: 2366-1313

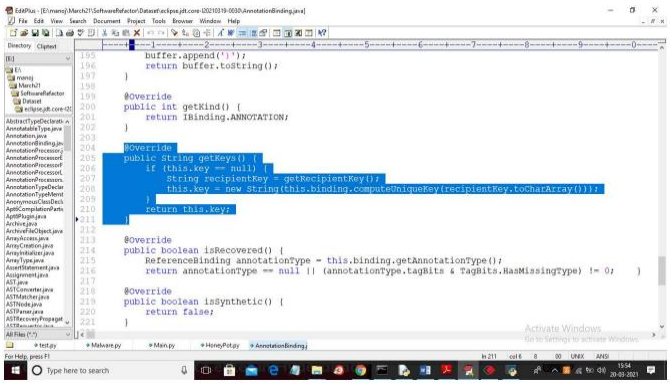

Fig 4.9 Represents duplicate code

In above screen we can see in same program two functions are there with same code and different name as getKeys() in selected text and in next screen we have another method as getKey()

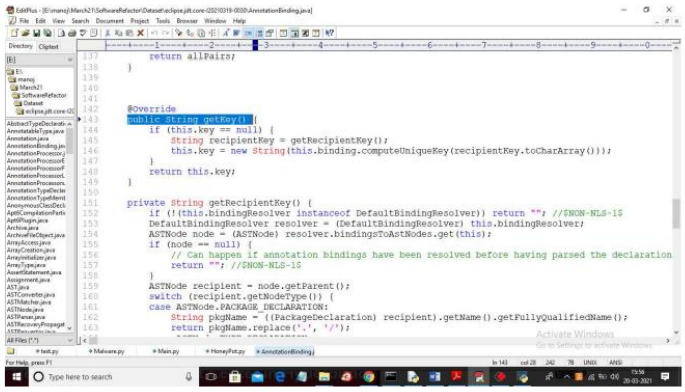

Fig 4.10 getKeys() and getKeys() contains duplicate code so refactor require

5. CONCLUSION

This project proposes a novel learning method aimed at assisting developers in refactoring code clones more effectively. The method automatically extracts features from detected code clones and uses machine learning models to advise developers on which clones need refactoring and their respective types. One key innovation is the introduction of a method to convert clone type outliers into an "Unknown" clone category, which enhances classification accuracy. The study

includes an extensive comparative analysis and evaluation of the proposed method using state-of-the-art classification models. The results demonstrate the effectiveness of the approach in achieving high accuracy in automated advising for refactored clones. In future work, the authors aim to expand the scope of their research to achieve further improvements. One potential direction is to explore set classification and deep learning techniques to enhance the capabilities of the model. These enhancements could lead to even more accurate and efficient refactoring recommendations for developers dealing with code clones.

6.FUTURE ENHANCEMENTS

For future enhancements, this project could explore advanced methods for feature extraction from code clones, such as using natural language processing (NLP) techniques to better understand code semantics. Additionally, integrating advanced machine learning models like convolutional neural networks (CNNs) or recurrent neural networks (RNNs) could improve the accuracy of clone refactoring recommendations. Integrating the refactoring advisor tool into popular Integrated Development Environments (IDEs) would provide developers with real-time refactoring suggestions as they write code. Implementing a feedback loop for developers to provide feedback on refactoring recommendations could improve the tool's accuracy over time. Supporting a wider range of programming languages and automating the refactoring process based on recommendations are also potential enhancements. Finally, optimizing the tool's performance for handling large codebases would ensure its practicality in real-world software development projects.

7. REFERENCES

[1] Y. Dang, S. Ge, R. Huang, and D. Zhang, ‘‘Code clone detection experience at microsoft,’’ in Proc. 5th Int. Workshop Softw. Clones, 2011, pp. 63–64.

[2] R. Yue, Z. Gao, N. Meng, Y. Xiong, X. Wang, and J. D. Morgenthaler, ‘‘Automatic clone recommendation for refactoring based on the present and the past,’’ in Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME), Sep. 2018, pp. 115–126.

[3] S. Kodhai and S. Kanmani, ‘‘Method-level code clone modification using refactoring techniques for clone maintenance,’’ Adv. Comput. Int. J., vol. 4, no. 2, pp. 7–26, Mar. 2013.

[4] M. Kim, V. Sazawal, D. Notkin, and G. Murphy, ‘‘An empirical study of code clone genealogies,’’ ACM SIGSOFT Softw. Eng. Notes, vol. 30, no. 5, 2005, pp. 187–196.

[5] N. Göde and R. Koschke, ‘‘Frequency and risks of changes to clones,’’ in Proc. 33rd Int. Conf. Softw. Eng., 2011, pp. 311–320.

[6] W. Wang and M. W. Godfrey, ‘‘Recommending clones for refactoring using design, context, and history,’’ in Proc. IEEE Int. Conf. Softw. Maintenance Evol., Sep. 2014, pp. 331–340.

[7] Y. Higo, T. Kamiya, S. Kusumoto, and K. Inoue, ‘‘Refactoring support based on code clone analysis,’’ in Proc. 135Int. Conf. Product Focused Softw. Process Improvement. Cham, Switzerland: Springer, 2004, pp. 220– 233.

[8] Y. Higo, T. Kamiya, S. Kusumoto, K. Inoue, and K. Words, ‘‘ARIES: Refactoring support environment based on code clone analysis,’’ in Proc. IASTED Conf. Softw. Eng. Appl., 2004, pp. 222–229.

[9] Y. Higo, S. Kusumoto, and K. Inoue, ‘‘A metric-based approach to identifying refactoring opportunities for merging code clones in a java software system,’’ J. Softw. Maintenance Evol. Res. Pract., vol. 20, no. 6, pp. 435–461, Nov. 2008.

[10] M. F. Zibran and C. K. Roy, ‘‘A constraint programming approach to conflict-aware optimal scheduling of prioritized code clone refactoring,’’ in Proc. IEEE 11th Int. Work. Conf. Source Code Anal. Manipulation, Sep. 2011, pp. 105–114.