AUTO CLONE FIX MACHINE LEARNING DRIVEN CODE REFACTORING

[1]K.UDAY KIRAN , [2]RAVULAPALLI NAVEEN

#1 Assistant Professor, Department of Master of Computer Applications (MCA), QIS College of Engineering &Technology (Autonomous), Vengamukkapalem (V), Ongole, Prakasam, AP, India

#2 MCA Student, Department of Master of Computer Applications (MCA), QIS College of Engineering &Technology (Autonomous), Vengamukkapalem (V), Ongole, Prakasam, AP, India

Abstract: To assist developers refactored code and to enable improvements to software quality when numbers of clones are found in software programs, we require an approach to advice developers on what a clone needs to refactor and what type of refactoring is needed. This paper suggests a unique learning method that automatically extracts features from the detected code clones and trains models to advise developers on what type needs to be refactored. Our approach differs from others which specify types of refactored clones as classes and creates a model for detecting the types of refactored clones and the clones which are anonymous. We introduce a new method by which to convert refactoring clone type outliers into Unknown clone set to improve classification results. We present an extensive comparative study and an evaluation of the efficacy of our suggested idea by using state-of- the-art classification models.

INTRODUCTION Code clones are pairs of code fragments which have a high degree of similarity or which are identical. Code clones might cause software maintenance to be more difficult and a system's source codes more

difficult to understand. Code cloning is a popular practice in the software development

process for a number of reasons, such as reusing code by "copy- and-paste" to increasing the speed of writing the code. There are various clone detector techniques which attempt to find code fragments which have a high number of similarities in the system's source code.Additionally, there have been variousrefactoring clone tools developed which change the structure of detected code clones without altering code fragment behavior. The refactoring code clones are a method by which to minimize thechances of introducing a bug. Refactoring,or removing, is utilized for improving software comprehensibility and maintainability. Although have shown that clone refactoring cannot solve software quality improvements for two reasons. Firstly, clones often have a short lifespan. Refactoring is less effective if there are block branches in a short distance. Secondly, longer living clones which havebeen altered with another element in the same class are difficult to remove or refactor. Additionally, it is a bug which can be simply corrected as the source code can be easily understood, which allows improvement of malleability resulting in code extensibility. Our approach provides different types of refactoring recommendation to a developer for preventing to remove the positive side of code clones and builds a training model after removing outliers to improve the results. Our tool can be built and used to minimize bugs in a system. Our study can improve clone maintenance by removing duplication code by identifying refactoring clones. Also, the possibility of bad design for a system, difficulty in a system improvement or modification, introducing a new bug, can be decreased by identifying and refactoring clones. In addition, our study can be utilized by various applications such as source codeor text plagiarism, malware detection,obfuscated code detection.

Code clones (or duplicated code) present challenges to software maintenance. They may require developers to repetitively apply similar edits to multiple program locations. When developers fail to consistently update clones, they may incompletely fix a bug or introduce new bugs [4], [13], [9], [12]. To mitigate the problem, programmers sometimes apply clone removal refactoring’s (e.g., Extract Method and Form Template Method [12]) to reduce code duplication and eliminate repetitive coding effort. However, as studied by existing work [15], [27], [5], not all clones need to be refactored. Many clones are not worthwhile to be refactored or difficult to be refactored [27], and programmers often only refactor a small portion of all clones [5]. Thus, when aclone detection tool reports lots of clones in a program, it would be difficult for the developer to go through the (usually long) list and locate the clones that should be refactored. A technique that automatically recommends only the important clones for refactoring would be valuable.

2. LITERATURE SUREVY

2.1 Code Cloning Detection Experience At Microsoft

Cloning source code is a common practice in the softwaredevelopment process. In general, the number of code clones increases in proportion to the growth of the code base. It is challenging to proactively keep clones consistent and remove unnecessary clones during the entire software development process of large-scale commercial software. In this position paper, we briefly share some typical usage scenarios of code clone detection that we collected from Microsoft engineers. We also discuss our experience on building XIAO, a code clone detection tool, and the feedback we have received from Microsoft engineers on using XIAO in real development settings.

2.2 AUTOMATIC CLONE RECOMMENDATION FOR REFACTORING BASED ON THE PRESENT AND THE PAST

When many clones are detected in software programs, not all clones are equally important to developers. To help developers refactor code and improve software quality, various tools were built to recommend clone-removal refactoring based on the past and the present information, such as the cohesion degree of individual clones or the co- evolution relations of clone peers. The existence of these tools inspired us to buildan approach that considers as many factorsas possible to more accurately recommend clones. This paper introduces CREC, a learning-based approach that recommends clones by extracting features from the current status and past history of software projects. Given a set of software repositories, CREC first automatically extracts the clone groups historically refactored (R-clones) and those notrefactored (NR-clones) to construct the training set. CREC extracts 34 features to characterize the content and evolution behaviors of individual clones, as well as the spatial, syntactical, and co-changerelations of clone peers. With these features, CREC trains a classifier that recommends clones for refactoring. We designed the largest feature set thus far for clone recommendation, and performed an evaluation on six large projects. The results show that our approach suggested refactorings with 83% and 76% F-scores in the within-project and cross-projectsettings. CREC significantly outperforms a state-of-the-art similar approach on our data set, with the latter one achieving 70% and 50% F-scores.

2.3 Method-level code clone modification using refactoring techniques for clone maintenance.

Researchers focused on activities such as clone maintenance to assist the programmers. Refactoring is a well-known process to improve the maintainability of the software. Program refactoring is a technique to improve readability, structure, performance, abstraction, maintainability or other characteristics by transforming a program. This paper contributes to a more unified approach for the phases of clone maintenance with a focus on clone modification. This approach uses the refactoring technique for clone modification. To detect the clones ‘CloneManager’ tool has been used. This approach is implemented as an enhancement to the existing tool CloneManager. The enhanced tool is tested with the open-source projects and the results are compared with the performance of other three existing tools.

3. EXISTING SYSTEM

Several studies are related to code clone refactoring. Higo et al. [7] suggest a method that refactors code clones using existing refactoring patterns such as the Extract and Pull Up Method. This research performed fully automated refactoring without developer intervention. The developer should evaluate refactoring based on their preference and indicate any clone which is a probable candidate for refactoring. Conversely, our work extracts feature and relies on machine learning to build our model and classify clones according to the type of refactored clone and those which are not refactored. Next, the developer evaluated the refactoring clones. Higo et al. [8] suggested a method that detects refactoring-oriented code clone to improve the usefulness and applicabilityof the software maintenance method. Higo et al. [9] proposed a refactoring method for merging software clones. Their technique can detect a refactoring-oriented codeclone in a general clone detected by token-based or text-based clone detection tools. We refactor clones using AST-based and PDG-based clone detection tools.

DRAWBACK IN EXISTING SYSTEM

Less Accuracy

3.1 PROPOSED SYSTEM

In this paper for the Unknown set classification, our adopted work model combines supervised learning classifiers and outlier detection for unknown classes model. This paper discusses the common and recent classification algorithms usedfor refactoring code clone classification and an outlier detection model combined for classifying the test examples as belonging to known or unknown class sizes. The improved performances of our classifier model are reliant upon its closed set validation. Model validation in machine learning is the process where by trained models are evaluated with testing datasets. The testing dataset in closed set validation contains examples which belongto known classes. We ran an outlier algorithm for datasets to find the data points which have considerably dissimilarity or inconsistency with theother given data points. Then, the data point classes are changed into unknownclasses. After detecting outlier data points, we build our model for closed-set classification and perform analysis of their performance after training. We train and test our classifier with vectors of datasets.

ADVANTAGES

Increased

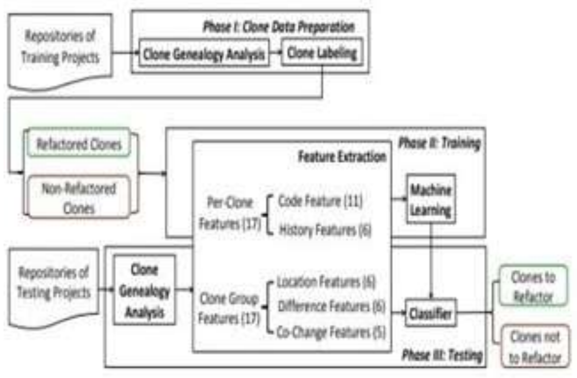


Fig: Architecture

4. IMPLEMENTATION

Python is a general-purpose language. It has wide range of applications from Web development (like: Django and Bottle), scientific and mathematical computing (Orange, SymPy, NumPy) to desktop graphical user Interfaces (Pygame, Panda3D). The syntax of the language is clean and length of the code is relatively short. It's fun to work in Python because itallows you to think about the problem rather than focusing on the syntax.

History of Python:

Python is a fairly old language created by Guido Van Rossum. The design began in the late 1980s and was first released in February 1991.

Why Python was created?

In late 1980s, Guido Van Rossum was working on the Amoeba distributedoperating system group. He wanted to usean interpreted language like ABC (ABC hassimple easy-to-understand syntax) that could access the Amoeba system calls. So, he decided to create a language that was extensible. This led to design of a new language which was later named Python.

Why the name Python?

It wasn't named after a dangerous snake. Rossum was fan of a comedy series from late seventies. The name "Python" was adopted from the same series "Monty Python's Flying Circus".

A simple language which is easier to learn

Python has a very simple and elegant syntax. It's much easier to read and write Python programs compared to other languages like: C++, Java, C#. Python makes programming fun and allows you tofocus on the solution rather than syntax.

If you are a newbie, it's a great choice to start your journey with Python.

Free and open-source

You can freely use and distribute Python, even for commercial use. Not only can you use and distribute software's written in it, you can even make changes to the Python's source code.

Python has a large community constantly improving it in each iteration.

Portability

Free and open-source

You can freely use and distribute Python, even for commercial use. Not only can you use and distribute software's written in it, you can even make changes to the Python's source code.

Python has a large community constantly improving it in each iteration.

Portability

You can move Python programs from one platform to another, and run it without any changes.

It runs seamlessly on almost all platforms including Windows, Mac OS X and Linux.

Extensible and Embeddable

Suppose an application requires high performance. You can easily combine pieces of C/C++ or other languages with Python code.

This will give your application high performance as well as scripting capabilities which other languages may not provide out of the box.

A high-level, interpreted language

Unlike C/C++, you don't have to worry about daunting tasks like memory management, garbage collection and soon.

Likewise, when you run Python code, it automatically converts your code to the language your computer understands. You don't need to worry about any lower-level operations.

5. ALGORITHM:

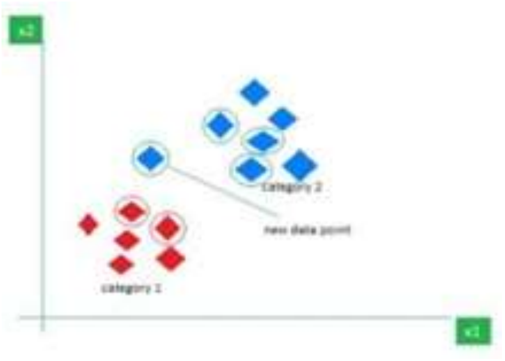
5.1 KNN

```
recall.clear()
fmeasure.clear()
global X_train, X_test, y_train, y_test
text.delete('1.0', END)
knn KNeighborsClassifier()
knn.fit(X_train, y_train)
predict knn.predict(X_test
acc accuracy_score(y_test.predict) * 100
100
p = precision_score(y_test, predict, average='macro') *
r = recall_score(y_test, predict,average='macro') * 100
f=fl_score(y_test, predict,average='macro')
100
text.insert(END, "KNN Accuracy: "+str(acc)+"\n")
text.insert(END, "KNN Precision: "+str(p)+"\n")
text.insert(END, "KNN Recall: "+str(r)+"\n")
text.insert(END, "KNN FMeasure: "+str(f)+"\n\n")
accuracy.append(acc)
precision.append(p)
recall.append(r)
fmeasure.append(f)
```

KNN is one of the most basic yet essential classification algorithms in machine learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining, and intrusion detection.

It is widely disposable in real-life scenarios since it is non-parametric, meaning it does not make any underlying assumptions about the distribution of data (as opposed to other algorithms such as GMM, which assume a Gaussiandistribution of the given data). We are

given some prior data (also called training data), which classifies coordinates into groups identified by an attribute.As an example, consider the following table of data points containing two features:



KNN Algorithm working visualization

Now, given another set of data points (also called testing data), allocate these points toa group by analyzing the training set. Note that the unclassified points are marked as 'White'.

Intuition Behind **KNN Algorithm**

If we plot these points on a graph, we may be able to locate some clusters or groups. Now, given an unclassified point, we can assign it to a group by observing what group its nearest neighbors belong to. This means a point close to a cluster of points classified as 'Red' has a higher probabilityof getting classified as 'Red'. Intuitively, we can see that the first point (2.5, 7) should be classified as 'Green', and the second point (5.5, 4.5) should be classified as 'Red'.

Why do we need a **KNN algorithm**?

(K-NN) algorithm is a versatile and widely used machine learning algorithm that is primarily used for its simplicity and ease of implementation. It does not require any assumptions about the underlying data distribution. It can also handle both numerical and categorical data, making it a flexible choice for various types of datasets in classification and regression tasks. It is a non-parametric method that makes predictions based on the similarity of data points in a given dataset. K-NN is less sensitive to outliers compared to other algorithms.

make predictions based on the local structure of the data.

Distance Metrics Used in **KNN Algorithm**

As we know that the KNN algorithm helpsus identify the nearest points or the groups for a query point. But to determine the closest groups or the nearest points for a query point we need some metric. For this purpose, we use below distance metrics:

Euclidean Distance

This is nothing but the cartesian distance between the two points which are in the plane/hyperplane. Euclidean distance canalso be visualized as the length of the straight line that joins the two points which are into consideration. This metric helps uscalculate the net displacement donebetween the two states of an object.

5.2 SVM ALGORITHM

```
def runSVM():
global X_train, X_test, y_train, y_test
```

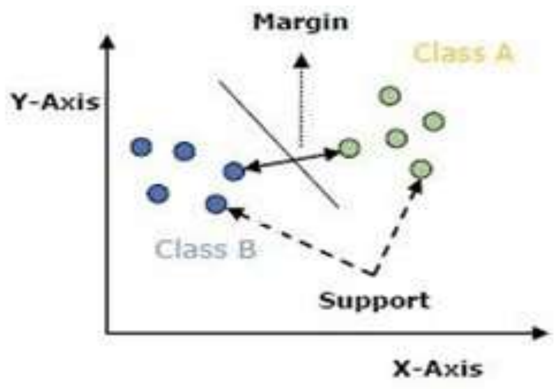
```
rf = svm.SVC()
rf.fit(X_train, y_train)
predict = rf.predict(X_test)
acc = accuracy_score(y_test,predict) * 100
p = precision_score(y_test, predict, average='macro') *
100
r = recall_score(y_test, predict, average='macro') * 100
f = fl_score(y_test, predict, average='macro') * 100
text.insert(END, "SVM Accuracy: "+str(acc)+"\n")
text.insert(END, "SVM Precision: "+str(p)+"\n")
text.insert(END, "SVM Recall: "+str(r)+"\n")
 text.insert(END, "SVM FMeasure: "+str(f)+"\n\n")
accuracy.append(acc)
precision.append(p)
recall.append(r)
fmeasure.append(f)
```

Introduction to SVM

Support vector machines (SVMs) are powerful yet flexible supervised machine learning algorithms which are used both for classification and regression. But generally, they are used in classification problems. In 1960s, SVMs were first introduced but later they got refined in 1990. SVMs have their unique way of implementation as compared to other machine learning algorithms. Lately, they are extremely popular because of their ability to handle multiple continuous and categorical variables.

Working of SVM

An SVM model is basically a representation of different classes in a hyperplane in multidimensional space. The hyperplane will be generated in an iterativemanner by SVM so that the error can be minimized. The goal of SVM is to divide the datasets into classes to find a maximum marginal hyperplane (MMH).



SVM Kernel

In practice, SVM algorithm is implemented with kernel that transforms an input data space into the required form. SVM uses a technique called the kernel trick in which kernel takes a low dimensional input space and transforms it into a higher dimensional space. In simple words, kernel converts non-separable problems into separable problems by adding more dimensions to it. It makes SVM more powerful, flexible and accurate. The following are some of the types of kernels used by SVM.

Linear Kernel

It can be used as a dot product between any two observations. The formula of linear kernel is as below

$K(x,xi)=sum(x*xi)K(x,xi)=sum(x*xi)$

From the above formula, we can see thatthe product between two vectors say & x is the sum of the multiplication of eachpair of input values.

Polynomial Kernel

It is more generalized form of linear kerneland distinguish curved or nonlinear input space. Following is the formula for polynomial kernel

k(X,Xi)=1+sum(X∗Xi)^dk(X,Xi)=1+sum(X∗Xi)^d

Here d is the degree of polynomial, which we need to specify manually in the learning algorithm.

Radial Basis Function (RBF) Kernel

RBF kernel, mostly used in SVMclassification, maps input space in indefinite dimensional space. Following formula explains it mathematically

K(x,xi)=exp(−gamma∗sum(x−xi^2))K(x,xi) =exp(−gamma∗sum(x−xi^2))

RESULTS AND DISCUSSION

Accuracy: The accuracy of a test is itsability to differentiate the patient and healthy cases correctly. To estimate the accuracy of a test, we should calculate the proportion of true positive and true negative in all evaluated cases. Mathematically, this can be stated as:

Accuracy = TP + TN TP + TN + FP + FN.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

F1-Score: F1 score is a machine learning evaluation metric that measures a model's accuracy. It combines the precision and recall scores of a model. The accuracy metric computes how many times a model

Here, *gamma* ranges from 0 to 1. We need to manually specify it in the learning algorithm. A good default value of *gamma* is 0.1.

As we implemented SVM for linearly separable data, we can implement it in Python for the data that is not linearly separable. It can be done by using kernels.

Example

The following is an example for creating an SVM classifier by using kernels. We will be using *iris* dataset from *scikit- learn*

$$\textbf{F1 Score} = \frac{2}{\left(\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}\right)}$$

$$\textbf{F1 Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Precision: Precision evaluates the fraction of correctly classified instances or samples among the ones classified as positives. Thus, the formula to calculate theprecision is given by:

Precision = True positives/ (True positives + False positives) = TP/(TP + FP)

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

6. COMPARISION GRAPH

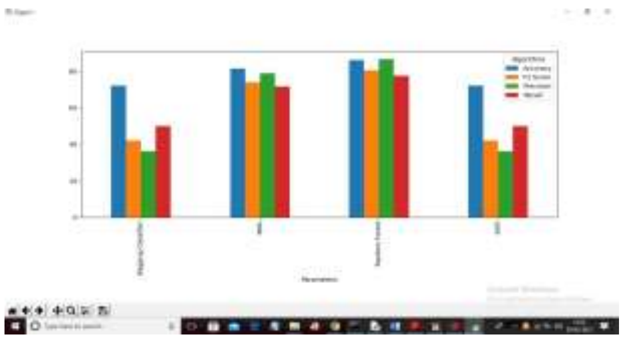


Fig.3 : Generating Features vector

Fig.4 : To Calculate local outlier factor

7.FINAL OUTPUT

To click on run button

Fig.1: Display the algorithms

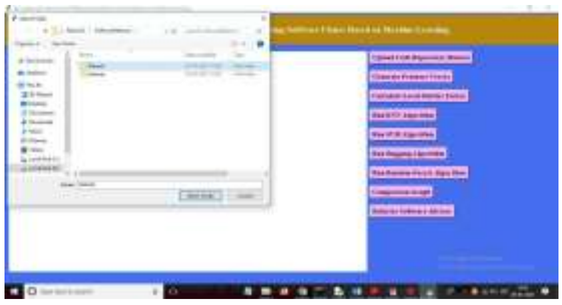


Fig.2 : Upload code repostory dataset

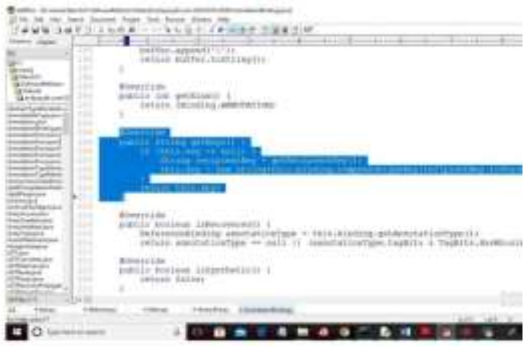


Fig.5 : Refactoring software advisor

CONCLUSION

This paper suggests a learning method which automatically extracts features from the detected code clones and trains the models to advise the developers in regard to what a clone needs to be refactored and what is its type. We introduce a new method of converting clone type outliers into an Unknown clone to improve classification results. We present an extensive comparative study and perform an evaluation of the efficacy of our suggested idea by using state-of-the- art classification models. We present a newmachine learning framework that automatically extracts features from the detected code clones and trains models to advise the developers on the type of refactored clone code and those which are not refactored. We explore a new method by which to clone types of outliers into an Unknown clone from the training categories, which significantly improves the classification results. We present an extensive comparative study and anevaluation of the efficacy of our suggested idea by using stateof-the-art classification models. We used four classification models to obtain their relative performance.The experimental results suggest that our approach has high value in achieving high automated advising refactored clone accuracy. In future, we would like to increase the scope of work to achieve additional improvements, for example, by using set classification and deep learning.

FUTURE SCOPE

To assist developers refactored code and to enable improvements to software quality when numbers of clones are found in software programs, we require an approachto advise developers on what a clone needsto refactor and what type of refactoring is needed. This paper suggests a unique learning method that automatically extracts features from the detected code clones and trains models to advise developers on what type needs to be refactored. Our approach differs from others which specify types of refactored clones as classes and creates a model for detecting the types of refactored clones and the clones which are anonymous. We introduce a new method by which to convert refactoring clone type outliers into Unknown clone set to improve classification results. We present an extensive comparative study and an evaluation of the efficacy of our suggested idea by using state-of-the-art classification models

MODULES

1. Upload Module: Using this module author ploading code clone/repositoryfrom eclipse, Net beans etc.

2. Extract Features: Using this module author applying natural language tool kit (NLTK) to convert code words in to feature vector and then normalize feature vector to 0 and 1 by taking average of eachword count.

3. Local Outlier Factor (LOF): Using this algorithm author identifying important attributes/columns from features vectorand this algorithm return 1 if attribute is anomaly (not outlier) and -1 if attribute is not important

or outlier. So using this algorithm we can extract important attributes from features vector.

4 Run Machine learning Algorithms: Features will be converted into train and test records and then based on similarity between code modules class label will be assigned as 0 or 1. If code contains somany similar words then 1 will be assigned other wise 0 will be assigned. Using this module various machinelearning algorithms will be applied such as SVM, KNN, Bagging classifier andRandom Forest,
5.Code Advisor: After building machine learning trained model we can predict code which require refactoring and then developer will analyse predicted code to refactor.

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.

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.

N. Göde and R. Koschke, "Frequency and risks of changes to clones," in Proc. 33rd Int. Conf. Softw. Eng., 2011, pp. 311-320.

[5] W. Wang and M. W. Godfrey, "Recommending clones for refactoringusing design, context, and history," inProc. IEEE Int. Conf. Softw. Maintenance Evol, Sep. 2014, pp. 331- 340.

[6] 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:

[7] 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.

[8] 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.

[9] 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.

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.

[10] K. Hotta, Y. Higo, and S. Kusumoto, "Identifying, tailoring, and suggesting form template method refactoring opportunities with program dependence graph," in Proc. 16th Eur. Conf. Softw. Maintenance Reeng.. Mar. 2012, pp. 53-62.

[11] R. Tairas and J. Gray, "Increasing clone maintenance support by unifying clone detection and refactoring activities," Inf. Softw. Technol., vol. 54, no. 12, pp. 1297-1307, Dec. 2012.

[12] N. Tsantalis, D. Mazinanian, and G.

P. Krishnan, "Assessing therefactorability of software clones," IEEE Trans. Softw. Eng., vol. 41, no. 11, pp. 1055-1090, Nov. 2015,

[13] M. Mondal, C. K. Roy, and K. A. Schneider, "SPCP-miner: A tool for mining code clones that are importantfor refactoring or tracking," in Proc. IEEE 22nd Int. Conf. Softw. Anal., Evol., Reeng. (SANER), Mar. 2015, pp. 484-488,