ISSN(Online): 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(A High Impact Factor, Monthly Peer Reviewed Journal)

Vol. 5, Issue 2, February 2016

Improved Analysis of Refactoring in Forked Project to Remove the Bugs Present in the System

Inderjeet Kour Jhans[1], Dr.V.Krishna Priya[2]

Research Scholar, Department of Computer Science, Sri Ramakrishna College of Arts and Science for Women, Coimbatore, Tamil Nadu, India[1]

Director and Head, Department of Computer Science, Sri Ramakrishna College of Arts and Science for Women, Coimbatore, Tamil Nadu, India[2]

ABSTRACT: Software forking is a process of creating the new project by using the present coding of the existing program without affecting the nature of the existing project. When developing the new project by using the existing source knowledge, there is a chance of bugs due to configuration adaptation problems. This bugs needs to be analysed and avoided for the better system performance in terms of the improved code. In the existing work, repertoire tool is used for analysing the forked software project for bugs. Repertoire tool analyses the work in terms of the history patches where the previous existing code would be compared with the newly generated code to find the difference and the bugs. Bug refactoring is carried out for improving the design of existing code without changing its observable behaviour.The performance of the existing research work is improved by applying the code quality metrics in the proposed research methodology. The code quality metrics that are used in the proposed research methodology are introduced by Chidamber and Kemerer based on object oriented programming language. The experimental tests conducted were proves that the proposed methodology leads to improved performance of the proposed methodology than the existing approach by analysing the code quality in the efficient manner.

KEYWORDS: Software Forking, Bug refactoring, forked code, ckjmmetrics.

I. INTRODUCTION

It has become increasingly common to create a variant software product or to introduce a new feature by copying code fragments from similar software products. As copying code fragments across products is common, there are names referring to this process: forking—copying an existing product to create a slightly different product and porting—copying an existing feature or bug fix from one program context to another. Forking is particularly common in free and open source software projects. The open source community often forks an existing project due to a conflict in vision or personality clash. For instance, the split of FreeBSD and NetBSD from 386BSD, XEmacs from GNU Emacs, and LibreOffice from OpenOffice are well known forks. Proprietary software is also forked to support different customer needs. Some notable proprietary forks include EnterpriseDB (a fork of PostgreSQL), Mac OS X (based on the proprietary Next step and the open source FreeBSD), and Cedega and CrossOver (proprietary forks of Wine).

With the practice of using forked codes in the projects, software developers find it difficult to use it, because the forked code which is obtained from already existing code may contain bugs and may not fit into the new project in which it is going to be used. There are some tools which can be used to detect the code similarities and duplications. But for the detection of bugs which has been copied from the forked code, the code must be implemented and tested. To improve the process of developing the code, the forked code is refactored, so that the code is cleaned up to reduce the introduction of bugs.

ISSN(Online): 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(A High Impact Factor, Monthly Peer Reviewed Journal)

Vol. 5, Issue 2, February 2016

II. RELATED WORK

To create a new variant of an existing project, developers often copy an existing code base and modify it. This process is called software forking. After forking software, developers often port new features or bug fixes from peer projects. [3]Repertoire analyses repeated work of cross-system porting among forked projects. It takes the version histories as input and identifies ported edits by comparing the content of individual patches. It also shows users the extent of ported edits, where and when the ported edits occurred, which developers ported code from peer projects, and how long it takes for patches to be ported. In the existing system repertoire tool is used to predict the different port codes that are present in the source code which are created by software forking. Repertoire tool attempts to detect such clone parts of the programs in two steps[11]. Those are

1. Clone detection using CC-Finder
2. Similarity matching of cloned codes using N-gram methodology

The above steps are followed to detect the cross port codes present in the software, thus the further forking can be made without any violation.

In the existing work, forking is done only within the software application in various functions and methods

1. Re-using the forked code may lead to the occurrence of bugs, if already present.
2. Coding quality metrics is not considered which might lead to performance degradation in the case of output generation.

III. SOFTWARE FORKING

In software engineering, [1]a project fork happens when developers take a copy of source code from one software package and start independent development on it, creating a distinct and separate piece of software. The term often implies not merely a development branch, but a split in the developer community, a form of schism.

Free and open-source software is that which, by definition, may be forked from the original development team without prior permission without violating copyright law. However, licensed forks of proprietary software (e.g. Unix) also happen.

Four possible outcomes of a fork [2], with examples:

1. The death of the fork. This is by far the most common case. It is easy to declare a fork, but considerable effort to continue independent development and support.
2. A re-merging of the fork (e.g., egcs becoming "blessed" as the new version of gcc.)
3. The death of the original (e.g. the X.Org Server succeeding and XFree86 dying.)
4. Successful branching, typically with differentiation (e.g., OpenBSD and NetBSD.)

Distributed revision control (DVCS) tools have popularised a less emotive use of the term "fork", blurring the distinction with "branch". With a DVCS such as Mercurial or Git, the normal way to contribute to a project is, to first branch the repository, and then seek to have the changes integrated with the main repository. Sites such as GitHub, Bitbucket and Launch pad provide free DVCS hosting expressly supporting independent branches, such that the technical, social and financial barriers to forking a source code repository are massively reduced, and GitHub uses "fork"[8] as its term for this method of contribution to a project.

Forks often restart version numbering from 0.1 or 1.0 even if the original software was at version 3.0, 4.0, or 5.0. An exception is when the forked software is designed to be a drop-in replacement for the original project, e.g. MariaDB for MySQL or LibreOffice for OpenOffice.org.

With the practice of using forked codes in the projects, software developers find it difficult to use it, because the forked code which is obtained from already existing code may contain bugs and may not fit into the new project in which it is going to be used. There are some tools which can be used to detect the code similarities and duplications[10]. But for the detection of bugs which has been copied from the forked code, the code must be implemented and tested[9]. To improve the process of developing the code, the forked code is refactored, so that the code is cleaned up to reduce the introduction of bugs.

The main purpose of using forked code to develop a software project is

1. To reduce the code complexity and readability.
2. To increase reusability of the forked code, to build new software with the help of existing code.

ISSN(Online): 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(A High Impact Factor, Monthly Peer Reviewed Journal)

Vol. 5, Issue 2, February 2016

IV. REFACTORING

Refactoring is the process of changing a software system in such a way that it does not alter the external behaviour of the code yet improves its internal structure. It is a disciplined way to clean up code that minimizes the chances of introducing bugs. In essence when refactoring is done the design of the code is improved after it has been written. "Improving the design after it has been written."[5] That's an odd turn of phrase. In current understanding of software development, the software is first designed and then its coded. A good design comes first, and the coding comes second. Over time the code will be modified, and the integrity of the system, its structure according to that design, gradually fades. The code slowly sinks from engineering to hacking.

Code refactoring is the process of restructuring existing computer code – changing the factoring – without changing its external behaviour. Refactoring improves non-functional attributes of the software. Advantages include improved code readability and reduced complexity; these can improve source code maintainability and create a more expressive internal architecture or object model to improve extensibility. Typically, refactoring applies a series of standardised basic micro-refactoring, each of which is (usually) a tiny change in a computer program's source code that either preserves the behaviour of the software, or at least does not modify its conformance to functional requirements. Many development environments provide automated support for performing the mechanical aspects of these basic refactoring. [6]If done extremely well, code refactoring may also resolve hidden, dormant, or undiscovered computer bugs or vulnerabilities in the system by simplifying the underlying logic and eliminating unnecessary levels of complexity. If done poorly it may fail the requirement that external functionality not be changed, and/or introduce new bugs.

V. PROCESS OF BUG REFACTORING

Forks are generally thought of as being bad for software projects, because they reduce the total pool of developers available to any one software system, slowing evolution. To maintain co-evolution, developers need to monitor the peer projects and when a necessary feature or a bug fix is introduced to one project; developers may need to port it to the other. This involves lot of manual repetitive work. However, it is not entirely clear what is the overhead involved in cross-system porting to maintain forked projects in parallel.

Once the code of the forked project is obtained it is then uploaded to the server to find and rectify the bugs that are occurred in the system. It is done based on the various performance parameters that are involved in the system.

This work presents REPERTOIRE, a tool which facilitates the analysis of software forks using several quantitative measures. REPERTOIRE allows users to analyse the number of lines of code ported between forks, the developers responsible for those ports, and the time between when a patch is first generated for an intended project and when it is ported. It also presents a graphical means for users to observe a pattern of ports. These analyses are designed to aid managers and architects to make informed decisions about the extent of porting within projects.

After detecting the corresponding variation between the cloned projects, the client would attempt to remove the variation and bugs present in the clone nodes by invoking modify request. This modify request would attempt to alter the coding parts that are detected by the previous nodes.

To do so, REPERTOIRE associates the identified ported edits with developers, commit dates, and file locations, and characterizes cross-system porting in developers, temporal, and spatial dimensions. This helps project managers or architects to make an informed decision on how to maintain a product family.

This is done to remove the variation present between the various performance parameters resides in the system

VI. CHIDAMBER AND KEMERER OBJECT-ORIENTED METRICS

Chidamber&Kemerer Java Metrics is an open source command-line tool[2]. The program ckjm calculates Chidamber and Kemerer object-oriented metrics by processing the bytecode of compiled Java files. The program calculates for each class the following six metrics, and displays them on its standard output, following the class's name:

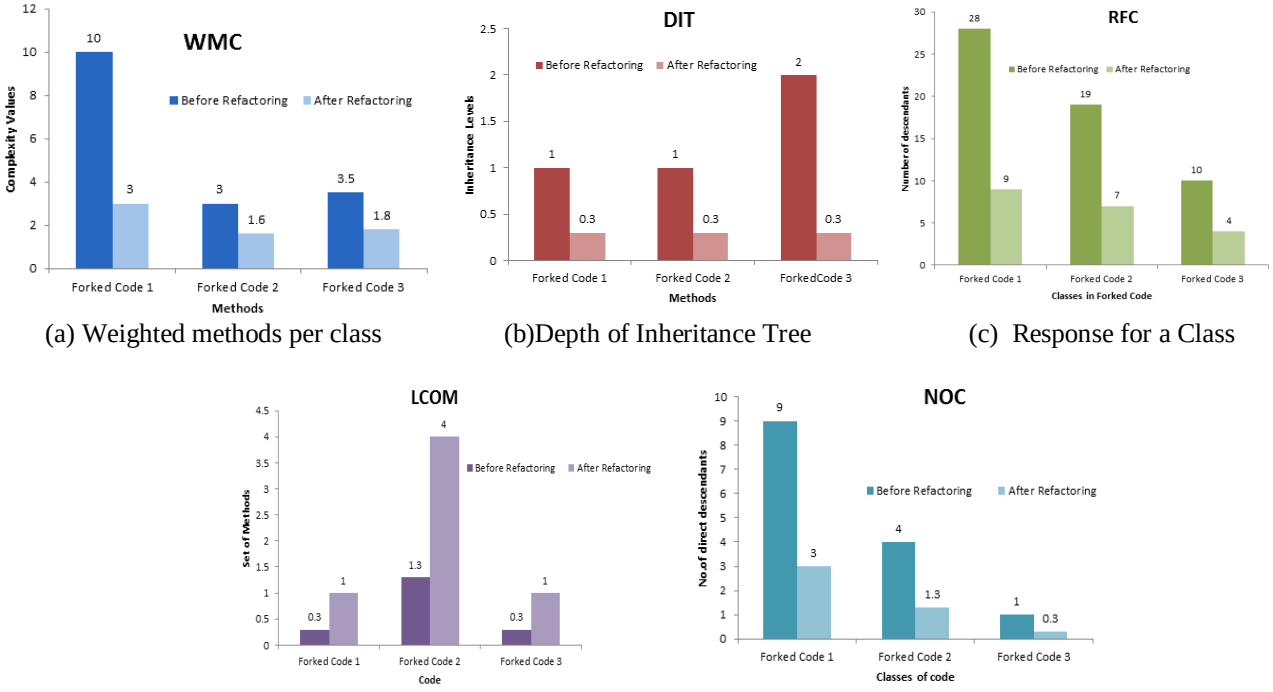
The ckjm metrics calculated for the forked project is as shown.

ISSN(Online): 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(A High Impact Factor, Monthly Peer Reviewed Journal)

Vol. 5, Issue 2, February 2016



(a) Weighted methods per class (b)Depth of Inheritance Tree (c) Response for a Class

(d)Lack of cohesion in methods (e) NOC - Number of Children

Figure (a), shows WMC (Weighted methods per class)-A class's weighted methods per class WMC metric is simply the sum of the complexities of its methods. As a measure of complexity we can use the cyclomatic complexity, or a complexity value can be arbitrary value of 1 to each method. The ckjm program assigns a complexity value of 1 to each method, and therefore the value of the WMC is equal to the number of methods in the class.In the figure the weighted methods per code shows a decrease after the forked code is refactored.

Figure (b), shows DIT (Depth of Inheritance Tree)-The depth of inheritance tree (DIT) metric provides for each class a measure of the inheritance levels from the object hierarchy top. In Java where all classes inherit Object the minimum value of DIT is 1.The inheritance levels are decreased after refactoring which reduces the depth of the inheritance tree. This lowers the size metrics of the Forked code.

Figure (c), describes NOC (Number of Children)- This metric is the number of direct descendants (subclasses) for each class. Classes with large number of children are considered to be difficult to modify and usually require more testing because of the effects on changes on all the children. They are also considered more complex and fault-prone because a class with numerous children may have to provide services in a larger number of contexts and therefore must be more flexible.

Figure (d), represents RFC - Response for a Class The metric measures the number of different methods that can be executed when an object of that class receives a message (when a method is invoked for that object). Ideally, it is found for each method of the class, the methods that class will call, and repeat this for each called method, calculating what is called the transitive closure of the method's call graph. This process can however be both expensive and quite inaccurate. In ckjm, a rough approximation is calculated to the response set by simply inspecting method calls within the class's method bodies.The RFC measure is decreased in the following figure which means the complexity of the code is also reduced.
Figure (e), LCOM (Lack of cohesion in methods) - A class's lack of cohesion in methods (LCOM) metric counts the sets of methods in a class that are not related through the sharing of some of the class's fields. The original definition of this metric (which is the one used in ckjm) considers all pairs of a class's methods. In some of these pairs both methods

ISSN(Online): 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

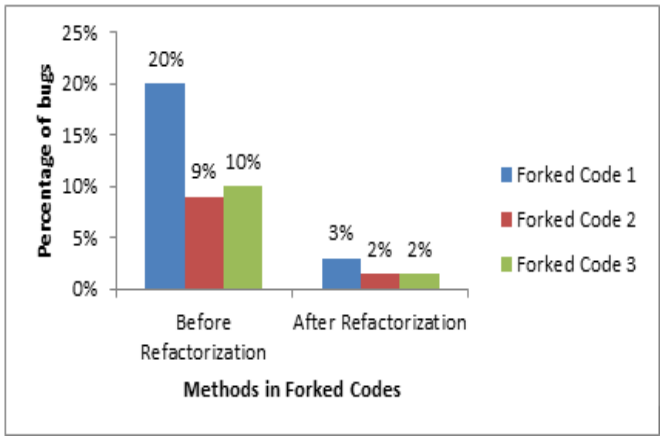
(A High Impact Factor, Monthly Peer Reviewed Journal)

Vol. 5, Issue 2, February 2016

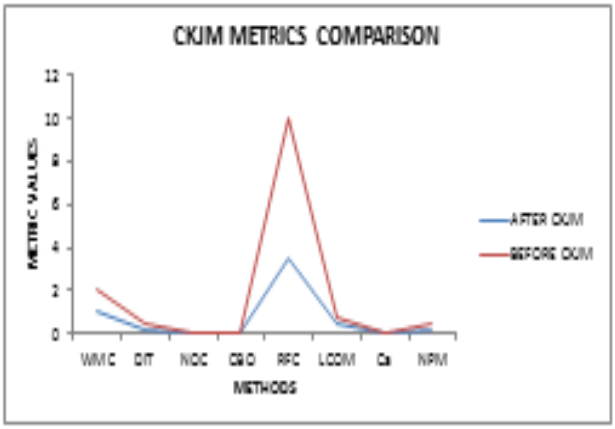
access at least one common field of the class, while in other pairs the two methods to not share any common field accesses. The lack of cohesion in methods is then calculated by subtracting from the number of method pairs that don't share a field access the number of method pairs that do. Note that subsequent definitions of this metric used as a measurement basis the number of disjoint graph components of the class's methods. Others modified the definition of connectedness to include calls between the methods of the class.LCOM count is increased in the analysis, which means the cohesion is high.Modules with high cohesion tend to be preferable because high cohesion is associated with several desirable traits of software including robustness, reliability and reusability

VII. EXPERIMENTAL RESULTS

ANALYSIS OF BUGS AND CKJM METRICS



(f)



(g)

Figure (f). shows the Percentage of Bugs present in the Forked Code before and after refactoring,which means there is a drop in the percentage of bugs when refactoring is carried out.

Figure (g). shows the comparison of metric values in methods of the class before and after applying ckjm metrics,which proves the quality improvement of the forked project where the bug fixing would be more efficient after considering these parameter values. Software metric comparison is done in terms of various performance evaluation parameter values. The comparison result is shown in the following figure.

ANALYSIS OF METRICS BEFORE AND AFTER REFACTORING:

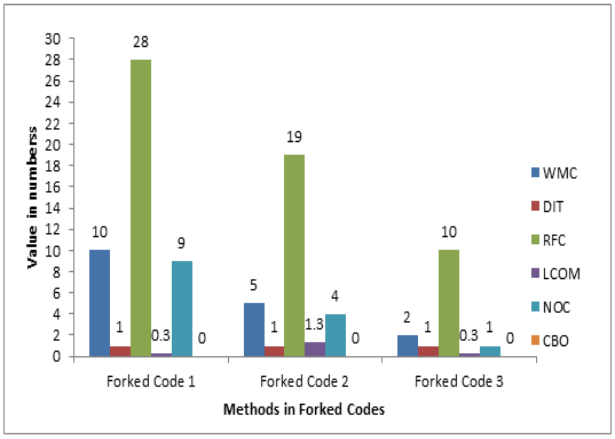
The values of RFC and DIT are reduced after the process of refactoring. This means the complexity of the project is reduced, as the values RFC represents the complexity, the higher the rate ofRFC, the more the complexity of the project.The DIT is the Depth of the inheritance tree, which is when reduced, decreases the size of the forked code. The following figures represent the difference between the metrics analysis before and after

ISSN(Online): 2319-8753
ISSN (Print) : 2347-6710

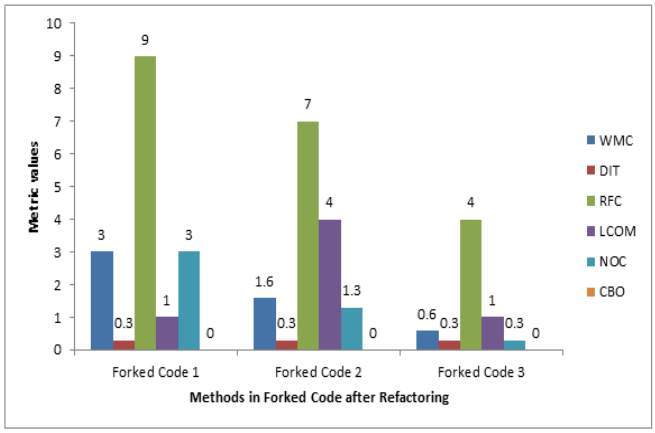
International Journal of Innovative Research in Science, Engineering and Technology

(A High Impact Factor, Monthly Peer Reviewed Journal)

Vol. 5, Issue 2, February 2016



(h)CKJM Metrics before Refactoring

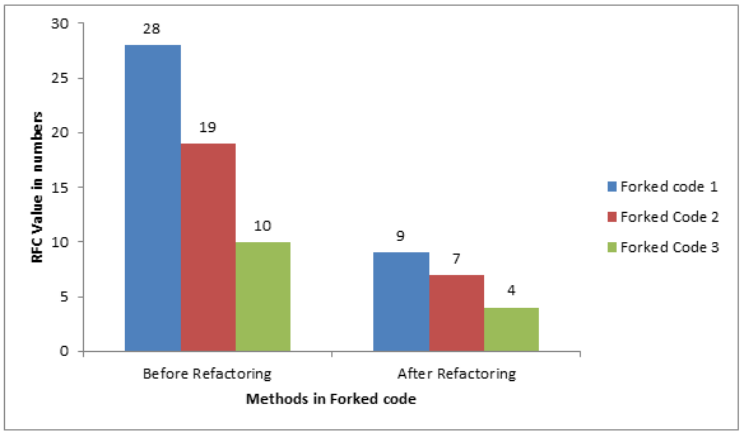


(i) CKJM Metrics after Refactoring

Figure (h) and(i) , represents the changes in the value of metrics in the respective forked codes before and after refactoring.

Refactoring improves non-functional attributes of the software. Advantages of refactoring include improved code readability and reduced complexity; these can improve source code maintainability and create a more expressive internal architecture or object model to improve extensibility.

CODE COMPLEXITY EVALUATION:



(j)Comparison of RFC in forked codes before and after refactoring

The metric called the response for a class (RFC) measures the number of different methods that can be executed when an object of that class receives a message (when a method is invoked for that object). The RFC for a class should usually not exceed 50 although it is acceptable to have a RFC up to 100. Refactor IT recommends a default threshold from 0 to 50 for a class. If the RFC for a class is large, it means that there is a high complexity.
[3] For example, if a method call on a class can result in a large number of different method calls on the target and other classes. It can be hard to test the behaviour of the class and to debug problems since comprehending class behaviour requires a deep understanding of the potential interactions that objects of the class can have with the rest of the system.

ISSN(Online): 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(A High Impact Factor, Monthly Peer Reviewed Journal)

Vol. 5, Issue 2, February 2016

The analysis shows that the implementation of a forked and refactored code in an application can reduce the dependency factor and can help in developing a scalable architecture. Here, the quality has been improved by reducing the coupling and introducing the cohesion measure.

VIII. CONCLUSION

Extending prior approaches to find duplicate bug reports, similar bug reports across different forked projects can be found and their corresponding fixed patches can be extracted. Then, REPERTOIRE can verify how similar these fixed patches are. This will help us to determine whether two similar bug reports in two related projects undergo similar fixes. If we confirm that a large number of fixes are actually similar, it would motivate building a patch recommendation system. When a new bug will be reported in a project, then a similar bug report in the sibling projects is analysed and its fix is suggested.

In the future work, the same metrics can be obtained by using the source code from a large java based project, which can be forked and reused easily by the developers. Along with the calculation of object-oriented metrics, other parameters like compatibility, maintainability and Time Complexity etc. are obtained to improve the quality of software.

REFERENCES

[1] RiabovAnton.V, Eric Bouillet, MarkFeblowitz .D, Zhen Liu And RanganathanAnand,"Wishful Search: Interactive Composition Of Data Mashups" WWW '08 Proceedings of the 17th international conference on World Wide Web, 2008.

[2] RüdigerLincke, Jonas Lundberg, WelfLöwe , "Software Metrics Tool Selection", ISSTA '08 Proceedings of the international symposium on Software testing and analysis. Pages 131-142,2008.

[3] CKJM tool- https://www.spinellis.gr/sw/ckjm.

[4] Baishakhi Ray, Christopher Wiley, MiryungKim,"REPERTOIRE: A Cross-System Porting Analysis Tool for Forked Software Projects", Proceedings of the ACM SIGSOFT, 2012.

[5] Ernst NA, Easterbrook SM, MylopoulosJ , "Code forking in open-source software: a requirements perspective",CoRR abs/1004.2889,2010.

[6] Kathryn T. Stolee, Sebastian Elbaum, "Refactoring Pipe-Like Mashups For End-User Programmers", 33rd International Conference on Software Engineering (ICSE),2011.

[7] R.Viseur , "Forks impacts and motivations in free and open source projects", (IJACSA), 2012

[8] William G. Griswold, David Notkin, "Automated Assistance For Program Restructuring" ACM SIGSOFT Software Engineering Notes, 17(1), 2012.

[9] Gerardo Canfora, Luigi Cerulo, Marta Cimitile, and Massimiliano Di Penta. "Social interactions around cross-system bug fixings: the case of freebsd and openbsd. In Proceeding of the 8th working conference on Mining software repositories", MSR '11, pages 143–152, New York, NY, USA, ACM,2011.

[10] RüdigerLincke, Jonas Lundberg, WelfLöwe,"Software Metrics Tool Selection", ISSTA '08 Proceedings of the international symposium on Software testing and analysis, Pages 131-142,2008.

[11] Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue, "CCFinder: A multilinguistictoken-based code clone detection system for large scale source code" ,IEEE Transactions on Software Engineering, 28(7):654–670, 2002.

[12] William G. Griswold, David Notkin, "Automated Assistance For Program Restructuring" ,ACM SIGSOFT Software Engineering Notes, 17(1), 2012.

[13] https://en.wikipedia.org/wiki/Fork (software_development)