Algorithm to Detect Non-Contiguous Clones with High Precision

Sonam Gupta
Research Scholar,
Suresh Gyan Vihar University, Jaipur, Rajasthan, India

Dr. P.C. Gupta
Department of Computer Science and Engineering
Kota University, Kota, Rajasthan , India

Abstract- Researchers have proved that duplication of code occur frequently in several systems because of various reasons [5,7]. It had been proved that almost 70% of the effort is wasted in resolving the clones during maintenance [8] since if the clones are not removed then it will lead to further more problems like hindrance to comprehension of the program, independent evolution of clones, bad design etc. Although code clones are a major problem but still they are evolved in the system because of the limitation of the programmer to finish the work as soon as possible. This paper focuses on detection of clones in large software systems so as to reduce the effort during maintenance. It describes a novel approach to design and implement a tool for detecting cloned codes in the system. The algorithm is developed in such a manner that it is precise and scalable with performance factor.

Keywords – Software Maintenance, clones, collectors, statement grouping.

I. INTRODUCTION

The impacts of clones on the standard of ASCII content document don't appear to be well caught on. In a great part of the writing on the subject, cloned code is considered harming to the standard of the ASCII content record, on the grounds that it is normally accepted that code clones will result in additional support exertion [1,2,3,4,5,6,7]. For example, changes to one period of code could need to be proliferated to numerous others, heightening upkeep costs [14]. Considering this, routes for programmed refactoring are expeditious [5,11], and apparatuses particularly to help engineers in the manual refactoring of code clones have moreover been produced [15]. There is little uncertainty that clone is an indication of messy style and in such cases should be considered to be a kind of improvement "awful stench". Notwithstanding, explores found that there are a few occasions wherever this can be simply not the situation. Case in point, cloning is additionally usual acquaint test enhancements with center subsystems while not contrarily impactful the consistent quality of the most code base [9]. These trial changes can be utilized as a piece of the get together framework, allowing completion clients to just switch between test alternatives and stable ones. Cloned code can even be usual dodge too much propelled code following from interleaving two or a great deal of sections of similar however non-indistinguishable code portions. The regular inclination to establish inside the writing that clone is inexorably unsafe to code quality and will be destroyed at whatever point potential. Presently the expression "cloning considered unsafe" — seemed wrong to U.S.A., even "destructive" itself. Narratively, the inclination to have discovered that clones may be utilized as a part of an exceptionally mixture of the path and for assortment of reasons, a few of that looked like high-principled designing decisions. Accordingly the need emerges to reliably research the improvement of clones in certifiable code frameworks. Before refactoring is endeavored; a code designer or companion should endeavor to see the purpose for making the code clone before choosing what activity (if any) to take. To help these decisions, a rundown portraying basic employments of code clones should be built, just about like the lists acclimated depict style designs [13] or hostile to examples [12]. Then again, the present writing doesn't offer controlling for experts on an approach to oversee code clones in the event that they are to exist inside the framework.

The grammar of ANTLR is used for the algorithm. Since ANTLR gives a solitary reliable documentation to indicating lexers, parsers, and tree parsers because of which it is not difficult to utilize and subsequently, it is utilized here. From a sentence structure, ANTLR produces a parser that can assemble and stroll along the parse trees. ANTLR can be utilized to produce tree parsers. These are recognizers that process theoretical linguistic structure trees which can be consequently produced by parsers. These tree parsers are remarkable to ANTLR and

significantly streamline the preparing of unique punctuation trees. It can also deal with the direct left recursion in the grammar which can be represented as a grammar having any production of the type A-> AP where P is(v U T)* i.e. any combination from terminals and non-terminals of the grammar.
The rest of the paper is organized as follows. Proposed algorithm IS explained in section II. The experimental results are described in section III followed by conclusion in section IV.

II. PROPOSED ALGORITHM

This algorithm looks at the improvement of the purposeful or accidental cloned codes by contrasting the code portions in light of the fact that it happens in medium-sized to monster code frameworks. Most code frameworks contain a huge amount of code cloning; typically 10–15% of the ASCII content document in goliath code frameworks is a component of single or a great deal of code clones [16,17]. Code clones is likewise presented as programming phrases connected with dialect or libraries, basic utilization of system or library APIs, or perhaps executions upheld normal samples. Essentially, not all duplicate and glue exercises should be considered code natural exploration. Duplicate and-gluing inconsequential portions of code, in the same way as change of for circles or variable names, isn't normally considered code cloning, on the grounds that the resulting code sections typically impart almost no consideration getting phonetics content.

(A) ALGORITHM TO DETECT THE CHUNK OF CLONES

The working of algorithm will be as follows:

Firstly, it will select file or files to be operated on and will store the selected files in the array and for all the files in the array it will get the java source. Now, initialize the Collector that will collect the tokens identifiers and other duplicate code with, the files to be operated upon and their configuration. With the help of the initialized Collector it will calculate chains by the initialized collector and refine the file list to remove the file that contain no duplicated chunk. Then it will find the project number of statement, N, and read each file from selected file and add no of statements in file to the total project number of statement, N. Then make a matrix of statements by using project number of statements, N. It will use the 'Collector' which had been initialized before, with the statementGroups and for each statement group compare one statement with the others in the same bucket.
Then find the different chains from the statements and make a sorted list of Chains. For each chain in Chains Display Chain and also print the "No of unique chains as" size of sorted list of Chains.
The matrix used in it for setting the values as true is a square matrix of size equal to no of statements in an item of the navigable collection and the matrix is updated only in the upper half portion i.e. above the upper right principal diagonal only with no operation on lower half of matrix.

(B) WORKING OF MODULES

The main aim of this research is to make a detector that can detect the duplicate chunk of code .
There are four modules: parser, categorizer, collector and displayer.

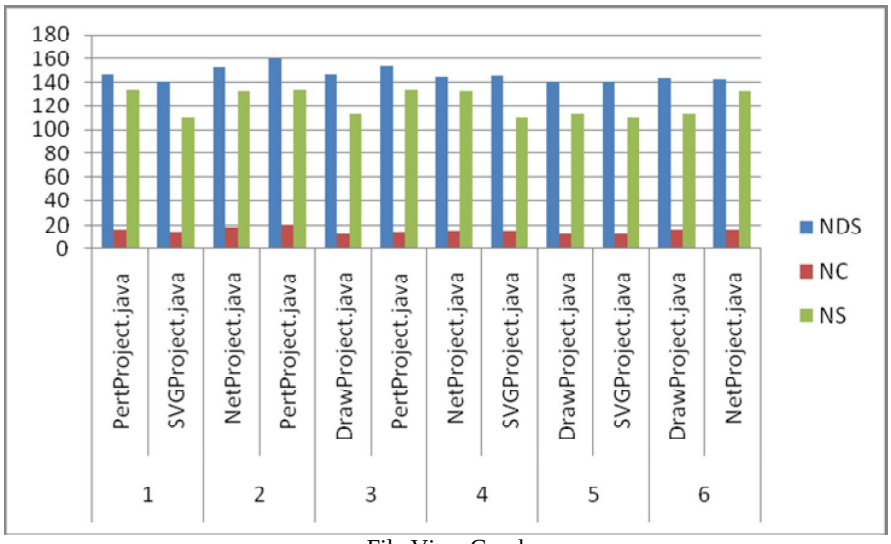
1. Parser module: This module is used for the parsing by using a predefined Plug-in ANTLR that uses the two grammar files {java15.g and java15.tree.g} from the given link : http://www.antlr.org/grammar/1090713067533/index.html in order to parse Java source code. Triggers are added in the java15.g in order to put back these decorating characters such as ';' , '{' , '}', etc on the Abstract Syntax Tree.
2. Categorizer module: This module makes an invert index of statements. Basically, a statement of language X is defined as any string generated from the grammar and is declared inside its grammar file . Therefore, the invert index of statements will be a map whose keys are statement type and whose values are list of statement of the same type. The invert index of statements is built while traversing the AST tree. This way reduces a lot of comparison computation to compare blindly one statement with all others and thereby reducing the number of false positives. This module also prepares the matrix by filling the value 0(false) or 1(true) according to two compared statements are different of the same. Makeinvertedlist: it is used to parse the file through which it make upper right triangular matrix for storing the unique code in it. And it resets the statements after making the matrix for previous statements group.

3. Collector module: This module walks along the diagonal matrix in order to build the list of unique chains that are present in the selected file. A unique chain is the list of sequence statements.

4. Displayer: This is the last and optional module for the project and it do not contribute to the functionality of the project. It only collect statistic number and display it via Swing for the better appearance.

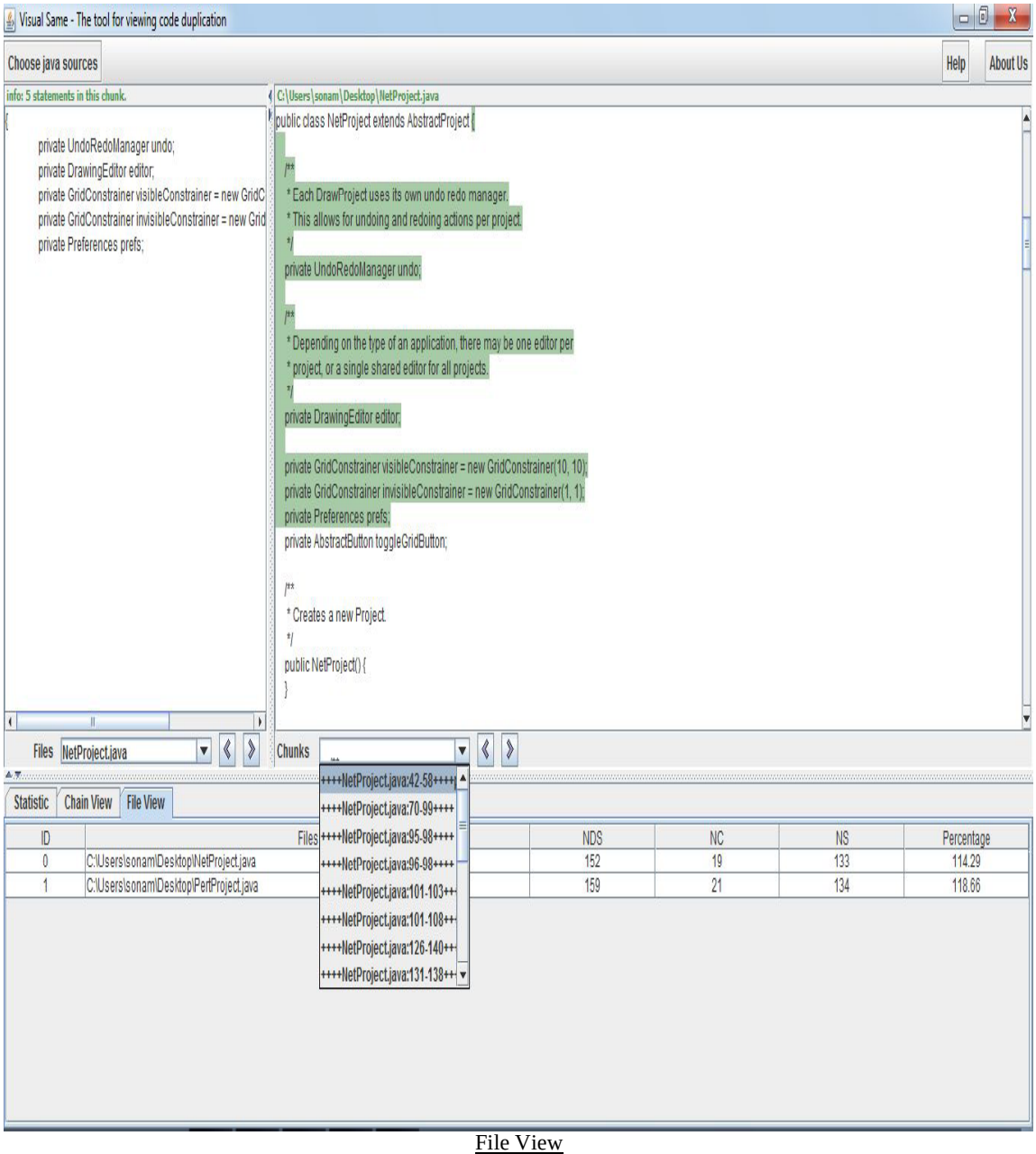
III. Experiment and Result

In this paper four modules of JHotDrawJava are compared by the help of the tool developed. The graph below shows the file view of clone detection of modules of JHotDrawJava.



File View Graph

The above graph clearly shows the results based on NDS, NC and NS, where NDS is the number of duplicated statements in the particular file (each statement can be counted more then once if it belongs to multiple chunks), NC is total number of chunks and NS is the total number of statements in the java file. Since this tool can detect clone in only maximum two files so the graph shows the comparison between two files at a time only. Likewise total 6 combinations of four java files were compared. By the help of this information the user can easily detect that whether the clones evolved are actually a threat to the maintenance or not. Nevertheless if the user finds the threat than he can go to a particular chunk as shown in figure below named as file view.



File View

The clones of these files when calculated manually were not much deviated as that calculated by the help of this tool and moreover the false positives are also very less in number. So, the precision rate of this tool is also very high.

IV.CONCLUSION

The clone detection tool developed in this research has been experimented on various modules of JHotDraw written in JAVA language. The results clearly show that the tool is able to detect all types of clones including non-contiguous clone. The algorithm is developed in such a manner that it is able to detect the cloned codes of maximum two files only. The advantage of this approach lies in the fact that the programmer can easily compare the two

versions of the file simultaneously. The tool has been basically developed for the programmer, so that he can detect the cloned codes at his level and can take the appropriate measures regarding it, i.e. if he founds that the detected cloned code is a bad smell then he will remove it else the good smells clones are well documented in the documentation , so as to reduce the efforts during maintenance.

The efficiency of the tool developed is very high as compared to other tools. The precision of various tools had been given in [10]. It clearly shows that except Clone DR all none of the tools have 100% efficiency, however the tool developed has 100% precision value, i.e. it detects all types of clones. The major difference between Clone DR and the tool developed is that the tool developed can be used to detect the clones of the whole system filewise whereas CloneDr cannot be used to detect all the clones of the system.

The approach used in this research also reduces the efforts at the maintenance site to a much extent as now the work to detect and remove the clones can be easily done by the programmer and moreover it will work to detect the cloned codes withing the same file or within two files.

REFERENCES

[1] Stephane Ducasse, Matthias Rieger, Serge Demeyer. A Language Independent Approach for Detecting Duplicated Code. In Proceedings of the 15th International Conference on Software Maintenance (ICSM'99), pp. 109-118, Oxford, England, September 1999.

[2] B. Baker, "On finding duplication and near-duplication in large software systems:, In *Proc. IEEE Working Conf. on Reverse Eng.*, pages 86–95, July 1995.

[3] Toshihiro Kamiya, Shinji Kusumoto, Katsuro Inoue. CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code. Transactions on Software Engineering, Vol. 28(7): 654- 670, July 2002.

[4] K. Kontogiannis, R. DeMori, E. Merlo, M. Galler, and M.Bernstein. Pattern Matching for Clone and Concept Detection. In Automated Software Engineering, Vol. 3(1-2):77- 108, June 1996.

[5] Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier, "Clone detection using abstract syntax trees", In *Int. Conf. on Software Maintenance*, pages 368–378, 1998.

[6] Jean Mayrand, Claude Leblanc, Ettore Merlo. Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics. In Proceedings of the 12th International Conference on Software Maintenance (ICSM'96), pp. 244-253, Monterey, CA, USA, November 1996.

[7] Howard Johnson, " *Substring Matching for Clone Detection and Change Tracking*", In Proceedings of the International Conference on Software Maintence (ICSM), pages 120-126, 1994.

[8] Ian Sommerville, *Software Engineering*, Addison-Wesley, 1996.

[9] Cory Kapser and Michael W. Godfrey. "Cloning Considered Harmful" Considered Harmful: Patterns of Cloning in Software. *Empirical Software Engineering*, Vol. 13(6):645–692 (2008).

[10] C.K. Roy and J.R. Cordy. *A Survey on Software Clone Detection Research.* Queen's School of Computing Technical Report:541, 115 pp., September 2007.

[11] Magdalena Balazinska, Ettore Merlo, Michel Dagenais, Bruno Lague, and Kostas Kontogiannis. Partial redesign of java software systems based on clone analysis. In *Proceedings of the Sixth Working Conference on Reverse Engineering (WCRE-99),6–8 October, 1999, Atlanta, Georgia, USA*, pages 326–336. IEEE Computer Society, 1999.

[12] William J. Brown, Raphael C. Malveau, Hays W. McCormick (III), and Thomas J. Mowbray. *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*. Wiley, Hoboken, NJ, 1st edition, 1998.

[13] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns:Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1st edition, 1995.

[14] Reto Geiger, Beat Fluri, Harald Gall, and Martin Pinzger. Relation of code clones and change couplings. In *Proceedings of the Ninth International Conference Fundamental Approaches to Software Engineering (FASE-06), 27–28 March, 2006, Vienna, Austria*, volume 3922 of *Lecture Notes in Computer Science*, pages 411–425. Springer, 2006.

[15] Yoshiki Higo, Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. Aries: Refactoring support environment based on code clone analysis. In *Proceedings of the Eighth IASTED International Conference on Software Engineering and Applications (SEA-04), 9–11 November, 2004, Cambridge, USA*, pages 222–229. ACTA Press, 2004.

[16] Cory Kapser and Michael W. Godfrey. Aiding comprehension of cloning through categorization. In *Proceedings of the 7th IEEE International Workshop on Principles of Software Evolution (IWPSE-04), 6-7 Sept. 2004, Kyoto, Japan*, pages 85–94. IEEE Computer Society, 2004.