International Journal of Advances in Engineering and Management (IJAEM)
Volume 5, Issue 5 May 2023, pp: 203-208 www.ijaem.net ISSN: 2395-5252

Studyof Refactored Code Cloneon Software Product Line Maintain ability

Prof. Rauki Yadav
Assistant Professor, Mahavir Swami College of Engineering and Technology,Vesu,Surat.

Date of Submission: 01-05-2023 Date of Acceptance: 08-05-2023

ABSTRACT: In order to reduce the cost of quality assessment in later phases of the development cycle, and to reduce the complexity of products over time, its early assessment is advocated. Early assessment technique comprises of studying the product and identifying code clones in them, removing the clones, a process called Refactoring, and identifying factors influencing various quality attributes. The current research focuses on the identification of Code Clones in SPL and effectively removing them by way of Refactoring. After the codes have been refactored, a correlation is established between the Maintainability and its sub-characteristics and a few metrics. Hence, the research objective was fulfilled of designing a prediction model to assess the impact of Refactored Codes on Maintainability.

KEYWORDS:Software Clone, Code Clone, Duplicated Code Detection, Clone Detection

I. INTRODUCTION

This paper focuses on defining research on associations between the cloning of code and maintaining quality software components as the main element of cloning code.

In the programming sector, numerous copying and usage operations take place in separate parts of the software which makes little or no modification. As a software clone, the process of replicating code is recorded and normal software cloning is used in this method. In instances where the maintenance of programming is catastrophic, clones may be changed or withdrawn.

The characteristics and relations of Code Clones with the SPL maintenance components were not investigated in this type of study. The research focuses on defining associations between the cloning of code and maintaining quality software components as the main element of cloning of code.

Step. 1: Pre-processing: This is starting phase in which entire inappropriate source code like whitespace and comments will be discarded. Production of source units: residual code will be separated in the dissimilar set of disjoint fragments identified as source components.

Step. 2: Transform: In this phase, source units got in past advance are changed over into selected middle structure which can be provided contribution to correlation algorithm. There is want of this progression in every one of methods aside from content-based strategy. It tends to be accomplished either utilizing standardization or extraction. Extraction is additionally subdivided into3 subcategories agreed underneath: Tokenization: Gathered source units as of past advance are changed over in tokens utilizing a few techniques or lexical conventions subsequent to eliminating remarks, deletes and so forth. Tokens are additionally organized into groupings. Parsing: Abstract Syntax Tree (AST) is produced by examining whole source code. AST is additionally partitioned into sub trees (Koschke et al., 2006). Those sub trees are thought-about to clone identification. Control & data flow analysis: A program Dependency Graph (PDG) graph is made with certain instruments in which control & data dependency is symbolized by edges & explanations by hubs. PDG sub graphs are thought-about to clone recognition.

Step. 3: Match Detection: In this progression, an appropriate match is created in contrasting yield of change stage, for example, changed units utilizing correlation calculation. The clones are spoken to as clone sets, family & classes. Addition trees & hashing are a portion of examination methods that can be utilized. In this phase, an appropriate match is found by looking at the yield of change stage, for example, changed units utilizing examination calculation. The clones are spoken to as clone sets, domestic& classes. The addition trees & hashing is a portion of examination methods that can be utilized.

Step. 4: Formatting: This stage is very unique in relation to the last stage in this stage, clone sets of

International Journal of Advances in Engineering and Management (IJAEM)
Volume 5, Issue 5 May 2023, pp: 203-208 www.ijaem.net ISSN: 2395-5252

changed code are recorded to unique source code utilizing document area.

Step. 5: Post-processing: Now this progression, the programmed heuristic or physical investigation is utilized to rank & sift through clones. Human specialists are utilized to sift through false positives via doing physical investigations. Heuristics dependent on decent variety, span, recurrence & different attributes of clones are utilized to consequently sift through or rank clone hopefuls.

Step. 6: Aggregation: It is the last advance of the clone identification procedure which incorporates legitimate information compression and investigation. The clone family & classes are shaped by joining distinguished clone sets..

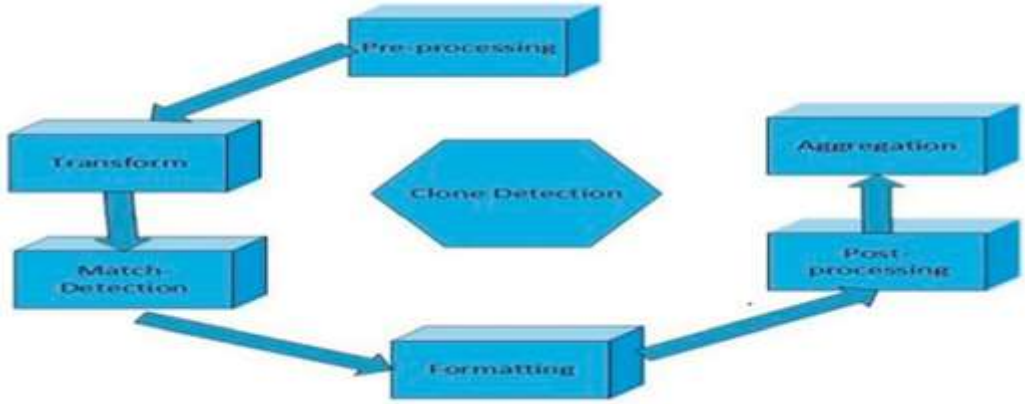

Figure1.0:StepsforCloneDetection

II METHODOLOGY

After establishing the relationship between Maintain ability and the effect of Code Clones, using the proposed model, the next step will be to remove those code clones using t h e Refactoring technique (with the help of existing Refactoring methods) which is a known technique for removing code clones. And lastly, once the code has been refactored, the impact of Maintainability and there factored code will be assessed.

III. EXPERIMENTATION

ExperimentNumber	ExperimentName	The objectiveoftheExperiment
Experiment1	CodeCloneMetricsforSPLs	Tostudythevariousmetricsrelatedtocodeclonesin SPLs
Experiment1.1	CodeClonemetrics relatedtoFiles	Tostudy file-relatedmetrics
Experiment1.2	CodeClonesmetrics relatedtocloneset	Studyingcloneset-relatedmetrics
Experiment1.3	CodeClonesrelatedto linebasedmetrics	Tostudylinebasedmetrics

Experiment1–Code Clone Metrics for SPLs

The commencement of our experiments is by making a detailed study of Software Product Line methodology, Code Cloning problem in SPL, identify the code clones in them and classifying them. This was done with the help of Code clone detection tool CCFinder. The graphs given in Fig 1.1 identifies clones based on Files, Clone set, Lines and Syntax and provides the respective metrics which is helpful in deriving correlation in the later stages

Experiment 1.1 – Code Clone metrics related to Files

The Fig 1.1 graphically depicts code clone metrics based on Files. It pictorially depicts the

International Journal of Advances in Engineering and Management (IJAEM)
Volume 5, Issue 5 May 2023, pp: 203-208 www.ijaem.net ISSN: 2395-5252

file-based code clone metrics in each of the SPL products taken for the experiment. From the graph, it is indicated that clones are present across different projects and within files of the same project. It helps to suggest clone fragments in files. It is observed that code similarity across the files are lower than similarity within a file. Thus these projects have clones within files.

Results- The graph classifies the clones according to files with the help of File-based metric. This helps in giving an idea of how clones are distributed across files, according to file set and file size.

Experiment 1.2 – Code Clone Metrics related to Clone Set

The below given Fig 1.2 pictorially depicts code clone metrics related to the clone set. The experiment shows the various clone set metrics spread across the projects for the experiments undertaken.

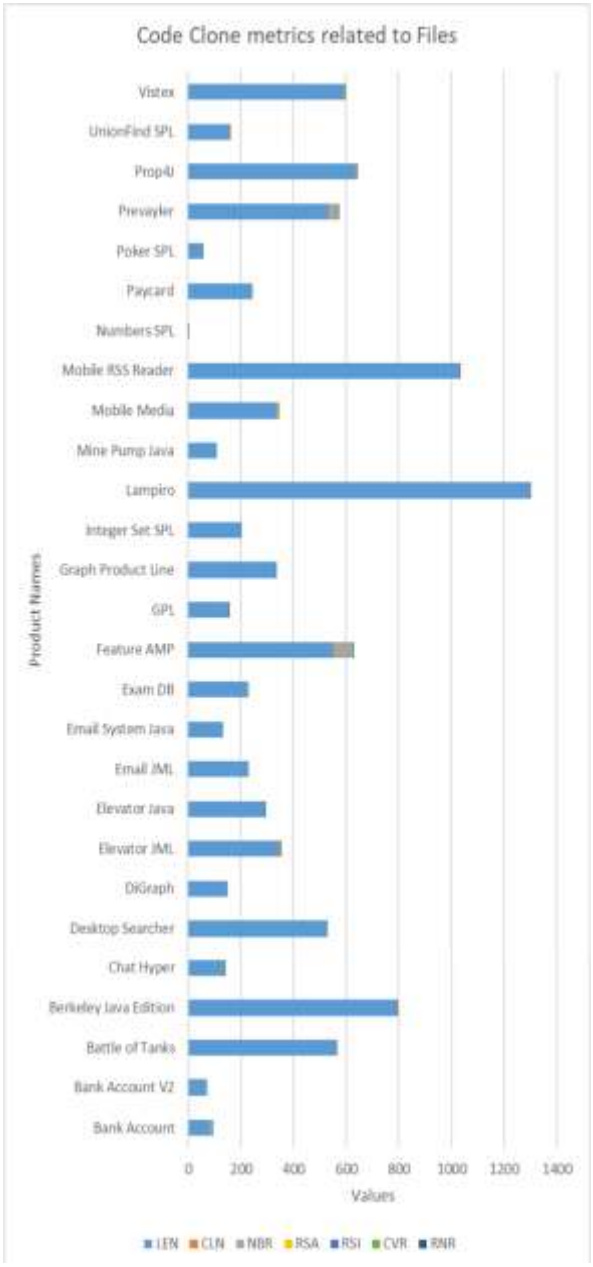

Fig1.1Code Clone metrics based on Files

International Journal of Advances in Engineering and Management (IJAEM)
Volume 5, Issue 5 May 2023, pp: 203-208 www.ijaem.net ISSN: 2395-5252

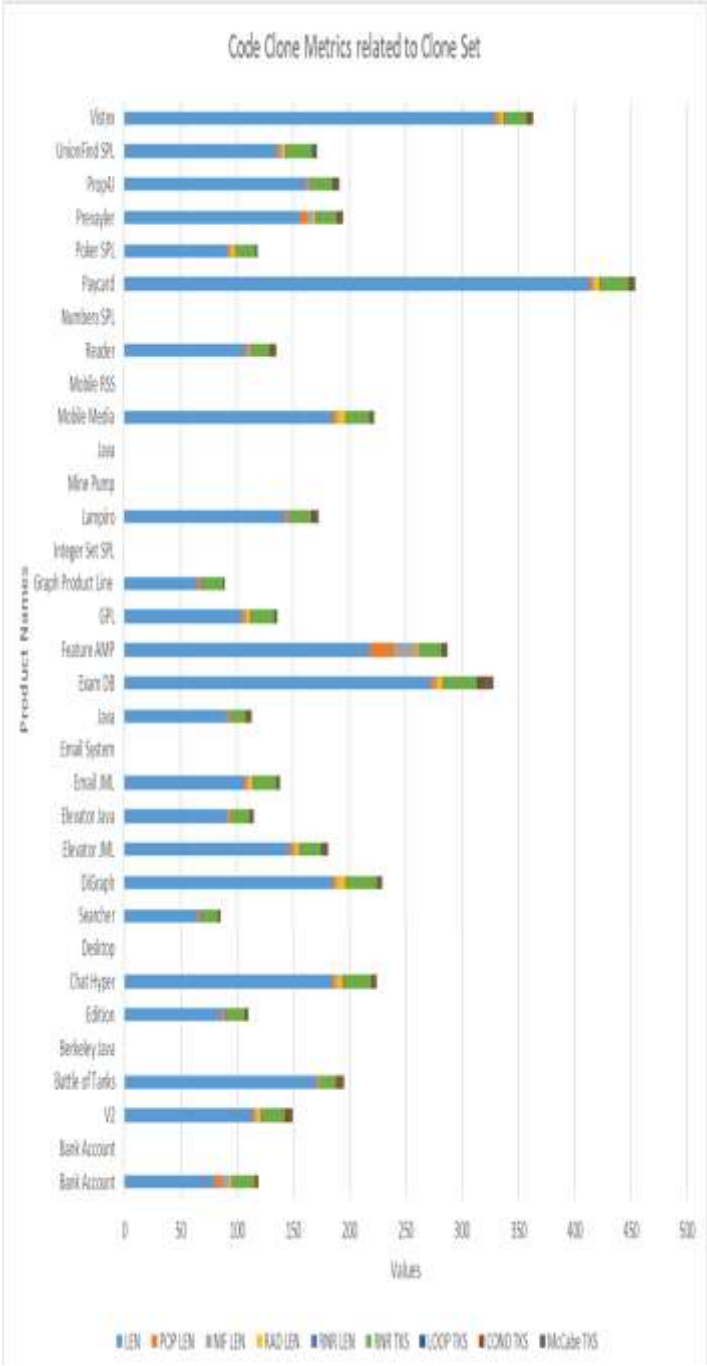

Fig1.2Clone set related metrics

From the above graph, it is indicated that the a graph gives a number of each of the identical or similar fragments in each of the files of SPLs. Code duplication considering token sequence, clones in loop statements, can be easily identified and also it indicates how presence of code clones in loop statements and other condition statements tends to increase the complexity of the overall code.

Results- Graph 1.2 classifies the clones according to Clone set-based metrics. This helps in giving an idea of how clones are distributed across products based on clone set metrics.

Observation: From the experiment conducted above, it is observed that code clones exist based on clone set which also includes decision and

International Journal of Advances in Engineering and Management (IJAEM)
Volume 5, Issue 5 May 2023, pp: 203-208 www.ijaem.net ISSN: 2395-5252

condition statements. On an average, there are less number of LOOP statements and less conditional statements.

Experiment1.3–Code clone metrics based on Lines

The below given Fig 1.3 graphically depicts code clone metrics based on Lines. It pictorially depicts the Line based code clone metrics in each of the SPL products taken for experiment.

From the above graph, it is indicated that the above graph depicts the clone concentration across products based on Line metrics. The total and average of each metric for each product has been calculated.

Results- Graph 1.3 classifies the clones according to Line based metrics. This helps in giving an idea of how clones are distributed across products based on clone set metrics. The above graph gives line-based clone values present in each of the selected SPL files. The clones present in LOC, including and excluding blank spaces, comments and the likes are classified in the above table. Also, the Total and average of each of the metric listed.

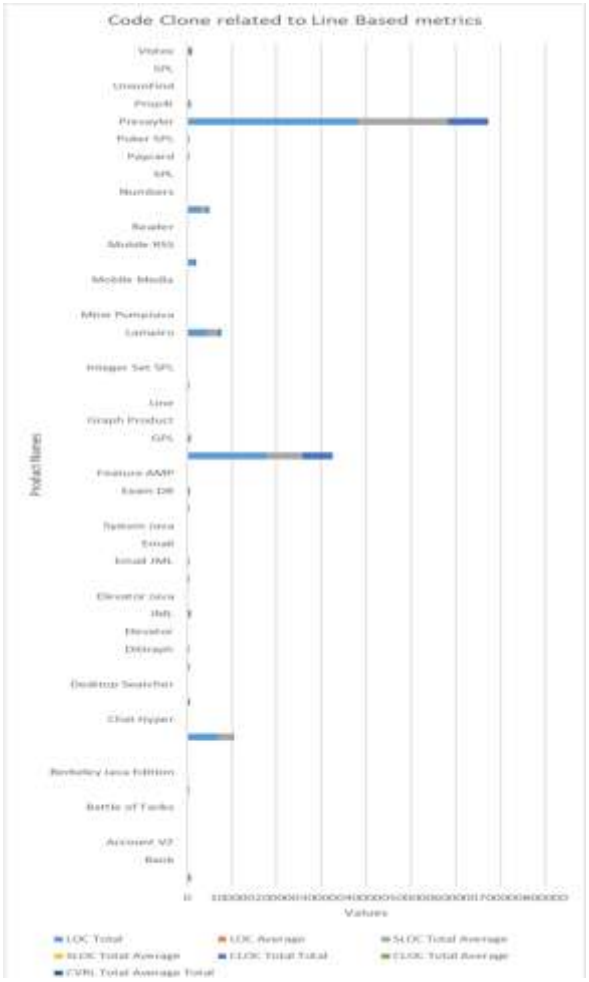

Fig 1.3 Line based metrics

IV. LIMITATION

This segment highlights the limitation of the current scope of study and also the scope for further research:

1)One drawback which could be stated is that the other external quality parameters were not taken into account.

2)In the current research, the limited number of SPLs were considered. In future, the scope of the investigation can be expanded and the outcomes can be predicted to be significant. Based on the variation of future experiments, the model can be refined to predict a tightly-knit trend.

3) More refactoring metrics can be formulated for a better understanding of the overall system in the long run.

V. CONCLUSION

Understanding the limited scope and nature of our study, based on the experimental analysis and results, and conclusions from the experiments conducted in the research undertaken, it was observed that SPL project suffers from code-cloning. They surely contain clones and it becomes important to remove them. Clones were distributed across products based on files, clone set, lines and syntax based metrics. It also included decision and condition statements and on an average, there were less number of LOOP statements and less conditional statements. It was observed from the experiments, that even in programs consisting of simple statements, clones did appear and presence of clones affected the overall program. It was also observed that the sub-characteristic of maintainability has various metrics and the values in the files are classified according to the metrics of Analyzability, Changeability, Testability, Stability and Maintainability which helped us in understanding which metric of which sub characteristic was more

The code clone metrics after refactoring has 70% of projects with lesser number of decision points and the refactored codes consisted of simple statements. The line-based metric values of projects, before and after refactoring showed that there is an overall decline in the values from the average value of the refactored project. It was observed that with the help of a new set of refactoring metrics, syntactical refactoring could be done and if for statements are used in a limited measure, the code will result in being less maintainable.

It was cumulatively observed that the for statement was the most frequently used and refactored. Maintainability values of the refactored projects were recorded and it was observed that changeability of the product was affected greatly. The least of the maintainable product, after refactoring, has been Integer Set and Union.

REFERENCES

[1]. Bosch J. and Bosch-Sijtsema P. M.. Introducing agile customer- centered development in a legacy software product line. Softw. Pract. Exper., 41(8):871–882, July 2011. Doi: 10.1002/spe.1063.

[2]. Bellon S., Koschke R., Antoniol G., Krinke J. and Merlo E. (2007). Comparison and Evaluation of Clone Detection Tools. in IEEE Transactions on Software Engineering, vol. 33, no. 9, pp. 577-591. doi: 10.1109/TSE.2007.70725.

[3]. Chen L., Babar M. A. and Ali N. (2009). Variability Management in Software Product Lines: A Systematic Review. In Proceedings of the 13th International Software Product Line Conference. Pittsburgh, Pa, pp. 81-90.

[4]. Fenske W., Meinicke J., Schulze S., Schulze S. and Saake G. (2017). Variant- Preserving Refactorings for Migrating Cloned Products to a Product Line. In International Conference on Software Analysis, Evolution and Reengineering. IEEE, pp. 316–326. doi: 10.1109/SANER.2017.7884632.

[5]. Jia Y., Binkley D., Harman M., Krinke J. and Matsushita M. (2009). KClone: A Proposed Approach to Fast Precise Code Clone Detection. computer science. pp. 1-5. JBoss Application Server.Retrieved from:http://www.jboss.org.

[6]. Jia Y., Binkley D., Harman M., Krinke J. and Matsushita M. (2009). KClone: A Proposed Approach to Fast Precise Code Clone Detection. computer science. pp. 1-5. JBoss Application Server.Retrieved from:http://www.jboss.org.

[7]. Kodhai. E, Perumal. A, and Kanmani. S. (2010). Clone Detection using Textual and Metric Analysis to figure out all Types of Clones. in International Journal of Computer Communication and Information System (IJCCIS). Vol2. No1, pp. 99-103. ISSN: 0976–1349.

[8]. Koschke R., Falke R. and Frenzel P. (2006). Clone detection using abstract syntax suffix trees. WCRE'06. Working Conference on reverse engineering IEEE(13): 253-262. doi: 10.1109/WCRE.2006.18.

[9]. Koschke R., Frenzel P., Breu A. P. J. and Angstmann K. (2009). Extending the Reflexion Method for Consolidating Software Variants into Product Lines. Software Quality Journal. Vol. 17, no. 4, pp. 331–366. DOI: https://doi.org/10.1007/s11219-009-9077-8.