ADV MATH SCI JOURNAL
Advances in Mathematics: Scientific Journal **9** (2020), no.6, 4001–4013
ISSN: 1857-8365 (printed); 1857-8438 (electronic)
https://doi.org/10.37418/amsj.9.6.81 Spec Issiue on ICAML-2020

FUNCTION CLONE REMOVAL USING REFACTORING TECHNIQUES

H. KAURR[1] AND R. MAINI

ABSTRACT. Refactoring is an activity and a technique to improve maintenance, performance and quality of software. Several studies in literature shed light on code clone refactoring to support software maintenance. This paper illustrats the experience on clone refactoring with tools cloneDR and JDeodorant. We investigated the impact of clone refactoring on detected clones in JhotDraw variants (5.2, 6.0b1and 7.0.6) and JEdit-4.2 software. Clone detection has been performed using cloneDR tool. Impact of refactoring has been observed before and after refactoring on detected clones. Code clones have been refactored using ExtractMethod, ExtractSuperClass and MoveMethod refactoring techniques. After refactoring clones are reduced in JhotDraw 5.2, JhotDraw 6.0b1, JhotDraw 7.0.6 and Jedit-4.2. SLOC in all software systems has also reduced to a certain amount.

1. INTRODUCTION

The process that deals with improvement of source code without affecting the external functionality of a program is known as refactoring. The term refactoring has been originally confronted in PhD dissertation of William Opdyke, [20]. Software evolution and enhancements in real environment, modification and adaptation to new emerging requirements, all these issues increases the complexity of code and the program then drifts from its primary design, results in lowering the software quality. Due to this, majority of software development

[1]*corresponding author*

2010 *Mathematics Subject Classification.* 68-04, 68N01.

Key words and phrases. code clone, function clone, refactoring.

cost is devoted to software maintenance. To manage this maintenance issue, there is an utmost need of methods and techniques that can lower down complexity of software by continually improving the internal software quality. The research area that addresses this issue is introduced as restructuring, [2, 10] or, it is also known as refactoring, [6, 10]. In software programs, programmers replicate several code fragments for the software reuse. Usually, this type of replication is known as code clones. Code clones also mean "duplicate code", which is a kind of bad smell as defined by [5]. Bad smells create many problems in source code such as duplicates of code portions, maintenance cost and bug propagation. Therefore, there is necessity to manage these code clones to improve the quality and structure of code. Refactoring is an activity to restructure the source code to remove code clones, [9]. In current research work, Clones have been detected through cloneDR. Afterwards, refactoring of detected clones will give view of any improvements to reduce the maintenance efforts of software. Contribution of experimentation is as follows: • Code clone detection has been performed using cloneDR (Clone Doctor) in three versions of open source software JhotDraw (5.2, 6.0b1 and 7.0.6) and JEdit-4.2. Results reflected number of clones in selected input systems. • Analysis of refactoring of code clones on detected clones has been assessed. For the purpose of refactoring JDeodorant 5.0 plug-in of Eclipse has been used. Tremendous research has been carried out to refactor code fragments, either these fragments can be functions or statements, [3, 5, 8, 14, 16]. Some research has also been carried out that returns function/method level clones [15, 17, 21]. Unlike other techniques that detect clones with fixed number of lines, the techniques which identify function/method level clones are suitable to architectural refactoring as they represent a meaningful segment and most reusable part of code. Various tools have been proposed in the literature for refactoring support, [11, 12, 18]. Because function/method clones are meaningful, so these are useful for software maintenance and evolution phases. Hence, it boosted the researcher/scientists to choose the granularity as function/method level, [15] and [17]. Main objective of current work is also to decrease clones in source code and to reduce size of source code in terms of SLOC (Source Lines of Code) to improve readability, structure, performance and maintainability based on function granularity.

TABLE 1. Software Systems under Investigation

System	Size	Files	LOC	Classes	Total Functions
JHotdraw 5.2	5.72 MB	160	14,577	208	1037
JHotdraw 6.0 b1	14 MB	484	21,091	405	4256
JHotdraw 7.0.6	7.0 MB	310	32,447	350	2811
JEdit 4.2	25.5	571	132926	426	5418

The paper is organized through the following sections. Overall methodology is represented in section 2. Section 3 briefly introduces the selected clone detection and refactoring tool. In Section 3, the results on clone detection for selected versions of two open source systems have been represented. In Section 4, the impact of clone refactoring on detected clones has been analyzed. Finally, Section 5 concludes the research work.

2. RESEARCH METHOD

First, LOC (lines of code), number of classes and total number of functions in open source softwares under consideration have been accessed. Function Clone detection has been performed using automated tool cloneDR and results are fed to next phase for clone refactoring using JDeodorant plug-in. Overall steps are shown in figure 1. Detected clones are then analyzed, if can't be refactored, it is left as un-factored, otherwise suitable refactoring technique is applied. To assess the impact of refactoring, clones are re-detected from the same input (open source software here) after applying refactoring methods.

2.1. **Input System Selection.**

Open source softwares JhotDraw, [22], (5.2, 6.0b1 and 7.0.6) and JEdit 4.2 have been considered for experimentation and have been widely used by researchers [1, 2, 13–15, 23]. Table 1 shows details of all the chosen systems for experimentation.

As the number of detected functions of JHotdraw versions 5.2, 6.0b1, 7.0.6 and JEdit 4.2 ranges 1037 to 5418. Therefore, it is very challengeable task to detect function clones in such huge software. Stefan Bellon faced challenge to develop an oracle for huge Source Lines of Code (SLOC) and manually evaluated only 2% of total source code. Same investigation approach of Stefan Bellon in [4]has been followed in current work also. Experimentation has been performed

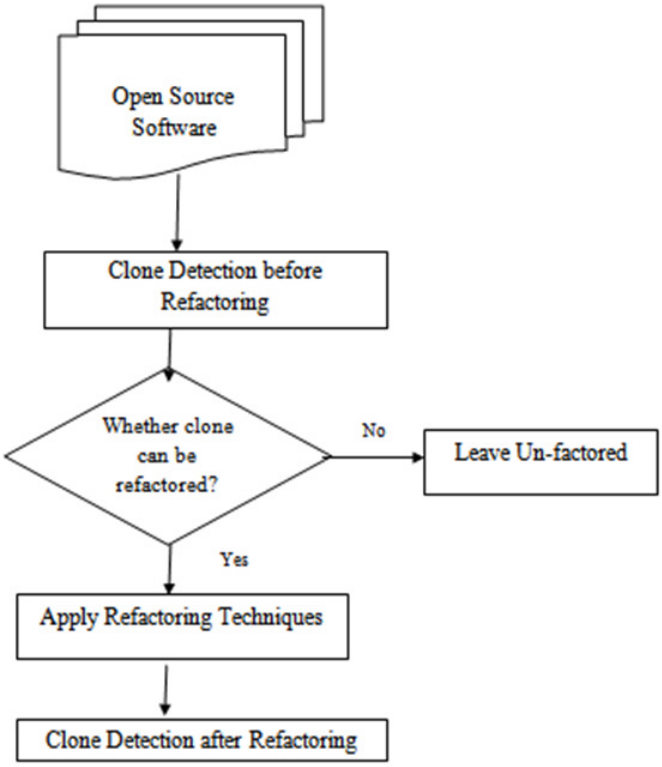

FIGURE 1. Refactoring Methodology

TABLE 2. Functions with Varying LOC (Lines of Code)

System	Functions (Lines of Code)			
	Minimum	Average	Above Average	Maximum
JhotDraw 5.2	2	6	>15	35
JhotDraw 6.0b1	2	6	>25	195
JhotDraw 7.0.6	2	8	>32	235
JEdit 4.2	4	8	>25	72

by considering only 10% detected functions of input systems to compute results. Functions with varying LOC have been considered as represented in Table 2.

3. TOOL FOR CLONE DETECTION AND REFACTORING

In this section, clone detection tool and plug-in used for refactoring has been discussed.

TABLE 3. *SLOC* and *Clones reported* before refactoring (first 10 samples)

Software System	**Total Function Clones Detected**	**Function clones before refactoring (in first 10 samples)**	**SLOC in clones %**
JhotDraw 5.2	122	18	10.6
JhotDraw 6.0b1	498	19	12.6
JhotDraw 7.0.6	309	21	22.4
JEdit 4.2	140	11	8.27

3.1. **CloneDR and JDeodorant Plug-in.** CloneDR is an automated tool to identify exactly similar and nearly-similar code portions in large software systems. Baxter et al. discussed all technical background of cloneDR, the author used a tree-based code clone detection technique. Various methods exist in literature to detect code clones, [1, 7, 19, 23]. In present work, clones are identified using cloneDR tool. The reported results: total clones detected, function clones and SLOC (source Lines of Code) in clones are shown in Table 3. It is not possible to show all results in this paper, therefore only first 10 samples reported by cloneDR are considered here for refactoring purposes. JDeodorant 5.0 is a plug-in to Eclipse used for refactoring purpose. In current work, it has been used to refactor detected clones. A pairs of duplicated code fragments or duplicated methods are fed as input to JDeodorant and it determines if they can be safely refactored. When two input clones have an identical structure, they will be treated for refactoring opportunity. JDeodorant pictures the two clone fragments/methods side-by-side to catch differences in those fragments, [18].

4. REMOVING FUNCTION CLONES USING REFACTORING TECHNIQUES

In this section, function clone refactoring techniques have been applied to reduce function clones in the software systems under consideration. Results of cloneDR for JHotdraw and JEdit have been imported in Eclipse. Following subsection discusses refactoring methods applied on similar functions.

4.1. **Extract SuperClass.** When two classes perform similar tasks, then Extract Superclass is applied to classes that already have a super class. Extract Superclass creates a new Superclass on the basis of existing class or the original class

can be renamed to create new Superclass. Extract Superclass is also applicable to duplicate functions/methods. Steps applied in JDeodorant:

(1) Open the clone groups with similar instances in the JDeodorant editor.
(2) Select Refactor or right click on the selected the duplicate highlighted code to select Refactor option.
(3) Select option Extract—- Select option Superclass.
(4) In the opened dialog, mention a name for super class and class members that are to be included to form new superclass Click Refactor. (Figure 2 represents ExtractSuperclass refactoring)

4.2. **Extract Method.** ExtractMethod is applied to those code fragments that can be grouped together and move those in a new method. After refactoring, duplicated code is replaced with a caller statement of the new method. Clone Group 32 (Type-1 clone) (Extract Method) New method name: contentRemovedExtracted(). Steps applied in JDeodorant: Select option Extract—-Method—Refactor. Figure 3 represents ExtractMethod refactoring.

4.3. **Move Method.** When a function/method has been used more in one class as compared to another. That method is then moved to more usable class. For example in figure 4, method:handleKey(KeyEventTranslator.Key) is moved from class:KeyEventWorkaround to class:DefaultInputHanldler.

5. Results and Discussion

The motive of this research work is to assess the impact of refactoring on code clones. Function clones are accessed using cloneDR before and after refactoring. As described earlier, number of clones reported before refactoring are shown in Table 3. After implementing refactoring methods, function clones are accessed once more by running cloneDR. The results reported reduced number of clones in selected systems under investigation. Table 4 shows refactoring opportunities possible on the chosen systems. Last column of the table shows number of non-refactorable cases. Table 5 shows the number of clone reported after refactoring.

As evaluation has been performed at 10% level, total number of functions in all chosen input systems was 253,725, 481 and 295 in JHotDraw (5.2, 6.0b1, 7.0.6) and JEdit 4.2 respectively. Percentage of cloning has been computed

FIGURE 2. Refactoring using *ExtractSuperclass*

before and after refactoring in selected versions. Figure 5 depicts decrease in percentage of cloning after applying possible refactoring opportunities.

It has been observed that percentage of cloning has been decreased from 48.22 to 41.11, 68.88 to 66.20, 64.24 to 62.57 and 47.45 to 42.37 percent in JHotDraw (5.2, 6.0b1, 7.0.6) and JEdit 4.2 respectively. SLOC has also been

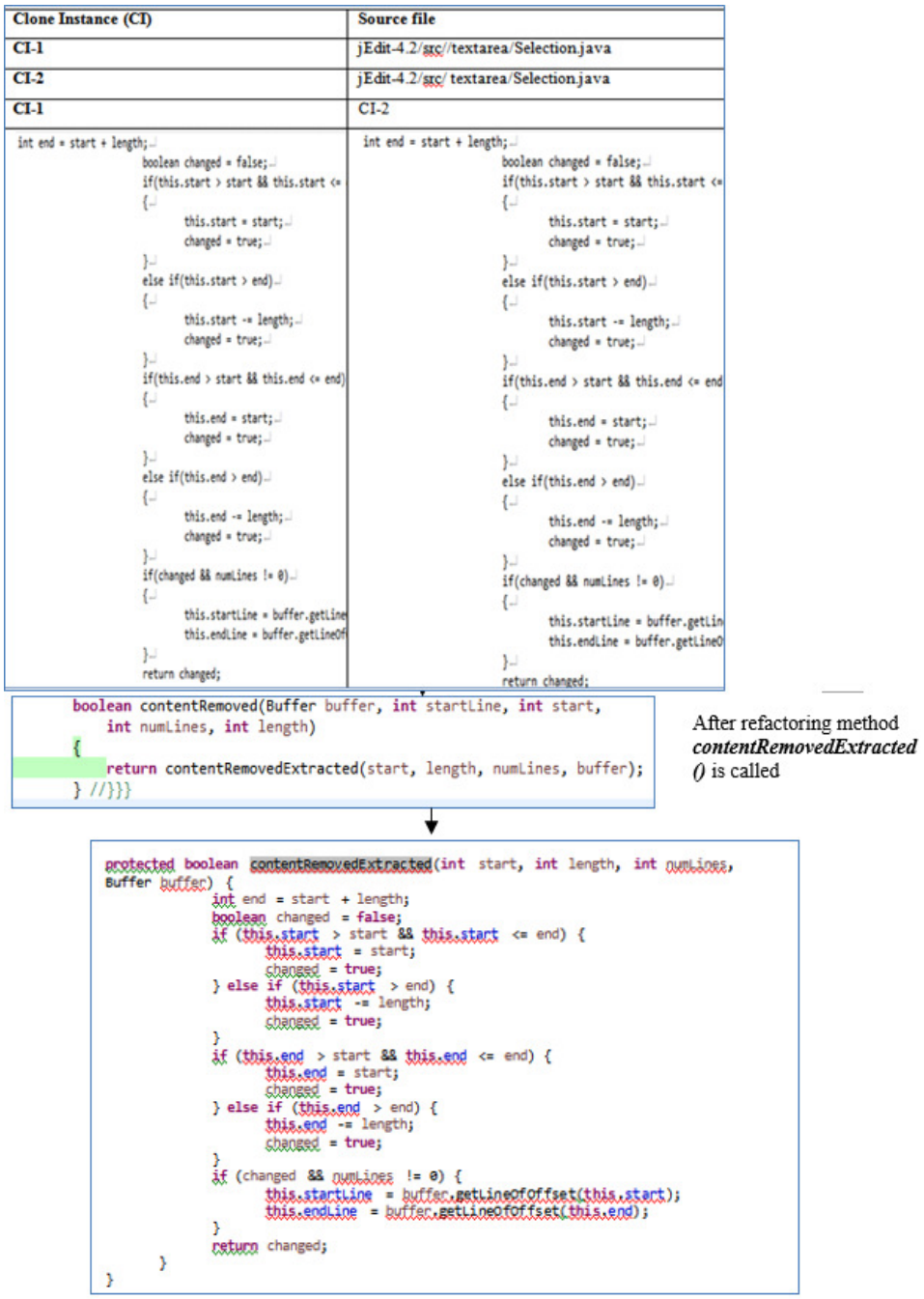

FIGURE 3. Refactoring using *ExtractMethod*

accessed using CloneDR before and after refactoring. Decrease in SLOC % has been reported after refactoring as plotted in Figure 6.

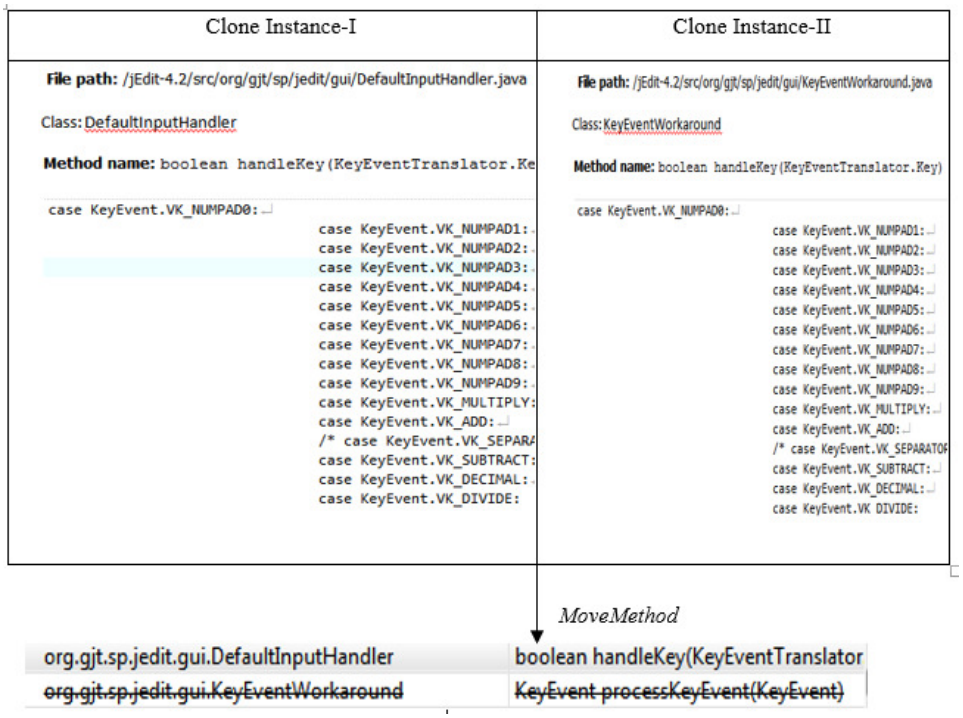

FIGURE 4. Refactoring using Move Method

TABLE 4. Number of Refactoring Opportunities

Software System	Extract Method	Extract Super class	Move Method	Non Refactorable Clones
JhotDraw 5.2	3	3	2	6
JhotDraw 6.0b1	2	3	3	5
JhotDraw 7.0.6	4	2	2	9
JEdit 4.2	3	1	1	4

6. CONCLUSION

In this current experience of clone detection using cloneDR tool has been demonstrated by analyzing open source systems. Impact of refactoring on function clones has also been assessed using JDeodorant 5.0 plug-in of Eclipse Mars. Different refactoring opportunities have been assessed. Experimentation has been performed on 4 open source projects and it has been noticed that number of function clones has been reduced from 122 to 104, 498 to 480, 309 to 301

TABLE 5. *SLOC* and *Clones reported* after refactoring

Software System	Total Function Clones Detected	Function clones after refactoring (in first 10 samples)	SLOC in %
JhotDraw 5.2	104	10	10.3
JhotDraw 6.0b1	480	11	12.2
JhotDraw 7.0.6	301	13	22.1
JEdit 4.2	125	6	8.11

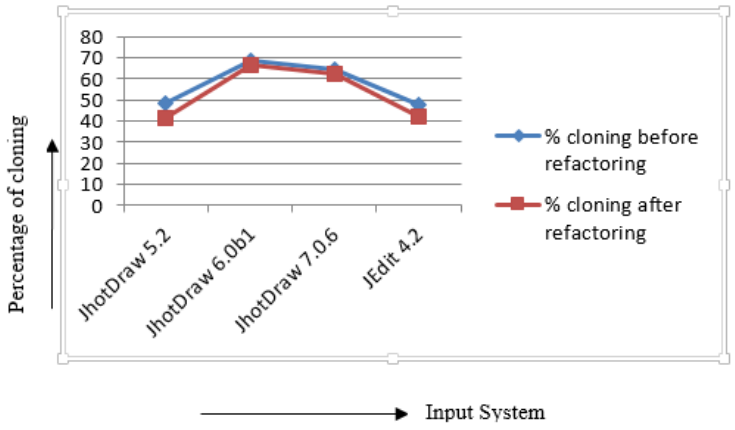

FIGURE 5. Percentage of Cloning before and after Refactoring

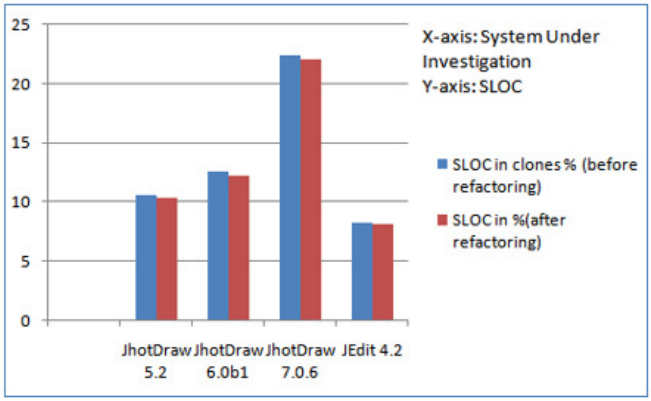

FIGURE 6. Comparison of SLOC before and after Refactoring

and 140 to 125 in JhotDraw 5.2, JhotDraw 6.0b1, JhotDraw 7.0.6 and Jedit-4.2 respectively, after implementing refactoring opportunities. Another parameter, SLOC has also been decreased from 10.6 to 10.3, 12.6 to 12.2, 22.4 to 22.1 and 8.27 to 8.11 percent in JHotDraw (5.2, 6.0b1, 7.0.6) and JEdit 4.2 respectively, which shows the reduction in size of the software without affecting the functionality of the system.

REFERENCES

[1] M. Ó. CINNÉIDE, L. TRATT, M. HARMAN, S. COUNSELL, I. H. MOGHADAM: *Experimental assessment of software metrics using automated refactoring*, Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement, (2012), 49–58.

[2] R. S. ARNOLD: *Tutorial on software restructuring*, IEEE Computer Society Press, 1986.

[3] M. BADRI, L. BADRI, O. HACHEMANE, A. OUELLET: *Measuring the effect of clone refactoring on the size of unit test cases in object-oriented software: an empirical study*, Innovations in Systems and Software Engineering, **15**(2) (2019), 117–137.

[4] S. BELLON, R. KOSCHKE, G. ANTONIOL, J. KRINKE, E. MERLO: *Comparison and evaluation of clone detection tools*, IEEE Transactions on software engineering, **33**(9) (2007), 577-591.

[5] S. BOUKTIF, G. ANTONIOL, E. MERLO, M. NETELER, *A novel approach to optimize clone refactoring activity*, Proceedings of the 8th annual conference on Genetic and evolutionary computation, Washington, USA, (2006), 1885–1892.

[6] F. M. T. CALEFATO, F. LANUBILE: *Function clone detection in web applications: a semiautomated approach*, J. Web Eng., **3**(1) (2004), 3–21.

[7] X. CHEN, A. Y. WANG, E.TEMPERO: *A replication and reproduction of code clone detection studies*, Proceedings of the Thirty-Seventh Australasian Computer Science Conference, **147** (2014), 105–114.

[8] Z. CHEN, Y. KWON, M. SONG: *Clone refactoring inspection by summarizing clone refactorings and detecting inconsistent changes during software evolution*, Journal of Software: Evolution and Process, **30**(10) (2018), e1951.

[9] M. FOWLER, K. BECK, J. BRANT, W. OPDYKE, D. ROBERTS: *Refactoring: Improving the design of existing code addison-wesley professional*, First edition, Addison Wesley Longman, Inc., Berkeley, USA, 1999.

[10] W. G. GRISWOLD, D.NOTKIN: *Automated assistance for program restructuring*, ACM Transactions on Software Engineering and Methodology (TOSEM), **2**(3) (1993), 228–269.

[11] Y. HIGO, T. KUSUMOTO, S. INOUE, K. ARIES: *Refactoring support environment based on code clone analysis*, Proceedings of the 8th IASTED International Conference on Software Engineering and Applications, (2004), 222–229.

[12] D. JEMEROV: *Implementing refactorings in intellij idea*, Proceedings of the 2nd Workshop on Refactoring Tools, (2008), 1–2.

[13] M. A. A. KHAN, C. K. ROY, K. A. SCHNEIDER: *Active clones: Source code clones at runtime*, Electronic Communications of the EASST, **63**(2014), 1–18.

[14] E. KODHAI, S. KANMANI: *Method-level code clone modification using refactoring techniques for clone maintenance* Advanced Computing: An International Journal, **4**(2) (2013), 7–26.

[15] E. KODHAI, S. KANMANI: *Method-level code clone detection through lwh (light weight hybrid) approach*, Journal of Software Engineering Research and Development, **2**(1) (2014), 12.

[16] J. KRÜGER, M. AL-HAJJAJI, S. SCHULZE, G. SAAKE, T. LEICH: *Towards automated test refactoring for software product lines*, Proceedings of the 22nd International Systems and Software Product Line Conference, **1** (2018), 143–148.

[17] B. LAGUE, D. PROULX, J. MAYRAND, E. M. MERLO, J. HUDEPOHL, *Assessing the benefits of incorporating function clone detection in a development process*, Proceedings International Conference on Software Maintenance, (1997), 314–321.

[18] D. MAZINANIAN, N. TSANTALIS, R. STEIN, Z. VALENTA: *Jdeodorant: clone refactoring* Proceedings of the 38th International Conference on Software Engineering Companion, (2016), 613–616.

[19] A. F. MUBARAK-ALI, S. M. SYED-MOHAMAD, S. SULAIMAN: *Enhancing generic pipeline model for code clone detection using divide and conquer approach*, Int. Arab J. Inf. Technol., **12**(5) (2015), 510–517.

[20] W. F. OPDYKE: *Refactoring: A program restructuring aid in designing object-oriented application frameworks*, PhD Thesis, 1992.

[21] W. F. OPDYKE: Refactoring object-oriented frameworks, University of Illinois at Urbana-Champaign, Department of Computer Science, 1992.

[22] J. SAVOLSKYTE: *Review of the jhotdraw framework*, Information and Media Technologies Matriculation No. 20668, 2004.

[23] M. WHITE, M. TUFANO, C. VENDOME, D.POSHYVANYK: *Deep learning code fragments for code clone detection*, Proceeding of 31st IEEE/ACM International Conference on Automated Software Engineering (ASE), (2016), 87–98.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
PUNJABI UNIVERSITY PATIALA
PUNJAB, INDIA
Email address: khasria.harpreet@gmail.com

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
PUNJABI UNIVERSITY PATIALA
PUNJAB, INDIA
Email address: research.raman@gmail.com