Detecting Model Clones using Design Metrics

Ritu Garg, R. K. Singh

Abstract: ***The cloning in software is a frequent phenomenon that leaves a negative impact among the product lines or the version control where the developer is involved in the evolution of software system due to any enhancement or changing requirements that leads to a release of new version. With the advent of MDD, identification of clones shifted from code to models to tackle risks at early stages. Due to the renaming of model elements, some model clones are missed that reports secondary clones instead of primary. So, in order to increase recall of clones in models we have proposed a hybrid approach based on the tree, lexical and metric approaches and validated it using SDMetric tool followed by analysis of detection of exact, renamed clones and modified clones. It provides one to one mapping in the form of corresponding primary clones with maximal matching that helps to reduce the domain for comparison of code at the implementation level. Such clones need to be identified among the versions and the stable part with least changes acts as a pattern and can be reused by the product lines.***

Keywords: Cloning Model Clones Reengineering Software Evolution Software Maintenance.

I. INTRODUCTION

Fowler considered cloning as a bad smell during software development [1]. In literature, various approaches for detection of clones using code are proposed. With the use of Model Driven Engineering approach, the detection of clones can be done in early stages of the Software Development Life Cycle [2]. Model is an abstract representation of the system within a particular context from a specific point of view. Alternately, models are the collection of logical entities, which describes a system at various levels of abstraction through class diagrams, collaboration diagram, state charts, etc. [7]. For example, the class diagram has logical elements in the form of classes and relationship between the classes. Recursively the class has model elements in the form of attributes and operations. Renaming of model elements and large domain of comparison of model fragments poses difficulty in the process of clone detection. Clones or redundancies in the form of model elements can be detected in models using various approaches such as lexical, token, tree, metric, graph, etc. In literature, these approaches have been used in isolation or in combination for clone identification [2] In this paper we have used the hybrid approach using a tree, metric and lexical approaches for detection of model clones. This approach increases recall and reduce the domain of comparison by providing the one to one mapping of model elements at the class level. The paper is organized as follows, section2 discusses Definitions and Terminology related to cloning. Section 3 classifies Model Clones followed by section 4 that elaborates our hybrid approach with different phases used in clone detection as pre-processing phase, Clone Analysis at stage-1 in models and Clone Analysis at stage-2 in models respectively. Section 5 concludes and represents the future scope of the paper.

II. DEFINITIONS AND TERMINOLOGY

Clones are represented as clone pairs or clone classes. Clone Pair represents a set of two clone instances of a particular granularity (block, function, class, package) that are similar to each other [4, 17]. Clone Class represents a set of all the clone instances of a particular granularity (block, function, class, package) that are similar to one another in such a way that a clone instance in the set can form clone pair with any other clone instance present in the same set [4]. Stability of clones can be predicted with the help of detection of SPCP clones in clone genealogy and lineage. SPCP clones are the clones during co-evlution that can preserve their similarity (being clone instance in same clone group in spite of difference in clone group among versions) are termed as Semantic Preserving Change Pattern clone [6]. Clone Genealogy represents the cloning relationship of a set of clone groups among multiple versions forming an evolution pattern [5]. A clone genealogy is live if it forms cloning relationship in subsequent versions. In case it doesn't form any clone relationship after that then such a genealogy is said to be dead. Clone Lineage represents cloning relationship of a clone group among multiple version as evolution history [5].

MagicDraw is used to draw the domain models and convert them into XML representation. SDMetric 2.33 tool (Demo version) released in April 2017, which allows comparison of two XML files and provides detailed statistics related to various metrics corresponding to the class, interface, package, interaction, use case, state machine, activity, component, node and diagram in the form of vectors [12]. This tool is used as a quality indicator by measuring design metrics from the perspective of fault-proneness, complexity, and effort prediction [19, 21].

III. CLASSIFICATIONS OF MODEL CLONES

Clones are the redundancies in the software in the form of similar fragments mainly due to copy and paste approach. "Model clone is a connected sub-model, which is structurally equivalent to another one, up to a certain threshold" [8]. Types of model clones are as under: -

Type 1: - Exact Model Clone –a model fragment that is identical except for layout, secondary notation, internal identifiers and notes [3,9].

Type 2: - Renamed Model Clone– a model that is identical except for changes of

Revised Manuscript Received on September 25, 2019
Ritu Garg, 1IGDTUW, Delhi-110006, India.
R. K. Singh, 1IGDTUW, Delhi-110006, India.

element names, attribute names and parts [3,9].
Type 3: - Modified Model Clone –a model that is identical other than the changes such as addition or removal of parts (sub model inside the ME) [3,9].
Type 4: - Semantic Model Clone – a model that is identical in functionality except for content, that may be due to model fragment copying, methodology or language constraints, convergent development or other processes [3,9].

3.1 Effect of Model clones in software evolution

Software evolution is a process of developing the software and then modifying the same as subsequent versions. This is done due to change in requirements, fast response time, fixing of bugs and defect and improving quality of software. Detecting clones helps in depicting the stability in different parts of the software in the form of classes, functions, packages, and blocks [10]. Based on the history of changes to the software among the various versions, one can identify the portions of the software that are more prone to changes and are thus highly risky from a maintenance point of view. More the change in the history of the versions, lesser will be the stability and lesser the change in the history of the versions, more will be the stability [6]. This can be achieved by identification of SPCP clones [6]. SPCP clones represent the change that is stable/consistent in nature and act as a refactoring candidate if change patterns are within same clone group. If change patterns are among the different clone groups, then it acts as tracking candidate. The clones that represent inconsistent changes are termed as non-SPCP clones as shown in fig-1. By SPCP and Non-SPCP clone analysis (dead or live genealogy) on change pattern among the various versions, one can proactively manage the clones in software development before the release of next version.

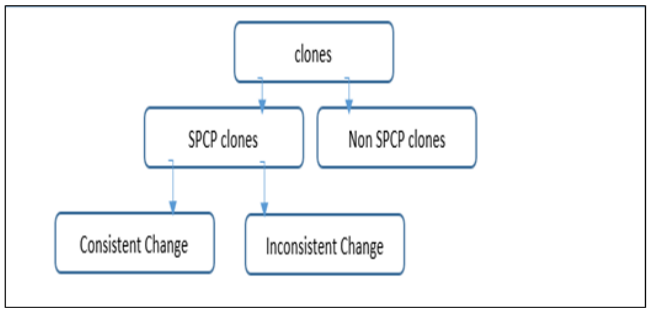

Fig. 1. Clones classification by similarity [6]

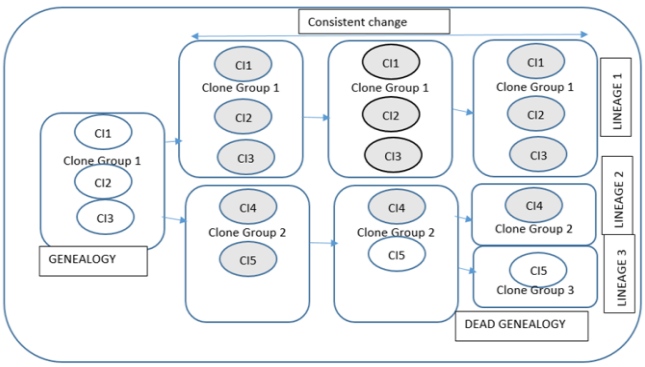

Fig. 2. Clone genealogy with three lineages

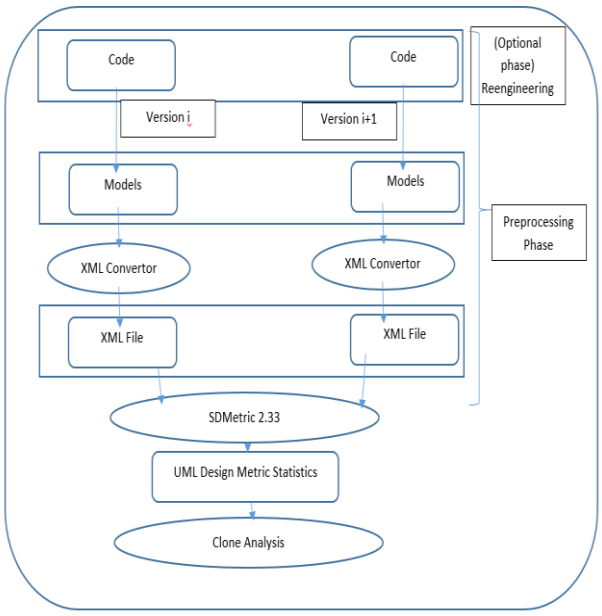

Fig. 3. The proposed approach is showing pre-processing, detection and analysis phase that will form the basis of clone management for model

IV. PROPOSED APPROACH

Detecting renaming in models taking into account the hierarchical structure with a reduction in a number of comparisons instead of pairwise matching of model elements is a very difficult task. We have used tree, lexical and metric approach to detect clones in domain models. This hybrid approach will help in generating feature vector using metric approach corresponding to each model element with textual similarity at class level in a tree structure. We have detected the first three types of clones defined by Störrle [3] using metrics as first stage analysis for the clone detection. It limits the comparison domain for the ME in the second stage using hybrid approach followed by fingerprinting technique and computation of similarity between the ME using cosine similarity [18]. It provides one to one mapping for comparison, thus reducing the domain of comparison code. The models consist of static and dynamic diagrams used in UML that are provided as input in which clones are to be detected at the class level. Such models may be

- Created and maintained by software designer as UML (Unified Modelling Language) models
- Created and maintained through reengineering of code as UML models

We have taken a case study generating three mutated versions of part of the base version named library management system in MagicDraw details of which are presented in table 1 with four diagrams consisting of a class diagram, state diagram, and two collaboration diagrams. There are four types of model clones, but our study is limited to type-1, 2 and 3 only. We used the same model as Base Version (BV) and seeded mutations to create three variants of the same software at class level in class diagram [14, 15]. One with type-1 clones; second with type-1, type-2 clones and third with type-1, type-2, type-3 clones named as Mutated Version 1 (MV1), Mutated Version 2 (MV2) and Mutated Version 3 (MV3) respectively. The

classification of mutations and number of mutations seeded in base version is presented in table 2 for all types of clones among mutated versions [14]. We have analyzed how clones can be detected in these versions using our approach in three phases

1. Preprocessing Phase

Clone Analysis at stage-1 in models

2. Clone Analysis at stage-2 in models

3.1 Preprocessing Phase

These models can't be processed as such but should be converted to XML to generate hierarchical representation using a tree structure. MagicDraw implicitly has the feature to save any project in the form of XML file that constitutes our pre-processing phase as shown in fig. 4(a). After that, the XML file is provided as input to SDMetric 2.33 tool.

3.2 Clone Analysis at stage-1 in models

For clone detection in models SDMetric, offers a comparison of two XML files. Selecting the type of the metric after selection of XML files, the design metrics or descriptive statistics as per selection is listed in work area, as the same for the base version is represented in fig. 5 where the values of parameters can act as a feature vector for the concerned model element. Fig. 6 shows class metrics related to the model comparison between the base version and mutated version. There are three possibilities

- A 'zero' value as an entry indicates no difference in the corresponding parameter (as Number of operations in the first column shown in decreasing order of effect of parameters).
- A 'positive' value as an entry indicates the given magnitude as the addition of corresponding parameter (Number of operations in the first column).
- A 'negative' value as an entry indicates the given magnitude as the deletion or leftover in corresponding parameter (as Number of operations in the first column).

These metrics design metric corresponding to diagram elements of various versions and comparison metrics between the versions are exported as database files from SDMetric tool which is read in the Microsoft Visual Studio.

Fig. 4. (a) XML file view for Base Version

Table 1. Description of Base Version and Mutation Versions

Versions	XML File Size (in KB)	Number of Diagrams	Number of mutated ME (Class level)
Base Version (BV)	362	4	0
Mutated Version 1 (MV1) with Type-1 clones	362	4	9
Mutated Version 2 (MV2) with Type-1 clones	366	4	19
Mutated Version 3 (MV3) with Type-1 clones	373	4	34

Algorithm
Input: AST for both versions
Output: Clone Pairs CP (ME, ME, L/F/C) as trident

1. For each model element (ME) at class level in tree
   a. Set all corresponding metrics as feature vectors FV []
   b. Generate fingerprints FP
   c. Set MEC=false
2. For each model element at class level in tree
   a. If lexical match found for ME
      i. Add both ME as clone pair to CP with L
   b. else if FP match found and cosine similarity =1 for ME
      ii. Add both ME as clone pair to CP with F
   c. else if Compute cosine similarity for ME
      iii. Add both ME as clone pair to CP with C
   d. If match found then set MEC=true for both ME
3. End for
4. List all the clone pairs

Fig. 4. (b) Algorithm for Clone Analysis

Table 2. Classification of mutation and number of mutations with respect to types of clone

Classification of Type-1 clones	Number of mutations	Classification of Type-2 clones	Number of mutations	Classification of Type-3 clones	Number of mutations
Change in position of ME within work area	2	Renaming class model element	2	Addition of class model element	1
Change in ordering of ME within containment structure	4	Renaming any attribute	3	Removal of class model element	0
Change in color of ME	2	Renaming any operation	2	Addition of attributes	3
Use of notes or comments	1	Changing data type for attributes	2	removal of attributes	5
		Changing return type for operations	1	Addition of operations	2
				removal of operations	3
				Addition of relationship	1
				Removal of relationship	0
Total mutations	9		10		15
Cumulative Mutations	9		19		34

Classes	NumOps	NumPubOps	NumAttr	MsgRecv	MsgSent	LLInst	EC_Attr	NumAssEl_ssc	NumAssEl_sb	Diags	Getters	MsgSelf	Setters	Nesting
book	7	7	8	4	3	3	3	1	1	1	0	0	0	0
librarian	17	17	2	6	6	1	1	4	4	1	1	1	0	0
transaction	6	6	5	3	3	3	2	1	1	1	0	0	0	0
member record	8	8	6	7	7	2	2	1	1	1	1	0	0	0
bill	5	5	4	2	2	1	1	1	1	1	0	1	0	0

Fig. 5. Few details of part of the software design named library management system that is created in magic draw

Classes	NumOps	NumPubOps	MsgSent	MsgRecv	NumAttr	NumAssEl_ssc	NumAssEl_sb	Getters	EC_Attr	LLInst	MsgSelf	Diags
book	1	1	0	0	0	0	0	0	0	0	0	0
librarian	-3	-3	0	0	1	0	0	0	0	0	0	0
transaction	0	0	0	0	-1	0	0	0	0	0	0	0
memberrecord	7	7	7	7	6	1	1	2	2	2	0	1
bills	5	5	2	2	4	3	3	0	1	1	1	1
store	0	0	0	0	1	1	1	0	0	0	0	1
member record	-8	-8	-7	-7	-6	-1	-1	-1	-2	-2	0	-1
bill	-5	-5	-2	-2	-4	-1	-1	0	-1	-1	-1	-1

Fig. 6. Few class metrics related to models created above and its variant (Type-3: modified clone)

3.2 Clone Analysis at stage-2 in models

At this stage database, files are read, and design metric corresponding to diagram elements of various versions is converted to fingerprints (aggregate of parameters in feature vector) for each model element at the class level. The different ME are compared to one another using the hybrid approach with cosine similarity as shown in fig 10 and 11. Also, the design metrics of model comparison can be extracted corresponding to all ME and compared using three approaches with the difference that the metric values are relative here other than absolute as depicted in the former case. The corresponding clones are listed as output in the form of one to one mapping as discussed above with any of the technique as mentioned above as presented in fig-4(b). The use of third entity in Trident (L/F/C) is for clone management when required. We did this stage analysis manually and in the process of developing the prototype corresponding to it. In case of exact clones, all the values of parameters should be zero for metric details as shown here in the fig. 7. We have shown the class level metric; similar is the case with other metrics. This is because there is no change in ME within the model, so the computation of metric values is same for both the models, thereby making the difference in the metric values as nil in the form of single row entry indicating clones that exist in both the software with the same name. Therefore, all corresponding five classes of primary clones with the same name are listed as clone pairs.

Type-2 clones or renamed clones are similar to type-1

clones except for the change in literals, identifiers, constants, sub ME name that exists within a model element (as class in case of class level metric) but if the name of the sub model element is modified then

- All values corresponding to the same class in one model is positive and in the other one as negative. However, both are same in magnitude (for feature vectors using Metric approach) as shown in fig. 8.
- The concept of Levenshtein distance (Textual similarity) is used to identify renaming by the hierarchical and containment relationship (Tree-based similarity) among versions.
- In case of descriptive statistics, we can even refer the differences in the delta, sum, mean and standard deviation, if their value is 'zero', then it depicts type-2 clones. Row-wise symmetric entries are clones of each other as shown in fig-9. In this case, all corresponding classes that follow above rules in addition to type-1 are reported as clone pairs.

In case of type-3 clones or modified clones, first of all, few conditions should be satisfied: -

- There is any non-zero entry for value of any parameter corresponding to any metric in the metric details section.
- The difference in the delta, sum, mean and standard deviation, their value is 'non zero' for at least one of the mentioned differences.
- The aggregate of metrics is considered for each primary element bound to a particular threshold value corresponding to base version to detect corresponding clones, which is set to 10% in order to differentiate the ME that are added or removed from the modified model element.

This is due to change in the model in the form of addition or deletion of a model element or change of containment ship of any model element within another model element as shown in fig. 6. More is the difference in the value of the metric parameter; more is the difference among the models, lesser will be the cloning. However, these are only relative values not absolute. Criteria of clone visualization are similar to that of exact and renamed clone. The class named store is not mapped to any other class in the previous version due to the higher difference that is above the threshold. Considering the number of changes required with respect to various class ME among the different versions is least in transaction ME, so it represents a stable clone and librarian ME being the most unstable in case renaming is considered. In case renaming is not considered then bill ME acts as the most unstable ME. Such unstable nature based on a number of changes helps in the further prediction of SPCP/ non-SPCP clones. Also if any of the class is removed from fig 7, then out of 5 ME only 4 will remain as clones in the form of active genealogy, and one of them will lead to dead genealogy. Using hierarchical matching with a tree structure, the comparison is done with branch and bound technique using the metric and lexical approach. This is done to identify renaming which is very common during the versioning. Example- if a class in the class diagram with name 'teacher' comprises of 10 ME is renamed to 'Teaching Staff" without any change in other ME.

Classes	NumAttr	NumOps	NumPubOps	Setters	Getters	Nesting	IFImpl	NOC	NumDesc	NumAnc	DIT	CLD	OpsInh	AttrInh
book	0	0	0	0	0	0	0	0	0	0	0	0	0	0
librarian	0	0	0	0	0	0	0	0	0	0	0	0	0	0
transaction	0	0	0	0	0	0	0	0	0	0	0	0	0	0
member record	0	0	0	0	0	0	0	0	0	0	0	0	0	0
bill	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Dep_Out	Dep_In	NumAssEl_ssc	NumAssEl_sb	NumAssEl_nsb	EC_Attr	IC_Attr	EC_Par	IC_Par	Connectors	InstSpec	LLInst	MsgSent	MsgRecv	MsgSelf	Diags
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fig. 7. All class metrics related to models created above and its variant (Type-1: exact clone)

Classes	NumOps	NumPubOps	MsgSent	MsgRecv	NumAttr	EC_Attr	LLInst	Getters	NumAssEl_ssc	NumAssEl_sb	MsgSelf	Diags	Setters
book	0	0	0	0	0	0	0	0	0	0	0	0	0
librarian	0	0	0	0	0	0	0	0	0	0	0	0	0
transaction	0	0	0	0	0	0	0	0	0	0	0	0	0
memberrecord	8	8	7	7	6	2	2	1	1	1	0	1	0
bills	5	5	2	2	4	1	1	0	1	1	1	1	0
member record	-8	-8	-7	-7	-6	-2	-2	-1	-1	-1	0	-1	0
bill	-5	-5	-2	-2	-4	-1	-1	0	-1	-1	-1	-1	0

Fig. 8. Few class metrics related to models created above and its variant (Type-2: Renamed clone)

parametrs	Posdeltas	Negdeltas	PosD_na	NegD_nd	Sum_1	Sum_2	Sum_D	N_1	N_2	N_D	Mean_1	Mean_2	Mean_D	StdDev_1	StdDev_2	StdDev_D
NumOps	13	13	0	0	43	43	0	10	10	0	4.3	4.3	0	5.558777	5.558777	0
NumPubOps	13	13	0	0	43	43	0	10	10	0	4.3	4.3	0	5.558777	5.558777	0
NumAttr	10	10	0	0	25	25	0	10	10	0	2.5	2.5	0	3.02765	3.02765	0
MsgSent	9	9	0	0	21	21	0	10	10	0	2.1	2.1	0	2.643651	2.643651	0
MsgRecv	9	9	0	0	22	22	0	10	10	0	2.2	2.2	0	2.699794	2.699794	0
EC_Attr	3	3	0	0	9	9	0	10	10	0	0.9	0.9	0	1.100505	1.100505	0
LLInst	3	3	0	0	10	10	0	10	10	0	1	1	0	1.247219	1.247219	0
NumAssEl_ssc	2	2	0	0	8	8	0	10	10	0	0.8	0.8	0	1.229273	1.229273	0
NumAssEl_sb	2	2	0	0	8	8	0	10	10	0	0.8	0.8	0	1.229273	1.229273	0
Diags	2	2	0	0	5	5	0	10	10	0	0.5	0.5	0	0.527046	0.527046	0
Getters	1	1	0	0	2	2	0	10	10	0	0.2	0.2	0	0.421637	0.421637	0
MsgSelf	1	1	0	0	2	2	0	10	10	0	0.2	0.2	0	0.421637	0.421637	0

Fig. 9. Few class descriptive metrics related to models created above and its variant (Type-2: Renamed clone)

$$similarity = \frac{\sum_{i=1}^{n} A_i * B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} * \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Fig. 10. Cosine similarity computation [20]

Similarity (member record, member record)=259/(√247*√274)
=259/(15.7*16.5)
=259/260.15
=99.55%

Fig. 11 Clone Class Pair reported-(member record, member record, C)

The lexical approach or lexical with levenshtein distance could not detect these corresponding classes as a clone, but our approach can detect it as primary clone instead of 10 secondary clones as a maximal matching model element. This will improve the recall and precision related to clone detection with improve in performance because of one to one mapping for code level. After that comparison of code at the implementation level can be extended, if required, instead of comparing every model element of one version to every other model element of another version.

V. CONCLUSION AND FUTURE SCOPE

Clone detection with structural similarity in the form of hybrid approach at higher granularity in models can tackle the risks earlier. It can provide most promising clones and reduces the comparison domain that makes it viable and quick approach other than clone detection in code. We have validated our approach by detecting type-1 and type-2 clones with higher accuracy and completeness. The concept of fingerprinting and cosine similarity will improve the performance than clone detection in code. This can assist in the prediction of risky portions of the software that frequently changes so that those clones should be on top priority and moreover, the resources are assigned accordingly from a maintenance point of view. This information is useful for clone management and supports software evolution and maintenance. The stable portions of the versions; if can exist as a standalone component then acts as a pattern and reused in other product line software.

It can detect clones at a higher granularity that is at class level in case of the class diagram, interaction in case of interaction or collaboration diagram, a state in case of state diagram, however lower level granularity is not supported. This approach can be used as a first step to filter out the most promising results in less time. It can also be used with other techniques to increase completeness and accuracy. Based on one to one mapping, the source codes can be further analyzed for better precision of the clone detection; however, we have limited our work to detection of one to one mapping. Moreover, the priority list of parameters should be accounted for creating feature vectors that can help in clone detection at the code level. In case of Type-3 clones, the threshold is user specific manual value, but it should automatically vary on the basis of nature of software automatically.

Mainly the parameters are structural in nature with emphasis on size and complexity-based parameters in metrics. It can be extended with semantic similarity. For scalable approach with better performance, the different weighing scheme should be applied to different ME. SPCP clones among the versions can be predicted to a greater extent that can help in the evolution of the software using this approach. With mutations mentioned above in table-2, manually we detected the clones with the shortcomings that difference in the type of relationship is not captured. Moreover, the study is based on mutation analysis in an academic environment but not an industrial one.

REFERENCES

[1] Fowler, Martin, Kent Beck, Refactoring: improving the design of existing code, Addison-Wesley Professional, 1999.
[2] D. Rattan, M. Singh, R. Bhatia, Design and development of an efficient software clone detection technique, Diss. 2015.
[3] H. Störrle, Effective and efficient model clone detection, Software, Services, and Systems, Springer International Publishing (2015) 440-457.
[4] C. K. Roy, James R. Cordy, A survey on software clone detection research, Queen's School of Computing TR 541.115 (2007) 64-68.
[5] Jeremy R. Pate, Robert Tairas, Nicholas A. Kraft. Clone evolution: a systematic review, Journal of software: Evolution and Process 25.3 (2013) 261-283.
[6] F. Mondal, Analyzing Clone Evolution for Identifying the Important Clones for Management. Diss. 2017.
[7] N. H. Pham, et al, Complete and accurate clone detection in graph-based models, Proceedings of the 31st International Conference on Software Engineering, IEEE Computer Society, 2009.
[8] F. Deissenboeck, et al., Model clone detection in practice., Proceedings of the 4th International Workshop on Software Clones. ACM, 2010.
[9] D. Rattan, R. Bhatia, M. Singh, Software clone detection: A systematic review, Information and Software Technology 55.7 (2013) 1165-1199.
[10] MdR Islam, Minhaz F. Zibran, A Comparative Study on Vulnerabilities in Categories of Clones and Non-Cloned Code, Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on. Vol. 3. IEEE, 2016.
[11] H. Störrle, Towards clone detection in UML domain models, Proceedings of the Fourth European Conference on Software Architecture: Companion Volume. ACM, 2010.
[12] Https://msdn.microsoft.com/en-us/library/dd409416.aspx.
[13] Http://www.sdmetrics.com/downman.html.
[14] M. Stephan, A mutation analysisbased model clone detector evaluation framework, Diss. 2014.
[15] M. Stephan, J. R. Cordy, Model Clone Detector Evaluation Using Mutation Analysis, ICSME (2014) 633-638.
[16] G. Mahajan, M. Bharti, Implementing a 3-way approach of clone detection and removal using PC Detector tool, Advance Computing Conference (IACC), 2014 IEEE International IEEE 2014.
[17] U. Mansoor, Handling High-Level Model Changes Using Search Based Software Engineering, (2017).
[18] I. Keivanloo, Source Code Similarity and Clone Search. Diss. Concordia University, 2013.
[19] P. V. Kavita, an analysis of OO design quality measurement tool for UML
[20] Fowler, Martin, Kent Beck, Refactoring: improving the design of existing code, Addison-Wesley Professional, 1999.
[21] D. Rattan, M. Singh, R. Bhatia, Design and development of an efficient software clone detection technique, Diss. 2015.
[22] H. Störrle, Effective and efficient model clone detection, Software, Services, and Systems, Springer International Publishing (2015) 440-457.
[23] C. K. Roy, James R. Cordy, A survey on software clone detection research, Queen's School of Computing TR 541.115 (2007) 64-68.
[24] Jeremy R. Pate, Robert Tairas, Nicholas A. Kraft. Clone evolution: a systematic review, Journal of software: Evolution and Process 25.3 (2013) 261-283.
[25] F. Mondal, Analyzing Clone Evolution for Identifying the Important Clones for Management. Diss. 2017.
[26] N. H. Pham, et al, Complete and accurate clone detection in graph-based models, Proceedings of the 31st International Conference on Software Engineering, IEEE Computer Society, 2009.
[27] F. Deissenboeck, et al., Model clone detection in practice., Proceedings of the 4th International Workshop on Software Clones. ACM, 2010.
[28] D. Rattan, R. Bhatia, M. Singh, Software clone detection: A systematic review, Information and Software Technology 55.7 (2013) 1165-1199.
[29] MdR Islam, Minhaz F. Zibran, A Comparative Study on Vulnerabilities in Categories of Clones and Non-Cloned Code, Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on. Vol. 3. IEEE, 2016.

[30] H. Störrle, Towards clone detection in UML domain models, Proceedings of the Fourth European Conference on Software Architecture: Companion Volume. ACM, 2010.
[31] Https://msdn.microsoft.com/en-us/library/dd409416.aspx.
[32] Http://www.sdmetrics.com/downman.html.
[33] M. Stephan, A mutation analysisbased model clone detector evaluation framework, Diss. 2014.
[34] M. Stephan, J. R. Cordy, Model Clone Detector Evaluation Using Mutation Analysis, ICSME (2014) 633-638.
[35] G. Mahajan, M. Bharti, Implementing a 3-way approach of clone detection and removal using PC Detector tool, Advance Computing Conference (IACC), 2014 IEEE International IEEE 2014.
[36] U. Mansoor, Handling High-Level Model Changes Using Search Based Software Engineering, (2017).
[37] I. Keivanloo, Source Code Similarity and Clone Search. Diss. Concordia University, 2013.
[38] P. V. Kavita, an analysis of OO design quality measurement tool for UML using SDMetrics, International Journal of Innovation in Engineering Research and Management ISSN 2348-4918, ISO 2000-9001 certified, E 4.3 (2017).
[39] Ragkhitwetsagul, Chaiyong, J.Krinke, D. Clark., Similarity of source code in the presence of pervasive modifications, Source Code Analysis and Manipulation (SCAM), 2016 IEEE 16th International Working Conference on IEEE 2016.
[40] Al Hussein, Abdullah, An Object-Oriented Software Metric Tool to Evaluate the Quality of Open Source Software, International Journal of Computer Science and Network Security (IJCSNS) 17.4 (2017) 345