Detecting Similarities and Clones Using UML Diagrams

[1]Dr.Sandeep Kumar Nain and [2]Manila
[1] Assistant Professor, [2]Research Scholar
[12] Department of Computer Science and Engineering, Galaxy Global Group of Institutions, Dinarpur, Ambala, Haryana, India

Abstract- The copying of code has been studied within application engineering frequently within the subject of clone evaluation. Program clones are regions of source code which are tremendously an identical; these regions of similarity are referred to as clones, clone lessons, or clone pairs. To effectually create process for mannequin clones we can need to be trained the entire items defined in UML. UML models have two parallel constitution, visual representation as diagrams; and an inner, tree-like constitution. While there are several motives why two areas of code is also identical, nearly all of the clone analysis literature attributes cloning recreation to the intentional copying and duplication of code with the aid of programmers; clones can be attributable to robotically generated code, or the constraints imposed by the use of a targeted framework or library. The Tree like constitution is typically provided in XML like modeling language, considering of the free nature of XML layout. To attain any basis for comparisons we are going to have to parse and system the consequent XML of an UML model. Parsing of XML will also be dealt with making use of any excessive stage language aiding XML DOM(document object mannequin) akin to C#, Java, python and so on.

Keywords: UML, class diagrams, multi-version program analysis, concept clones,

I. INTRODUCTION & MOTIVATION

Copying current code fragments and pasting them with or without changes into other sections of code is a established process in software development. The copied code is referred to as program clone and the process is called program cloning. A malicious program detected in a single component of code consequently requires correction in all the replicated fragments of code. Consequently, it's major to seek out all related fragments in the course of the source code. In view that the excessive preservation price, application clone detection has emerged as an lively study field. Distinct programming paradigms and languages have ended in number of clone versions and detection strategies.

- Software clones are reward in lots of distinct software artifacts. As a result, our be trained on detection methods goes beyond supply code.
- Upon assessing state of artwork in program clone research, we realized the dearth of systematic literature assessment. Hence we summarized the prevailing study centered on large and systematic database search and file the study gaps for additional investigation. .

1. Software clones

The Merriam-Webster dictionary defines a clone as one that appears to be a duplicate of an fashioned kind, for this reason being synonymous to a duplicate. In supply code and other application artifacts, the original (code) fragment is copied and pasted with or without changes. The pasted (code) fragment is alleged to be a clone and this activity is referred to as (code) cloning. Nonetheless in software engineering area, the time period code clones is still browsing for a suitable definition. Its vagueness used to be effectively mirrored in Ira Baxter's phrases:
''Clones are segments of code which might be similar according to a couple definition of similarity''.
Roy and Cordy mentions code duplication or cloning as a type of application reuse. The gain knowledge of means that as much as 20–30% of giant application systems include cloned code. Fowler mentions duplication of code as one of the crucial unhealthy practices in software progress growing upkeep price. The increasing use of open source program and its editions also multiplied code reuse. Present code can be modified to cater to new standards thereby facilitating and advancing open source development. Pressing have to realize application clones has invigorated software clone detection as an energetic research discipline.

2. Types of clones

It's quite pertinent to mention that requirements in case of nomenclature are nonetheless lacking thereby leading to specific taxonomies through specific researchers. We list here normal forms of clones.
Sort 1 (certain clones): application fragments which are same besides for variations in white area and feedback.
Kind 2 (renamed/parameterized clones): software fragments that are structurally/syntactically an identical except for changes in identifiers, literals, types, layout and feedback.
Variety three (near omit clones): software fragments which were copied with additional changes like declaration insertions/deletions moreover to alterations in identifiers, literals, forms and layouts. Variety four (semantic clones): program fragments which are functionally similar without being textually identical.

Structural clones: These are patterns of interrelated courses rising from design and evaluation house at architecture level. Structural Clones replicate design stage similarities which help in upkeep.
Operate clones: The clones which are confined to the granularity of a perform/system or procedure. A couple of reports devised the clone detection ways that located the clones at function degree which can also be extracted in a different procedure.
Model headquartered clones: these days graphical languages are exchanging the code as core artifacts for process progress. Surprising overlaps and duplications in models are termed as model founded clones.

3. Why clones

Although reduce–copy–paste–adapt methods are viewed bad practices from a maintenance factor of view, many programmers use them. We list probably the most motives of application cloning.

- **Programmers dilemma and time constraints:** The application is seldom written beneath ideal conditions. Limitations of programmer's skills and rough time constraints inhibit suitable software evolution. Best means out is copying/pasting/enhancing.
- **Complexity of the method:** The situation in figuring out enormous systems best promotes copying the present functionality and good judgment.
- **Language boundaries:** Kim performed an ethnographic be trained on why programmers replica and paste code. They concluded that mostly programmers are compelled to repeat and paste code as a result of boundaries in programming languages. Many languages lack inherent support for code reuse, leading to duplication.
- **Phobia of recent code:** Programmers ordinarily fear to bring in new ideas in present application. They fear that introduction of recent code may just result in a lengthy application development lifestyles cycle. Furthermore, it is less complicated to reuse the prevailing code than to develop a recent solution due to the fact that new code may introduce new mistakes.
- **Lack of restructuring:** Programmers extend restructuring (refactoring, abstraction, etc.) of code as a result of deadlines. Often, restructuring gets delayed until after product delivery which increases subsequent protection fees.
- **Forking/templating:** Forking is the reuse identical solutions, with the hope that evolution of the code will occur independently at least in brief time period. Use of structural and realistic templates is mainly recounted as reuse mechanisms.

4. Advantages of clones

Commonly software developers deliberately introduce code clones into current application. The learn by way of Kapser and Godfrey discusses this limitation. Probably the most facets are mentioned below:

- it is a speedy and instantaneous approach of addressing trade standards.
- Some programming paradigms inspire using Templates in programming.
- If a programming language lacks reuse and abstraction mechanism, it's the only approach left to rapidly increase the present performance.
- The overhead of process calls routinely promotes code duplication for efficiency considerations.

5. Disadvantages of clones

- **Larger upkeep charges:** Two stories verify that presence of code clones in program greatly increase the post implementation maintenance (preventive and adaptive) effort.
- **Trojan horse propagation:** If a code fragment involves a malicious program and that fragment is pasted at specific locations, the same trojan horse will probably be reward in all of the code fragments. So code cloning raises the probability of bug propagation.
- **Bad influence on design:** Code cloning discourages using refactoring, inheritance, and so forth. It leads to bad design apply.
- **Affect on system figuring out/improvement/change:** It is fairly long-established that the character who developed the original approach will not be the one who is preserving it. Moreover the presence of duplicated code now not only complicates the design however results in lowered understanding thereby hampering enhancements and changes. In the long run, the software may just become so tricky that even minor alterations are tough to make.
- **Stress on resources:** Code cloning raises the dimensions of the application procedure thereby putting a strain on procedure resources. It degrades the overall performance in terms of compilation/execution time and house necessities.

II. UNIFIED MODELING LANGUAGE

Unified Modeling Language (UML) is describing as a standardized general purpose modeling language in object-oriented program engineering. The UML involves a collection of picture notation strategies to create visual items of object-oriented application-intensive systems.
The UML was once developed thus of Grady Booch, Ivar Jacobson and James Rumbaugh at Rational application in the 1990s. It's used to specify, visualize, modify and record the artifacts of an object-oriented application-intensive process beneath progress.
UML combines tactics from information, business, object and component modeling. It's used with all approaches, all over the application progress existence cycle and across special implementation applied sciences.
The Unified Modeling Language (UML) presents a average way to feel about a procedure's architectural blueprints, with elements such as:

- Activities
- Actors
- Business Processes

- Database Schemas
- (Logical) Components
- Programming Language Statements
- Reusable Software Components.

Unified Modeling Language is a specification language that's used in the software engineering field. It is defined as a extensive intent language that makes use of a graphical designation which can create an summary mannequin. This summary model can be used in a approach. This method is called the UML model. Nonetheless, it should be emphasized that UML will not be restrained conveniently modeling application. It can be used to assemble models for process engineering, industry tactics, and institution buildings. A certain language known as systems Modeling Language was designed to manage programs that have been outlined within UML 2.Zero.

The UML is essential for a number of explanations. First, it has been used as a system for the development of applied sciences which are model driven and some of these include model pushed progress and mannequin driven architecture. For the reason that an emphasis has been placed on the value of graphics notation, UML is in a position in meeting this demand, and it may be used to signify behaviors, classes, and aggregation.

It should also be famous that UML items may also be transformed into various other representations. One illustration of this is the capacity to transform UML models into Java representations. The Unified Modeling Language has a number of elements and characteristics which separate it from other languages within the same category. Many of these attributes have allowed it to be valuable for builders.

III. RELATED WORK

Nguyen HoanAnh et al, in 2012 [1] The authors describe Recent research results suggest a need for code clone management. In this paper, they introduce JSync, a novel clone management tool. JSync provides two main functions to support developers in being aware of the clone relation among code fragments as software systems evolve and in making consistent changes as they create or modify cloned code. JSync represents source code and clones as (sub) trees in Abstract Syntax Trees; measures code similarity based on structural characteristic vectors, and describes code changes as tree editing scripts. The key techniques of JSync include the algorithms to compute tree editing scripts, to detect and update code clones and their groups, to analyze the changes of cloned code to validate their consistency, and to recommend relevant clone synchronization and merging. Their empirical study on several real-world systems shows that JSync is efficient and accurate in clone detection and updating, and provides the correct detection of the defects resulting from inconsistent changes to clones and the correct recommendations for change propagation across cloned code.

Harald Störrle et al., 2013 [2]In this paper code clones (i.e., duplicate fragments of code) have been studied for long, and there is strong evidence that they are a major stheirce of software faults. Anecdotal evidence suggests that this phenomenon occurs similarly in models, suggesting that model clones are as detrimental to model quality as they are to code quality. However, programming language code and visual models have significant differences that make it difficult to directly transfer notions and algorithms developed in the code clone arena to model clones. In this article, they develop and propose a definition of the notion of "model clone" based on the thorough analysis of practical scenarios. They propose a formal definition of model clones, specify a clone detection algorithm for UML domain models, and implement it prototypically. They investigate different similarity heuristics to be used in the algorithm, and report the performance of their approach. While they believe that their approach advances the state of the art significantly, it is restricted to UML models, its results leave room for improvements, and there is no validation by field studies.

Zhang Gang et al., in 2013 [3] The authors describe Effective clone management is essential for developers to recognize the introduction and evolution of code clones, to judge their impact on software quality, and to take appropriate measures if required. Their previous study shows that cloning practice is not simply a technical issue. It must be interpreted and considered in a larger context from technical, personal, and organizational perspectives. In this paper, they propose a contextual and on-demand code clone management approach called CCEvents (Code Cloning Events). Their approach provides timely notification about relevant code cloning events for different stakeholders through continuous monitoring of code repositories. It supports on-demand customization of clone monitoring strategies in specific technical, personal, and organizational contexts using a domain-specific language. They implemented the proposed approach and conducted an empirical study with an industrial project. The results confirm the requirements for contextual and on-demand code clone management and show the effectiveness of CCEvents in providing timely code cloning notifications and in helping to achieve effective clone management.

XieShuai et al., 2013 [4] The authors describe When implementing new features into a software system, developers may duplicate several lines of code to reuse some existing code segments. This action creates code clones in the software system. The literature has documented different types of code clone (e.g., Type-1, Type-2, and Type-3). Once created, code clones evolve as they are modified during both the development and maintenance phases of the software system. The evolution of code clones across the revisions of a software system is known as a clone genealogy. Existing work has investigated the fault-proneness of Type-1 and Type-2 clone genealogies. In this study, they investigate clone genealogies containing Type-3 clones. They analyze three long-lived software systems Apache-Ant, Argo UML, and J Boss, which are all written in Java. Using the NiCad clone detection tool, they build clone genealogies and examine two evolutionary phenomena on clones: the mutation of the type of a clone during the evolution of a system, and the migration of clone segments across the repositories of a software system. Results show that

1. mutation and migration occur frequently in software systems;
2. the mutation of a clone group to Type-2 or Type-3 clones increases the risk for faults;
3. increasing the distance between code segments in a clone group also increases the risk for faults.

R.K. Saha, et al, 2013 [5]The authors describe Understanding the evolution of clones is important both for understanding the maintenance implications of clones and building a robust clone management system. To this end, researchers have already conducted a number of studies to analyze the evolution of clones, mostly focusing on Type-1 and Type-2 clones. However, although there are a significant number of Type-3 clones in software systems, they know a little how they actually evolve. In this paper, they perform an exploratory study on the evolution of Type-1, Type-2, and Type-3 clones in six open source software systems written in two different programming languages and compare the result with a previous study to better understand the evolution of Type-3 clones. Their results show that although Type-3 clones are more likely to change inconsistently, the absolute number of consistently changed Type-3 clone classes is higher than that of Type-1 and Type-2. Type-3 clone classes also have a lifespan similar to that of Type-1 and Type-2 clones. In addition, a considerable number of Type-1 and Type-2 clones convert into Type-3 clones during evolution. Therefore, it is important to manage type-3 clones properly to limit their negative impact. However, various automated clone management techniques such as notifying developers about clone changes or linked editing should be chosen carefully due to the inconsistent nature of Type-3 clones.

S. Schulze et al, in 2013 [6] The authors describe Code clones are a common reuse mechanism in software development. While there is an ongoing discussion about harmfulness and advantages of code cloning, this discussion is mainly centered around aspects of software quality. However, recent research has shown that code cloning may have legal implications as well such as license violations. From this point of view, a developer may favor to hide his cloning activities. To this end, he could obfuscate the cloned code to deceive clone detectors. However, it is unknown how robust certain clone detection techniques are against code obfuscations. In this paper, they present a framework for semi-automated code obfuscations. Additionally, they present a case study to evaluate the robustness of selected clone detectors against such obfuscations.

I. Keivanloo et al, **2013 [7]**The authors describe they introduce semantic-enabled clone detection, as an approach that emphasizes on importance of the token semantics during the pattern matching for clone detection. This approach can be realized using Semantic Web and its support for knowledge modeling. While the Semantic Web has found wide acceptance in various application and research domains, it still lacks the same acceptance in the source code analysis domain. In this paper they focus on how the Semantic Web and its support for semantic modeling and knowledge retrieval can be applied towards source code clone detection and search. They discuss both, open challenges in the source code clone detection domain, as well as both theoretical and practical aspects on how the Semantic Web can address some of these challenges.

Svajlenko, J. et al, **2013 [8]** The authors describe Detecting clones from large datasets is an interesting research topic for a number of reasons. However, building scalable clone detection tools is challenging and it is often impossible to use existing state of the art tools for such large datasets. In this research they have investigated the use of their Shuffling Framework for scaling classical clone detection tools to ultra large datasets. This framework achieves scalability on standard hardware by partitioning the dataset and shuffling the partitions over a number of detection rounds. This approach does not require modification to the subject tools, which allows their individual strengths and precisions to be captured at an acceptable loss of recall. In their study, they explored the performance and applicability of their framework for six clone detection tools. The clones found during their experiment were used to comment on the cloning habits of the global Java open-source development community.

Cho Kwantae et al., **2013 [9]** The authors describe Wireless sensor networks (WSNs) consist of tiny sensor nodes that communicate with each other over wireless channels, often in a hostile environment where nodes can be captured and compromised. Consequently, an adversary may launch a clone attack by replicating the captured nodes to enlarge the compromised areas employing clones. Thus, it is critical to detect clone nodes promptly for minimizing their damage to WSNs. Recently; various clone detection schemes were proposed for WSNs, considering different types of network configurations, such as device types and deployment strategies. In order to choose an effective clone detection scheme for a given sensor network, the selection criteria play an important role. In this paper, they first investigate the selection criteria of clone detection schemes with regard to device types, detection methodologies, deployment strategies, and detection ranges. They then classify the existing schemes according to the proposed criteria. Simulation experiments are conducted to compare their performances. It is concluded that it is beneficial to utilize the grid deployment knowledge for static sensor networks; the scheme using the grid deployment knowledge can save energy by up to 94.44% in comparable performance (specifically in terms of clone detection ratio and the completion time), as compared to others. On the other hand, for mobile sensor networks, no existing approach works efficiently in reducing detection error rate

Roy, C.K. et al, 2014 [10] The authors describe Duplicated code or code clones are a kind of code smell that have both positive and negative impacts on the development and maintenance of software systems. Software clone research in the past mostly focused on the detection and analysis of code clones, while research in recent years extends to the whole spectrum of clone management. In the last decade, three surveys appeared in the literature, which cover the detection, analysis, and evolutionary characteristics of code clones. This paper presents a comprehensive survey on the state of the art in clone management, with in-depth investigation of clone management activities (e.g., tracing, refactoring, and cost-benefit analysis) beyond the detection and analysis. This is the first survey on clone management, where they point to the achievements so far, and reveal avenues for further

research necessary towards an integrated clone management system. They believe that they have done a good job in surveying the area of clone management and that this work may serve as a roadmap for future research in the area.

XieShuai et al., 2014 [11] The authors describe Copy and paste activities create clone groups in software systems. The evolution of a clone group across the history of a software system is termed as clone genealogy. During the evolution of a clone group, developers may change the location of the code fragments in the clone group. The type of the clone group may also change (e.g., from Type-1 to Type-2). These two phenomena have been referred to as clone migration and clone mutation respectively. Previous studies have found that clone migration occur frequently in software systems, and suggested that clone migration can induce faults in a software system. In this paper, they examine how clone migration phenomena affect the risk for faults in clone segments, clone groups, and clone genealogies from three long-lived software systems JBoss, APACHE-ANT, and ARGOUML. Results show that:

1. migrated clone segments, clone groups, and clone genealogies are not equally fault-prone;
2. when a clone mutation occurs during a clone migration, the risk for faults in the migrated clone is increased;
3. migrating a clone that was not changed for a longer period of time is risky.

Mandal, M. et al., 2014 [12]In this paper, they present an in-depth empirical study on identifying clone fragments that can be important refactoring candidates. They mine association rules among clones in order to detect clone fragments that belong to the same clone class and have a tendency of changing together during software evolution. The idea is that if two or more clone fragments from the same class often change together (i.e., are likely to co-change) preserving their similarity, they might be important candidates for refactoring. Merging such clones into one (if possible) can potentially decrease future clone maintenance effort. They define a particular clone change pattern, the Similarity Preserving Change Pattern (SPCP), and consider the cloned fragments that changed according to this pattern (i.e., the SPCP clones) as important candidates for refactoring. For the purpose of their study, they implement a prototype tool called MARC that identifies SPCP clones and mines association rules among these. The rules as well as the SPCP clones are ranked for refactoring on the basis of their change-proneness. They applied MARC on thirteen subject systems and retrieved the refactoring candidates for three types of clones (Type 1, Type 2, and Type 3) separately. Their experimental results show that SPCP clones can be considered important candidates for refactoring. Clones that do not follow SPCP either evolve independently or are rarely changed. By considering SPCP clones for refactoring they not only can minimize refactoring effort considerably but also can reduce the possibility of delayed synchronizations among clones and thus, can minimize inconsistencies in software systems.

Dhavleesh Rattan et. al, 2015 [13], This paper presents a novel method to compute similarity across object oriented programs at different levels of granularity. The tool is able to detect concept level similarities by applying latent semantic indexing and principal component analysis. Previous research has shown that detection of high level similarities can help in comprehending the design of the system for better maintenance. Moreover, model driven development has become a standard industry practice; we have extended our tool to detect similarities for UML diagrams by measuring the distance between two class models. In addition, we mined important change patterns at method level using multiversion program analysis by applying the proposed technique throughout the evolutionary history of the software. We have validated the similarity score by applying the tool at function level in the source code. We assess the usefulness and scalability of the proposed technique by empirical evaluation on source code of open source subject systems and multi-version program analysis on 8 releases of dnsjava.

Work Done

In this paper we compare four different models i.e Hospital Model, School Model, Transport Management and Insurance Management for detecting clones and to check duplicity and similarities between them.
Model Methods and Model Properties are different columns Names.

Table 1 and Table 2 shows the comparison between four different models of their properties and methods.

Table 3 shows comparison between actual cloning and detected cloning.

IV. RESULTS

Model Name	Model Properties	Model Methods
Hospital Model	15	12
School Model	16	14
Transport Management	19	21
Insurance Management	31	25

Table-1

Model Name	Model Properties	Model Methods
Hospital Model	90	59
School Model	96	42
Transport Management	76	126
Insurance Management	124	150

Table-2

Model Name	Actual Cloning (%)	Detected Clones (%)
Hospital Model	85	81
School Model	85	80
Transport Management	89	83
Insurance Management	92	88

Table-3

Reference Model	Candidate Model	Cloning (%)
Person	Person	100%
Satff	Satff	75%
Student	Faculty	65%
Student	Student	100%
Faculty	Faculty	75%

Table-4

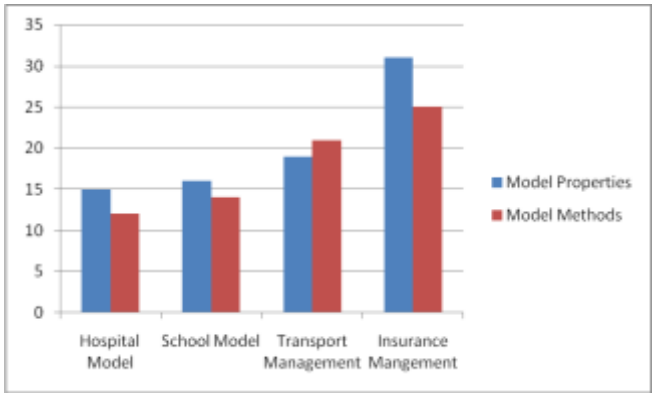

Figure-1

As a result of Table 1 Fig.1 shows that there is an increase in Model Properties with the increase in Model Method.

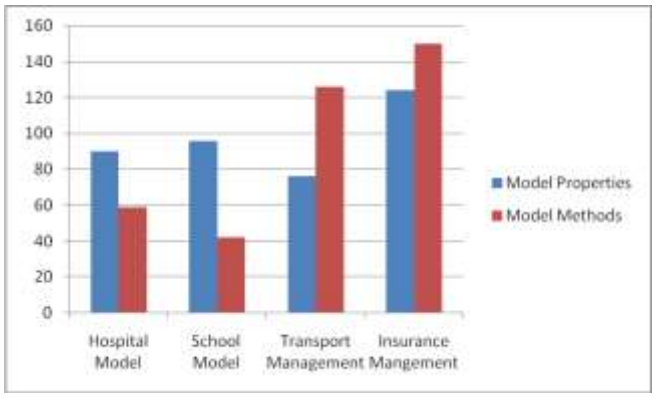

Figure-2

Fig 2 shows first model properties increased with the decrease in model methods and then model methods increase with the decrease in model properties.

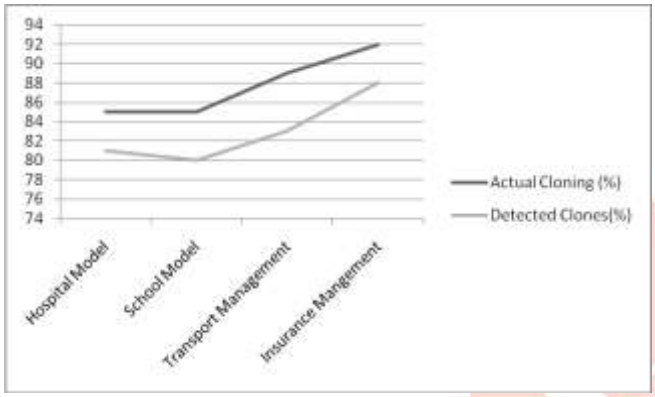

Figure-3

Fig. 3 shows percentage of cloning gradually increases.

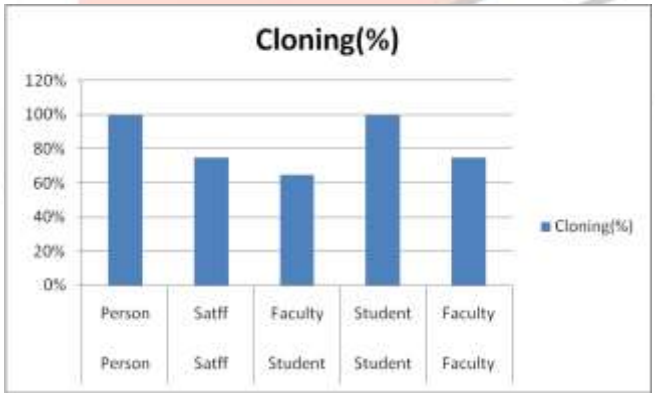

Figure-4

Table 4 shows the comparison between two models i.e Reference Model (Hospital Model) & Candidate Model (School Model). As a result of this we find the percentage between actual cloning and detected cloning.

V. CONCLUSION

Four different attributes are compared separately i.e (Person, Staff, Student, Faculty) As a Result we find percentage (%) of cloning in models.

Our Algorithm is used for finding the similarity and cloning in attributes and Methods. Reference Model is used to search clones in candidate model.

As a result our algorithm is successful in finding out 90% cloning in candidate model.

If a system is to be developed, its clones will have to be identified as a way to make constant changes. Cloning is often a strategic method for evolution.

Clone detection systems play a main position in application evolution study where attributes of the identical code entity are discovered over more than one types. To effectually create any process or process for model clones detection we can ought to gain knowledge of the entire items defined in UML together with inside and outside constitution of UML We have now learned that our approaches for detecting clones work very well for model clones and we have been able to become aware of program clone for the same. Some Heuristics, corresponding to 60% constraint are still required to with clone detection but the procedure produces enough results for any no. of units.

VI. FUTURE SCOPE

In code-based development cloning in models creates several difficulties in software package maintenance. However, existing clone detection tools for models have limitations on accuracy and completeness. during this wprl, In Furture We will try to implement algorithms that ar able to consistently observe clones and clone teams within the UML based models with a high degree of completeness and accuracy. The core concepts embody the systematic generation of the candidate clones with the optimisation techniques, and therefore the precise structural feature extraction for candidate subgraphs whereas technique has been tested and hand validated on an outsized set of publically accessible UML models, it's however to be used on software examples, and that expect to operating with software partners to investigate their systemsit's not clear that every one of those attributes transfer to its use on graphical models, and that are presently running a bigger scale experiment to statistically validate and compare technique on a way larger set of models. We also try to implement plugin systemes to observe systematically renamed model clones, at the instant In future, We will also try to generate UML colorization scripts to indicate results directly within the UML IDE

VII. REFERENCES

[1]. Nguyen, HoanAnh, Tung Thanh Nguyen, Nam H. Pham, Jafar Al-Kofahi, and Tien N. Nguyen. "Clone management for evolving software." Software Engineering, IEEE Transactions on 38, no. 5 (2012): 1008-1026.

[2]. Harald Störrle, "Towards clone detection in UML domain models." Software & Systems Modeling 12, no. 2 (2013): 307-329.

[3]. Zhang, Gang, Xin Peng, Zhenchang Xing, Shihai Jiang, Hai Wang, and Wenyun Zhao. "Towards contextual and on-demand code clone management by continuous monitoring." In Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on, pp. 497-507. IEEE, 2013.

[4]. Xie, Shuai, FoutseKhomh, and Ying Zou. "An empirical study of the fault-proneness of clone mutation and clone migration." In Proceedings of the 10th Working Conference on Mining Software Repositories, pp. 149-158. IEEE Press, 2013.

[5]. Saha, Ripon K., Chanchal K. Roy, Kevin A. Schneider, and Dewayne E. Perry. "Understanding the evolution of type-3 clones: an exploratory study." In Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on, pp. 139-148. IEEE, 2013.

[6]. Schulze, Sandro, and Daniel Meyer. "On the robustness of clone detection to code obfuscation." In Proceedings of the 7th International Workshop on Software Clones, pp. 62-68. IEEE Press, 2013.

[7]. Keivanloo, Iman, and JuergenRilling. "Semantic-Enabled Clone Detection." In Computer Software and Applications Conference (COMPSAC), 2013 IEEE 37th Annual, pp. 393-398. IEEE, 2013.

[8]. Svajlenko, Jeffrey, ImanKeivanloo, and Chanchal K. Roy. "Scaling classical clone detection tools for ultra-large datasets: An exploratory study." In Proceedings of the 7th International Workshop on Software Clones, pp. 16-22. IEEE Press, 2013.

[9]. Cho, Kwantae, Minho Jo, Taekyoung Kwon, Hsiao-Hwa Chen, and Dong Hoon Lee. "Classification and experimental analysis for clone detection approaches in wireless sensor networks." Systems Journal, IEEE 7, no. 1 (2013): 26-35.

[10]. Roy, Chanchal K., Minhaz F. Zibran, and Rainer Koschke. "The vision of software clone management: Past, present, and future (keynote paper)." In Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on, pp. 18-33. IEEE, 2014.

[11]. Xie, Shuai, FoutseKhomh, Ying Zou, and ImanKeivanloo. "An empirical study on the fault-proneness of clone migration in clone genealogies." In Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on, pp. 94-103. IEEE, 2014.

[12]. Mandal, Manishankar, Chanchal K. Roy, and Kevin A. Schneider. "Automatic ranking of clones for refactoring through mining association rules." In Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on, pp. 114-123. IEEE, 2014.

[13]. Rattan, Dhavleesh, Rajesh Bhatia, and Maninder Singh. "Detecting High Level Similarities in Source Code and Beyond." (2015).