Lampyrid 2023: Volume 13, 61–72
ISSN: 2041-4900
https://lampyridjournal.com

Code Clone Detection: An Empirical Study of Techniques for Software Engineering Practice

Harshita Kaushik[1], Dr K D Gupta[2]
[1]. PhD Scholar, Department of Computer Science Apex University, Jaipur,
harshitasharma061194@gmail.com
2. Associate Professor, Department of Computer Science Apex University, Jaipur,
Kdevgupta@gmail.com

Abstract— *Software cloning has been a prevalent practice in software development for several decades, wherein code fragments are duplicated and reused throughout the codebase. While cloning can help boost productivity and code maintainability, it also has the potential to introduce new problems like bugs, inconsistencies, and code smells. The detection of code clones, which entails the identification of code segments that are structurally similar or identical, is one of the key issues in dealing with software cloning. Various types of code clones include Type 1 clones (identical code fragments), Type 2 clones (structurally similar code fragments), Type 3 clones (code fragments with semantic similarity), and Type 4 clones. Textual analysis, token-based analysis, and tree-based analysis are only a few of the methods explored to identify code clones. Another method that has shown promise in clone identification is probabilistic software modeling, in which code is modeled as a probabilistic network and clones are found by analysis of the graph structure. Herein, we survey the state-of-the-art in software cloning and code clone detection methods. The paper also covered the numerous kinds of code clones along with their benefits and drawbacks. We next explore and evaluate many methods for identifying code clones, including probabilistic software modeling. Finally, we investigate the ways in which probabilistic software modeling may be used for various software engineering purposes, such as predictive and generative.*

Keywords—*Software Clone, Source Code, Code Clone, Code Clone Detection, Probabilistic Software Modeling.*

---

1. INTRODUCTION

Cloning is a prevalent practise in the software and scientific communities. The accessible resources that encourage code cloning are often utilised to reuse the existing code. It creates a difficult challenge for software maintenance. It takes a significant amount of time and work, which raises the cost of building any tool, programme, etc. Syntactic and semantic clones are two basic forms of clonesType-1, Type-2, and Type-3 syntactic clones exist, although Type-4 semantic clones exist. [1]. Type-1 clones are two or more pieces of code that are otherwise identical save for a few formatting details. Type-2 clones are defined as code segments which are identical to one another but have minor name changes, such as variables, renaming identifiers, etc. As a result, Type-2 clones are often referred to as renamed clones. Type-3 clones are close clones that include some extra additions and deletions of instructions but are nonetheless comparable code fragments. Semantic clones, also known as Type-4 clones, are copies that do the same objective but may have a distinct syntactic structure. Table 1 provides an example of a semantic clone which describes the swap of two integers based on distinct logics..

abstract syntax tree-based, token-based Text-based, programme dependency graph-based, etc. methods have all been utilised in past to find code clones. [2]. Source code clone detection has recently benefited from deep learning &machine learning[3] [4] based approaches. The use of deep learning in identifying copying codes has been on rise recently. (e.g., [5][6][7][8][9][10]). Embedding methods like graph2vec, node2vec, & word2vec, are used to discover structural similarities between pieces of source code represented in abstractions like tokens, abstract syntax trees, and control flow graphs. Existing research mostly focuses on identifying duplicates of code written in same language.

[11]. However, these days software is typically developed on a multilanguage platform, wherein a variety of languages are employed to accomplish the same tasks. APIs for large data processing, like Apache Spark, are widely used due to their similarity in name and call patterns across languages. [12][13]. When changes are made to one clone in such an environment, updates must be performed consistently across all clones, which often includes clones in several languages. Some study has been conducted on the identification of cross-language code clones. Nevertheless, their methods underperform because they rely on poor quality features for learning and predicting. In this study, we provide a concise and effective overview & literature analysis to assist future researchers in becoming acquainted with methodologies utilised to identify semantic code clones.[14] The fundamental goal of this research is to give a thorough systematic & comparative examination of semantic clone detection algorithms, together with their benefits and drawbacks.

Table 1. An Example of a Semantic Clone

main ()
{int I, J;
int temp;
temp=I;
I=J;
J=temp;
}
	main ()
	{int P, Q;
	P=P+Q;
	Q=P-Q;
	P=P-Q;
	}

2. Software Cloning

Clones of software are described in terms of syntax or semantics as comparable (near-miss) or the same (precise) code fragments. In general, these code fragments are produced by the copy-pasting of code by programmers that generate similar clones. Yet, if the parts of copied code include little amendments, they lead to clones nearly miss. The code may no longer be regarded as a clone, as the consequence of significant alterations to the copied code. Likewise, once programmers do a common task or even after they utilize libraries or APIs, certain clones are accidentally placed into software systems. If two data pieces have the same functionality and also have alternative syntax implementations, semantic clones are termed. Mostly during the development phase, software code cloning delivers benefits. Application inventors reuse their private code fo2 save time reworking, or utilize the code of others to circumvent some constraints on programming & design. Further attention is given to skilled developers to pick higher quality, tested & bug-free cloning code. In contrast, the copied code might contain a significant issue, i.e. bugs which require additional maintenance tests or updating [15]. Jamshoro has the highest average pace contrasted to other zones. [16].

In the process of developing software, programmers often prefer to clip and paste a section of source code from another source segment exactly, even if this requires making some minor adjustments in order for two sections to seem equal or similar. This is referred to as "software/code cloning," and some researchers also do it. Programmers may complete their task more quickly by using code clones. There are several reasons for copying the code. Due to this sort of conduct, programming or maintenance problems emerge. If, for example, a defect is perceived in a cloned software system code fragment, the programmer must find and repair this bug everywhere, therefore increasing software maintenance problems.

In addition, code clones may contribute to vulnerabilities spread in terms of software system security when a susceptible portion of code is cloned. Although software developers are attempting to design safe source code & reduce source vulnerabilities throughout their system development, software programming will unavoidably cause code clone behavior and spread system faults. If two code pieces are very identical in software engineering with little changes or, because of copy-paste behavior, are even identical [17].

3. Code Clones

Code clones are code fragments that are in pairs, inside or between software systems. When reusing code via cut/paste, software developments produce clones, but clones can be created for a variety of reasons. The influence of clones on software design may be detrimental. Their size increases unnecessarily, software maintenance & re-engineering expenses rise. The problem is reproduced all across the system, making debugging & bug repair difficult when bugging code has been cloned. Clones might introduce additional issues if they are not updated on development of original code snippet. Cloning can also have advantages including acceleration & decoupling software [1]. Nevertheless, to limit its negative impacts, designers must maintain track of their clones. It has been proven that clone statistical models have apps in code search, new API exploitation, bug identification, detection of security vulnerability, malware detection, etc. Figure 1 shows an example of code clone.

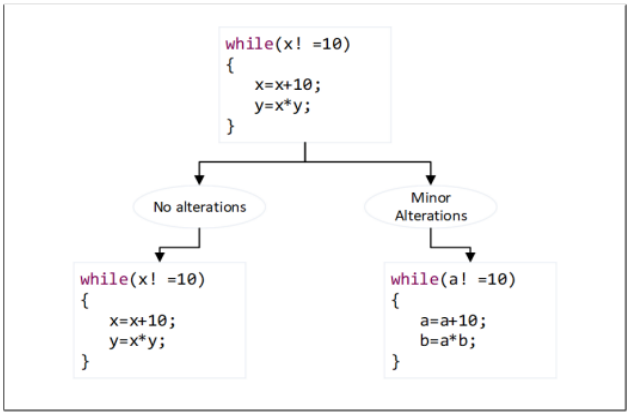
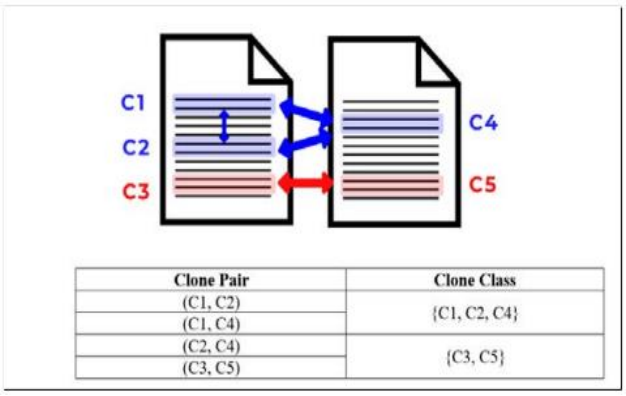


Figure 1. An Example of a Code Clone

The two pieces of code which make up a clone pair, or clone classes, are said to have a clone relation if and only if they are equivalent. [56]. To be an equivalence relation, it must be capable of holding all other relations (reflexive, symmetric, & transitive). When two pieces of code share the same sequences—whether they be original strings, sequences of token type, strings with whitespace, or sequences of converted tokens—they are said to have a clone relation. A pair of code parts or code segments that match is what is meant by a clone pair for a particular clone relation, and the term "clone class" refers to the equivalence class of a clone relation. In other words, if the relation holds between any two code segments, then all code segments in a clone class create a clone pair. Clone pairs and clone classes are shown in Figure 2.



Clone Pair	Clone Class
(C1, C2)	{C1, C2, C4}
(C1, C4)	
(C2, C4)	{C3, C5}
(C3, C5)	

Figure 2: A description of clone class & clone pair

A. Code Clone Types

Experts suggest four major categories of clones that are mutually excluding and characterized regarding their detecting capability [1] [18]. Clones are divided into four types depending on their similarities: textual (types I, II, and III) & functional (type IV). In particular, these:

a) Type-1

Type-1 clones are equivalent fragments of code if trivia such as foreign white space, code styling & comments are ignored. Type 1 clones are normally produced when the layout & comments are copied and pasted without changes or alteration. Identical code segments, which ignores white space variances, code formatting/style & comments.

Code Segment 1	Code Segment 2

`int a=0, b=1, c=2;`
`while (c>0)`
`{`
`  a= a+b;`
`  b= b*c;`
`  c= c-1;`
`}`
	`int a=0, b=1, c=2;`
	`while (c>0)`
	`{`
	`  a= a+b;`
	`  b= b*c;`
	`  c= c-1;`
	`}`

Figure 3: Type -1 (or precise) clones

b) Type-2

Type-2 clones are generated often by copying and pasting with a tiny change like the rename of a variable, argument, or a literal. Many detectors can easily detect Type 2 clones. Comparable code fragments structurally/syntactically, ignoring changes in identification names, literal data & variations in blank space, code formatting / style & comments. Figure 4 is an illustration of a Type II clone.

Code Segment 1	Code Segment 2

`int a=0, b=1, c=2;`
`while (c>0)`
`{`
`  a= a+b;`
`  b= b*c;`
`  c= c-1;`
`}`
	`int a=0, b=1, d=2;`
	`while (d>0)`
	`{`
	`  a= a+b;`
	`  b= b*c;`
	`  d= d-1;`
	`}`

Figure 4: Type-2 (or rechristened clones)

c) Type-3

Type 3 clones have changes in the declaration, and code fragments which include statements have been added/removed or changed. Code fragments are syntactically identical with statements differing. The code fragments are accompanied by statements that are added, deleted, or modified. When further changes are performed to a line-level clone code fragment, like deletion, addition, or alteration of one or more lines, a Type 3 copy is generated. Type III clones, like clone 5, are very similar to the original but not quite[19].

Code Segment 1	Code Segment 2

int a=0, b=1, c=2;
while (c>0)
{
a= a+b;
b= b*c;
c= c-1;
}
	int a=0, b=1, c=2, e=5;
	while (c>0)
	{
	a= a+b;
	b= b*c;
	e= a+b;
	c= c-1;
	}

Figure 5: Clones of Type-3 (or near miss)

d) Type-4

Type-4 clones may emerge when several distinct syntactic versions have the same functionality performed. The code segments are syntactically different, implementing the same or comparable functions. Semantic similarities between two or more pieces of code (also called "functional clones" or "dependency clones") are often found to be 4. Type IV clones are seen in Figure 6.

Code Segment 1	Code Segment 2

int a=0, b=1, c=2;
while (c>0)
{
a= a+b;
b= b*c;
c= c-1;
}
	int a=0, b=1, c=2, f=5;
	while (c>0)
	{
	a= a+b;
	b= b*c;
	f= a+b;
	c= c-1;
	}

Figure 6. Type-4 (or) semantic clones

There are many degrees of complexity needed to detect different kinds of clones. Although lexical-based analysis may be employed to spot examples of Types I and II, it becomes more difficult to find Types III and IV because matching functionally identical code fragments requires more sophistication.

B. Advantages of Code Clone

In software systems, clones are often added after restructuring so that various maintenance advantages may be obtained. Some of the benefits provided by clones are discussed below.

- **Danger in Writing New Code:** When a developer desires to eliminate hazards associated with developing new code, he or she will utilise existing code. Writing new code introduces the possibility of errors and defects, whereas the extant code has been thoroughly examined. According to Cordy, clones occur frequently in a financial software system, despite the fact which financial products rarely alter, notably within same financial institution. Mostly because the current system requires constant maintenance and improvements to accommodate new features that are functionally equivalent to those already present. In cases like these, it is common practise to ask the developer to modify an existing code block to meet the needs of novel product. This is mostly due to the much higher risk of introducing software defects detected in new code fragments as compared to using preexisting code that has already through comprehensive testing. Software errors. in an organisation may be highly expensive.
- **Software architecture that is clear and understandable:** In order to promote clear and understood software design, it is intended to incorporate clones into the system.
- **Maintenance Speedup:** In a multi-cloned system, two cloned code fragments are distinct from

one another in terms of both syntaxse & semantics and may develop independently at various rates without impacting one another. Testing can also be done and is necessary for updated fragments. Main training cloned parts in a system could facilitate maintenance, particularly in the absence of automatic regression checks.

- **Ensuring Robustness of Life-critical Systems:** Life-critical systems are often built with redundancies or clones in mind. In order to minimise the possibility of mistakes, numerous teams work on same functionality in life-critical systems to ensure that all safety measures are maintained and the system functions without problems. This not only enhances recommendation outcomes, but also decreases the difficulties caused by data scarcity by a significant amount [20].
- **The High Expense of Calling Functions in Real-Time Applications:** It may seem that function calls are too expensive to use in real-time applications. Although inlined functions are slightly faster than regular functions due to the elimination of function-calling overheads, they also consume more memory. There will be ten copies of function added to code if it is inlined ten times. Without inline functions, the computer autonomously decides which functions to inline, and if it doesn't, coder must write code that would have gone in function at the place where the function is called, which creates duplicates.
- **Disadvantages of Code Clone:**The use of code cloning may make the development of software systems simple, but it may also be essential for ongoing maintenance and improvement of software system's quality. Code cloning may simplify the construction of software systems, but it may also be necessary for continuous maintenance and quality enhancement of such systems.
- **Increased Probability of Defects:** If the original code has a flaw, then the clone will have same problem. As a result, duplicating code can make it more likely for a system flaw to happen.
- **Increased Resource Requirements:** An rise in code clones may increase the size of system, the time it takes to compile code, and amount of memory system needs, all of which might lead to need for costly hardware and software updates.
- **Increase Maintenance Effort and Cost:** Code cloning during software maintenance significantly increases the amount of work that must be done. If an error or flaw is discovered during the maintenance stage, all of clones of that fragment should be checked to see if they share the same error or bug before problem can be fixed, which increases maintenance effort.
- **Increased Chances of Bad Design:** The number of code clones may rise, resulting in a larger system with more memory needs, longer compilation times, and larger system sizes. This might play a role in pricey hardware and software updates.

4. Code Clone Detection

Code clone detection has been an integral part of a number of software engineering processes. In context of aspect mining, comprehension of initiatives, plagiarism detection, copyright, code composition, analytics, software developments, quality analysis, bog detection as well as viral detection, to address just a few, text like syntactic, similar, or semantics code fragments should usually be recognized. There has been a lot of attention in detecting code clones recently. A clone is an object that occurs more than once in a development software output. Nowadays most clone detection focuses on code clones, although cloning may take place in any device. In code, this is often the consequence of regular programming practice: developers know that something similar has been accomplished elsewhere. Simply copy & change this section of the code to meet your new needs. So far, it's not an issue, because we anticipate the author to refactor to eliminate the new duplication. This often, even so, doesn’t happen either due to constraint of time or because the developer does not even know that this may be an issue [21].

A. Code Clone Detection Techniques

In general, methods for detecting different types of clones in software systems may be grouped into many areas, that are discussed below. [22]:

a) Textual Approaches

Text-based languages The most popular and simple approach of finding clones is via use of clone detection tools. Such methods look for linguistic patterns in source code that may indicate cloning by analysing code's lexical structure. Particularly, string-based detection techniques are often applied to distinguish either identical sections of code (Type I) or clones with slight modifications, including renamed variables (Type II), and they may be utilised across a broad range of programming languages. This method involves a line-by-line comparison of strings without any rewriting of code. Recently, though, various text-based approaches have been developed that modify code by eliminating whitespace and comments. Since these approaches do not necessitate a semantical or syntactical analysis of the source code, their efficacy is superior to that of other techniques. Dup is an example of a clone detection instrument which employs textual analysis.

Token Based Technique

Token-based clone detection methods work by first translating code into a set of tokens, and then comparing those tokens to others in a sequence which have similarities with them. It is common practise to use a lexical analyzer for the process of converting code into tokens. This method runs less quickly than a text-based method since every code must be tokenized, which takes a lot of time. CCFinder is an example of a clone detection instrument which employs token analysis.

b) Code Metrics Analysis

These tools compare different sections of code using metrics derived from source code to see whether any of sections are identical or nearly identical. Particularly, a few metrics are employed as code fragment classification and representation signatures. The fundamental idea is that clones of two or more code fragments would have a variety of features, all of which can be accurately measured by metrics being employed. Because of this, signatures that are quite similar raise the possibility of cloning. Code metrics approaches are quicker, simpler, and easier to utilise than other clone detection techniques. They also take less time to identify code. Covet is an instrument that adheres to metrics-based clone detection method.

c) Parsing Techniques

Source code abstract syntax trees (ASTs) are compared and matched using such methods. Particularly, subtree similarities in the AST of the system are indicative of cloning. Type II clones may also be detected using these methods since names and literal values of variables are not taken into account while building AST. It takes a long time for this method to be applied on a huge source code. CloneDR is a programme that employs an AST-based strategy to find clones.

d) Graph Analysis

By comparing parallel subgraphs in a PDG (programme dependency graph), this method locates code snippets in a programme that are functionally equivalent to one another. A PDG is a visual representation of logic and data flow of a programme. This method improves the AST parsing method by taking into account both syntactic structure and the data flow of programmes. This specific method can detect interwoven clones as well as clones with matching code statements which have been rearranged. Duplix is an instrument for clone detection based on PDG.

e) Hybrid Technique

The applications of this method are flexible. Here, developers typically combine multiple methods to identify potential clone codes. The processes may be so complex that they are carried out in phases, with a complete technique constituting the initial stage and another method constituting the second. Kosche et al. proposed a method that ultimately proved effective. The members of the team contrasting the tokens of different AST (Abstract Syntax Tree) nodes without doing a direct comparison for each and every AST node. Tairas et al. built a methodology to find

already-existing, functioning cones in software by combining suffix tree and AST-based approaches into one system.

Table 2. Clone detection methods' capabilities and characteristics

Technique	Clone Type	Portability	Efficiency	Integrality
AST Based	Type – I, II, III	Low	High	Low
Token Based	Type – I, II	Medium	Low	High
Text Based	Type - I	High	High	Low
Metric Based	Type – I, II, III	Depends on Metrics used	High	Medium
PDG Based	Type – I, II, III	Low	High	Medium

Table 2 provides information on various approaches' efficacy, portability, and integrality depending on categories of discovered clones.

5. PROBABILISTIC SOFTWARE MODELING (PSM)

The software process is complicated with all interconnected components of requirements, features, revisions, modules, or software 2.0. In conventional software engineers, complexity-related concerns have numerous tools, techniques, and solutions to relieve problems (e.g., version control systems, requirements engineering, unit testing). Methods and instruments that incorporate analytics, testing, growth, integration, and maintenance may be adopted in future if artificial intelligence is tightly integrated into programme plans. These strategies have not yet been developed. [23].

Current PSM, modeling data-driven approach to software engineering predictive & generative approaches. PSM is an analytical process that constructs a program's probabilistic model to conventional software (e.g. Java). The PM enables developers to define programme semantics at the same level of abstraction as their source code (i.e. approaches, regions, or modules) without having to switch between project implementations or programming languages. It allows the benefits that are key in other areas of combinatorial optimization & complex formulas for software development (e.g.material simulation, medical biology, meteorology, economics,). In both traditional software and AI components with their unpredictability, PSM allows uses e.g. test case creation, semantic clone identification, or abnormal detection without delay. Our investigations show that PMs can model programs on which these applications build and enable causal reasoning & constant data creation. PSM has 4 major elements: Code, Runtime, Modeling & Inference. PSM contains four primary features. First, a programme structure (Code) is extracted with the assistance of static code analysis by PSM. Attributes, executable code, and types (for example, fields, methods, or classes in Java) make up various layers of abstraction. Secondly, it analyzes the behavior of the application by monitoring its runtime (Runtime). This provides access to properties & executable calls. This is defined as a structure & dynamic behavior that are then combined with PSM into a probabilistic model (Modeling). The primary contribution of this study is this phase as well. After that, anomaly detectors and test-case generators are used to calibrate the models by statistical inference .

A. PSM Applications

PSM is a general basis for a broad range of generative & predictive purposes. This section provides an assortment of available applications.

a) Predictive Applications

The aim is the measurement, visualization, inference, and prediction of a system's behavior & performance.

- **Visualization and Comprehension applications** Contribute to the understanding and behavior of the software. This involves viewing code components & non-functional characteristics, e.g. efficiency. PMs are visualization source that shows global but contextual behavior across the parts of code.
- **Semantic Clone-Detection applications** Intercept distinct, but conceptually identical sections of code, e.g. iterative as well as recursive algorithm version. Clone detection usually analyses pieces of source code focused on clones accurately or significantly adjusted. Semantic equivalence, though, does not have entirely static source code characteristics. By analyzing their models, PSM may discover technique-level clones. For example, the comparison was made by statistical analyses on statistical features, or by use of approaches like Q-Q visualization (complete manual decision), or combinations of sampled data.
- **Anomaly Detection applications** Measure the difference between such a persistent PSM model as well as an observable recently obtained. These technologies can be implemented in a live system that monitors and checks factors for their models. An improbable runtime monitoring threshold x (e.g., p(W eight = weightnew) <.1) is used to prompt extra measures due to a failure. x, as well as its implications on further aspects, may subsequently be explored for further decision-making through, for example, visualization and understanding approach.

b) Generative Applications

This is a useful insight from the frameworks, for example, operable inputs or property values.

- Test-Case Generation applications to produce test data, obtain observations via operational & property models. PSM may produce scanned test data for a given system situation with a certain probability or (system state). E.g., probability-scoped data may be utilized in the generation of various test cases, such as typical, rare, or invisible, by sampling x < P(Person) = P (Weight, Height) < y with pre-defined probability borders x and y. This improves overall process models with relevant, autonomously created, behavior-based tests.
- Simulation applications Example traces of operation designed to replicate the operating system from the network of models. It will probably be running without executing the original application. Simulators can link hardware-software interfaces & reduce the number of hardware dependencies in creation [23].

6. Related Work

A study has been conducted to examine the various methods currently in employed for clone detection in source code segments. It helps in identifying a variety of additional difficulties with clone detection in source code. This section provides a review of clone detection studies and associated fields.

Svajlenko & Roy (2021) introduced a benchmarking framework for mutation analysis that may be used not only to assess recalls of clone identification systems for various kinds of clones but also to evaluate particular types of clone modifications without manual effort. The system employs a clone synthesis editing taxonomy to create 1000 fake clones, injecting codes into bases & evaluating subject clones automatically using a method for mutation analysis. Furthermore, the framework

provides functionality in which individual clone pairs may also be utilized for the evaluation of an instrument's subject. This presents a chance to assess a tool's capability to find sophisticated type-4 clones or real-world clones without creating specialised mutation processes for certain situations. [24].

Thaller, Linsbauer, & Egyed (2020) The current semantic clone identification using the PSM system as a robust technique for the semi-semantically equivalent detection of the methodologies. PSM inspects the program structure and functionality and synthesizes the PMs network. Every PM in the recognized software method allows run-time events to be generated & evaluated. They use this to discover semantic clones properly. Findings indicate that the technique can recognize semantic clones with high efficiency & low error rates in conditions of syntactical similarity [25].

Yu et al. (2019) suggested a framework technique to detect semantic clones by using tree-based convolution, by both collecting structural code information from the AST as well as the code information lexical tokens for a code fragment. Furthermore, their method overcomes the restriction of the limitless vocabulary of tokens as well as models in the use of sources of lexical data from tokens frequently useless when it comes to unseen tokens. Specifically, they present a novel approach of embedding, known as position-aware character embedding (PACE), that mainly considers every token as a positional combo of single-hot character embedding. Their testing findings show that their method significantly improves previous state-of-the-art approaches by increasing F1 score from 0.42 to 0.15 in two prominent code-clone benchmarks (OJClone & BigCloneBench). PACE also shows that their technique is significantly more efficient when code clones have invisible tokens [26].

Sheneamer (2019) Suggests a detection tool for Java code obfuscation as well as for syntactic & semantic clones via integrating cluster data using CNN deep learning algorithm known as CCDLC. The CCDLC employs a new Java BDG along with PDG as well as the AST functionalities. To validate the efficacy of their approach, they employ numerous published code clones and Java obscured code datasets. Their testing findings and assessment show that the combination of classification as well as deep learning is a feasible approach because it improves clones detection and obfuscation code of corpus. The major advantage of this technique is that their tool may increase the accuracy of detection of shielding by 5.44% and enhance the accuracy of both clones of syntactic and semantic by approximately 12% [27].

Y. Yang et al. (2018) Concentration on an investigation and use of structural information to evaluate coding similitudes on function-level coding clones-based function. In order to create more abstract code descriptions, it first integrates a kind of (AST-Abstract Syntax Tree) that uses specified node types rather than a true node description. The method then computes the contrast scores between two pairs of code fragments at function level using Smith-Waterman local assessment technique. Trials carried out across 5 open-source datasets demonstrate that their approach can obtain 92.46 percent on average accuracy & up to 10.94 percent & 4.02 percent respectively above competitive methods. Meanwhile, testing findings reveal that in code clone identification their technique can reach an average of 90.73% of precision over cross-projects [28].

Misu & Sakib (2018) If they have comparable interfaces & execute comparable tasks, 2 techniques are prone to cloning. In this light, a new methodology is being presented to identify clones utilizing method interface similarities, which is a lightweight interface-driven code clone detection (IDCCD). First, the blocks of a technique from source files are tokenized. Interface information is collected & indexed using mapped tokens for these method block tokens. Identical interfaces from this index are then queried and the clone detection algorithm is similar to a similarity function. Using a BigCloneEval platform, IDCCD is evaluated alongside other cutting-edge methods. The testing findings show which IDCCD is identical in its efficiency to other less sophisticated current instruments [29].

Hu et al., (2017) attempt to implement a semantics-based solution to achieve the objective. This approach evaluates the binary functions in arguments as well as indirect jump targets, then emulates

their operation to obtain the semantic signatures that let us assess the similarity of such processes. The method has been put into action in a working model known as CACompare, which can identify duplicated functions across several architectures & compilation configurations. It can do binary analysis on Linux platforms, supporting comparisons between popular architectures (i.e. IA-32, ARM, & MIPS). The testing findings demonstrate that CA is capable of solving a wide range of issues associated with binary incompatibilities on diverse architecture including configuration-variant compilation configurations, Also, it is effective in binary code clone identification; nevertheless, compared to state-of-the-art methods, it yields high precision [30].

Sunayna et al., (2016) A number of studies have shown that between 5 and 20 percent of software systems include duplicate code as a result of copying in pieces of previously written code and that this wastes between 40 and 60 percent of an organization's work. Code duplication has the major drawback of necessitating the investigation of all related code pieces for the same issue if a flaw is found in one code fragment. Code clones may be spotted by using various clone detection methods, that improve software maintenance effectiveness & lowers the overall maintenance cost [31].

7. Conclusions and Future Work

This paper examined software cloning and code clone detection, discussing advantages & disadvantages of software cloning as well as the various categories of code clones that can be found in software systems. Methods for detecting code clones, including textual analysis, and tree-based analysis, among others. were also investigated. The potential method of probabilistic software modeling was also described; this method has been proved to be useful in detecting clones. Clone detection by graph analysis is only one of the numerous benefits of modeling code as a probabilistic graph, which has been proved to have widespread use in software engineering. We conclude that probabilistic software modeling is a feasible approach that may augment existing code clone detection methods, and that software cloning and recognition of code clones is a significant field of research. We believe that this review article serves as a helpful resource for scholars and practitioners in the area of software engineering since the detection and management of code clones will become more crucial as the complexity of software systems continues to expand.

Future research in this area might be focused in a number of different directions. To begin, studies may be undertaken to discover and create better methods for detecting code clones, such as hybrid approaches that integrate different methods in order to enhance clone detection accuracy. Second, there is a need for more research into the uses of probabilistic software modeling in software engineering, especially in the field of code synthesis, where probabilistic models may be used to automatically produce code from specifications. Finally, more study is required to learn how software quality is affected by code cloning and to create methods for effectively managing code clones to boost software maintainability.

References

[1] H. Min and Z. L. Ping, “Survey on software clone detection research,” 2019, doi: 10.1145/3312662.3312707.

[2] D. Rattan, R. Bhatia, and M. Singh, “Software clone detection: A systematic review,” *Information and Software Technology*. 2013, doi: 10.1016/j.infsof.2013.01.008.

[3] S. Jadon, “Code clones detection using machine learning technique: Support vector machine,” 2017, doi: 10.1109/CCAA.2016.7813733.

[4] A. Sheneamer, S. Roy, and J. Kalita, “An Effective Semantic Code Clone Detection Framework using Pairwise Feature Fusion,” *IEEE Access*, 2021, doi: 10.1109/ACCESS.2021.3079156.

[5] J. Zhang, X. Wang, H. Zhang, H. Sun, K. Wang, and X. Liu, “A Novel Neural Source Code Representation Based on Abstract Syntax Tree,” 2019, doi: 10.1109/ICSE.2019.00086.

[6] Y. Yuan, W. Kong, G. Hou, Y. Hu, M. Watanabe, and A. Fukuda, “From Local to Global Semantic Clone Detection,” 2020, doi: 10.1109/DSA.2019.00012.

[7] W. Hua, Y. Sui, Y. Wan, G. Liu, and G. Xu, “FCCA: Hybrid Code Representation for Functional Clone

Detection Using Attention Networks," *IEEE Trans. Reliab.*, 2021, doi: 10.1109/TR.2020.3001918.

[8] J. Zeng, K. Ben, X. Li, and X. Zhang, "Fast code clone detection based on weighted recursive autoencoders," *IEEE Access*, 2019, doi: 10.1109/ACCESS.2019.2938825.

[9] W. Wang, G. Li, B. Ma, X. Xia, and Z. Jin, "Detecting Code Clones with Graph Neural Network and Flow-Augmented Abstract Syntax Tree," 2020, doi: 10.1109/SANER48275.2020.9054857.

[10] Y. Meng and L. Liu, "A Deep Learning Approach for a Source Code Detection Model Using Self-Attention," *Complexity*, 2020, doi: 10.1155/2020/5027198.

[11] M. Lei, H. Li, J. Li, N. Aundhkar, and D. K. Kim, "Deep learning application on code clone detection: A review of current knowledge," *J. Syst. Softw.*, 2022, doi: 10.1016/j.jss.2021.111141.

[12] K. W. Nafi, T. S. Kar, B. Roy, C. K. Roy, and K. A. Schneider, "CLCDSA: Cross language code clone detection using syntactical features and API documentation," 2019, doi: 10.1109/ASE.2019.00099.

[13] D. Perez and S. Chiba, "Cross-language clone detection by learning over abstract syntax trees," 2019, doi: 10.1109/MSR.2019.00078.

[14] M. A. Alamri *et al.*, "Molecular and Structural Analysis of Specific Mutations from Saudi Isolates of SARS-CoV-2 RNA-Dependent RNA Polymerase and their Implications on Protein Structure and Drug–Protein Binding," *Molecules*, 2022, doi: 10.3390/molecules27196475.

[15] F. Al-omari, "Towards Semantic Clone Detection, Benchmarking, and Evaluation," 2021.

[16] R. Asghar *et al.*, "Wind Energy Potential in Pakistan: A Feasibility Study in Sindh Province," *Energies*, 2022, doi: 10.3390/en15228333.

[17] H. Zhang and K. Sakurai, "A Survey of Software Clone Detection from Security Perspective," *IEEE Access*. 2021, doi: 10.1109/ACCESS.2021.3065872.

[18] J. Svajlenko and C. Roy, "A Survey on the Evaluation of Clone Detection Performance and Benchmarking." 2020.

[19] S. U. Ahmed, M. Affan, M. I. Raza, and M. Harris Hashmi, "Inspecting Mega Solar Plants through Computer Vision and Drone Technologies," 2022, doi: 10.1109/FIT57066.2022.00014.

[20] V. Rohilla, M. Kaur, and S. Chakraborty, "An Empirical Framework for Recommendation-based Location Services Using Deep Learning," *Eng. Technol. Appl. Sci. Res.*, 2022, doi: 10.48084/etasr.5126.

[21] D. M. StefanWagner, "Chapter 3 - Analyzing Text in Software Projects," *Art Sci. Anal. Softw. Data*, pp. 39–72, 2015.

[22] M. Salman Khan, "A Topic Modeling approach for Code Clone Detection," UNIVERSITY OF NORTH FLORIDA SCHOOL OF COMPUTING, 2019.

[23] H. Thaller, L. Linsbauer, R. Ramler, and A. Egyed, "Probabilistic Software Modeling: A Data-driven Paradigm for Software Analysis." 2019.

[24] J. Svajlenko and C. K. Roy, "The Mutation and Injection Framework: Evaluating Clone Detection Tools with Mutation Analysis," *IEEE Trans. Softw. Eng.*, 2021, doi: 10.1109/TSE.2019.2912962.

[25] H. Thaller, L. Linsbauer, and A. Egyed, "Towards Semantic Clone Detection via Probabilistic Software Modeling," 2020, doi: 10.1109/IWSC50091.2020.9047635.

[26] H. Yu, W. Lam, L. Chen, G. Li, T. Xie, and Q. Wang, "Neural detection of semantic code clones via tree-based convolution," 2019, doi: 10.1109/ICPC.2019.00021.

[27] A. Sheneamer, "CCDLC Detection Framework-Combining Clustering with Deep Learning Classification for Semantic Clones," 2019, doi: 10.1109/ICMLA.2018.00111.

[28] Y. Yang, Z. Ren, X. Chen, and H. Jiang, "Structural Function Based Code Clone Detection Using a New Hybrid Technique," 2018, doi: 10.1109/COMPSAC.2018.00045.

[29] M. R. H. Misu and K. Sakib, "Interface Driven Code Clone Detection," 2018, doi: 10.1109/APSEC.2017.97.

[30] Y. Hu, Y. Zhang, J. Li, and D. Gu, "Binary Code Clone Detection across Architectures and Compiling Configurations," 2017, doi: 10.1109/ICPC.2017.22.

[31] S. Sunayna, K. Solanki, S. Dalal, and S. Sudhir, "Comprehensive Study of Software Clone Detection Techniques," *IOSR J. Comput. Eng.*, 2016, doi: 10.9790/0661-1804021519.