HBRP PUBLICATION

Journal of Advancement in Software Engineering and Testing
Volume 6 Issue 1

Towards Reliable Code Plagiarism Detection: A Survey on Software Clone Detection

Sanjay B. Ankali[1], Dr. S G Gollagi[2], Dr. Bahubali M. Akiwate[3],
[1]Associate Professor, Department of Computer science & Engineering, KLE College of Engineering and Technology, Chikodi-India
[2]Professor, Department of Computer science & Engineering, KLE College of Engineering and Technology, Chikodi-India
[3]Associate Professor Department of Computer science & Engineering, KLE College of Engineering and Technology, Chikodi-India
****Corresponding Author***
E-mail Id:-*sanjayankali123@gmail.com*

ABSTRACT
Despite substantial study over the past three decades resulting in the development of more than 250 clone detection technologies, there is no one framework that can accurately and reliably identify all four major types of clones. The lack of comprehensive, reliable, and language-neutral code clone detection has a significant negative influence on online learning systems like Coursera, which are unable to assess the proficiency of students in coding projects and assignments they submit to the online platforms. This survey paper can contribute to building more reliable code plagiarism detection by presenting various tools and techniques to find the same language and cross-language clone types with respect to the clone types they detect and the languages they work on. The paper highlights 3 major issues in terms of language agnostic nature and accuracy a) Most of the proposed techniques work only on a specific language like C, CPP, Java, or Python for detecting clones. b) Only 8 proposed works accurately classify all 4 basic clone types. c) 98% of the clone detection in the past is based on regular clones ignoring micro clones. The summary of the paper can provide proper directions in building a more reliable code plagiarism detection tool.

Keywords:- *Software clone detection, Code plagiarism**,** Clone types, Software Development Life Cycle*

INTRODUCTION

The practice of producing functionally comparable codes with syntactic alteration is known as software cloning or code cloning. Alternately, it can be described as pairs of semantically related code fragments with or without syntactical modification [1]. Numerous academics use various words to refer to this process, such as duplicate code [4,5], similar code [2], same code [3]. Large legacy systems have up to 30-50% of duplicate code, respectively, according to these two papers. According to the milestone and works of literature like [6,7,8,9], there are four different sorts of code clones that fall under the syntactic category:

Type 1 is commonly known as exact clones
Type 2 is also known as renamed clones
Type 3 is known as near-miss clones.
Type 4 is functionally similar clones that are implemented differently.

Different editing taxonomies provide the foundation for syntactic clones. A significant problem with earlier clone detections is that Type 4 clone detection is outside the capabilities of many outstanding clone identification techniques, like Siamese [12],

SourcererCC [11], and VUDDY [10]. These tools scale well on several MLOC repositories with excellent precision and recall. VUDDY is implemented to learn how Github repositories operate. VUDDY employs finger print matching of hash functions, and its body solely depends on its syntactic details. According to the results, the tool has processed 133,812 susceptible functions for each project in a second by processing 13.2 MLoC files of Github projects. Only Type 1, 2 clones are detected. The token-based clone detector SourcerrerCC performs clone detection and partial index construction in two steps. To limit the amount of false-positive clone pair filtering, it applies token position filtering to the filtered heuristics of sub-block overlaps. The tool processes 250 MLOC of code to classify upto nearmiss clone in a 4.5 days on a single system.

For scalable code indexing and retrieval, Siamese uses the high-performance text-based search engine "Elasticsearch." It works in two phases: the index phase and the query phase and performs multiple code representation, query reduction, and ranking function to enhance the performance of clone search. A corpus of 365 million lines of code yields several type 3 clones using tools in less than 8 seconds.

Although it addresses the most crucial aspect of scalability together with these three tools [13,14], and [15]), it falls short in terms of Type 4 clone identification. One of these [13] required 80 pricey student laboratories' expensive workstations to evaluate 400MLoC in two days. It is never easy to achieve recall, precision, and scalability all at once [12].
Code cloning is done for a variety of reasons, the main one being the development approach of employing strong system designs, code, and logic over and again [6]. Software cloning results from forking. Code must be copied and pasted because inheritance and polymorphism are not supported by the language [16].
Software developers are encouraged to look for code online by the abundance of MLOC that is available on the Internet. According to the poll of 72 developers conducted by vaibhav saini [17], before starting any coding task, 96% of developers prefer searching for the solution on internet and 33% of college students nationwide look for online code to complete programming tasks [18].

Current code plagiarism detection lacks in relating the clones types to the level of learning happens in submitting programming assignments by finding The good, the bad, and the ugly [19]. Code cloning has the largest impact on the Software Development Life Cycle many advantages like reusing the reliable, semantic, and syntactic constructs for system design, detecting library candidates, software maintenance through refactoring, and helps academia to detect code plagiarism [6]. Cloning supports software forking to create a variety of software products [4]. Despite debate between researchers whether cloning is harmful or not software cloning emerge an as fast and immediate way to address change requirement [8].

Code cloning has a bad impact on the design, bug propagation [6] we find evidence in [20] that around 75% of the cases bug pattern of the original code is duplicated “as-is” to the sibling that mainly increases maintenance cost. Zijian Jiang (2019) [21] in his previous research defines software maintenance as a complex process of editing different entities of programs like renaming identifiers, function name, class name to fix the bugs. He mentioned a scenario where 80% of multi edits are made by developers to fix the bugs which cost 70%

of the developer's time and resource. Studies on the detection of clones in the past have neglected micro clones. A recent study comparing software bugs in large and small clones [22] discovered that micro clones have 80% consistent update to the several files pervasively and are more likely to have severe bugs than large clones.

An effort to codify plagiarism detection began in late 1976 in an effort to support academic honesty. According to Chiver's survey from 2020 [23], 3 different techniques are used to identify code plagiarism. A technique that is attribute-based, structure-based or hybrid. Ottenstein (1976) [24] conducted the first attribute counting experiment. (Halstead, 1973)[25]

Investigation is based on the metrics consisting of many tokens with distinct operators and operands served as the foundation. The current metrics of [24] included conditional statements, looping statements, and other tokens like white space and lines. In the year 1981 (Grier, 1981),[26] added 16 more qualities. Programming features like loops and conditional statements are counted in a study[27].

An empirical method for detecting code similarity based on 24 metrics proposed by Faidhi and Robinson (1987) [28]. These early investigations relied solely on text or strings and tallied all of the program's properties. In a comparative research, (Whale., 1990) [29] debates that the structural aspects of the code and application specific metrics are to be considered.

RESEARCH QUESTION

1. How many clone detection tools are introduced in last 3 decades to find same language and cross language clone types?
2. Is there a reliable, complete and language agnostic clone detector?
3. Do the existing tools find all four types of clones?
4. Understand the more accurate clone detection method for code plagiarism detection.

Major contributions of the paper

- With regard to the detection of all four clone types and the language they work on, the paper highlights the most significant contributions in same language & cross language clone detections.
- The paper introduces the recent cross language clone detection techniques.
- The paper gives insights about the usage of clone detection approaches with respect to the accuracy in finding the all four clone types.

BACKGROUND

Clone detection in same language

In this section, we demonstrate our grasp of clone types with examples based on [6]. There are nine different sorts of clones, says [8]. On the basis of the taxonomy of editing, we describe 4 fundamental categories of clones. Comparative analysis [7] has shown numerous editing scenarios for each categories of clone. Few sample cloned codes from our case studies are presented below.

Type-1 clone: Similar codes (semantic & syntax) with change in comments and spaces [1].

```
main()    // summing code
{
int p=21,q=25, r;
r=p+q;      //logic
printf("addition is=%d",r);
```

```
main()
{ // addition code
int p=21,q=25, r;
r=p+q;      //main code
printf("addition is=%d",r);
```

```
}
```
Code-1

```
}
```
Code-2

Type 1 clones include such like Code-1 and 2. These are also known as copy/paste clones or exact clones. To prohibit students from engaging in distasteful learning habits and to prevent breaches in software integrity in the business world.

Type-2: Syntactically similar code created by multiple entity edits such as change in the variable name, function name, and class name.

```
main()
{
int f=1,index=1,number;
printf("Enter the +ve integer number");
scanf("%d",&number);
while(index<=number)
{
   f=f*index;
    index++;
}
printf( " factorial of a number is=%d", f);
}
```
Code-3

```
main()
{
int result=1,i=1,num;
printf("Enter the number");
scanf("%d",&num);
while(i<=num)
{
  result=result*i;
  i++;
}
printf("factorial is=%d", result);
}
```
Code-4

Type 2 clones include those of codes 3 and 4. Renamed clones are another term for type 2 clones. This terrible practice of renaming the various code entities, such as identifier, method name, and class name, still violates academic integrity.

Type 3: types-3 clones are the subset of type 2 codes that are created by addition or deletion of lines. Following code-1 and code-5 snippets are the example for type-3 clone

```
main()    // addition program
{
int m=22, n=32, sum:
sum= m+n;
printf("Hello everyone sum is=%d”,
sum);
}
```
Code-1

```
main()    // addition program
{
int m=22, n=32, sum:
sum= m+n;
printf(“here we add two values”);
printf("Hello everyone sum is=%d”,
sum);
}
```
Code-5

Generating the type-3 clones are treated as a bad coding practice in academia and are matter of interest to the clone detection research [7].

Type 4: codes that behave similarly and implemented differently (recursion & iteration) are called as type-4 clones or functional clones.

```
int func(int n)
{
if (n==0)
  return 1;
else
```

```
main()
{
int index=1,f=1,num;
printf("Enter the + ve integer
number");
```

```
return (n* func(n-1));
}
```

Code-6

```
scanf("%d",&num);
while(index<=num)
{
  f=f*index;
   index++;
}printf("factorial of number
is=%d",f);
}
```

Code-7

Code-6 and Code-7 are semantic clones, sometimes known as type 4 clones. Both business and academia are interested in type 4 clones. These kinds of clones were not picked up by any of the text-based, token-based, or tree-based detection approaches discussed in the [7]. Excellent scalable clone detection technologies weren't included in the study's scope despite being introduced in section 1. The main problem with current code plagiarism detection technologies is that they label these codes as clones, although from an academic standpoint, these techniques raise students' levels of learning.

Following section introduces the cross language code clones with .c and .cpp code snippets and present the most significant detection techniques in cross clone detection [7].

Cross Language Clone Detection

Definition: According to the definition of clones, two or more codes that provide the same output but are implemented in different languages with various four sorts of differences are referred to as cross-language clones. For a instance the C program that find factorials of numbers is referred to as cross clones of C++ code that do find factorial of a number [30]. following C and CPP code snippets were used as case studies for the demonstration.

```
1.  main()
2.  {
3.  float p=11,q=22,r;
4.  r=p+q;
5.  printf("sum of two numbers=%d", r);
6.  }
```

C-code-1

```
1.  main()
2.  {
3.  float p=11,q=22,r;
4.  r=p+q;
5.  cout<<"sum of two numbers ="<<r;
6.  }
```

CPP-code-1

The above snippets are the gem examples of cross language type-1 clone

```
1.  main()
2.  {
3.  int r=7,s=77,t;
4.  t=s+t;
5.  printf("sum of two numbers=%d",t);
6.  }
```

C-code-2

```
1.  main()
2.  {
3.  int x=7,y=77,z;
4.  z=x+y;
5.  cout<<"sum of  two numbers is
="<<z;
6.  }
```

CPP-code-2

The above CPP example is the copied C code with the change in the variable names.

```
1.  main()
```

```
1.  main()
```

```
2.  {
3.  int index,f=1,num;
4.  printf("Enter the number");
5.  scanf("%d",&num);
6.  for(index=1; index<=num; index++)
7.  f=f*index;
8.  printf("factorial of number
is=%d",f);
9.  }
```

C-code-3

```
2.  {
3.  int index=1,f=1,num;
4.  cout<<" Enter the number";
5.  cin>>num;
6.  while(index<=num)
7.  {
8.  f=f*index;
9.  index++;
10. }
11. cout<<"Factorial of Given Number
is ="<<f;
12. }
```

CPP-code-3

In spite of similar semantics, above C and CPP implementations of factorial finding codes generate different pare tree structures and are most challenging to detect.

RELATED WORK

SAME LANGUAGE CLONED CODE DETECTION

Software clone detection has been the subject of extensive research. We categorise all the tools and techniques under five classes based on detection techniques, including *text-based, token-based, tree-based, PDG-based, and metric-based,* and we briefly discuss the problems these tools face as mentioned in [7]. This is based on the seminal literature works of [7], [8]. We summarize clone detection techniques/methodologies and divide the chronology into two time periods: before 2010 and after 2010.

Clone detection research till 2010

Clone detection based on text similarity: The tools like simian [32], EqMiner[33], Duploc[5], (Johnson, 1994) [31], NICAD [34], DuDe [35] have efficiently present the text similarity of two code fragments. These great tools limit in their capabilities to detect type 3 clones (except NICAD). The tools [31],[5],[35] perform better in detecting type 1 clone and most of the text based tools were good in detecting type 1 and 2 clones.

Token-based Techniques: To extract the tokens from the source code, lexical analysis is used. The suffix tree or suffix array is created using these extracted tokens for matching. D-CCFinder, CP-Miner, iClones, Dup, and CCFinder are the few examples of such tools [36, 37]. These tools have successfully identified types 1 and 2, and the tool CP-Miner made a modest effort to identify type 3.

Tree-based Techniques: work by parsing the source code to parse tree or abstract syntax tree. These approaches perform better in case of code refactoring and proved to be better with precision of clone detection [39]. Many approaches/tools like CloneDigger [43], Deckard [40], CloneDR [41], simScan[32] , Asta [42], sim [44], ClemanX [45], cpdetector [48] , JCCD API [46], CloneDetection [47].

PDG-based Techniques: the control and data flow of source code is converted to get PDG and similarity metrics are applied to find the clone pairs. These techniques have proved to be the most suitable candidates to detect type 4 clones. Tools like Scorpio [35], PDG-DUP[49], Duplix[2], and Choi[50] have detected the functional similarities between the 2 codes.

Metrics-based Techniques like CLAN/Covet [51], Antoniol[52], and Dagenias [53] work by counting the number of different token classes and stores them in a matrix/ tables and apply similarity matching on the matrices to get the clones pairs. Metric based techniques are proved to be the false positives in detecting type 3 and 4 clones.

Clone detection research after timeline 2010

Detecting software clones or code plagiarism is not an easy task. All the mentioned tools in this section do excellently well in detecting clone types they are aimed to and in terms of scalability to the large repositories. In table 1 presents text based clone detection tools/techniques in terms of types of clones they detect and language they work on based on the literature survey of [1],[9].

Table 1:*-Text based tools/techniques*

Sl.No	Tool/Author/Citation	Language Supported	Clone Type detection
1	(Ragkhitwetsagul C. a., 2017) [54]	Java	1,2,3
2	(Kim S. a., 2018) [55]	C/C++	1,2
3	(Jadon, 2016) [56]	C	3
4	(D Yu, 2017) [57]	Java	1,2,3
5	VUDDY (Kim S. S., 2017) [10]	C/C++	1,2
6	(Y Nakamura, 2016) [58]	HTML, Javascript	3
7	(Lyu, 2016) [59]	Layout XML Files	1,2,3
8	(Xue et al., 2020) [60]	Assembly	1,2,3
9	(Chen, 2015) [61]	Java	1,2,3
10	(Thaller, 2017) [62]	C/C++, ST	1,2
11	(Newman, 2016) [63]	C/C++	1,2
12	vfdtect (Liu, 2017) [64]	C/C++	1,2
13	(E. Kodhai, 2010) [65]	C	1,2
14	(Maeda, 2010) [66]	Java	1
15	CCCD (Shihab, 2013) [67]	C	3,4
16	(A. Cuomo, 2012) [18]	Java	2
17	SimCad(M. S. Uddin, 2013) [68]	Java	1,2,3
18	(S. Park, 2013) [69]	C	Product similarity
19	(J.-S. Lim, 2011) [70]	C	No classification
20	(B. Lesner, 2010) [71]	Any	No classification
21	(Yadav, 2013) [72]	Java	1,2,3

Table 2 presents' token based clone detection tools/techniques. Detecting clones up to type 3 and tools like SourcereCC, Saimese, ScaleClone, Li, and Nishi scale well on large data sets like IJaDataSet which is the major contributions to clone detection research.

Table 2:-*Token based tools/techniques*

Sl.No	Author/Citation	Language Supported	Clone Type detection
1	(Nishi, 2018) [73]	IJaDataset 2.0[i] (Java)	1,2,3
2	Saimese (Ragkhitwetsagul et al., 2019) [12]	Java	1,2,3
3	(Tekchandani, 2017) [74]	N/A	1,2
4	ScaleClone(Farhadi, 2015) [75]	Assembly	1,2,3
5	CCAligner (Wang P. J., 2018) [76]	C, Java	1,2,3
6	(Yuki, 2017) [77]	Java files	1,2,3
7	SourcererCC (Sajnani, 2016) [11]	IJaDataset(Java)	1,2,3
8	(Semura, 2017)[78]	From Rosetta Code[ii]	1,2
9	(Li L. H., 2017) [79]	IJaDataset[i] (Java)	1,2,3
10	(J. Y. Poon, 2010)[80]	Java	No type classification
11	(Toomey, 2012)[81]	Unspecified	1,2
12	(Roy J. S., 2017)[82]	Java	Near miss
13	SHINOBI(S. Kawaguchi, 2009) [83]	C,C++, C#	1,2,3
14	CodeEase(S. Abid, 2017)[84]	Java	structural
15	SaCD(Qing Qing Shi, 2013)[85]	Java/C/C++	1,2
16	Boreas(Guo, 2012)[86]	Java	1,2
18	(Y. Semura N. Y., 2018)[87]	Any	1,2,3
19	(Merlo, 2012)[88]	Java	1,2,3
20	(Koschke, 2009) [38]	Java/C	1,2,3
21	(M. Elsabagh, 2018)[89]	Java	1,2
22	(B. Hummel, 2010) [15]	C	1,2
23	(M. Dong, 2012)[90]	Binary	No classification
24	CCfindersw(Y. Semura N. Y., 2017)[78]	Any	1,2
25	NICAD (Roy J. R., 2011)[91]	C/C#/Java/ Python/WSDL	Near Miss
27	(Bharti, 2014)[92]	C/C++	1,2,3
28	(Rilling, 2013)[93]	Any	1,2,3

Table 3 presents a tree based clone detection tools/techniques. These tools are designed mainly for the refactoring purpose and to achieve high precision.

Table 3:-*Tree based tools/techniques*

Sl.No	Author/Citation	Language Supported	Clone Type detection
1	(Yang, 2018) [94]	Java	Function
2	(Pati, 2017) [95]	ArgoUML	1,3
3	(Lavoie, 2019) [96]	Java	3,4
5	Clonemerge (Narasimhan, 2015)[97]	C/C++	Near miss
6	(Y. Yang, 2018) [98]	Java	1,2,3
7	(J. Zeng, 2019) [99]	Java	1,2,3,4
8	OOP (D. Li, 2014) [100]	Java/PHP	1,2
9	(Thompson, 2011) [101]	Erlang	Structural

Table 4 presents clone detection based on the metric. The main objectives is to detect all 4 clone types. We did not find the results to prove the proper classification of tools.

Table 4:-*Metric based tools/techniques*

Sl.No	Author/Citation	Language Supported / Dataset	Clone Type detection
2	(Svajlenko, 2017) [102]	IJaDataset[1] (Java)	1,2,3
3	(Sudhamani, 2016) [103]	C,CPP,Java	1,2,3,4
4	(Haque et al., 2016) [104]	N/A	1,2,3,4
5	Vincent (Ragkhitwetsagul et al.,2018) [105]	Java	1,2,3
8	(Y. Fukushima, 2009) [106]	Java	Structural
9	(Kusumoto, 2011) [107]	Java	1,2,3,4
10	(Singh R. a., 2017) [108]	Java	1,2,3,4

Table 5 presents the clone detection using semantic/PDG approach. These studies aim at finding the clone types that was not handled by text, token, and tree based techniques.

Table 5:-*Semantic/PDG based tools/techniques*

Sl.No	Author/Citation	Language Supported/Dataset	Clone Type detection
1	(Wang M. P., 2017) [109]	C	4
2	(Sabi, 2017) [110]	Java	1,2
3	(Crussell, 2015) [111]	Java	N/A
4	(Sargsyan, 2016)[112]	C	4
5	(Hu Y. Y., 2017)[113]	Assembly	4
6	(Kamalpriya, 2017)[114]	Java	4
7	(Avetisyan, 2015)[115]	C	1,2,3,4

8	(Leavens, 2012)[116]	Java	3,4
9	(Deepika, 2013)[117]	C#	File similarity
10	(R. Tekchandani, 2013)[118]	Any	Semantic
11	Agec (Kamiya T. , 2013)[119]	Java	Semantic
12	SeByte(I. Keivanloo, 2012)[120]	Java	1,2,3,4
13	(D. Yu, 2019)[121]	Java	No classification
14	(Kamiya et al., 2012) [37]	C	Structural
15	(Singh C. M., 2017)[122]	Java/C	1,2,3
16	(M. Wang, 2017)[123]	Java	1,2,3
17	(Y. Higo U. Y., 2011)[124]	Java	1,2,3
18	(Z. Xing, 2011)[125]	Java	1,2,3

Table 6 shows the hybrid clone detection tools that combine the key usage from subset of previous detection techniques. These involve complex computational tasks to achieve the completeness to clone detection process.

***Table 6**:-Hybrid clone detection tools/techniques*

Sl.No	**Author/Citation**	**Language Supported/Dataset**	**Clone Type detection**
1	(Misu M. R., 2017) [126]	IJadataset2.0[i] (Java)	1,2,3
2	(Sheneamer A. a., 2016)[127]	IJadataset2.0[i] (Java)	1,2,3,4
3	(Vislavski, 2018)[128]	Java,JavaScript,C,modula-2,scheme	1,2,3
4	(Ghofrani, 2017)[129]	Any	4
5	(Akram, 2018)[130]	Java	1,2,3
6	(Sheneamer A. S., 2018)[131]	Java	1,2,3,4
7	(Matsushita, 2017)[132]	ML Programs	1,2,3
8	(Kodhai, 2014)[133]	C, Java	1,2,3,4
9	(Uemura, 2017)[134]	HDL Code	1,2
10	(Nasirloo, 2018)[135]	C	4
11	(Singh M. a., 2015)[136]	C,C#,Java,Text files	Structural
12	(White, 2016)[137]	Java	3

In addition to the works described up until 2019, we also introduce two new works that were released in 2020. Twin-Finder [138] is a revolutionary method that combines symbolical execution of the methodology and machine learning-based clone verification method to attain precision. Step 1 of the strategy involves three steps. It uses static analysis to do domain-specific program slicing, which

includes isolating code using forward-backward slicing on each variable, dependency analysis, and pointer selection. In the next step the technique applies DECKARD for comparing weighted similarities of AST consisting of the most important feature vectors. The model uses recursive sampling for verification in order to validate the accuracy of the clones found.
Studies on the detection of tree-based clones in the past did not scale to huge repositories. Wang's [1] highly scalable tree-based clone detection works in two stages. In 1st stage, it builds flow augmented-AST by adding edges to indicate control flow and data flow. 2. Converts the AST into vector representation using the GNN and GMN (graph matching network). 3. Performs clone classification using similarity metrics. The technique is limited to the 1) semantic clone detection for Java code. 2) The flow-augmented AST has the limited AST information declaration and definition of methods and classes.

Cross Language Software Clone Detection

Compared to the similar language clone detection, we find very few significant works in finding cross language clone detection. The below section presents the most significant contributions.

3.2.1 Cross language clone detection has the major application in open source software categorization. The work CroLSim [139] detects the cross language software similarity in four phases.
A. Finding correlation between API and library methods through the continuous bag of words.
B. Filtering commonly used methods through SVD (singular value decomposition).
C. Determining semantic similarity between cross language software application using Doc2Vec model and cosine similarity.
D. Use of KNN algorithm to perform clustering to group similar applications.
Issues: Searches functionally similar code from the repository with only 28% precision.

3.2.2 Cross language clone identification is presented by the semantic cross clone detection program SLACC[140] that works on the input / output behavior of code. The SLACC finds duplicate code in a dynamically typed languages Java and python. The method works in the ways that follow.
A. The target code base chunked into smaller code snippets. *B.* Formation of blocks from the snippet functions grouped into declaration, assignment, blocks, loops, return value. *C.* generation of inputs for primitive objects, arrays, files using grey box testing. *D.* execution of functions on the generated input sets to store return values. E. measuring similarities of executed functions using jaccard similarity. E. clustering to group functions into clones.
Issues A. Do not support complex and long types of python
B. Works fine for semantic similarities.
C. Do not support more granular cross language clone types classification.
D. Dead code elimination

3.2.3 The tool [141] works in 4 steps. *A.* tokenization of source code using ANTLR. *B.* Application of Karp-Rabin algorithm to find the vector similarity. *C.* Creating language specific indexes by application of TF-IDF weighting. And finally *D.* Displaying the ranking based on similarity.

Issues A. Do not detect type 2 clones.
B. Computationally complex because of intermediate code generation.

3.2.4 (Perez, 2019) [142] proposed the cross-language clone detection approach based on the AST. The method is the semi-supervised machine learning model which uses the tree-based skip grammar algorithm and a token level vector generation to detect cross-clones. The method operates in three steps. A. Generation of token level vectors. Training the data sets, then, is step B. C. Siamese architecture-based clone detection stage.

Issues A. Shows only 75% confidence in detecting cross clones of Java and Python.
B. Do not perform clone type classification.

3.2.5 (K. W. Nafi, 2019) [143] proposed the tool CLCDSA which is more scalable and works with the action filters to filter out non-probable clones. It works in following steps.
A. Feature selection for selecting 9 features out of 24 presented by (Saini, 2018) [144] that are applicable to cross language clone detection.
B. Preprocessing to remove tokens, strings literals, and comments generated by ANTLR.
C. Finding similarity of API call.
D. feature metric extraction and similarity detection based on neural network model based on Siamese architecture.

Issues: Do not perform clone type classification.

3.2.6 BiNN's based technique was proposed by (Nghi D. Q. Bui, 2017) [145] which finds similarity in the structure based AST Using BTBCNN. It is based on 3 major constructs i. BiNN's using softmax for classification of structures. ii. Variation of tree based convolution neural network to encode each AST. iii. Unicode AST in multiple programming languages. The method has got 80% precision in program classification

Issues: A. The large codebase will slow down the training process.
B. Do not perform type classification.

3.2.7 Hu Y. a. (2017) [146] suggests a binary instruction-based method to identify semantically related functions. The technique works as follows
A. function argument reorganization using no. of arguments, return value.
B. Switch index branch target detection.
C. Semantic signature generation.
D. Signature comparison using Jaccard Similarity.

Issues: A. Works well only to detect type 4 clones (functional clones).
B. unreliable to code obfuscation.

3.2.8 LICCA [147] is an integrated tool that works by modified longest common subsequence algorithm on an enhanced concrete syntax tree (eCST) of source code.
Issues: limited to semantic clone detection

3.2.9 The first method for cross-clone detection for the Java & C# languages without intermediate code was proposed by (Cheng X. P., 2017) [148]. The method involves employing four stages to take revision histories that capture diffs that show changes in the software system through file differences. Technique works in the manner
A. Performs Log parsing to extract *diffs* from the version control system to find software evolution and their attributes from revision logs.
B. Performs Normalization where *diffs* are normalized to remove comments, punctuations so that the resulting text is ready for string matching next step.
C. Performs matching using "Bag of Words" to find the nearest file.

D. Reports cross language clones diffs similarity level.

Issues

A. Precision is less because of the weak correlation between attributes and *diffs*.

B. Performs well only with latest revision.

C. Not a language agnostic.

3.2.10 Al-Omari (2012) proposed the idea of clone detection within the.NET language family [149]. The process works by converting the .NET code into common intermediate language (CIL). It uses eight distinct filtering techniques in the second stage to lessen the noise in the CIL instructions and boost recall. The Common Intermediate Language of the .NET framework is then used to detect clone pairs by application of SimCad, NICAD, or Levenshtein Distance.

Issues.

A. No proper results to prove efficiency.

B. Works only for .NET family code.

3.2.11 The research by Lawton Nichols (2019) [30] extends earlier work [148] that identifies syntactic similarity using structural and nominal similarities. The approach now supports (C++, Java, and JavaScript) and operates on functions rather than VCS diffs. The procedure functions as shown below.

A. Generate parse tree using ANTLR grammar.

B. Normalize the parse tree to remove unnecessary length.

C. Map the different parse tree to find matching.

D. Apply preorder traversal to obtain linearity to matching result;

E. Apply "Smith Waterman local sequence alignment algorithm" on literalized functions.

F. Present the amount of matching in terms of clones.

Issues

A. Works well on small code repository.

B. Preprocessing hampers the time complexity.

C. Designed for object oriented programming languages.

SUMMARY

The below table 7 summarizes the number of clone detection tools developed to work on the coding language to find various clone types.

***Table 7**:- Clone types and techniques*

Clone type	**Number of Tool/Techniques**
Type-1	71
Type-2	71
Type-3	55
Type- 4	19
All 4 types	12
Function clones	3
File clone	1

There are just 12 ways that can identify all four types of clones, and a maximum of 68 tools can detect clones in Java, C, or C++ code.

Despite the enormous number of studies that have been done on clone detection, we still do not have a complete and accurate method for detecting clones in

submitted coding assignments and projects.

CONCLUSION

In this study, we present a survey of software clone detection tools and methods proposed in last 4 decades for identifying different sorts of code clones across all four clone categories and programming languages. We discover a serious lack of complete and language clone detection techniques that can significantly contribute in building reliable code plagiarism detection tools, despite significant research that happened in the last three decades introducing many scalable and reliable software clone detection tools and techniques. This study shows that AST-based tools are more accurate at detecting all four types of clones than text- and token-based methods. The capabilities of the ANTLR parser has created new opportunities for more accurate, thorough (identify all four types of clones), and language-neutral (any programming language) software.

1. Dr. Sanjay Ankali is currently working as an Associate Professor in the Department of CSE at KLECET, Chikodi, India-591201 having 12 years of teaching and 7 years of research experience. He obtained his Bachelor Degree in Computer Science & Engineering, M.Tech in Computer Networking & Ph.D. in Computer/ Information Science from VTU, Belagavi. His research interest is in the field of Software Engineering, Software clone detection and code plagiarism detection.

2. Dr. Bahubali M. Akiwate is an Associate Professor in the department of Computer Science and Engineering at KLE College of Engineering and Technology, Chikodi, Karnataka, India with more than 11 years of experience in teaching and research. Research areas of interest include Cryptography, Information Security & Privacy, and Networking. He received a Bachelor of Engineering degree in Computer Science and Engineering from Bahubali College of Engineering, Shravanabelagola, affiliated to VTU, Belagavi, during the year 2009. He completed M.Tech Degree in Digital Communication and Networking from Gogte Institute of Technology, Belagavi, affiliated to VTU, Belagavi during the year 2011. He completed Ph.D in Computer Science and Engineering from Visvesvaraya Technological University, Belagavi, Karnataka, India in the year 2022.

3. Dr. Shantappa G. Gollagi *is* currently working as a Professor in the Department of CSE at KLECET, Chikodi, India-591201 having 24 years of teaching and 7 years of research experience. He obtained his Bachelor Degree in Computer Science & Engineering from BVB College of Engineering & Technology, Hubli M.Tech. in Computer Engineering from College of Engineering, Pune & Ph.D. in Computer/ Information Science from VTU, Belagavi. His research interest is in the field of software Engineering, Image processing and Pervasive Computing

REFERENCES

1. Wang, W. L. (2020). Detecting Code Clones with Graph Neural Network and Flow-Augmented Abstract Syntax Tree. *27th International Conference on Software Analysis, Evolution and Reengineering (SANER)* (pp. 261-271). IEEE.
2. Krinke, J. (2001). Identifying similar code with program dependence graphs. *Proceedings of the 8th Working Conference on Reverse Engineering (WCRE'01),* , (pp. 301–309). Stuttgart, Germany.
3. I. D. Baxter, A. Y. (1998). Clone detection using abstract syntax trees. *Proceedings of the 14th International Conference on Software Maintenance (ICSM '98),* , (pp. Bethesda, Maryland, USA, 1998, pp. 368–). Bethesda, Maryland, USA.
4. Godfrey, C. K. (2006). clones considered harmful. *Reverse Engineering(WCRE'06)* (pp. 19-28). Benevento, Italy: IEEE.
5. Ducasse, S. R. (1999). A language independent approach for detecting duplicated code. *International Conference on Software Maintenance-1999 (ICSM'99)* (pp. 109-118). IEEE.
6. Chanchal Kumar Roy, J. R. (2007). A survey on software clone detection research. *Queen's School of Computing TR* , 64-68.
7. Chanchal K. Roy, J. R. (2009). Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Science of Computer Programming* , 470-495.
8. Dhavleesh Rattan, R. B. (2013). Software clone detection: A systematic review. *Information and Software Technology* , 1165-1199.
9. Ain, Q. U. (2019). A systematic review on code clone detection. . *IEEE access* , 86121-86144.
10. Kim, S. S. (2017). VUDDY: a scalable approach for vulnerable code clone discovery. *In Security and Privacy (SP), 2017 IEEE Symposium* (pp. 595-614). San Jose, CA, USA: IEEE.
11. Sajnani, H. V. (2016). SourcererCC: scaling code clone detection to big-code. *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference* (pp. 1157-1168). Austin Texas: IEEE/ACM.
12. Ragkhitwetsagul, C. K. (2019). Siamese: scalable and incremental code clone search via multiple code

representations. *Empir Software Eng.* , 2236–2284.

13. S. Livieri, Y. H. (2007). Very-Large Scale Code Clone Analysis and Visualization of Open Source Programs Using Distributed CCFinder: D-CCFinder. *29th International Conference on Software Engineering (ICSE'07)* (pp. 106-115). Minneapolis, MN: IEEE.
14. Koschke. (2014). Large-scale inter-system clone detection using suffix trees and hashing. *Journal of Software: Evolution and Process,* , 747–769.
15. B. Hummel, E. J. (2010). Index-based code clone detection: incremental, distributed, scalable. *IEEE International Conference on Software Maintenance* (pp. 1-9). Timisoara, Romania: IEEE.
16. Kim, M. B. (2004). An ethnographic study of copy and paste programming practices in OOPL.. *International Symposium on Empirical Software Engineering, 2004. ISESE '04.* (pp. 83-92). Redondo beach, CA, USA: IEEE.
17. Vaibhav Saini, H. S. (2015). *Instant Clone Finder: Detecting Clones During SoftwareDevelopment*. Retrieved April 12, 2020, from https://aftabhussain.github.io: https://aftabhussain.github.io/documents/pubs/tech-report15-instacf.pdf
18. A. Cuomo, A. S. (2012). A novel approach based on formal methods for clone detection. *2012 6th International Workshop on Software Clones (IWSC)* (pp. 8-14). Zurich: IEEE.
19. [19] Ossher, J. &. (2011). (2011). File cloning in open source Java projects: The good, the bad, and the ugly.. *IEEE 27th International Conference on Software Maintenance, ICSM 2011* (pp. 283-292). Williamsburg, VA, USA: IEEE.
20. Lopes, H. S. (2013). A parallel and efficient approach to large scale clone detection. *International Workshop on Software Clones (IWSC)*, (pp. 46-52). San Francisco: IEEE.
21. Zijian Jiang, Y. W. (2019). Automatic method change suggestion to complement multi-entity edits. *Journal of Systems and Software* .
22. J. F. Islam, M. M. (2019). A Comparative Study of Software Bugs in Micro-clones and Regular Code Clones. *," 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)* (pp. 73-83). Hangzhou, China: IEEE.
23. Chivers, K. a. (n.d.). *https://www.researchgate.net/publication/337953514*. Retrieved April 2020, from researchgate: https://www.researchgate.net/publication/337953514
24. Ottenstein, K. J. (1976.). An algorithmic approach to the detection and prevention of plagiarism. *ACM SIGCSE Bulletin* , 30–41.
25. Halstead., M. H. (1973). An experimental determination of the “purity” of a trivial algorithm. *ACM SIGMETRICS Performance Evaluation Review* , 10–15.
26. *the Twelfth SIGCSE Technical Symposium on Computer Science Education.* New York, NY, USA: Association for Computing Machinery.
27. J. L. Donaldson, M. P. (1981). A plagiarism detection system. *ACM SIGCSE Bulletin* , 21-25.
28. Robinson, J. A. (1987). An empirical approach for detecting program similarity and plagiarism within a university programming environment. *Computers & Education,* , 1–19.
29. Whale., G. (1990). Software metrics and plagiarism detection. *Journal of Systems and Software* , 131–138.

30. Lawton Nichols, M. E. (2019). Structural and Nominal Cross-Language. Clone Detection In: , LNCS 11424. *FASE 2019* (pp. 247–263). Prague: Springer.
31. Johnson. (1994). Substring matching for clone detection and change tracking. *International Conference on Software Maintenance (ICSM)* (pp. 120-126,). Victoria, BC, Canada: IEEE.
32. L. Barbour, H. Y. (2010). A technique for just-in time clone detection. *Proceedings of the 18th IEEE International Conference on Program Comprehension (ICPC'10),* , (pp. 76–79.). Washington DC, USA.
33. S. Ducasse, O. N. (2006). On the effectiveness of clone detection by string matching, . *Journal on Software Maintenance and Evolution: Research and Practice* , 37-58.
34. Cordy, C. K. (2008). NICAD: Accurate Detection of Near-Miss Intentional Clones Using Flexible Pretty-Printing and Code Normalization. *16th IEEE International Conference on Program Comprehension,* (pp. 172-181). Amsterdam: IEEE.
35. Y. Higo, U. Y. (2011). Incremental code clone detection: A pdg-based approach. *18th Working Conference on Reverse Engineering* (pp. 3-12). NW Washington, DCUnited States: IEEE.
36. Baker, B. S. (2007). Finding clones with dup: Analysis of an experiment. *IEEE Transactions on Software Engineering, 33(9), ,* 608-621.
37. Kamiya, T. (2013). Agec: An execution-semantic clone detection tool. *21st International Conference on Program Comprehension (ICPC)* (pp. 227-229). San Francisco, CA, USA: IEEE.
38. Koschke, N. G. (2009). Incremental clone detection. *13th European Conference on Software Maintenance and Reengineering* (pp. 219-228). Kaiserslautern, Germany: IEEE.
39. Li, Z. a. (2004). CP-Miner: A Tool for Finding Copy-Paste and Related Bugs in Operating System Code. *Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation - Volume 6* (p. 20). San Francisco, CA: USENIX Association.
40. L. Jiang, G. M. (2007). DECKARD: Scalable and accurate tree based detection of code clones. *Proceedings of 29th International Conference on Software Engineering (ICSE'07)*, (pp. 96-105). Minneapolis, MN, USA.
41. I. D. Baxter, A. Y. (1998). Clone detection using abstract syntax trees. *Proceedings of the 14th International Conference on Software Maintenance (ICSM '98),* , (pp. Bethesda, Maryland, USA, 1998, pp. 368–). Bethesda, Maryland, USA.
42. W.S. Evans, C. F. (2009). Clone detection via structural abstraction,. *Software Quality Journal* , 309–330.
43. A. Corazza, S. D. (2010). A tree kernel based approach for clone detection. *Proceedings of the 26th IEEE International Conference on Software Maintenance (ICSM'10)* (pp. 1-5). Timisoara, Romania: IEEE.
44. D. Gitchell, N. T. (1999). Sim: a utility for detecting similarity in computer programs, . *ACM SIGCSE Bulletin 31 (1)* , 266–270.
45. T.T. Nguyen, H. N.-K. (2009). ClemanX:Incremental clone detection tool for evolving software. *Proceedings of 31st International Conference on Software Engineering (ICSE'09),* , (pp. 437–438). Vancouver,Canada.
46. B. Biegel, S. D. (2010). Highly configurable and extensible code clone detection. *Proceedings of the 17th Working Conference on Reverse Engineering (WCRE'10),* (pp. 237–241). Beverly, MA, USA.

47. V. Wahler, D. S. (2004). Clone detection in source code by frequent itemset techniques,. *Proceedings of the 4th IEEE International Workshop Source Code Analysis and Manipulation (SCAM'04),* (pp. 128–135.). Chicago, IL, USA: IEEE.
48. R. Koschke, R. F. (2006). Clone detection using abstract syntax suffix trees. *Proceedings of the 13th Working Conference on Reverse Engineering (WCRE'06),* (pp. 253–262.). Benevento, Italy.
49. R. Komondoor, S. H. (2001). Using slicing to identify duplication in source code. *Proceedings of the 8th International Symposium on Static Analysis (SAS'01),* (pp. 40–56). Paris, France.
50. S. Choi, H. P. (2009). A static API birthmark for windows binary executables. *The Journal of Systems and software* , 862–873.
51. Elizabeth Burd, J. B. (2002). "Evaluating Clone Detection Tools for Use during Preventative Maintenance," . *2nd IEEE International Workshop on Source Code Analysis and Manipulation (SCAM)* (pp. 36-43). Montreal, Canada: IEEE.
52. G. Antoniol, U. V. (2002). Analyzing cloning evolution in the Linux kernel,. *Information and Software Technology* , 755-765.
53. M. Balazinska, E. M. (1999). Measuring clone based reengineering opportunities. *Proceedings of the 6th International Software Metrics Symposium (METRICS'99),* , (pp. 292–303). Boca Raton,Florida, USA.
54. Ragkhitwetsagul, C. a. (2017). Using compilation/decompilation to enhance clone detection. *11th International Workshop on Software Clone (IWSC'17)* (pp. 8-14). Klagenfurt, Austria: IEEE.
55. Kim, S. a. (2018). Software systems at risk: An empirical study of cloned vulnerabilities in practice. *Computers & Security* , 720-736.
56. Jadon, S. (2016). Code clones detection using machine learning technique: Support vector machine. *In Computing, Communication and Automation (ICCCA), 2016 International Conference* (pp. 399-303). Noida: IEEE.
57. D Yu, D. J. (2017). Detecting Java Code Clone swith Multi-granularities Based on Bytecode. *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)* (pp. 317-326). Torino · Italy: IEEE.
58. Y Nakamura, Y. E. (2016). Towards detection and analysis of interlanguage clones for multilingual web applications. *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference* (pp. 17-18). United States : IEEE.
59. Lyu, F. Y. (2016). SUIDroid: An Efficient Hardening-Resilient Approach to Android App Clone Detection. *In Trustcom/BigDataSE/ISPA, 2016 IEEE* , 511-518.
60. H. Xue, Y. M. (2020). Twin-Finder: Integrated Reasoning Engine for Pointer-Related Code Clone Detection. *14th International Workshop on Software Clones (IWSC)* (pp. 1-7). London, ON, Canada: IEEE.
61. Chen, J. M. (2015). Detecting android malware using clone detection. *Journal of Computer Science and Technology 30, no. 5 (2015)* , 942-956.
62. Thaller, H. R. (2017). Exploring code clones in programmable logic controller software. *arXiv preprint arXiv:1706.03934 (2017)* .
63. Newman, C. D. (2016). srcSlice: a tool for efficient static forward slicing. *Software Engineering Companion (ICSE-C), IEEE/ACM*

International Conference (pp. 621-624). Austin, TX, USA: IEEE/ACM.

64. Liu, Z. Q. (2017). VFDETECT: A vulnerable code clone detection system based on vulnerability fingerprint. *Information Technology and Mechatronics Engineering Conference (ITOEC)* (pp. 548-553). Chongqing: IEEE.
65. E. Kodhai, S. K. (2010). Detection of type-1 and type-2 using textual analysis and metrics. *2010 International Conference on Recent Trends in Information, Telecommunication and Computing* (pp. 241-243). NW Washington, DCUnited States: IEEE.
66. Maeda, K. (2010). An extended line-based approach to detect code clones using syntactic and lexical information. *2010 Seventh International Conference on Information Technology: New Generations* (pp. 1237-1240). Las Vegas, Nevada, USA: IEEE.
67. Shihab, D. E. (2013). Cccd: Concolic code clone detection. *2013 20th Working Conference on Reverse Engineering (WCRE)* (pp. 489-490). Koblenz, Germany: IEEE.
68. M. S. Uddin, C. K. (2013). An extensible and faster clone detection tool for large scale software systems. *2013 21st International Conference on Program Comprehension (ICPC)* (pp. 236-238). San Francisco, CA, USA: IEEE.
69. S. Park, S. K.-J. (2013). Detecting source code similarity using code abstraction. *7th International Conference on Ubiquitous Information Management and Communication, ICUIMC* (pp. 74:1-74:9). New York, NY, USA: ACM.
70. J.-S. Lim, J.-H. J.-G. (2011). Plagiarism detection among source codes using adaptive local alignment of keywords. *5th International Conference on Ubiquitous Information Management and Communication, ICUIMC* (pp. 24:1-24:10). New York, NY, USA: ACM.
71. B. Lesner, R. B. (2010). A novel framework to detect source code plagiarism: Now, students have to work for real! *2010 ACM Symposium on Applied Computing, SAC* (pp. 57-58). New York, NY, USA: ACM.
72. Yadav, A. A. (2013). A hybrid-token and textual based approach to find similar code segments. *2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)* (pp. 1-4). Tiruchengode, India: IEEE.
73. Nishi, M. A. (2018). Scalable code clone detection and search based on adaptive prefix filtering. *Journal of Systems and Software 137 (2018)* , 130-142.
74. Tekchandani, R. R. (2017). Code clone genealogy detection on e-health system using Hadoop. *Computers & Electrical Engineering 61 (2017)* , 15-30.
75. Farhadi, M. R. (2015). Scalable code clone search for malware analysis. *Digital Investigation 15 (2015)* , 46-60.
76. Wang, P. J. (2018). CCAligner: a token based large-gap clone detector. *Proceedings of the 40th International Conference on Software Engineering* (pp. 1066-1077). Gothenburg Sweden: ACM.
77. Yuki, Y. Y. (2017). A technique to detect multi-grained code clones. *Software Clones (IWSC), 2017 IEEE 11th International Workshop* (pp. 1-7). Klagenfurt, Austria: IEEE.
78. Y. Semura, N. Y. (2017). Ccfindersw:Clone detection tool with exible multilingual tokeniz[80] ation. *24th Asia-Pacific Software Engineering Conference* (pp. 654-659). Nanjing, Jiangsu, China: A PSEC.

79.] Li, L. H. (2017). CCLearner: A Deep Learning-Based Clone Detection Approach. *Software Maintenance and Evolution (ICSME), 2017 IEEE International Conference* (pp. 249-260). Shanghai, China: IEEE.
80. J. Y. Poon, K. S.-Y. (2010). Instructor-centric source code plagiarism detection and plagiarism corpus. *17th ACM Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE* (pp. 122-127). New York, NY, USA: ACM.
81. Toomey, W. (2012). Ctcompare: Code clone detection using hashed token sequences. *2012 6th International Workshop on Software Clones (IWSC)* (pp. 92-93). Zurich: IEEE.
82. Roy, J. S. (2017). Cloneworks: A fast and exible large-scale near-miss clone detection tool. *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)* (pp. 177-179). Buenos Aires, Argentina: IEEE.
83. S. Kawaguchi, T. Y. (2009). Shinobi: A tool for automatic code clone detection in the ide. *2009 16th Working Conference on Reverse Engineering* (pp. 313-314). Lille: IEEE.
84. S. Abid, S. J. (2017). Codeease: harnessing method clone structures for reuse. *2017 IEEE 11th International Workshop on Software Clones (IWSC)* (pp. 1-7). Klagenfurt, Austria: IEEE.
85. Qing Qing Shi, L. P. (2013). A novel detection approach for statement clones. *2013 IEEE 4th International Conference on Software Engineering and Service Science* (pp. 27-30). Beijing, China: IEEE.
86. Guo, Y. Y. (2012). Boreas: an accurate and scalable token-based approach to code clone detection. *2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering* (pp. 286-289). Essen, Germany: IEEE/ACM.
87. Y. Semura, N. Y. (2018). Multilingual detection of code clones using antlr grammar definitions. *25th Asia-Pacific Software Engineering Conference(A PSEC)* (pp. 673-677). Nara, Japan: A PSEC.
88. Merlo, T. L. (2012). An accurate estimation of the levenshtein distance using metric trees and manhattan distance. *6th International Workshop on Software Clones (IWSC)* (pp. 1-7). Zurich: IEEE.
89. M. Elsabagh, R. J. (2018). Resilient and scalable cloned app detection using forced execution and compression trees. *IEEE Conference on Dependable and Secure Computing (DSC)* (pp. 1-8). Kaohsiung, Taiwan: IEEE.
90. M. Dong, H. Z. (2012). A new method of software clone detection based on binary instruction structure analysis. *8th International Conference on Wireless Communications, Networking and Mobile Computing* (pp. 1-4). Shanghai, China: IEEE.
91. Roy, J. R. (2011). The nicad clone detector. *IEEE 19th International Conference on Program Comprehension* (pp. 219-220). Ontario, Canada: IEEE.
92. Bharti, G. M. (2014). Implementing a 3-way approach of clone detection and removal using pc detector tool. *IEEE International Advance Computing Conference (IACC)* (pp. 1435-1441). Haryana · India: IEEE.
93. Rilling, I. K. (2013). Semantic-enabled clone detection. *IEEE 37th Annual Computer Software and Applications Conference* (pp. 393-

398). NW Washington, DCUnited States: IEEE.

94. Yang, Y. Z. (2018). Structural Function Based Code Clone Detection Using a New Hybrid Technique. *IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)* (pp. 286-291). Tokyo, Japan: IEEE.
95. Pati, J. B. (2017). A Comparison Among ARIMA, BP-NN, and MOGA-NN for Software Clone Evolution Prediction. *IEEE Access 5* , 11841-11851.
96. Lavoie, E. M. (2019). Computing structural types of clone syntactic blocks. *16th Working Conference on Reverse Engineering* (pp. 274-278). Lille: IEEE.
97. Narasimhan, K. (2015). Clone merge - an eclipse plugin to abstract near-clone c++ methods. *30th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (pp. 819-823). NW Washington, DCUnited States: IEEE.
98. Y. Yang, Z. R. (2018). Structural function based code clone detection using a new hybrid technique. *IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)* (pp. 286-291). Tokyo,Japan: IEEE.
99. J. Zeng, K. B. (2019). Fast code clone detection based on weighted recursive autoencoders. *IEEE Access, 7* , 125062-125078.
100. D. Li, M. P. (2014). One pass preprocessing for token-based source code clone detection. *IEEE 6th International Conference on Awareness Science and Technology (iCAST)* (pp. 1-6). Paris, France: IEEE.
101. Thompson, H. L. (2011). Incremental clone detection and elimination for erlang programs. *Fundamental Approaches to Software Engineering,Springer* , 356-370.
102. Svajlenko, J. a. (2017). Fast and flexible large-scale clone detection with CloneWorks. *Software Engineering Companion (ICSE-C), 2017 IEEE/ACM 39th International Conference* (pp. 27-30). Buenos Aires, Argentina: IEEE.
103. [103] Sudhamani, M. a. (2016). Code clone detection based on order and content of control statements. *Contemporary Computing and Informatics (IC3I), 2016 2nd International Conference* (pp. 59-64). Noida (UP) India: IEEE.
104. Haque, S. M. (2016). Generic code Cloning method for detection of Clone code in software Development. *Data Mining and Advanced Computing (SAPIENCE), International Conference* (pp. 335-339). Ernakulam, India: IEEE.
105. Ragkhitwetsagul, C. J. (2018). A picture is worth a thousand words: Code clone detection based on image similarity. *Software Clones(IWSC), 2018 IEEE 12th International Workshop* (pp. 44-50). Campobasso, Italy: IEEE.
106. Y. Fukushima, R. K. (2009). Code clone graph metrics for detecting diffused code clones. *16th Asia-Pacific Software Engineering Conference* (pp. 373-380). NW Washington, DCUnited States: IEEE.
107. Kusumoto, Y. H. (2011). Code clone detection on specialized pdgs with heuristics. *15th European Conference on Software Maintenance and Reengineering* (pp. 75-84). Oldenburg, Germany: IEEE.
108. Singh, R. a. (2017). To enhance the code clone detection algorithm by using hybrid approach for detection of code clones. *2017 International Conference on Intelligent Computing and Control Systems (ICICCS)* (pp. 192-198). Madurai · Tamil Nadu · India: IEEE.

109. Wang, M. P. (2017). CCSharp: An Efficient Three-Phase Code Clone Detector Using Modified PDGs. *Asia-Pacific Software Engineering Conference (APSEC), 2017 24th* (pp. 100-109). Nanjing, Jiangsu, China: IEEE.
110. Sabi, Y. Y. (2017). Rearranging the order of program statements for code clone detection. *Software Clones (IWSC), 2017 IEEE 11th International Workshop* (pp. 1-7). Klagenfurt, Austria: IEEE.
111. Crussell, J. C. (2015). Andarwin: Scalable detection of android application clones based on semantics. *IEEE Transactions on Mobile Computing 14, no. 10* , 2007-2019.
112. Sargsyan, S. S. (2016). Scalable and accurate detection of code clones. *Programming and Computer Software 42, no. 1* , 27-33.
113. Hu, Y. Y. (2017). Binary code clone detection across architectures and compiling configurations. *25th International Conference on Program Comprehension* (pp. 88-98). Buenos Aires Argentina: IEEE.
114. Kamalpriya, C. M. (2017). Enhancing program dependency graph based clone detection using approximate subgraph matching. *oftware Clones (IWSC), 2017 IEEE 11th International Workshop* (pp. 1-7). Klagenfurt, Austria: IEEE.
115. Avetisyan, A. S. (2015). LLVM-based code clone detection framework. *Computer Science and Information Technologies (CSIT)* , 100-104.
116. Leavens, R. E. (2012). Semantic clone detection using method ioe-behavior. *6th International Workshop on Software Clones (IWSC)* (pp. 80-81). Zurich: IEEE.
117. Deepika, S. S. (2013). Unifying clone analysis and refactoring activity advancement towards C# applications. *Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)* (pp. 1-5). Tiruchengode, Tamil Nadu, India: IEEE.
118. R. Tekchandani, R. K. (2013). Semantic code clone detection using parse trees and grammar recovery. *The Next Generation Information Technology Summit (4th International Conference)* (pp. 41-46). Uttar Pradesh, India: IEEE.
119. Kamiya, T. K. (2002). CCFinder: a multilinguistic token-based code clone detection system for large scale source code. *IEEE Transaction on Software Engineering* , 54–67.
120. I. Keivanloo, C. K. (2012). Sebyte: A semantic clone detection tool for intermediate languages. *20th IEEE International Conference on Program Comprehension (ICPC)* (pp. 247-249). Passau,Germany: IEEE.
121. D. Yu, J. Y. (2019). Detecting java code clones based on bytecode sequence alignment. *IEEE Access, 7* , 22421-22433.
122. Singh, C. M. (2017). Enhancing program dependency graph based clone detection using approximate subgraph matching. *IEEE 11th International Workshop on Software Clones (IWSC)* (pp. 1-7). Klagenfurt, Austria: IEEE.
123. M. Wang, P. W. (2017). *24th Asia-Pacific Software Engineering Conference* (pp. 100-109). Nanjing, Jiangsu, China: APSEC.
124. Y. Higo, S. K. (2011). Code clone detection on specialized PDG's with heuristics,. *Proceedings of the 15th European Conference on Software Maintenance and Reengineering (CSMR'11),*, (pp. 75-84). Oldenburg, Germany.
125.] Z. Xing, Y. X. (2011). Clonedifferentiator:Analyzing clones by differentiation. *IEEE/ACM*

International Conference on Automated Software Engineering (ASE 2011) (pp. 576-579). NW Washington, DCUnited States: IEEE/ACM.

126. Misu, M. R. (2017). Interface Driven Code Clone Detection. *Asia-Pacific Software Engineering Conference (APSEC)* (pp. 747-748). Nanjing, Jiangsu, China.: IEEE.
127. Sheneamer, A. a. (2016). Semantic clone detection using machine learning. *Machine Learning and Applications (ICMLA), 2016 15th IEEE International Conference* (pp. 1024-1028). Anaheim, CA: IEEE.
128. Vislavski, T. G. (2018). LICCA: A tool for cross-language clone detection. *IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)* (pp. 512-516). Campobasso, Italy: IEEE.
129. Ghofrani, J. M. (2017). A conceptual framework for clone detection using machine learning. *Knowledge-Based Engineering and Innovation (KBEI), 2017 IEEE 4th International Conference.* Tehran · Iran: IEEE.
130. Akram, J. Z. (2018). DroidCC: A Scalable Clone Detection Approach for Android Applications to Detect Similarity at Source Code Level. *IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)* (pp. 100-105). Tokyo,Japan: IEEE.
131. [131] Sheneamer, A. S. (2018). A detection framework for semantic code clones and obfuscated code. *Expert Systems with Applications 97* , 405-420.
132. Matsushita, T. a. (2017). Detecting code clones with gaps by function applications. *2017 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation* , 12-22.
133. Kodhai, E. a. (2014). Method-level code clone detection through LWH (Light Weight Hybrid) approach. *Journal of Software Engineering Research and Development 2* .
134. Uemura, K. A. (2017). Detecting and analyzing code clones in HDL. *Software Clones (IWSC), 2017 IEEE 11th International Workshop* (pp. 1-7). Klagenfurt, Austria: IEEE.
135. Nasirloo, H. a. (2018). Semantic code clone detection using abstract memory states and program dependency graphs. *4th International Conference on Web Research (ICWR)* (pp. 19-27). Tehran, IRAN: IEEE.
136. Singh, M. a. (2015). Detection of file level clone for high level cloning. *Procedia Computer Science 57* , 915-922.
137. White, M. M. (2016). Deep learning code fragments for code clone detection. *31st IEEE/ACM International Conference on Automated Software Engineering* (pp. 87-98). New York,NY,United States: IEEE/ACM.
138. H. Xue, Y. M. (2020). Twin-Finder: Integrated Reasoning Engine for Pointer-Related Code Clone Detection. *14th International Workshop on Software Clones (IWSC)* (pp. 1-7). London, ON, Canada: IEEE.
139. Kawser Wazed Nafi, B. R. (2020). A universal cross language software similarity detector for open source software categorization. *Journal of Systems and Software, //doi.org/10.1016/j.jss.2019.110491* .
140. George Mathew, C. P. (2020). SLACC: Simion-based Language Agnostic Code Clones ,arXiv:2002.03039 [cs.SE]. *Accepted at ICSE 2020 technical track,* (p. 11).
141. Karnalim, O. (2020). TF-IDF Inspired Detection for Cross-Language Source Code Plagiarism

and Collusion. *https://doi.org/10.7494/csci.2020.21.1.3389* .

142. Perez, D. a. (2019). Cross-Language Clone Detection by Learning over Abstract Syntax Trees. *16th International Conference on Mining Software Repositories* (pp. 518–528). Montreal, Quebec, Canada: IEEE Press.
143. K. W. Nafi, T. S. (2019). CLCDSA: Cross Language Code Clone Detection using Syntactical Features and API Documentation. 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), (pp. 1026-1037). San Diego, CA, USA: IEEE.
144. Saini, V. a. (2018). Oreo: Detection of Clones in the Twilight Zone. *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (pp. 354–365). New York, NY, USA: Association for Computing Machinery.
145. Nghi D. Q. Bui, L. J. (2017). *Cross-Language Learning for Program Classification using Bilateral Tree-Based Convolutional Neural Networks.* arXiv:1710.06159.
146. Hu, Y. a. (2017). Binary Code Clone Detection across Architectures and Compiling Configurations. *Proceedings of the 25th International Conference on Program Comprehension* (pp. 88-98). Buenos Aires, Argentina: IEEE.
147. T. Vislavski, G. R. (2018). LICCA: A tool for cross-language clone detection. *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)* (pp. 512-516). Campobasso: IEEE.
148. Cheng, X. P. (2017). CLCMiner: Detecting Cross-Language Clones without Intermediates. . *IEICE Transactions on Information and Systems* , 273-284.
149. Al-Omari, F. K. (2012). Detecting Clones Across Microsoft .NET Programming Languages. *19th Working Conference on Reverse Engineering* (pp. 405-414). IEEE.