Comprehending Code Fragment in Code Clones: A Literature-Based Perspective

Sarveshwar Bharti and Hardeep Singh

Abstract As Code Clones are defined on the notion of similarity in code fragments, it is necessary to first know what a code is meant by in accordance with Code Clones. A Source Code Fragment, which is a sequence of source code lines, is the basic entity that is used to analyze similarity/relation between Code Clones. For analysis, removal, avoidance, and management of Code Clones we have to first detect clones in software systems. There are more than 40 clone detection tools that implement some clone detection techniques to detect clones, but it is not well-defined what could be the appropriate minimum threshold for Clone length and with which unit of estimation. This paper, on the basis of Code Clone literature, presents different Units of Measurement of Clone Size and a comprehensive review of minimum Clone Size based on a particular technique used in Clone Detection and also argues that a unique Unit of Measurement and Minimum Clone Size should be presented.

Keywords Clone · Clone coverage · Clone granularity · Code fragment · Comparison granularity · CSIR rule · Minimum clone size · Unit of measurement

1 Introduction

Baxter et al. [1] defined Clone as:

> A clone is a code fragment that [is] identical to another fragment.

And Koschke [2] presented the definition by Baxter, 2002 as:

> Clones are segments of code that are similar according to some definition of similarity

The term 'Clone' is utilized by the software community in two different ways [3]: 'Clone' as a noun—refers to a code fragment that is similar to one or more code

S. Bharti (✉) · H. Singh
Guru Nanak Dev University, Amritsar, Punjab 143005, India
e-mail: sarveshwar.dcsrsh@gndu.ac.in

H. Singh
e-mail: hardeep.dcse@gndu.ac.in

P. K. Singh et al. (eds.), *Proceedings of ICRIC 2019*, Lecture Notes in Electrical Engineering 597, https://doi.org/10.1007/978-3-030-29407-6_56

fragments. 'Clone' as a verb—indicates the act of producing a code segment (e.g. by Copy-pasting). To study the relation between code clones, and, as code clones are defined on the notion of similarity between code fragments, it is necessary to know what a code fragment is.

Walenstein et al. [4] wrote in a paper on similarity in programs:

> In order to clarify the overview [what constituted a different similarity 'type'] we shall take cues from established notions of what a "program" [or code fragment] is. Having a clean definition is critical since it is logical to assume that only when the definition of "program" is nailed down can one hope to properly pin the notion of similarity in programs.

For analysis, removal, avoidance, and management of code clones, we have to first detect clones in software systems. There are more than 40 clone detection tools [5] that implement different clone detection methods to detect clones, but it is not clear what could be the appropriate minimum threshold for clone length. So, to understand the concept of code fragment and thus the relationship between them, the first question that arises is

> How much of code can be regarded as a code segment. [3]

There are various studies in the literature that answers the issue of minimum clone size but each study provides minimum clone size based on the technique used. So, it is as yet not clear what should be the least clone size and with which unit of estimation. When the unit of estimation is picked, it ought to be chosen what could be the suitable least limit for clone length dependent on this unit.

This paper presents a comprehensive review of various units of measuring clone length and minimum clone length for respective clone detection technique used in the literature.

The rest of this paper is organized as follows: In the next segment i.e. Sect. 2, we discuss the basic definition of Code Fragment and various types of Clone Granularities as found in the literature. Section 3, presents various units of measuring clone size as found in the literature. This discussion is supported by presenting various Comparison Granularities. In Sect. 4, we have provided a review of Minimum Clone Lengths used in the literature. We presented the literature review in a tabular form depicting Clone detection techniques, unit of measurement, minimum clone size and finally references in support of our findings. In Sect. 5, we discuss the impact of minimum clone size using a study from literature. Then finally this paper ends with a conclusion and future work, and acknowledgments along with references.

2 Understanding Code Fragment

Bellon et al. [6] defined code fragment as:

> A Code Fragment is a tuple (f, s, e) which consists of a name of the source file f, the start line s, and the end line e, of the fragment. Both line numbers are inclusive

From the above definition, the length of the Code Fragment is determined as
Code Fragment Length $= e - s + 1$
(where *e* is End Line and *s* is Start Line).

In clone detection process, to use comparison algorithms more efficiently, the transformation of source text into an internal format is carried out. When representing source in this internal format we can identify a code fragment accordingly, e.g. in token representation, tuple (*f*, *s*, *e*) would be, *f* as file name, *s* as start token number, *e* as end token number.

Now, the question is how much of contiguous source code can be contemplated as code fragment so that it can fulfill the criteria to be a clone candidate for Clone Detection. In literature, contiguous portions of source code at different levels of granularities have been used like at the level of statements, entire file, class definitions, method body and code block [3]. A granularity of Clones can be "Fixed" with predefined syntactic boundary or "Free" with no syntactic boundary, i.e. "clones are similar code fragments without considering any limit or boundary on their structure or size" [7]. Figure 1 shows different types of clone granularities found in the literature, which are broadly classified as Fixed and Free. While free granularity has no syntactic boundary, Fixed has a predefined syntactic boundary and thus can be seen at the different level of granularity. Figure 1 is created using the concepts of Clone Granularity as found in [7] and [3]. The first part of figure, i.e. free and fixed granularity types are taken from [7] and fixed granularity type is then extended using [3].

To detect clones that are useful from the maintenance perspective Zibran and Roy [3] proposed the following characteristics of chosen granularity of the code segment as motivated by their experiences and the criteria suggested by Giesecke [8]:

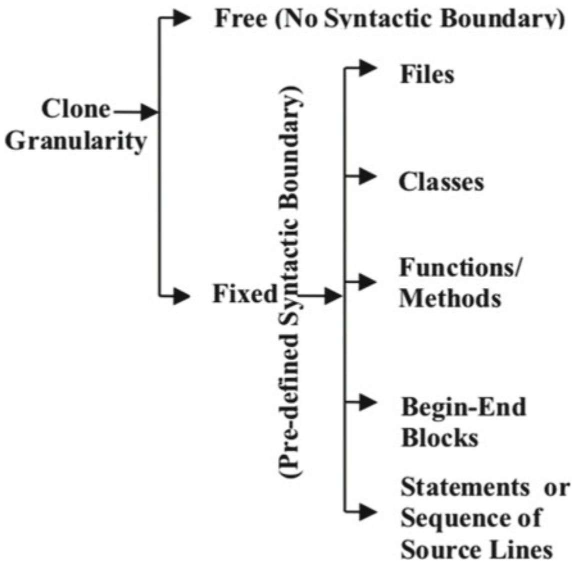
Clone Granularity
Free (No Syntactic Boundary)
Fixed
(Pre-defined Syntactic Boundary)
Files
Classes
Functions/ Methods
Begin-End Blocks
Statements or Sequence of Source Lines

Fig. 1 Types of clone granularities

Coverage: "The set of all code segments should cover maximal behavioral aspects of the software".
Significance: "*Each code segment should possess implementation of a significant functionality*".
Intelligibility: "Each code segment should constitute a sufficient amount of code such that a developer can understand its purpose with little effort".
Reusability: "The code segments should feature a high probability for informal reuse".

And finally suggested that code segment at the level of blocks or functions can be the utmost appropriate granularity for dealing with clones, especially for maintenance. For convenience, authors call these four characteristics as 'CSIR Rule' in rest of this paper, where *C* stands for *Coverage*, *S* for *Significance*, *I* for *Intelligibility* and *R* stands for *Reusability*.

3 Units of Measuring Clone Size

There are more than 40 Clone Detection Tools [5] available. Each tool implements an algorithm that works on a particular code representation. Each clone detection tool is developed using a particular clone detection technique. There are different Clone detection Techniques as found in literature viz. String-based, Token-based, Tree-based, PDG-based, Metrics-based and Hybrid Approach and each clone detection technique use a particular level of Clone Granularity for comparison. In literature, there is a number of different comparison granularities used by clone detection techniques. Figure 2 lists all such comparison granularities. For easy understanding, authors listed comparison granularities in the form of tree representation as shown in Fig. 2. This figure is created from [5]. Roy et al. [5] while comparing Clone Detection Tools, described comparison granularity as one of the technical facets and then described its various attributes viz. Line, Substring/Fingerprint, Identifiers and Comments, Tokens, Statements, Subtree, Subgraph, Begin-End Blocks, Methods, Files, and, Others. In order to stipulate a comparison of both general methods and distinct tools, they gathered citations of the identical group jointly using a category annotation, *L* for Lexical i.e. Token-based, *T* for Text-based, *S* for Syntactic i.e. Tree-based, *G* for Graph-based and *M* for Metrics-based, for each attribute. In Fig. 2 each attribute of Comparison Granularity is further categorized based on the above-mentioned category annotations.

Now, as Juergens et al. [9] described:

> Code is interpreted as a sequence of units, which for example could be characters, normalized statements, or lines.

And, as there are different units for representing code and thus different comparison granularities, so the particular type of comparison granularity used, corresponds to a particular "Unit of Measurement" for measuring Clone Length. Figure 3 shows

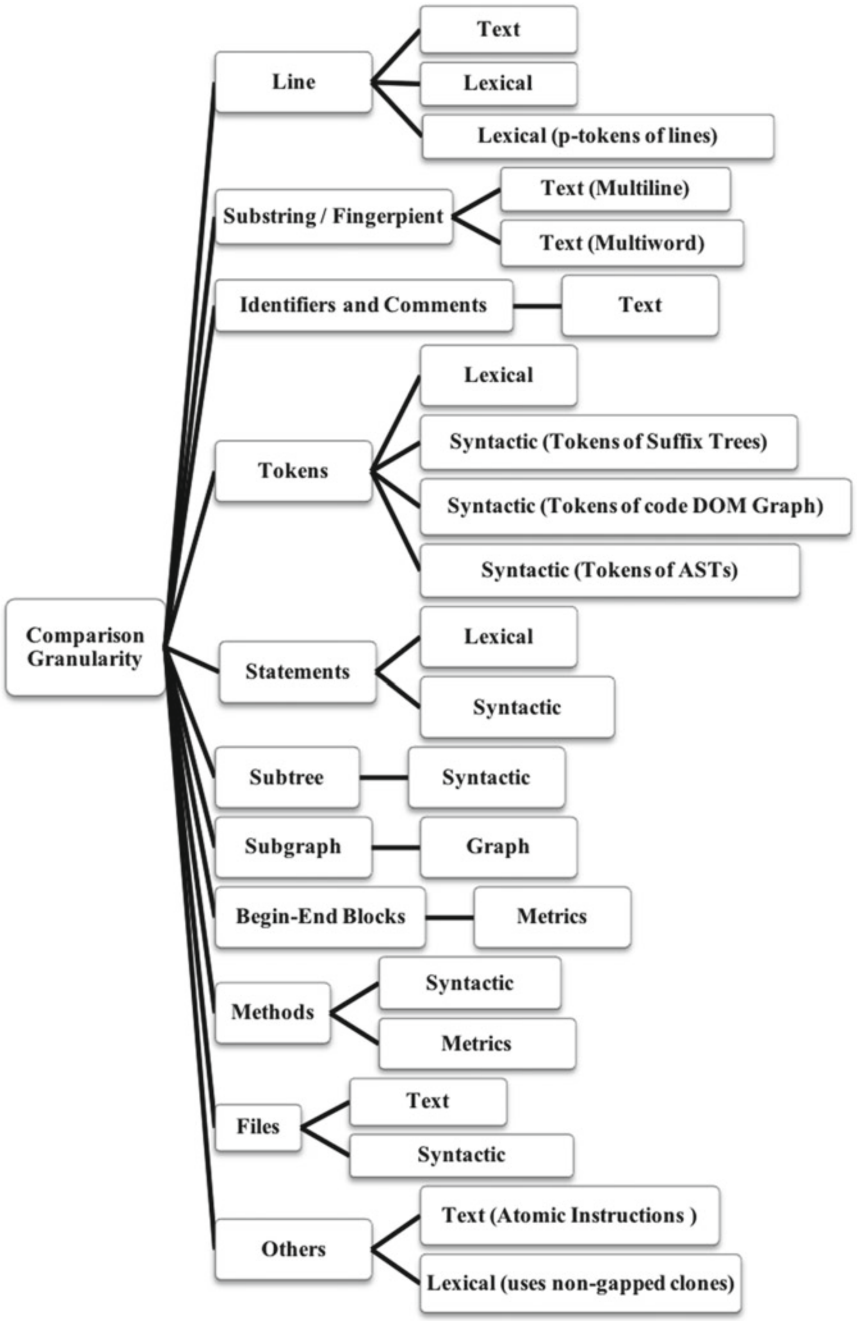

Fig. 2 Comparison granularity (based on [5])

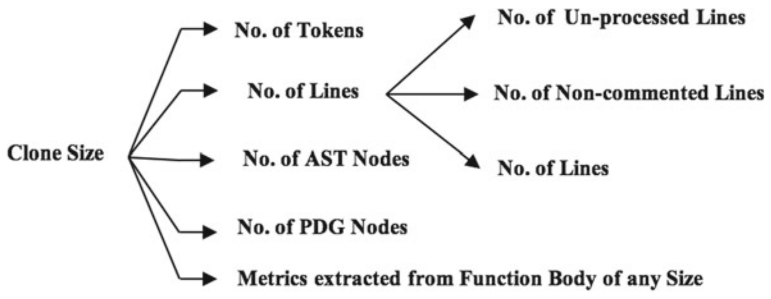
Clone Size
No. of Tokens
No. of Lines
No. of AST Nodes
No. of PDG Nodes
Metrics extracted from Function Body of any Size
No. of Un-processed Lines
No. of Non-commented Lines
No. of Lines

Fig. 3 Different units of measurement of clone size

different Units of Measurement as found in literature viz. Number of Tokens, Number of Source Lines, Number of AST Nodes, Number of PDG Nodes and Metrics value from Function Body of any Size. “Number of Lines” Unit of Measurement is seen as different Units viz. A number of Un-processed Lines, Number of Non-Commented Lines and Number of Lines.

4 Minimum Clone Length

Now to answer the question, as pointed in Sect. 1:

How much of code can be regarded as a code segment [3],

Authors present a summary of the Minimum Clone Size with respective Unit of Measurement, as found in the literature. Authors evaluated the findings related to minimum clone length in [7] and extended the findings in a more elaborated and systematic way in a tabular form as revealed in Table 1.

As discussed earlier, there are more than 40 different Clone Detection Tools [5] available, these tools implement particular Clone Detection Technique. In the ‘Clone detection Technique’, column of Table 1 authors list different Clone Detection Techniques found in the literature. Each Clone Detection Technique uses a particular unit of measuring Clone Size as listed in ‘Unit of Measurement’ column of Table 1. To optimize the results, every Clone Detection Tool used some minimum threshold for Clone Length. In ‘Minimum Clone Size’ column of Table 1, authors list the Minimum Clone Length used by different researchers and the references in support of their findings are listed in ‘Citations’ column of Table 1.

Kamiya et al. [10] used 30 tokens as their minimum clone length while detecting clones with their token-based clone detection tool CCFinder. Various other researchers like Kapser and Godfrey [11], Kim and Murphy [12], Jiang et al. [21], Higo et al. [22], etc. also used the same minimum clone length.

In line-based techniques, different researchers have different opinions. Bellon et al. [6] used 6 unprocessed lines as their minimum clone length, Baker [14] used

Table 1 Summary of minimum clone size and units of measurement

Clone detection technique	Unit of measurement	Minimum clone size	Citations
Token-based	Tokens	30 tokens	Kamiya et al. [10] Kapser and Godfrey [11] Kim and Murphy [12] Li et al. [13]
Line based	Un-processed lines	6 lines	Bellon et al. [6]
	Non-commented Lines	15 lines	Baker [14]
	Lines	50 lines	Johnson [15]
Abstract syntax tree based	AST node	Subtree	Baxter et al. [1]
Program dependency graph-based	PDG node	Sub graph	Komondoor and Horwitz [16] Krinke [17] Komondoor and Horwitz [18]
Metrics-based (Function clone detection)	Function metrics	Function body of any size	Mayrand et al. [19] Lague et al. [20]

15 non-commented lines as minimum clone length and Johnson [15] used 50 lines. There are other opinions too, but the authors presented these three to give a general idea of what is the range of value for minimum clone length when considering line-based techniques.

Baxter et al. [1] introduced clone detection by means of an Abstract Syntax Tree. AST (Abstract Syntax Tree) was produced by parsing the source code and then algorithm was applied on AST to detect the Sub-Tree Clones, thus, using sub-tree as the Minimum Clone Size and AST nodes as Unit of Measurement.

In literature, a number of researchers used PDG (Program Dependence Graph) based clone detection technique for detecting clones. Komondoor and Horwitz [16] used PDG (Program Dependence Graph) to find isomorphic Sub-Graphs. Krinke [17] presented an approach to identify alike code in programs established on locating similar Sub-Graphs in an attributed directed graph. This approach was used on the Program Dependence Graph. Komondoor and Horwitz [18] defined an algorithm for extricating "difficult" set of statements. Control-Flow Graph of a method and set of nodes in that CFG that have been selected for removal are the inputs to the algorithm. At the point when the algorithm finishes, the marked nodes will form a hammock (they described it as a sub-graph of CFG that has a single entry node, and from this entry point control moves to a specific outside-exit node), and thus extricating them into a distinct method and swapping them with a method call. Thus PDG based clone detection technique used PDG Nodes as a Unit of Measurement and Sub-Graph as a Minimum Clone size.

Another Clone Detection Technique found in the literature is Metrics-Based Technique. Function Clone Detection Technique uses Function Metrics values calculated from functions to detect clones at the function level of granularity. Mayrand et al. [19] presented an identification method to automatically recognize duplicate as well as near-duplicate functions in huge size systems based on metrics extricated from the source code exploiting tool DatrixTM. This Clone discovery method utilizes 21 function metrics grouped into four points of evaluation. Each point of correlation is utilized to compare functions and determine their level of cloning. Eight cloning levels are then defined as an ordinal scale ranging from exact copy clones to distinct functions. Lague et al. [20] also used the same above mentioned Clone Detection Technique. They compared two subsequent versions of the functions on the basis of the above mentioned 21 DatrixTM metrics used by the clone detection methodology. Thus function clone detection technique, which is a metrics-based technique uses metrics as the Unit of Measurement of clone size and function body of any size for extracting metrics, so, using function body as a Minimum Clone Size.

Table 1 provides a general understanding of the minimum threshold for Clone Size used in the literature by different researchers, while the citations in Table 1 may not be complete.

5 Impact of Clone Length

In the previous sections, this paper discussed what a code fragment is and what is the minimum clone length used by different researchers. Now, the question arises:

Does Code Clone Length matter in Code Cloning?

To answer this question, the authors present a study by Gode et al. [23]. Gode et al. illustrated their findings, from identifying clones within an industrial C/C++software system with 1400 KLOC, as revealed in Fig. 4.

In Fig. 4, horizontal axis symbolizes Minimum Clone Length (granularity used was statements) and a vertical axis represents Clone Coverage in percent. Gode et al. defined Clone Coverage as “the percentage of source code being part of at least one clone.”

To show that parameters utilized for clone identification impact the clone coverage, they chose three noticeable parameters, a minimum size of clone, the omission of generated code (G) and whether identifiers and literals are normalized (N) or not. Now from Fig. 4, it is clear that different value of minimum clone length and different combinations of the parameters have a strong effect on clone coverage that ranges from 19 to 92%. They also compared the clone coverage for different systems with minimum clone length varying but other parameters were fixed, as shown in Fig. 5.

Gode et al. observed that clone coverage of all software systems declines with growing minimum clone length. From this study, they established that exploiting

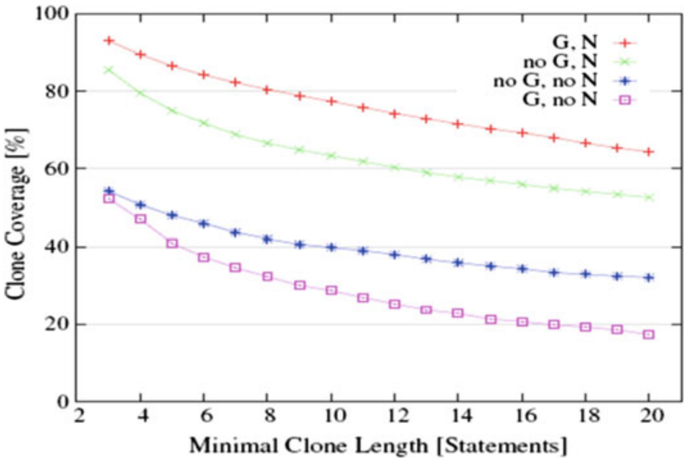

Fig. 4 Clone coverage using different parameters [23] (figure reproduced with permission)

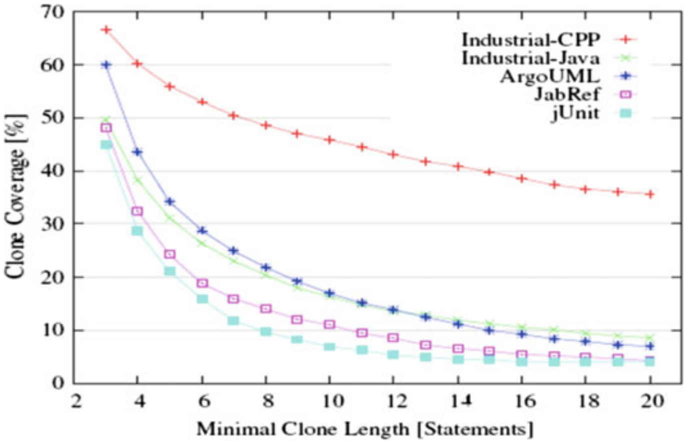

Fig. 5 Clone coverage of different systems [23] (figure reproduced with permission)

clone coverage for comparing software systems should be taken carefully, as the result depends on the parameters used.

6 Conclusion and Future Work

As discussed earlier, each Clone detection Technique uses some level or type of clone granularity i.e. comparison granularity for comparing Code Fragments to detect Clones, but as discussed, there are number of different types of clone granularities and thus number of different Units of measurement for measuring clone length, so, we don't have a unique Unit of Measurement, and thus standard Minimum Clone Size. Different researchers use different 'Units of Measurement' and 'Minimum Clone Size', so, our research community should work toward this direction and make a standard for 'Unit of Measurement' and 'Minimum Clone Size' and obviously this standard must obey the 'CSIR Rule' discussed in Sect. 2, only then we can have better results from the clone detection process and thus better maintenance. To ascertain the exact extent of clone length required and thus analyze Code Clones, more comprehensive survey is required which is beyond the scope of this short paper.

Present Clone Detection Tools needs Minimum Clone Size specified by the user, so, as discussed in Sect. 5, selection of Minimum Clone Length should be done with care.

Like Code Clone Detection, calculating Minimum Clone Size can also be a Tool supported, where Minimum Clone Size will be calculated with the help of a tool, so our community should also work towards this direction.

From the literature survey we did, it is found that this matter of Unit of measurement and Clone Size found a little space in related publications, and thus authors think that this paper will be a keynote paper towards the study of Code Fragment and thus the Code Clones.

Acknowledgements We would like to thank University Grants Commission, a statutory body of Govt. of India for offering the financial support for this research in the form of Rajiv Gandhi National Senior Research Fellowship to the first author and also would like to thank Guru Nanak Dev University (Department of Computer Science), Amritsar for the scholastic and infrastructural support towards the current research work.

References

1. Baxter, I.D., Yahin, A., Moura, L., Anna, M.S., Bier, L.: Clone detection using abstract syntax tree. In: Proceedings of 14th International Conference on Software Maintenance (ICSM'98), Bethesda, Mayland (1998)
2. Koschke, R.: Survey of research on software clones. In: Dagstuhl Seminar Proceedings 06301: duplication, Redundancy, and Similarity in Software (Dagstuhl 2007) (2007)
3. Zibran, M.F., Roy, C.K.: The road to software clone management: a survey (2012)
4. Walenstein, A.: Similarity in programs. In: Dagstuhl Seminar Proceedings 06301: Duplication, Redundancy, and Similarity in Software (Dagstuhl 2007) (2007)
5. Roy, C.K., Cordy, J., Koschke, R.: Comparison and evaluation of code clone detection techniques and tools: a quantitative approach. Sci. Comput. Progr. **74**(7), 470–495 (2009)
6. Bellon, S., Koschke, R., Antoniol, G., Krinke, J., Merlo, E.: Comparision and evaluation of clone detection tools. IEEE Trans. Softw. Eng. **33**(9), 577–591 (2007)
7. Roy, C.K., Cordy, J.R.: A survey on software clone detection research. Qween's University, Kingston (2007)
8. Giesecke, S.: Generic modelling of code clones. In: Dagstuhl Seminar Proceedings 06301: Duplication, Redundancy, and Similarity in Software (Dagstuhl 2007) (2007)
9. Juergens, E., Deissenboeck, F., Hummel, B., Wagner, S.: Do code clone matter? In: Proceedings of 31st International Conference on Software Engineering ICSE 2009, Vancouver, BC (2009)

10. Kamiya, T., Kusumoto, S., Inoue, K.: CCFinder: a multilinguistic token-based code clone detection system for large scale source code. IEEE Trans. Softwa. Eng. **28**(7), 654–670 (2002)
11. Kapser, C.J., Godfrey, M.W.: Supporting the analysis of clones in software systems: a case study. In: IEEE International Conference on Software Maintenance ICSM 2005 (2005)
12. Kim, M., Murphy, G.: An empirical study of code clone genealogies. In: Proceedings of the 10th European Software Engineering Conference (ESEC) held jointly with 13th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-13) ESEC/SIGSOFT FSE 2005 (Lisbon 2005) (2005)
13. Li, Z., Lu, S., Myagmar, S., Zohu, Y.: CP-Miner: finding copy paste and related bugs in large scale Software Code. IEEE Trans. Softw. Eng. **32**(3), 176–192 (2006)
14. Baker, B.: On finding duplication and near duplication in large software systems. In: Proceedings of the 2nd Working Conference on Reverse Engineering (WCRE'95) (1995)
15. Jonson, J.: Substring matching for clone detection and change tracking. In: Proceedings of International Conference on Software Maintenance ICSM'94, Victoria, BC (1994)
16. Komondoor, R., Horwitz, S.: Using slicing to identify duplication in source code. In: Proceedings of 8th International Symposium on Static Analysis SAS 2001, Paris (2001)
17. Krinke, J.: Identifying similar code with program dependence graphs. In: Proceedings of the 8th Working Conference on Reverse Engineering (WCRE'01), Stuttgart (2001)
18. Komondoor, R., Horwitz, S.: Effective, automatic procedure extraction. In: Proceedings of the 11th IEEE International Workshop on Program Comprehension IWPC 2003, Portland (2003)
19. Mayrand, J., Leblane, C., Merlo, E.: Experiment on the automatic detection of function clones in a software systems using metrics. In: Proceedings of International Conference on Software Maintenance (IWSM'96), Monterey (1996)
20. Lague, B., Proulx, D., Mayrand, J., Merlo, E.M., Hudepohl, J.: Assessing the benefits of incorporating function clone detection in a development process
21. Jiang, Z.M., Hassan, A.E., Holt, R.C.: Visualizing clone cohesion and coupling. In: XIII Asia Pacific Software Engineering Conference APSEC 2006, Bangalore (2006)
22. Higo, Y., Kamiya, T., kusumoto, S., Inoue, K.: Method and implementation for investigating code clones in a software system. Inf. Softw. Technol. **49**(5), 985–998 (2006)
23. Gode, N., Hummel, B., Juergens, E.: What clone coverage can tell. In: Proceedings of 6th International Workshop on Software Clones IWSC 2012, Zurich (2012)