Survey on Diverse Code Clone Detection Methods and Tools

Geetika Chatley, Aarti
Department of Computer science and engineering
Lovely Professional University.

Abstract:

In this paper, we discussed several methods and tools for detecting code replication in different dimensions. This survey provided an in-depth study of the clone detection methods and tools. From clone perceptions, the classification of clones and a general variety of selected methods and tools are discussed. This review paper covers the total worldview of clone identification and presents open research procedures in the recognition of code clones.

Keywords: clone, refactoring, duplicate code, replication

1) INTRODUCTION:

Code segments generally occur due to replication from one location and then rewrite them in another section of code with or without changes, they are copied code also known as clone. Several researchers have reported more than 20 to 59% code replication. The problem with this copied code is that an error detected in the original must be checked on each copy for the similar type of error. In addition, the copied code increases the effort that must be made when increasing the lines of code. However, analysis on the improved quality code, identification of replication, bug recognition, facet extraction, and error exposure are other software engineering tasks. These tasks require the extra efforts on semantically or syntactically identical code segments. Fortunately, there are a number of comparison and evaluation studies related to many clone detection methods. Recently, Rattan presented a methodical investigation into the detection of clones. In addition, possible studies have evaluated the approach to detecting clones in another context.

In this paper, we have given an extensive survey of presently available clone investigation methods and tools. We will start with the fundamentals of code clones after their classification and comparison of methods and tools in two distinct ways. First of all, the classification of various types of clones and their methods and the subsequent categorisation of clone spotting tools. The rest of the document is structured as follows. Second part presents the taxonomy of code clones. Third part covers the various methods for spotting clones. Fourth part explores the tools for detecting code clones. Research gaps are reviewed in the fifth part and finally, sixth part concludes the article.

2) Classifying Clones in Code: Figure 1 characterizes the classification of code clones. Classifications of clones are helpful for extending re-engineering and investigation methods. Based on the classification of clones, we have discussed the most important types of duplicate codes or clones, which occur at the time of reengineering. Then, the duplicate codes are classified according to three facets, such as: (1) the likeliness between two code parts, (2) the location of the duplicate codes in the program and (3) the chances for refactoring with the clones.

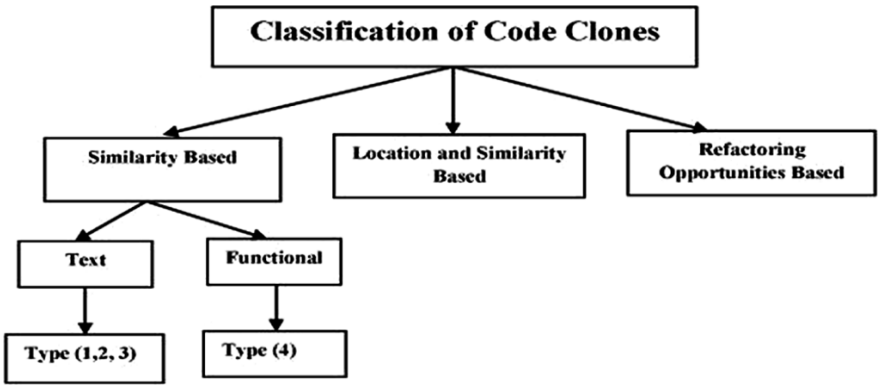

Fig. 1. Classification of code clones

On the basis of similarities, clones are of two types, such as: (1) two code parts can be similar depending on the similarity of the content in the program and (2) can have same functionality without being textually similar. However, clones based on textual similarities are of 3 types such as type one (similar parts of code without changes, except changes in whitespace and comments) type two (syntactically similar copied code, except for modifications in the names of the identifying functions, variables used, types) and type three (similar code parts with or without additional changes; the declarations have been modified, inserted or deleted). The syntactic code must be measured in this classification which has been changed by the coder after duplication. Type 4 clone (similar calculation but different structure) based on similar performances.

Mayrand gave the similarity between two functions of three types which are based on comparison points, such as the function name, the code design, lexis in the functions and the flow of control of functions. A taxonomy for the cloning methods proposed by Balazinska with 18 different categories that each group of cloning methods considers according to the differences between them. Categories specify the amount of method content that has been copied and also the type of syntax elements that have been changed. At the level 1, two categories for general similarities are identical and external changes.

The second instances, symbolic variations and aspects of the method based on three categories. The third point is based on the importance of the particular token in the body of the method and, moreover, the fourth phase is based on the distinction of the sequence of tokens in the body of the function. The 3 types of clones, such as exact duplicate codes, parametric duplicate codes and duplicate codes which have other dominant characteristics illustrated by Bellon and Koschke to make a good assessment between various investigation tools. The objective of this categorisation is to analyse the investigation and the adequacy of the classification of the different duplicate code spotting tools. A clone topology with an additional type (type 4) presented by Davey which is based on Bellon and Koschke. In addition to this, the researchers detected type one to type three clones and type four detection leaving it as task for the future. Kontogiannis details 4 types of clones, which are based on a functional replication scheme, such as exact clones, similar clones, except those analytically replaced by names of the variables and data types; the third are duplicate codes with more adaptations. Fourth is duplicate codes with statements that have been inserted or removed.

In addition, the taxonomy is related to the position and similarity of the duplicate codes. This categorisation is based on position distinctions, as well as on the physical extent between clone instance positions. Obstacles are based on the code segments that are in the same file, or files in distinct directories can be improved without affecting their external behaviour by using refactoring. Cloning instances in an object-oriented system occupy a specific position in the class hierarchy. Rudimentary analysis technology is sufficient to extract such assortments for a few clones. The illustrations of these types of classifications provided by various researchers are as

follows. The first taxonomy of duplicate codes is based on their substantial position in the program text. The next taxonomy of clones is based on the type of domain in which they are found, and the third is based on clones of short codes. Monden described how to simplify the relationship between software quality and code through categorization based on code cloning modules and also provided a module taxonomy.

Finally, classifications based on refactoring chances have discussed the simplicity of extracting copied code from the point of view of refactoring. The classifications of these types of variations are based on methods that have been defined outside of the code snippet and on the use of variables. Context analysis was proposed by Balazinska to complete the difference analysis of code clones. Basically, there are four types of clones and each type is classified as shown in Table 1 below.

Type 1	Type 2	Type 3	Type 4
Exact clone	Renamed clone	Near-miss clone	Structural clone
Structural clone	Parameterized clone	Gapped clone	Function clone
Function clone	Near-miss clone	Non-contiguous clone	Reordered clone
	Function clone	Reordered clone	Intertwined clone
		Structural clone	Semantic clone
		Function clone	

Table 1. Code clone properties

Types of clones and their sub-types are mentioned in table 1. Text-based (type-one to three) and on the basis of function (type-four) are some types of similarity based classification. Type one clone has been explained as follows. (1) The exact clone (identical code excluding a few variations in comments), Structural type one, three, four as (2) (identical); Structural subset (Type one, two, three, four as (3) function). Type two as (1) (modifying the copied code), (2) set (renamed duplicate code with renaming). Form 2, 3 as (3) near-miss duplicate codes. (2)non-contiguous (Like near-misses between fragments of code are permissible). The gapped form is (1) (add, erase, change a section between segments) and (2). Type three, four as (3) reorganized (rearranged some lines). Type 4 is interwoven as (1) (making two parts in one part).

3) Code Clone Detection Methods:

Analysing similar code investigation methods can be done on the basis of miscellaneous clone properties. These code clone properties are listed in figure 2.

The property of normalization is also listed there which means doing number of purifications in the code such as removing white spaces, comments etc. before comparing the code. The actual result is shown after the transformation. Crucial role is played by comparison algorithms in investigation of dissimilar type of code clones. Complexity of the algorithm is based on its type of transformation and comparison technique. The language independency property

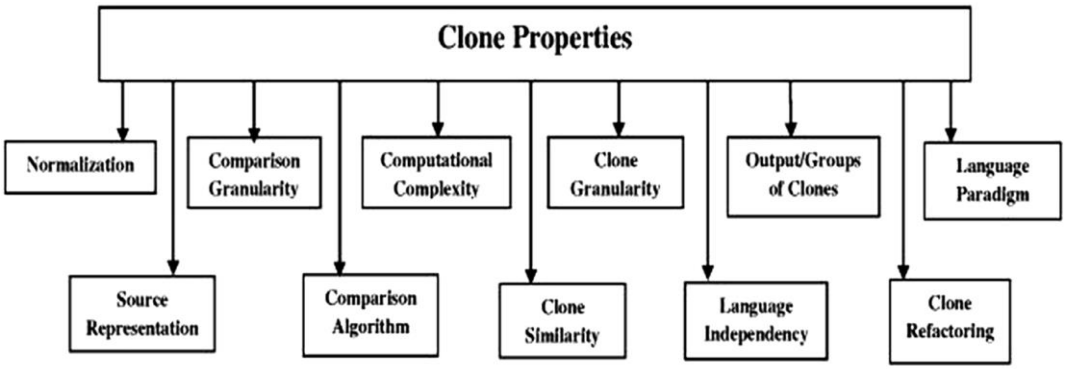

Fig. 2. Various types of code clone properties

verified language sustain of a detection tool. The output aspect indicates what sort of output, or both, will occur as a clone class. Duplicate code refactoring involves internal code redesign without altering the external behavior. The language paradigm implies programming paradigm that is aimed at the specific interesting process.

3.1) Taxonomy of Code Clone investigation Methods

There are following types of clone detection methods.

(1) The text based method: The source fragments are analyzed as a subsequence of text in this method. There are various alterations, such as whitespace, newline and eliminating comments, all segments are textually compared to one another for the purpose of finding related string sequences. A number of scientists have proposed numerous clone detection methods based on string / text. Both the lexer and the line-based string algorithm are used on tokens for Baker's text line using Dup. A function was also employed to classify clones (who have different names of variables).

Nevertheless, clone written in a different style could not be identified and discovery and navigation between the copied numbers could not be helped. By using tokens and non-parameterized suffixes, Koshke have solved this issue. While authors could neither detect the same and parameterized clones nor differentiate them. In fact, the clones that Koshke et al. proposed does not validate that the identifiers have been continuously updated. Karp-Rabin fingerprint something-rithm for clone detection by Johnson as well as for the measurement of the text fines for all the source code length substring. The whole text of the substring shall be separated in to a series of substrings, because each character of this technique comprises of at least one substring. The limitation of this method, however, is that the correspondence of 50 lines decreases the number of false positives. The grammar system used for the definition of syntactic structures by Cordy. In fact, the developer found almost absent copies for HTML web pages. The limitations of that technique are that any code could not be normalized and that the comparisons were small. The string-based dynamic pattern matching algorithm suggested by Ducasse et al., which is language-independent. Furthermore, this technology cannot identify meaningful duplicate code resolution linguistically. The Marcuss approach to Late Semantic indexing. In spite of comparing the entire source code, this approach detects clones in comments by extremity their comparative domain. The identification names cannot detect such types of clones with the same structure. All of the aforementioned detection approach shows that the recent approach proposed by Ducasse has used several raw source transformations and that the source code is not modified. Although it is not possible to analyze and detect whether the cost to the text-based approach is significantly smaller except for code that have identifying changes, line split, parenthesis amputation, type, etc.

(2) The token method: The token approach are also known as lexical approach. In this method, the complete source code is divided into tokens by way of token analysis and then lexical comes out as a set of token sequence. Finally, the order is scanned for figuring out similar type of code. One of the primary device of token primarily based method named as CCFinder proposed by using

Kamiya. Foremost, the lexer partitions each line of text in to tokens and subsequently forms a single lexical order and moreover, the suffix tree matching algorithm is used to find similar sub-sequences of token sequence. Although, Dup is also a token-based approach tool in the sense that it is also used lexer for tokenization as well as for comparisons based on suffix tree matching algorithm proposed by Baker. CP-Miner has been introduced to overwhelm the problem of CCFinder and Dup, in which a frequent subsequence mining technique is used for duplicate code investigation rather than ordered analysis in CCFinder and Dup. Approach for C ++ and C# was proposed by Kawaguchi and it could not overcome the problem.

(3) The syntactic method: In this method, the code is viewed as abstract syntax tree. One of the initial abstract syntax tree tools known as CloneDR proposed by Baxter. It creates AST with the aid of compiler generator and then compares its sub tree by using metrics which is based on hash functions. Although, it was not detecting identical clones. To solve this issue, the Bauhaus has provided a ccdiml tool by avoiding the uses of hashing and similarity metrics. However, it was incompetent to verify the renamed identifiers. Yang has presented one of the grammar-based approach. It is used for finding the syntactic variations by creating their parse tree and then apply run-time programming technique for identifying similar sub tree. Wahler explored the approach to detect the exact and parameterized clone. This approach foremost converts the AST into XML and subsequently used frequent item set data mining technique for extracting the clones.

(4) The semantic/PDG method: This approach used the control flow and data flow for clone detection semantically and syntactically. PDG-DUP is one of the most prominent PDG-based clone detection approach proposed by Komondoor and Horwitz. It is based on program slicing technique for identifying PDG sub graph without changing its semantics behaviour. Further, the same slicing based clone analysis approach accomplished by Gallagher and Lucas. They compute slices of program on all the data variables of a program but could not figure out any analysis result.

(5) The syntactic method: In syntactic methods dissimilar metrics are gathered such as functions, lines etc. from code parts and then evaluates that metrics. Mayrand computed metrics from expression, layouts and control flow for each function elements of a program and then similar metrics returned as software clones. It detects function-based copy-paste instead of segment-based copy-paste which occurs recurrently. The feasible matches identified by an abstract pattern tool which is based on markov model provided by Kontogiannis. The authors used metrics for clone detection which is extorted from an AST of the code and then match detection is done by using dynamic programming. However, it was unable to identify copy-pasted code rather than it only measures similarity between the codes.

(6) The semantic method: The semantic method is a collection of several approaches. The tree and token based-hybrid approach proposed by Koschke for finding type-one and type-two code clones. In this method authors generate suffix tree for serialized Abstract syntax tree. The Microsoft's new phoenix framework, was also used for the detection of function level clones with the same approach. It can detect exact function clone as well as parameterized clone with identifier renaming not data type changes.

The analogous approach was proposed by Greenan for detection of method level clones with the sequence matching algorithm. Jiang et al. who explored AST in Euclidean space which is used for the computing vectors as well as group and these vectors are on the basis of similarity and it is used through the Locating Sensitive Hashing (LSH). A dynamic pattern matching as well as characterized based hybrid approach which is provided by Balazinska. In this method of cloning each body is computed with the quality metrics and then it evaluates identified clusters and it can be used on the basis of Patenaude's

metric-based approach. Further, it was also unable for data flow detection, in this method clones are detected at the coder's level.

The above-explained survey has been presented in the graph.

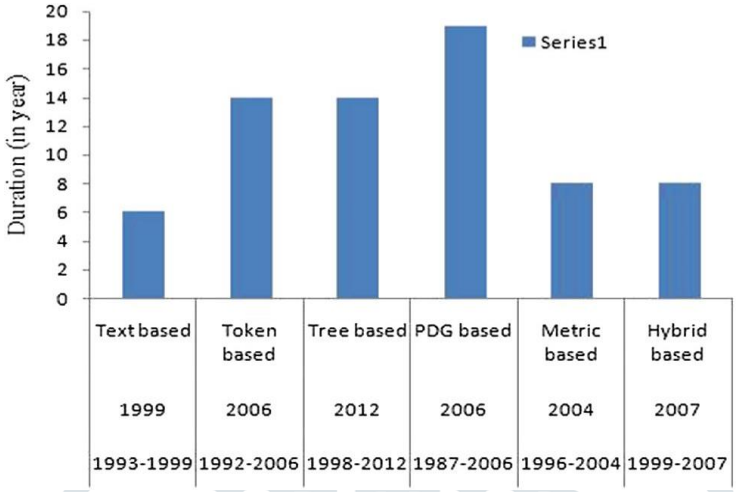

Fig. 3. Survey of code clone detection methods.

4) Comparing various Clone Detection Tools:

The software tools which is used to detect clones are multivariate and their abstraction which entails a methodical scheme for recounting their property. This table explains the assessments of different tools and methods. The first column has the name of the author, and the second column has the name of the tools. The third column signifies the methods, fourth column imply the supported languages and fifth column implies the tools domain.

Author	Tools	Methods	Supported-language	Domain
Baker	Dup	Line/Text based	C, C++, Java	CD/Linux
Kamiya,	CCFinder	Transformation/Token comp. with suffix tree	C, C++, Java, COBOL etc.	CD/Windows/NT
Bellon	Ccdiml (Bauhaus)	AST/Tree Matching	C, C++	CD/Linux
Krinke	Duplix	PDG, graph Matching	C?	CD

Table 2. Assessment of different tools and methods

5) Issues in Spotting Code Clone:

If we talk about the clone investigation approaches for the investigation of non-trivial duplicate code. So there is no such a code clone technique. As we know that every tool has its own issues, which makes it difficult to define which is realistic for clone detection. The type-one as well as type-two duplicate codes are easily spotted if we compare with type-3 and type-4. The PDG method can only figure out type-three and type-four duplicate code but only issue of this algorithm is that it is producing many variants of the similar code. Therefore, it is an important tool for such type of methods and tools that may overcome the restrictions of existing methods for similar code investigation.

6) Conclusion:

The code-clone investigation is an issue in the software industry so it degrades the software's unambiguousness as well as maintainability. Therefore, its analysis and detection is

very necessary for improving the quality. In this paper, our main discussion is in terms of attributes based on duplicate code categorization, classification of similar code investigation tools as well as these approaches like text based, token based, tree based, PDG based, metric based and hybrid methods on the basis of their property and sub-property. However, there are so many other algorithms which have been developed and these are based on above-mentioned approaches, but still some investigation of clones whose precision and productivity is a potential issue. Therefore, there are various algorithms for clone detection in which some algorithms are less efficient when a huge system is to be compared. This report represents an extensive comparison of tools and methods as well as research gaps in clone detection so that can easily select an appropriate method according to the necessity and can analyze chances for hybridizing various methods that may overcome the existing research gaps in similar code spotting algorithms.

References

1. Baker, B.S.: On finding duplication and near-duplication in large software systems. In: Proceedings of 2nd IEEE Working Conference on Reverse Engineering, Toronto, Ontario, Canada, pp. 86–95, July 1995
2. Ducasse, S., Rieger, M., Demeyer, S.: A Language independent approach for detecting duplicated code. In: Proceedings of 1st IEEE International Conference on Software Maintenance, Oxford, UK, ICSM 1999, pp. 109–118 (1999)
3. Mayrand, J., Leblanc, C., Merlo, E.: Experiment on the automatic detection of function clones in a software system using metrics. In: Proceedings of 1st IEEE International Conference on Software Maintenance, Monterey, CA, pp. 244–254 (1996)
4. Kontogiannis, K., Mori, R.D., Merlo, E., Galler, M., Bernstein, M.: Pattern matching for clone and concept detection. J. Autom. Softw. Eng. 3(1), 79–108 (1996)
5. Lague, B., Proulx, D., Mayrand, J., Merlo, E.J., Hudepohl, J.: Assessing the benefits of incorporating function clone detection in a development process. In: Proceedings of 1st IEEE International Conference on Software Maintenance, Washington, DC, USA, pp. 314–321 (1997)
6. Roy, C.K., Cordy, J.R.: A survey on software clone detection research. Technical report 541, Queen's University at Kingston (2007)
7. Rattan, D., Bhatia, R., Singh, M.: Software Clone detection: a systematic review. Inf. Softw.Technol. 55(7), 1165–1199 (2013)
8. Roy, C.K., Cordy, J.R., Koschke, R.: Comparison and evaluation of code clone detection techniques and tools: a qualitative approach. Sci. Comput. Program. 74(7), 470–495 (2009)
9. Bellon, S., Koschke, R., Antoniol, G., Krinke, J., Merlo, E.: Comparison and evaluation of clone detection tools. IEEE Trans. Softw. Eng. 33(9), 577–591 (2007)
10. Patil, R.V., Joshi, S., Shinde, S.V., Ajagekar, D.A., Bankar, S.D.: Code clone detection using decentralized architecture and code reduction. In: Proceedings of IEEE International Conference on Pervasive Computing, Pune, India (ICPC 2015), pp. 1–6, January 2015
11. Keivanloo, I., Zhang, F., Zou, Y.: Threshold-free code clone detection for a large-scale heterogeneous Java repository. In: Proceedings of 22nd IEEE International Conference on Software Analysis, Evolution and Reengineering, Montreal, QC (SANER 2015), pp. 201–210 (2015)
12. Chodarev, S., Pietrikova, E., Kollar, J.: Haskell clone detection using pattern comparing algorithm. In: Proceedings of 13th IEEE International Conference on Engineering of Modern Electric Systems (EMES 2015), Oradea, Romania, pp. 1–4 (2015)
13. Kamiya, T.: An execution-semantic and content-and-context-based code-clone detection and analysis. In: Proceedings of 9th International Workshop on Software Clones, Montreal, QC (IWSC 2015), pp. 1–7 (2015)
14. Singh, M., Sharma, V.: Detection of file level clone for high level cloning. In: Proceedings of 3rd Elsevier International Conference on Recent Trends in Computing (ICRTC 2015), India, pp. 915–922 (2015)
15. Basit, H.A., Jarzabek, S.: A data mining approach for detecting higher-level clones in software. IEEE Trans. Softw. Eng. 35(4), 497–514 (2009)

16. Balazinska, M., Merlo, E., Dagenais, M., Lague, B., Kontogiannis, K.: Measuring clone based reengineering opportunities. In: Proceedings of the 6th IEEE International Symposium on Software Metrics (METRICS 1999), USA, pp. 292–303, November 1999
17. Bellon, S:. Vergleich von techniken zur erkennung duplizierten quellcodes. Master's thesis no. 1998, University of f Stuttgart (Germany). Institute for Software Technology, September 2002, 664 P. Gautam and H. Saini
18. Koschke, R., Falke, R., Frenzel, P.: Clone detection using abstract syntax suffix trees. In: Proceedings of the 13th IEEE Working Conference on Reverse Engineering, Italy, pp. 253–262, October 2006
19. Davey, N., Barson, P., Field, S., Frank, R., Tansley, D.: The development of a software clone detector. J. Appl. Softw. Technol. 1(3/4), 219–236 (1995)
20. Kontogiannis, K.: Evaluation experiments on the detection of programming patterns using software metrics. In: Proceedings of the 4th IEEE Working Conference on Reverse Engineering, Netherlands, pp. 44–54, October 1997
21. Kapser, C., Godfrey, M.W.: Aiding comprehension of cloning through categorization. In: Proceedings of the 7th IEEE International Workshop on Principles of Software Evolution, Japan, pp. 85–94, September 2004
22. Monden, A., Nakae, D., Kamiya, T., Sato, S.I., Matsumoto, K.I.: Software quality analysis by code clones in industrial legacy software. In: Proceedings of 8th IEEE International Symposium on Software Metrics, Canada, pp. 87–94, June 2002
23. Fanta, R., Rajlich, V.: Removing clones from the code. J. Softw. Maintenance 11(4), 223–243 (1999)
24. Koni-N'sapu, G.G.: A scenario based approach for refactoring duplicated code in object oriented systems. Diploma thesis, University of Bern, Germany (2001)
25. Johnson, J.H.: Identifying redundancy in source code using fingerprints. In: Proceeding of the 1993 Conference of the Centre for Advanced Studies on Collaborative Research, Canada, pp. 171–183, October 1993
26. Marcus, A., Maletic, J.: Identification of high-level concept clones in source code. In: Proceedings of 16th IEEE International Conference on Automated Software Engineering (ASE 2001), pp. 107–114, November 2001
27. Baker, B.S.: A program for identifying duplicated code.de. In: Proceedings of Computing Science and Statistics, 24th Symposium on the Interface, pp. 49–57, March 1993
28. Baker, B.S.: Parameterized difference. In: Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms (SODA 1999), Maryland, USA, pp. 854–855, January 1999
29. Cordy, J.R., Dean, T.R., Synytskyy, N.: Practical language-independent detection of near-miss. In: Proceedings of the 14th Conference of the Centre for Advanced Studies, Canada, pp. 1–12, October 2004
30. Cox, I.J., Linnartz, J.P.M.: Some general methods for tampering with watermarks. J. Sel. Area Commun. 16(4), 587–593 (1998)
31. Dumais, S.T.: Latent Semantic Indexing (LSI) and TREC-2. In: Proceedings of the 2nd Text Retrieval Conference (TREC 1994), Maryland, pp. 105–115, March 1994
32. Kamiya, T., Kusumoto, S., Inoue, K.: CCFinder: a multilinguistic token-based code clone detection system for large scale source code. IEEE Trans. Softw. Eng. 28(7), 54–67 (2002)
33. Baker, B.S.: Finding clones with dup: analysis of an experiment. IEEE Trans. Softw. Eng. 33 (9), 608–621 (2007)
34. Li, Z., Lu, S., Myagmar, S., Zhou, Y.: CP-miner: a tool for finding copy-paste and related bugs in operating system code. In: Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI 2004), Berkeley, CA, USA, vol. 4, no. 19, pp. 289–302, December 2004
35. Li, Z., Lu, S., Myagmar, S., Zhou, Y.: CP-miner: finding copy-paste and related bugs in large-scale software code. IEEE Trans. Softw. Eng. 32(3), 176–192 (2006)
36. Juergens, E., Deissenboeck, F., Hummel, B.: Clone detective - a workbench for clone detection research. In: Proceedings of the 31st IEEE International Conference on Software Engineering, Vancouver, BC, pp. 603–606 (2009)Various Code Clone Detection Techniques and Tools 665
37. Kawaguchi, S., Yamashina, T., Uwano, H., Fushida, K., Kamei, Y., Nagura, M., Iida, H.:

SHINOBI: a tool for automatic code clone detection in the idea. In: Proceedings of 16th IEEE Working Conference on Reverse Engineering (WCRE 2009), Lille, pp. 313–314 (2009)
38. Baxter, I.D., Yahin, A., Moura, L, Anna, M.S.: Clone detection using abstract syntax trees. In: Proceedings of the 14th IEEE International Conference on Software Maintenance (ICSM 1998), Maryland, pp. 368–377, November 1998
39. Raza, A., Vogel, G., Plödereder, E.: Bauhaus – a tool suite for program analysis and reverse engineering. In: Pinho, L.M., González Harbour, M. (eds.) Ada-Europe 2006. LNCS, vol. 4006, pp. 71–82. Springer, Heidelberg (2006). doi:10.1007/11767077_6
40. Yang, W.: Identifying syntactic differences between two programs. J. Softw. Prac. Exp. 21(7), 739–775 (1991)
41. Wahler, V., Seipel, D., Fischer, G.: Clone detection in source code by frequent item set techniques. In: Proceedings of the 4th IEEE International Workshop on Source Code Analysis and Manipulation (SCAM 2004), USA, pp. 128–135, September 2004
42. Evans, W.S., Fraser, C.W., Ma, F.: Clone detection via structural abstraction. J. Softw. Qual.17(4), 309–330 (2009)
43. Duala-Ekoko, E., Robillard, M.P.: Clone tracker: tool support for code clone management.In: Proceedings of the 30th ACM International Conference on Software Engineering, Washington, DC, USA, pp. 843–846 (2008)
44. Nguyen, H.A., et al.: Clone management for evolving software. IEEE Trans. Softw. Eng. 38(5), 1008–1026 (2012)
45. Komondoor, R., Horwitz, S.: Using slicing to identify duplication in source code. In: Cousot,P. (ed.) SAS 2001. LNCS, vol. 2126, pp. 40–56. Springer, Heidelberg (2001). doi:10.1007/3-540-47764-0_3
46. Krinke, J.: Identifying similar code with program dependence graphs. In: Proceedings of the 8th IEEE Working Conference on Reverse Engineering (WCRE 2001), Germany, pp. 301–309, October 2001
47. Liu, C., Chen, C., Han, J., Yu, P.S.: GPLAG: detection of software plagiarism by program dependence graph analysis. In: Proceedings of the 12th ACM International Conference on Knowledge Discovery and Data Mining (KDD 2006), Philadelphia, pp. 872–881, August 2006
48. Komondoor, R.V.: Automated duplicated-code detection and procedure extraction. Doctoral thesis, University of Wisconsin- Madison, USA (2003)
49. Gallagher, K., Layman, L.: Are decomposition slices clones? In: Proceedings of the 11th IEEE International Workshop on Program Comprehension (IWPC 2003), USA, pp. 251–256, May 2003
50. Ferrante, J., Ottenstein, K.J., Warren, J.D.: The program dependence graph and its use in optimization. ACM Trans. Program. Lang. Syst. 9(3), 319–349 (1987)
51. Di Lucca, G.A., Di Penta, M., Fasolino, A.R., Granato, P.: Clone analysis in the web era: an approach to identify cloned web pages. In: Proceedings of the 7th IEEE Workshop on Empirical Studies of Software Maintenance, Italy, pp. 107–113, November 2001
52. Di Lucca, G.A., Di Penta, M., Fasolino, A.R., Granato, P.: An approach to identify duplicated web pages. In: Proceedings of the 26th International Conference on Computer Software and Applications, England, pp. 481–486, August 2002
53. Calefato, F., Lanubile, F., Mallardo, T.: Function clone detection in web applications: a semi-automated approach. J. Web Eng. 3(1), 3–21 (2004)
54. Lanubile, F., Mallardo, T.: Finding function clones in web applications 2003. In: Proceedings of 7th IEEE European Conference on Software Maintenance and Reengineering (CSMR 2003), Italy, pp. 379–386, March 2003
666 P. Gautam and H. Saini
55. Tairas, R., Gray, J.: Phoenix-based clone detection using suffix trees. In: Proceedings of the 44th ACM Annual Southeast Regional Conference (ACM-SE 2006), Melbourne, pp. 679–684, March 2006
56. Greenan, K.: Method-level code clone detection on transformed abstract syntax trees using sequence matching algorithms. Student report, University of California, Santa Cruz, USA (2005)
57. Jiang, L., Misherghi, G., Su, Z., Glondu, S.: Scalable and accurate tree-based detection of code clones. In: Proceedings of the 29th IEEE International Conference on Software

Engineering (ICSE 2007), USA, pp. 96–105, May 2007
58. Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.S.: Locality-sensitive hashing scheme based on p-stable distributions. In: Proceedings of the 20th ACM Annual Symposium on Computational Geometry (SoGG 2004), New York, pp. 253–262, June 2004
59. Balazinska, M., Merlo, E., Dagenais, M., Lagüe, B., Kontogiannis, K.: Measuring clone based reengineering opportunities. In: Proceedings of the 6th IEEE International Software Metrics Symposium (METRICS 1999), Florida, USA, pp. 292–303, November 1999
60. Patenaude, J.F., Merlo, E., Dagenais, M., Laguë, B.: Extending software quality assessment techniques to java systems. In: Proceedings of the 7th IEEE International Workshop on Program Comprehension (IWPC 1999), USA, pp. 49–56, May 1999
61. De Wit, M., Zaidman, A., Van Deursen, A.: Managing code clones using dynamic change tracking and resolution. In: Proceedings of IEEE International Conference on Software Maintenance (ICSM 2009), Edmonton, AB, pp. 169–178 (2009)