Code Clone Detection: A Systematic Review

Iqra Yaqub[1] Khubaib Amjad Alam[2]

Abstract

Code cloning in software systems has gained significant development in past few years. Cloning is a general mean of reusing software as existing code snippets can be utilized either by copy and paste methods or by minor modifications in the current code in software systems. However, this may lead to produce bugs and maintenance issues. A plethora of various code clone detection tools and techniques have emerged from last few decades. However, there are no comprehensive studies reviewing all the available techniques since 2013. The aim of this Systematic Literature Review (SLR) is to fill this research gap by systematically reviewing all the available research and extending the research on this particular topic. The main objectives of the study are to identify, categorize and synthesize relevant techniques related to this particular topic. After analyzing initial set of 1181 studies gathered from four large databases, 37 studies relevant to defined research questions were identified by following a systematic and unbiased selection procedure according to standard PRISMA guidelines. This selection process is followed by the data extraction, detailed analysis and reporting of findings. The results of this SLR reveals that different tools and techniques have widely been used for code clone detection, but graph-based and metric-based approaches are most prolific approaches. These approaches have also been used as a part of hybrid approaches. Different match detection techniques are also reported. However, to cope with rapidly evolving clones in software systems, the need is to develop more efficient techniques to improve the state of current research. This study concludes with new recommendations for future research.

Keyword: Software clone, Code clone, Duplicated code, Clone detection, Detection techniques, Reuse, Similarity, Clone detection tools

1 Introduction

In software engineering, the word “abstraction” is used frequently. This technique is widely used by developers to manage the complexity of the software by establishing a level of simplicity. Abstractions at all levels of granularity involves implementation. For doing such implementations, we can start coding from scratch or use some existing code by code cloning [1]. Reusing an existing code is a situation that often occurs during software development process. Existing code can be used as it is if it is fulfilling the requirements or it can be used with minor or even major changes that can be performed at different levels. All of these can be achieved by code cloning which could be of any type that all depends on the programmer’s technique and capability of using the code [2]. Code cloning is a common activity during software development in which existing code snippets can be utilized either by copy and paste methods or by doing minor modifications in the current code in software systems. The pasted code itself

[1]*National University of Computer & Emerging Sciences, Islamabad | f179022@nu.edu.pk*
[2]*National University of Computer & Emerging Sciences, Islamabad | khubaib.amjad@nu.edu.pk*

with or without modifications is called clone of the original. In other words, code clones are like different code fragments that produce similar results on same input.

Code cloning has gained significant importance in our research community. During the development of a software, code cloning can be done intentionally using copy and paste methods by programmers. They can also be introduced unintentionally due to lack of technical knowledge in developers. For example, such accidental code clones are produced due to use of certain design patterns, use of certain APIs, etc. Code cloning has some positive as well as negative effects on the development and maintenance of the software. This activity is adopted or used by the programmers as a common practice to increase productivity, reduce advancement costs and enhance product quality [3, 4, 5].

In software maintenance, duplication of code or reusing code by copy and paste methods with or without modifications is considered a well-known code smell. Although, reusing existing code is a standard practice in modern programming paradigms. But, adapting this approach too much has some negative impact on software systems [2]. It has been observed that code clones have some bad effect on maintenance of software as it increases the chances of bug propagation and produces code that is difficult to maintain. Code clones have bad effect on maintainability and reusability of the software. Software code clones also lack software quality. Considering its harmfulness and to improve the quality of the code, it is important to detect code clones in software systems. So, the negative side of code cloning part needs more attention for detecting code clones and removes them for not becoming a hindrance in the process of software development. However, it is very difficult to identify the original code from copied code after development [6].

Detection of code clones in software systems is very important for avoidance of their side effects. In recent years, many code clone detection methods based on different types of clones have been proposed. Various code clone detection techniques are used according to the characteristics and representation of source code [7]. These code clone detection techniques fall under different categories which will be discussed later.

Recently, it has been investigated that different studies used different tools for detection of different types of code clones having different environment. According to a study [8], there are no general results about the harmfulness of code clones in software systems. It was concluded in the study that "not all code clones make software maintenance more difficult". So, it is unsuitable to remove all the code clones for efficiency of program. However, it is significant to reduce the risk of code clones instead of totally removing them which requires more cost and seems impossible [8].

Code clone detection is a wider field and has gained significant importance from research point of view. Previous research includes different types of code clone detection tools and techniques. However, most of the research is carried out regarding software clone detection in general. There are many surveys and comparative studies in this domain but, there exists no comprehensive review, or systematic study from 2013 to present the state-of-the-art research in this domain. This paper reports a Systematic Literature review (SLR) to fill this research gap

by analyzing and reporting the findings of code clone detection tools and techniques from 2013 to 2018. The purpose of this SLR is to identify, summarize and analyze the existing code clone detection tools and techniques.

The contribution of this SLR involves the taxonomies for understanding the structure of code clone detection tools, techniques and different types of datasets used. Moreover, major findings on code clone detection are uncovered by detailed analysis of the identified solutions. All the studies in this SLR are selected to ensure inclusion criteria and are selected through a quality assessment process. This SLR also considers the overall research productivity of this research field. In this study, section 2 consists of a brief description of related work. Section 3 explains the detailed research method including research questions, study selection process, inclusion and exclusion criteria, quality assessment criteria and data extraction process. Section 4 explains the results, discussion and detailed analysis of findings of the selected primary studies. These selected primary studies consider five research questions. Section 5 describes the conclusion of the study.

2 Background Knowledge

Comparative analysis of different code clone detection techniques observes that text-based techniques can detect Type-1 clones only [6], token-based techniques detect Type-1 and Type-2 clones and tree-based approaches detect Type-1, Type-2 and Type-3 clones. According to a review [5], textual and token-based approaches are good for problem detection. Type-1 and Type-2 are easier to detect than Type-3 and Type-4 clones. PDG-based approach is used to identify Type-3 clones [9]. Graph-based approaches are used more in number than tree-based and metric-based approaches [10].

According to a comprehensive and detailed analysis [11] on software clone detection, the research in this field is increasing day by day. Mainly, semantic clone detection and model-based detection are discussed in this extensive study. This existing SLR is about software clones in general and software clone detection in particular. Different types of clones, different clone detection techniques/tools and their evaluation are discussed. The purpose is to identify the importance of software clone detection techniques. This study identifies that reliable detection of similar code is an open area for research. However, this study does not discuss clone detection techniques/tool from 2013. So, this study fills this research gap by reporting a comprehensive and detailed analysis of the current techniques used since 2013. Neural Networks [17], [18], [19] (CNNs) are feed-forward deep neural networks best suited to solve visual imagery learning problems, e.g., image classification and recognition. They are famous because they eliminate the need to exact image features

3 Research Method

It is necessary to ensure that the search results or analysis must contain all the relevant studies. This can be ensured by performing a systematic literature review (SLR) which is done by identification, interpretation, evaluation and detailed analysis of all the available research associated to a particular domain. A SLR must contain a search plan which is quite fair, free of

biasness and must ensure the completeness of analysis. There is no comprehensive analysis or detailed review of available research on code clone detection since 2013. So, this study aims at conducting a comprehensive SLR on code clone detection by following the SLR guidelines of Kitchenham [12]. This strategy of systematically reviewing has a number of steps to be performed in a systematic way. Development of a review protocol, conduction of systematic review, analysis of results, reporting of results and visualization of results including discussion on findings are the steps of systematic review process.

A Research Questions

This study has a primary research question i.e "What is the state-of-art of code clone detection in software systems?" This main research question is divided into five RQ's. This SLR reports and answers only first two research questions due to shortage of space. The answer of all other research questions will be the part of the extended version of this SLR.

RQ1: What techniques/methods have been used to detect code clones in software systems?

RQ2: What is the overall research productivity in this domain?

RQ3: What type of commercial/open source tools have been used for code clone detection and what are their characteristics?

RQ4: What are the basic types of clones and their taxonomies according to different researchers?

RQ5: Which datasets have been widely used for code clone detection?

B Electronic Databases

Four different electronic databases are used in this process which are enlisted in Table I.

Table 1: Electronic Databases

ED1	ACM	http://dl.acm.org/
ED2	IEEE Xplore	http://ieeexplore.ieee.org/
ED3	Science Direct	http://sciencedirect.com/
ED4	Springer Link	http://link.springer.com/

C Search Terms

A search string was defined by combining different search terms to search all the related articles from the above-mentioned electronic databases enlisted in Table I. Following are the research terms for population, intervention, and outcome.

Population: code, source code, instructions, program, software

Intervention: clone, copy, duplicate, replica, image, dummy

Outcome: detection, recognition, identification, findings, exploration

The main search string includes different inter-related concepts e.g. code clone detection, code clone identification, code duplicate detection etc. All these concepts will be used as a combination.

("code" OR "source-code" OR "source code" OR "sourcecode" OR "program" OR "software") AND ("clone*" OR "cloning" OR "duplicat*" OR "copy*" OR "copies") AND ("detect*" OR "recogni*" OR "identif*")

D Study Selection Procedure

This SLR has a specific study selection procedure which follows the standard PRISMA guidelines for systematic review as visualized in Fig. 1. It has mainly three phases after extraction of results from databases and duplicate removal. Fairness and un-biasness ensured in this process when each phase was done by a detailed consensus meeting.

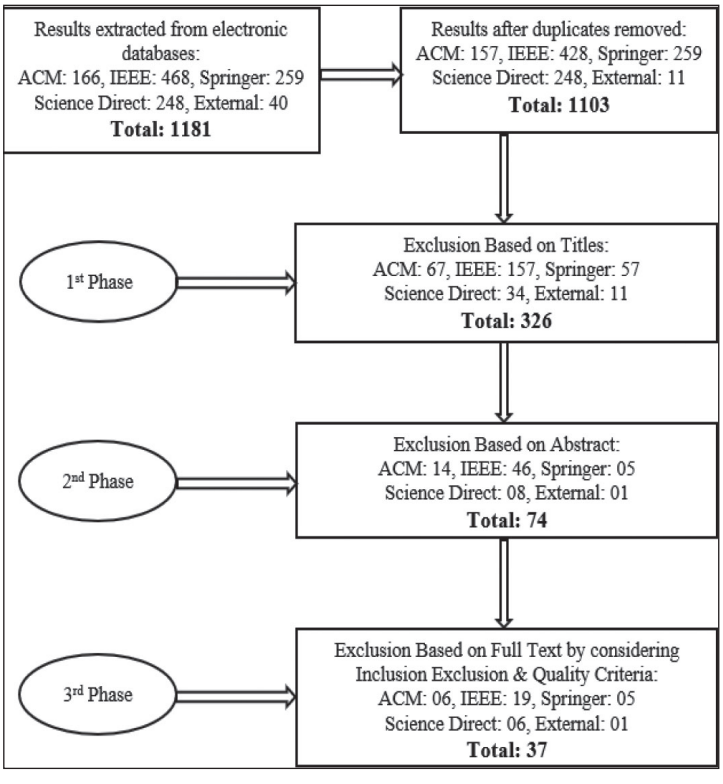

Figure 1: Study Selection Procedure

E Inclusion and Exclusion Criteria

Inclusion and exclusion criteria is defined for selection of relevant studies from databases to

answer the research questions which is listed in Table II.

Table 2: Inclusion and Exclusion Criteria

Inclusion Criteria	
IC1	Any primary study related to code clone detection
IC2	Studies published in 2013-2018
IC3	Only peer reviewed articles should be included
IC4	Only those articles are considered for which full texts are retrievable
Exclusion Criteria	
EC1	Studies other than English language
EC2	Studies with no validation of proposed techniques or comparative evaluation
EC3	Data from editorials, short papers, posters, extended abstracts, blogs or Wikipedia should not be included
EC4	Studies with ambiguous results or findings

These criteria ensured that the studies from 2013 to 2018 were included to fill the research gap of no comprehensive study on code clone detection since 2013. These criteria were applied to all the results in different stages of study selection procedure (Fig. 1). These criteria are mainly applied to 2nd stage for exclusion based on abstracts and 3rd stage for exclusion based on full-text articles considering inclusion & exclusion criteria and quality attributes. Initial set of studies were 1181. Final set of studies after filtration were 37. Fig. 2 depicts the proportion of selected studies.

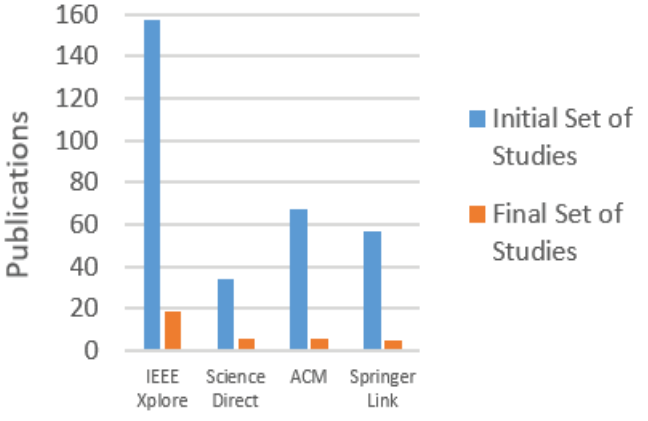

Figure 2: Proportion of selected studies

F Quality Assessment Criteria

Quality assessment was considered for exclusion of full-text articles in third phase of study selection process (Fig. 1). As, 74 articles were filtered out in stage 2. So, quality assessment criteria were considered for these 74 studies which were filtered based on abstracts. Fairness

and un-biasness were accomplished by reviewing each article by 2 reviewers. Scale of three (Fully, Partially, No) was used for conformance to quality ranking. This procedure retrieved 37 studies satisfying the quality attributes. Quality assessment criteria is given in Table III.

Table 3: Quality Assessment Criteria

Quality Assessment Criteria	
QC1	Primary studies must have proper validation
QC2	Primary studies must have clearly defined goals and objectives
QC3	Primary studies must include limitations
QC4	Are the methods used in the studies well defined?

4 Discussion and Results

The similarity of code can be occurred in the form of clone pairs or clone classes. Two code fragments can be similar due to textual similarity or on the basis of resemblance of their functionalities [5]. Textual similarity can be in terms of syntax and functional similarity in terms of functions or semantics of two or more code fragments. Textual similarity is further divided into Type-1, 2 and 3 clones and functional similarity as Type-4 clones [13]. Type-1 are the similar code fragments having some variations in comments, whitespaces and layouts. Type-2 are the similar code fragments having different identifiers, literals, layout and comments. Type-3 clones are the similar code fragments which are further modified by adding or changing statements. Type-4 clones are semantically similar performing same computations [5].

Clone detection process have several steps for finding clone pairs or classes. It has a proper mechanism and requires speedy computational results. Pre-processing, transformation, match detection process, formatting, post processing and aggregation are the phases of a generic work flow of clone detection process [7]. A clone detection technique can focus on one or more of the phases of generic clone detection process [9].

The first step of clone detection process involves pre-processing of code base for elimination of uninterested parts. In this phase, segmentation is performed on source code and then, area of comparison is figured out. Second step of clone detection process is transformation in which preprocessed code is converted into intermediate representations. Extraction, tokenization and parsing are performed to get transformed code [3]. Every transformed fragment is compared to all other fragments to find similarity using comparison algorithm [9]. A set of clones on transformed code are obtained in this phase. The next phase of clone detection involves formatting. In this phase, the code acquired in previous phase is further converted to some new clone pairs or classes related to the original source code. Post processing or filtration is performed in the next phase which can be done manually or by some automated heuristic. Next phase is aggregation which is considered as an optional phase. In this step, clone pairs

extracted from previous phase are aggregated into groups, sets or classes to reduce the amount of data [3]. A generic work flow of clone detection process is visualized in Fig. 3.

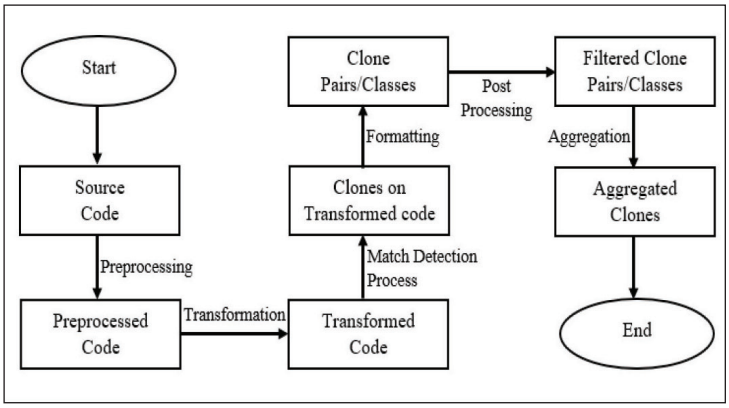

Figure 3: Work Flow of Clone Detection Process

The basic work flow of clone detection process and the idea of clone similarity detection are considered for answering top two previously defined research questions.

RQ1: What techniques/methods have been used to detect code clones in software systems?

This question is considered as one of the main questions of this study as the main thing in a clone detection are the techniques or methods that a clone detection process uses. There are different types of clones that can be detected by different approaches. Different types of techniques or methods for code clone detection include text-based, token-based, abstract syntax tree-based, program dependency graph-based, metric-based and hybrid approaches.

Text-based approach is one of the simple and fastest approach. It is used to detect Type-1 clones. Comparison is performed line by line on two code fragments and, in this way, similarity on the basis of text is detected as clones. Token-based approach converts code fragments into tokens and these tokens are compared by using matching algorithm to find similarity. It can detect both Type-1 and Type-2 clones. Tree-based approach converts source code into Abstract Syntax Tree (AST) and similar trees are identified using tree matching algorithm. Graph-based approach converts code fragments into Program Dependency Graph (PDG) which contains the semantic information of code fragments. Similar subgraphs are identified by using some sub-graph matching algorithm. This approach can detect Type-4 clones. In metric-based approach, different metrices are computed and values of metrices are compared to find similarity. Moreover, Hybrid approach can use these approaches as a combination to give the better results of similarity [3].

Different types of techniques have been used by different researchers depending on the nature of clones. Table IV provides an overview of all the techniques used in selected primary studies since 2013.

Table 4: Overview of Techniques in Selected Studies

Ref.	Technique/Method	Approach
[14]	Feature extraction from BDG, PDG, AST	Framework
[15]	Suffix array, token substrings	Method+Tool
[16]	Token-based approach, Filtering heuristic	Method
[17]	Count matrix clone detection (CMCD), AST	Method
[18]	Token-based approach, deep learning	Method
[19]	Formal methods, CCS transformation	Method+Tool
[20]	Tree-based (AST) + Token-based methods	Method
[21]	Coarse-grained + Fine-grained methods, Hash values + Levenshtein distance	Method+Tool
[22]	Dynamic dependence graphs	Method+Tool
[23]	Token-based + ASTs, computing LCPs	Method+Tool
[24]	Interface information + PDG	Method
[25]	PDG, Plan calculus to represent programs	Method+Tool
[26]	PDG, Approximate Subgraph Matching	Method
[27]	Concolic Analysis, Levenshtein distance	Method
[28]	Static data flow analysis, I/O profiles	Method+Tool
[29]	Metric Collection, Pairwise comparisons	Model
[30]	Method Interface Similarities, Jaccard similarity measure	Method
[31]	Concolic Analysis	Method+Tool
[32]	PDG generation, PDG's merging	Framework
[33]	Textual analysis (Island-drivern parsing approach) + Metrices	Method+Tool
[34]	Smith-Waterman algorithm, Fine-grained	Method+Tool
[35]	Smith-Waterman algorithm	Method+Tool
[36]	PDG, Spatial-based+graph based pattern mining	Framework
[37]	PDG, Method trials	Method
[38]	Hybrid (Metric-based + Token-based)	Model+Tool
[39]	PDG, Slice-based algorithm	Method+Tool
[40]	Token-based approach, Heuristics (prefix filtering + token position filtering + adaptive prefix filtering)	Method
[41]	Token Matching, Jaccard similarity	Model
[42]	AST + PDG, Vector representation	Method+Tool
[43]	Feature extraction, DBSCAN Clustering	Method
[44]	Token-based, Partial Index Creation	Method+Tool
[45]	Metric-based appoach, Distance Matrix	Method
[46]	Metric-based method, Metric comparison	Method

[47]	Partial indexes, Jaccard similarity metric	Method+Tool
[48]	K-means Clustering, Friedman method	Method
[49]	Hybrid (PDG + Metric-based) approach	Method
[1]	AST, Greedy method	Method

The overview of all the techniques in selected primary studies from Table IV shows that after 2013, mostly studies use hybrid approaches using two or more basic detection techniques. It has been observed that numerous studies used Graph-based [14, 22, 24-26, 32, 36, 37, 39, 42, 49] and Metric-based [24, 29, 30, 33, 38, 43, 45-47, 49] approaches to find similarity for code clone detection. Moreover, some studies also used Token-based [15, 16, 18, 20, 23, 38, 40, 41, 44], Hybrid [14, 20, 21, 23, 24, 33, 38, 42, 49] and Tree-based [1, 14, 17, 20, 23, 42] approaches. However, only a few studies included Textual analysis [33]. Some other detection techniques used in some studies are Formal methods [19], K-means clustering [48], Smith-Waterman algorithm [34, 35], Static flow analysis [28], and Concolic analysis [27, 31]. However, many studies used combination of different detection techniques to improve the efficiency of similarity detection of code clones.

RQ2: What is the overall research productivity in this domain?

The purpose of this research question is to identify the overall research productivity in code clone detection. This can be done by analyzing Chronological distribution of selected studies, most influential studies of the domain and potentially relevant publication sources.

Chronological distribution is used for the demonstration of increasing research interest in a particular field. Based on this, further research or future work can be conducted. This study basically fills the research gap of having no comprehensive systematic review since 2013. So, studies from 2013 to 2018 were selected accordingly. This distribution visualizes that the research on clone detection was on peak in 2015 and 2017. Fig. 4 visualizes the Chronological distribution of selected studies.

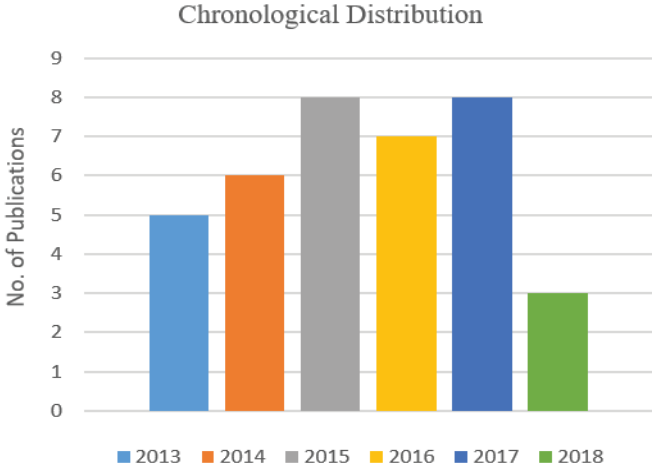

Figure 4: Chronological Distribution of Selected Studies

As, this SLR extracted studies from four main electronic databases which were IEEE Xplore, Springer Link, Science Direct and ACM. Most of the studies which were retrieved after selection procedure were from IEEE Xplore. However, full texts were retrieved from all of the four databases. Most influential studies of code clone detection having greatest count of citations are enlisted in Table V. Table shows top 10 studies in terms of citations having greatest citation count of [44].

Table 5: Most Influential Primary Studies

Ref.	Title	Citation Count
[44]	SourcererCC: Scaling Code Clone Detection to Big-Code	61
[1]	Deep Learning Code Fragments for Code Clone Detection	45
[34]	Gapped Code Clone Detection with Lightweight Source Code Analysis	34
[16]	A parallel and efficient approach to large scale clone detection	19
[36]	Pattern mining of cloned codes in software systems	16
[48]	Threshold-free code clone detection for a large-scale heterogeneous Java repository	14
[31]	CCCD: Concolic Code Clone Detection	13
[25]	Detecting Refactored Clones	13
[41]	SeByte: Scalable clone and similarity search for bytecode	10
[22]	Code Relatives: Detecting Similarly Behaving Software	09

Studies which were extracted from different databases have different publication venues. The overall research productivity can be found out by analysis of distribution of primary studies in these publication venues. Table VI enlists the distribution of primary studies along journals and conferences which clearly shows that the ratio of primary studies along conference proceedings are more than the journal articles. However, table shows that 22nd International conference on SANER has maximum count while the count of journals is same for all journal articles.

Table 6: Distribution of Primary Studies Along Journals and Conferences

Journals	#
Expert Systems with Applications	1
Journal of Software: Evolution and Process	1
Science of Computer Programming	1
Journal of Software Engineering and Applications	1
Computational Science and its applications	1
Journal of Software Engineering Research and Development	1
Information Sciences	1
Programming and Computer Software	1
Journal of Systems and Software	1
Science of Computer Programming	1
Procedia Computer Science	1

Conferences	#
Proceedings of the IEEE International Conference on Software Engineering and Service Sciences	1
Proceedings of the Thirty-Seventh Australasian Computer Science Conference (ACSC 2014), Auckland	1
Proceedings - 2017 IEEE International Conference on Software Maintenance and Evolution	1
Seventh International Conference on Intelligent Computing and Information Systems	1
Foundation of Software Engineering Conference	
Proceedings of the 2017 ACM SSIGPLAN workshop on partial evaluation and program manipulation	1
IEEE 12th International Workshop on Software Clones	1
European Conference on Object-Oriented Programming	1
11th IEEE International Workshop on Software Clones, co-located with SANER	1
Proceedings of the 30th annual ACM symposium on Applied computing	1
IEEE International Conference on Program Comprehension	1
24th Asia-Pacific Software Engineering Conference	1
Proceedings- Working conference on Reverse Engineering, WCRE	1
CSIT 2015 - 10th International Conference on Computer Science and Information Technologies	1
21st International Conference on Program Comprehension (ICPC)	
22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)	1
International Conference on Data and Software Engineering (ICODSE)	1
Confluence 2013: The Next Generation Information Technology Summit (4th International Conference)	1
15th IEEE International Conference on Machine Learning and Applications (ICMLA)	1
International Conference on Neural Information Processing (ICONIP)	1
38th IEEE International Conference on Software Engineering	1
2nd International Conference on Contemporary Computing and Informatics (IC3I)	1
IEEE International Conference on Software Engineering Companion	1
International Conference on Intelligent Computing and Control Systems ICICCS	1
International Conference on Automated Software Engineering	

5 Conclusion & Future Work

Code clone detection has emerged as a most dominant area of research. The detection of clones is necessary for improving the quality and maintenance of software systems. This SLR provides a comprehensive systematic review of all the existing research on code clone detection since 2013. After a detailed analysis, 37 primary studies were selected having different techniques to detect code clones which include text-based, token-based, tree-based, graph-based, metric-based and hybrid approaches. Results of this study reveals that PDGs and metric-based approaches are the mostly commonly used techniques to detect code clones. Although, many efficient hybrid approaches have been developed but still, the need is to improve the techniques in terms of accuracy and efficiency. Overall research productivity in code clone detection is defined by chronological distribution which visualizes the increasing research interest towards code clone detection in past few years. Lastly, this study presents preliminary results relevant to the two selective research questions. The extended version of this study will provide comprehensive discussion related to all defined research questions.

Acknowledgement

This research is supported by FAST-National University of Computer & Emerging Sciences (NUCES), Islamabad, Pakistan.

References

[1] M. White, M. Tufano, C. Vendome, and D. Poshyvanyk, "Deep learning code fragments for code clone detection," Proc. 31st IEEE/ACM Int. Conf. Autom. Softw. Eng. - ASE 2016, pp. 87–98, 2016.

[2] M. Tech, C. S. Engg, and M. Gobindgarh, "Hybrid Approach for Efficient Software Clone Detection," IRACST–Engineering Sci. Technol. An Int. J., vol. 3, no. 2, pp. 2250–3498, 2013.

[3] G. Chatley, S. Kaur, and B. Sohal, "Software clone detection: A review," Int. J. Control Theory Appl., vol. 9, no. 41, pp. 555–563, 2016.

[4] C. K. Roy and R. Koschke, "The Vision of Software Clone Management : Past , Present , and Future (Keynote Paper)," Softw. Maintenance, Reengineering Reverse Eng. (CSMR-WCRE), 2014 Softw. Evol. Week-IEEE Conf. on. IEEE, pp. 18–33, 2014.

[5] P. Prem, "A Review on Code Clone Analysis and Code Clone Detection," Int. J. Eng. Innov. Technol., vol. 2, no. 12, pp. 43–46, 2013.

[6] K. Kaur and R. Maini, "A Comprehensive Review of Code Clone Detection Techniques," vol. IV, no. Xii, pp. 43–47, 2015.

[7] M. Kapdan, M. Aktas, and M. Yigit, "On the Structural Code Clone Detection Problem: A Survey and Software Metric Based Approach," Iccsa 2014, vol. 8583 LNCS, no. PART 5, pp. 492–507, 2014.

[8] H. Murakami, K. Hotta, Y. Higo, H. Igaki, and S. Kusumoto, "Gapped code clone detection with lightweight source code analysis," IEEE Int. Conf. Progr. Compr., pp. 93–102, 2013.

[9] A. Sheneamer and J. Kalita, "A Survey of Software Clone Detection Techniques," Int. J. Comput. Appl., vol. 137, no. 10, pp. 1–21, 2016.

[10] K. Solanki and S. Kumari, "Comparative study of software clone detection techniques," 2016 Manag. Innov. Technol. Int. Conf., no. 1995, p. MIT-152-MIT-156, 2016.

[11] D. Rattan, R. Bhatia, and M. Singh, "Software clone detection: A systematic review," Inf. Softw. Technol., vol. 55, no. 7, pp. 1165-1199, 2013.

[12] S. E. Group, "Guidelines for performing Systematic Literature Reviews in Software Engineering," 2007.

[13] A. Gupta and B. Suri, "A Survey on Code Clone, Its Behavior and Applications," Netw. Commun. Data Knowl. Eng., vol. 27–39, 2018.

[14] A. Sheneamer, S. Roy, and J. Kalita, "A detection framework for semantic code clones and obfuscated code," Expert Syst. Appl., vol. 97, pp. 405–420, 2018.

[15] Q. Q. Shi, L. P. Zhang, F. J. Meng, and D. S. Liu, "A novel detection approach for statement clones," Proc. IEEE Int. Conf. Softw. Eng. Serv. Sci. ICSESS, pp. 27–30, 2013.

[16] M. Zanoni, F. Perin, F. A. Fontana, and G. Viscusi, "A parallel and efficient approach to large scale clone detection," J. Softw. Evol. Process, vol. 26, no. 12, pp. 1172–1192, 2014.

[17] X. Chen, A. Y. Wang, and E. Tempero, "A replication and reproduction of code clone detection studies," Conf. Res. Pract. Inf. Technol. Ser., vol. 147, no. ACSC, pp. 105–114, 2014.

[18] L. Li, H. Feng, W. Zhuang, N. Meng, and B. Ryder, "CCLearner: A deep learning-based clone detection approach," Proc. - 2017 IEEE Int. Conf. Softw. Maint. Evol. ICSME 2017, pp. 249–260, 2017.

[19] A. Cuomo, A. Santone, and U. Villano, "CD-Form: A clone detector based on formal methods," Sci. Comput. Program., vol. 95, pp. 390–405, 2014.

[20] R. Ami and H. Haga, "Code Clone Detection Method Based on the Combination of Tree-Based and Token-Based Methods," J. Softw. Eng. Appl., vol. 10, no. 13, pp. 891–906, 2017.

[21] A. Sheneamer and J. Kalita, "Code clone detection using coarse and fine-grained hybrid approaches," Intell. Comput. Inf. Syst. (ICICIS), 2015 IEEE Seventh Int. Conf. on. IEEE, pp. 472–480, 2015.

[22] F.-H. Su, J. Bell, K. Harvey, S. Sethumadhavan, G. Kaiser, and T. Jebara, "Code relatives: detecting similarly behaving software," Proc. 2016 24th ACM SIGSOFT Int. Symp. Found. Softw. Eng. - FSE 2016, pp. 702–714, 2016.

[23] T. Matsushita, "Detecting Code Clones with Gaps by Function Applications," Proc. 2017 ACM SIGPLAN Work. Partial Eval. Progr. Manip., pp. 12–22, 2017.

[24] R. Tajima, "Detecting Functionally Similar Code within the Same Project," Softw. Clones (IWSC), 2018 IEEE 12th Int. Work. IEEE, pp. 51–57, 2018.

[25] M. Shomrat and Y. A. Feldman, "Detecting refactored clones," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 7920 LNCS, pp. 502–526, 2013.

[26] C. M. Kamalpriya and P. Singh, "Enhancing program dependency graph based clone detection using approximate subgraph matching," IWSC 2017 - 11th IEEE Int. Work. Softw. Clones, co-located with SANER 2017, pp. 61–67, 2017.

[27] D. E. Krutz, S. A. Malachowsky, and E. Shihab, "Examining the effectiveness of using concolic analysis to detect code clones," Proc. 30th Annu. ACM Symp. Appl. Comput. - SAC '15, pp. 1610–1615, 2015.

[28] F. H. Su, J. Bell, G. Kaiser, and S. Sethumadhavan, "Identifying functionally similar code in complex codebases," IEEE Int. Conf. Progr. Compr., vol. 2016–July, pp. 1–10, 2016.

[29] M. S. Aktas and M. Kapdan, "Implementation of Analytical Hierarchy Process in Detecting Structural Code Clones," Int. Conf. Comput. Sci. Its Appl., vol. 2, pp. 652–664, 2017.

[30] R. H. Misu and K. Sakib, "Interface Driven Code Clone Detection," Asia-Pacific Softw. Eng. Conf. (APSEC), 2017 24th. IEEE, 2017.

[31] D. E. Krutz and E. Shihab, "CCCD: Concolic code clone detection," Proc. - Work. Conf. Reverse Eng. WCRE, pp. 489–490, 2013.

[32] A. Avetisyan, S. Kurmangaleev, S. Sargsyan, M. Arutunian, and A. Belevantsev, "LLVM-based code clone detection framework," CSIT 2015 - 10th Int. Conf. Comput. Sci. Inf. Technol., pp. 100–104, 2015.

[33] E. Kodhai and S. Kanmani, "Method-level code clone detection through LWH (Light Weight Hybrid) approach," J. Softw. Eng. Res. Dev., vol. 2, no. 1, p. 12, 2014.

[34] H. Murakami, K. Hotta, Y. Higo, H. Igaki, and S. Kusumoto, "Gapped code clone detection with lightweight source code analysis," IEEE Int. Conf. Progr. Compr., pp. 93–102, 2013.

[35] H. Murakami, Y. Higo, and S. Kusumoto, "ClonePacker: A tool for clone set visualization," 2015 IEEE 22nd Int. Conf. Softw. Anal. Evol. Reengineering, SANER 2015 - Proc., pp. 474–478, 2015.

[36] W. Qu, Y. Jia, and M. Jiang, "Pattern mining of cloned codes in software systems," Inf. Sci. (Ny)., vol. 259, pp. 544–554, 2014.

[37] B. Priyambadha and S. Rochimah, "Case study on semantic clone detection based on code behavior," 2014 Int. Conf. Data Softw. Eng., pp. 1–6, 2014.

[38] K. Raheja and R. K. Tekchandani, "An efficient code clone detection model on Java byte code using hybrid approach," 4th Int. Conf. Next Gener. Inf. Technol. Summit, Conflu. 2013, vol. 2013, no. 647 CP, pp. 16–21, 2013.

[39] S. Sargsyan, S. Kurmangaleev, A. Belevantsev, and A. Avetisyan, "Scalable and accurate detection of code clones," Program. Comput. Softw., vol. 42, no. 1, pp. 27–33, 2016.

[40] M. A. Nishi and K. Damevski, "Scalable code clone detection and search based on adaptive prefix filtering," J. Syst. Softw., vol. 137, pp. 130–142, 2018.

[41] I. Keivanloo, C. K. Roy, and J. Rilling, "SeByte: Scalable clone and similarity search for bytecode," Sci. Comput. Program., vol. 95, pp. 426–444, 2014.

[42] A. Sheneamer and J. Kalita, "Semantic Clone Detection Using Machine Learning," 2016 15th IEEE Int. Conf. Mach. Learn. Appl., pp. 1024–1028, 2016.

[43] M. Chawla and K. P. Miyapuram, "Software Clone Detection Using Clustering Approach," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 9490, pp. 467–474, 2015.

[44] H. Sajnani, V. Saini, J. Svajlenko, C. K. Roy, and C. V. Lopes, "SourcererCC: Scaling Code Clone Detection to Big Code," Softw. Eng. (ICSE), 2016 IEEE/ACM 38th Int. Conf. on. IEEE, no. 1, pp. 1157–1168, 2016.

[45] M. Sudhamani and L. Rangarajan, "Structural similarity detection using structure of control statements," Procedia Comput. Sci., vol. 46, no. Icict 2014, pp. 892–899, 2015.

[46] M. Sudhamani, "Code clone detection based on order and content of control statements," Contemp. Comput. Informatics (IC3I), 2016 2nd Int. Conf. on. IEEE, pp. 59–64, 2016.

[47] J. Svajlenko and C. K. Roy, "Fast and flexible large-scale clone detection with cloneworks," Proc. - 2017 IEEE/ACM 39th Int. Conf. Softw. Eng. Companion, ICSE-C 2017, pp. 27–30, 2017.

[48] I. Keivanloo, F. Zhang, and Y. Zou, "Threshold-free code clone detection for a large-scale heterogeneous Java repository," 2015 IEEE 22nd Int. Conf. Softw. Anal. Evol. Reengineering, SANER 2015 - Proc., pp. 201–210, 2015.

[49] G. Singh, "To Enhance the Code Clone Detection Algorithm by using Hybrid Approach for detection of code clones," Intell. Comput. Control Syst. (ICICCS), 2017 Int. Conf. on. IEEE, pp. 192–198, 2017.