BigCloneVis – A Clone Visualization Framework with BigCloneBench

Project Report

Zonayed Ahmed
Jenia Afrin Jeba

University of Saskatchewan
Department of Computer Science

Department of Computer Science
University of Saskatchewan

AALBORG UNIVERSITY
STUDENT REPORT

Title:
BigCloneVis – A Clone Visualization Framework with BigCloneBench

Theme:
Scientific Theme

Project Period:
Fall Semester 2018

Project Group:

Participant(s):
Zonayed Ahmed
Jenia Afrin Jeba

Supervisor(s):
Prof. Dr. Chanchal Roy

Copies: 1

Page Numbers: 9

Date of Completion:
January 5, 2019

Abstract:

For reusing code fragments, programmers frequently copy and paste code fragments for fast developments of software systems, but it yields into code clones. Code cloning results into various negative effects on software developments such as, programmers might have modified the code fragments in some places but not in others, henceforth resulting into code inconsistencies. Therefore, it is worth of using clone detection and visualization tool that might take input of code fragments and as output, finds out the clones and visualizes them. However, huge set of text-based data extracted by these tools is a challenge for the users due to lack of a common platform for evaluation, comparison and visualization. In this work, we propose an idea of a tool named BigCloneVis that aims to speed up the entire process of clone detection and visualization. Based on selected metrics and visualization techniques, BigCloneVis helps users to explore the results of different clone detection tools against tool-independent dataset, BigCloneBench.

Chapter 1

Introduction

1.1 What is Code Clone

A code segment which is similar (based on either textual representation or functionality) as any other code fragments in the source code is known as code clone. Code clones are the results of code copy-and-paste operations, forking, design, functionalities and logic reusing etc. These operations performed by the programmers during software development often make software maintenances tougher [1][2]. If programmers modify any code segment in a place (for e.g. for adding any new functionality or removing bugs etc.), they also need to check that its clones also get the same modification. Again, the practice of stable code cloning has many favorability (e.g. faster development) in software development [5,6,7]. Hence, it is important to detect, analyze and visualize code clones for further understanding the impact of clones in software maintenance and evaluation.
For detecting clones, researchers have put their efforts to develop clone detection tools which take particular code segment as input and then as output, they find out the code clones.

1.2 Why Clones exist in Software Systems?

There are various reasons for the existence of code clones in a system [2,4,5]. For meeting project deadlines, the developers may create clones since there might be shortage of enough time for creating proper abstractions; so generating or reusing the duplicated codes become necessary at that moment. Another reason might be that the developers need to solve the same problems that they've solved previously but now they are unwilling to rewrite the abstraction due to not understanding the codes, have lack of enough time or they do not understand the consequences generating code clones. A set of cases which influence the programming abstraction costly leading to the programmers to create cloned code is explained by Toomim et

al. [7]. Eight cloning patterns for explaining the interest of creating code cloning, has been identified by Kapser et al. [88]; they've also listed the detail advantages and disadvantages of code cloning. An extensive study of factors which initiate code cloning are explained by Roy and Cordy [8].

1.2.1 Adverse Consequences of Code Clones

Reusing a code fragment (with or without minor changes) which contains an error / bug in various parts of a system multiplies the possibilities of bugs [7,8].
Code base size is increased by code duplication, so extra time and effort is required by the developers for understanding the system, resulting into the requirement of more time and more challenges for making even simple system modifications. Again, if the developer has to change a particular part of a system he also needs to check other places of the system for detecting the other instances of duplication of that particular code.
Faulty design, bad inheritance structure, hidden dependencies among the duplicated codes in different parts of program or absence of programming abstraction etc. are introduced due to Code cloning making it harder in future projects.
The amount of time for understanding and deciding the adaptability of required changes into the system increases with the increase of the size of a code base. It needs to be thoroughly investigated to analyze the differences when many instances of the same code base exist in several parts of the system [77]; hence, increasing the effort. Furthermore, the code comprehension becomes a very challenging task when the copying-pasting of code fragments are not properly documented.

1.3 Existing Code Clone visualization techniques

There are various types of clone detectors, such as token-based [1], [2], text-based [4], [5], [6], syntax-tree-based [7], [8] metric based [9], [10], hybrid [11], [12] and graph-based [13], [14]. Again, various tools have been proposed for the visualization of those clones also [20]-[25] such as – clone pairs marked with file, line, program structure facts, clone metric and clone type etc. producing Hasse diagrams [15], flat lists [13], scatterplots [14], hyperlinked text, node-link views [11], polymetric views [2] and metric graphs [4] etc.
These clone visualization tools facilitate very fast-and-easy clone detection with a very easy-to-understand matching-unmatching ratio of the spotted clones for users.

1.3.1 Lexical Approaches

The idea of lexical or token based clone detection technique is to transform the source code into a series of tokens for detecting the cloned tokens. The first type, the DUP [1], can detect Type 1 and Type 2 clones, such as those types of duplicate code fragments which might be textually identical or similar after changing the variable / constants name with another name.
Kamiya et al. [2] has put their efforts for enhancing the technique with language specific transformation rules named as CCFinder.
To detect near-miss clones, Cordy et al. [3] also proposed a token and line-based approach. Since, syntax is not considered, clones detected by token-based techniques may sometimes overlap but applying pre-processing [1,2] or post processing techniques, clones corresponding to different syntactic units can be determined.

1.3.2 Text-based Approaches

In text-based approaches raw source codes are used for detecting copied codes. Therefore, this approach doesn't not require parsing. Either fingerprints on substrings of the source code [4] or SDD (Similar Data detection) algorithm, which uses N-neighbor distance concept [5] or language-independent technique which uses line based transformation (also provides scatter plot-based visualization) [6] – can be used for clone detection in this approach.

1.3.3 Graph-based Approaches

In graph-based technique [13] the program statements and expressions are represented by the nodes within the graph and the control flows and dependencies are represented by the edges. The source code is generally transformed into a sequence of program dependency graphs (PDG) though this approach can manage the deletion, insertion and non-contiguous code, reordering of statements, they fail to meet scalablity [14].

1.3.4 Syntax-Tree based Approaches

In this technique, at first the source code is transformed into an abstract syntax tree or a parse tree; then using tree matching algorithm, the duplicated code fragment is determined .
CloneDr is another token-based clone detection tool where the source code is converted into an abstract syntax tree to detect the duplicate code or code fragments [15].
Suffix trees for determining code clones in the abstract syntax tree representation has been proposed by Koschke et al. [16]. Another approach for detecting function clones using abstract syntax trees and suffix trees has been developed Tairas et al.

[17].
Jiang et al. [18] have introduced "Deckard" tool which utilizes the tree-based representation of source code to identify same subtrees with the help of particular numerical vectors for identifying code clones.

1.3.5 Metric-based techniques

In this technique, a set of metrics are collected which are used for characterization of the code fragments, named as fingerprinting functions. The determination of similar code fragments are done by doing the comparison of the metric vectors [9]. Cloned codes are detected either a function or syntactic units like a class or a method. For detecting similar web pages and clones in web documents, metric based clone detection techniques are very much useful [10].

1.3.6 Hybrid techniques

There are some existing works which use a mix of semantic and syntactic based clone detection technique [19]. NiCad [19], a hybrid technique tool which exploits the AST-based detection techniques to correctly identify near-miss clones.

1.4 Problem Statement

There are some difficulties in the aforementioned visualization tools specially for the new / general users; The results found from clone detectors are produced in textual format (listed with the names of source files and starting and ending locations of the lines where the clones were found) and those reports are bulky in size [11]-[14][17]. So, for users, comparing their own tool with the existing ones in terms of precision and recall on a single platform using one common dataset becomes complex because of the Limitation of visualizing techniques in terms of clone types and functionalities.

1.5 Motivation

The goal of the clone visualization technique is to create a platform to interact with programmers and clone researchers in representing clones and related information in an optical manner rather than raw textual data. Hence, a common platform to identify the differences among detectors and portray similarities and dissimilarities in a visual manner is a must.

1.6 Objective

In this work, we are going to propose a clone visualization tool named as BigCloneVis which intends to accelerate the whole procedure of assessment, evaluation and visualization of text-based data extracted by clone detection tools against tool-independent dataset, BigCloneBench.
The contribution of this work is to facilitate a common platform to compare all clone detection tools and evaluation based on their results, which, to the best of our knowledge hasn't been done before.

1.6.1 Research Methodology

A new visualization tool based on BigCloneEval, namely BigCloneVis is proposed that identifies the patterns of clone detection tools and answers the previous research questions.
1. The required input files for these tools are the reports generated by all the clone detection tools.
2. The tools will have different control panels consisting of clone types, functionality types of IJA dataset, tool types, clone classes, project types of BigCloneBench, revision types for different tools, type of visualization (dot plot, bar plot, hive plot).
3. Upon selection from the control panel by the user, the tool will automatically demonstrate the relationships among clones and tools, clones and functionalities, clones and projects.
4. One interface will generate tree structure based on genealogy of clone files. Zooming on the nodes will provide information about the clones.
5. Another interface will represent changing clones over revisions of a clone detection tool.
6. The last interface will represent commonality among the chosen tools in terms of clone types or functionality.

Chapter 2

Related works

We have already discussed some of the existing clone detection tools in chapter 1 in section 1.3. In this chapter, some more existing works and approaches are discussed.

2.1 Literature Review

Murakami [20] proposes a tool named ClonePacker that visualizes detected clones using Circle Packing with file hierarchies. The tool reduces the time of browsing detected clones comparing with Libra [26]. While it reduces the time to identify clones, it uses predefined token length, targets one software and uses experienced programmers.
SoftGUESS [21] is a code clone exploration system, built on GUESS [27], represents clone evolution in conjunction with containment relationships and structural dependencies. The tool definitely provides users an opportunity to investigate clones in single version snapshots as well as their changes over multiple revisions but it is limited to dataset generated for one system, does not compare among existing tools and is not specific to types or functionalities of clones.
D-CCFinder [22] visualizes a global view of open source software systems. The output of this tool is type-1 and type-2 clones (as defined in [28]) existing in target composed of multiple unique categories. But the limitation of their findings is consideration of one software system and not evaluating their framework compared to any benchmark dataset.
VisCad [23] supports and in-depth analysis of near-miss hybrid clone detection tool, NiCad [29]. While visualizes all exact, renamed and near-miss gapped clones detected by NiCad, identifying patterns among all clone detection tools is not possible by VisCad.

SolidSDD [24] offers visualization with four level of details: overview, zoom, filter, clone details. It also shows clones in file context and allows navigating among clone pairs, shows clone vs system structure as well as clone and file metrics. Though the tool uses several data representation techniques to make quantification and examination of clones easier and faster compared to when using separate tools, it doesn't implicate any relation among the clone detection tools.
Authors in [25] integrates two different tools namely CloneDR and AJDT Visualizer Plugin to detect and visualize both textual and graphically. The limitation is that they do not evaluate their tool against any benchmark data.
In [26] authors have showed the use of Hasse diagram for representing the textual resemblance between the source files. Afterwards, they suggested hyper-linked web pages for exploring the clone classes and files [27].
For offering an interactive appearance of clones, authors [27] proposed HTML based outline of the clone classes shown in a web page with hyperlinks; by clicking on those links users can peruse the details of individual clone class. Even if such illustrations offer speedy navigation, they cannot expose the high-level relations between clone classes.
Scatter plot is the most popular method of visualization technique among various visualizations for visualizing different levels of inter-system and intra-system cloning [28]. But there is a limitation of it which is that rather than depending on the amount of cloning, the size of scatter plot visualization depends on the size of input and non-contiguous sections encompassing the same set of clones cannot be assembled together. Therefore, use of this technique in any large system may become thought-provoking due to the big size of the plot. To solve this issue, authors [29] came with an idea of presenting individual source file using a bar and clones within the files using stripes (clones in the same class are presented using similar color).
Evaluation of modern clone detection tools to compare their precision and recall is a demanding task. Svajlenko and Roy proposes a framework named BigCloneEval [30] that evaluates clone detection tools using BigCloneBench [31], a real-world tool-independent benchmark of manually validated clones in the inter-project Java repository IJaDataset-2.0 [32] (25,000 Java systems). While it measures both inter-project and intra-project clones with recall summarized for all tools per clone type and per functionality in the benchmark, it only provides a tool evaluation report that summarizes the data without any graphical representation.

Chapter 3

Conclusion

Though the research on code cloning detection and analysis is very significant in current software research, the availability of a common platform to compare all clone detection tools and evaluation based on their results is uncommon. Moreover, the same platform providing visualization to portray similarities and identifying trends among all the tools is not present in the status-quo. Consequently, this tool, integrated in a web platform can help the current researchers to work faster and encourage new ones by reaching them easily.