Non-Trivial Software Clone Detection
Using Program Dependency Graph

Pratiksha Gautam, Jaypee University Of Information Technology, Solan, India
Hemraj Saini, Jaypee University Of Information Technology, Solan, India

ABSTRACT

Code clones are copied fragments that occur at different levels of abstraction and may have different origins in a software system. This article presents an approach which shows the significant parts of source code. Further, by using significant parts of a source code, a control flow graph can be generated. This control flow graph represents the statements of a code/program in the form of basic blocks or nodes and the edges represent the control flow between those basic blocks. A hybrid approach, named the Program Dependence Graph (PDG) is also presented in this article for the detection of non-trivial code clones. The program dependency graph approach consists of two approaches as a control dependency graph and a data dependency graph. The control dependency graph is generated by using a control flow graph. This article proposes an approach which can easily generate control flow graphs and by using control flow graph and reduced flow graph approach, the trivial software clone, a similar textual structure, can be detected. The proposed approach is based on a tokenization concept.

KEYWORDS

Control Flow, Cyclomatic Complexity, Graph Software Clones, Program Dependency Graph

1. INTRODUCTION

The process of reusing existing code fragments by copying and pasting it with or without minor modifications to another location is called code cloning and the pasted fragment is known as code-clone. The software becomes more and more complicated if a bug is detected in one section of the code due to replication and then it requires corrections in all the replicated fragments of the code. Thus, its analysis and detection is an emerging issue due to high maintenance cost (Sharma, 2011) as well as improving the quality, structure and design of the software system. The reported literature reveals that, 66% of the source codes are cloned (Marcus & Jonathan, 2001; Tairas & Jeff, 2006; Puri & Kumar, 2006) therefore it is potentially useful to find the code-clones to improve the quality of software systems. However, various code clone detection techniques have been proposed in the literature. Generally, clone detection technique consists of two phases, the first phase is transformation phase and the second phase is comparison phase. Further, in transformation phase the source code is transformed into internal representation as lines, tokens, abstract syntax tree or program dependency graphs while in comparison phase an appropriate comparison algorithm or an approach is used to detect duplicated clones from the source code. Therefore, based on the internal representation, clone detection techniques can be assorted as two types such as string-based, token-based and the other approach of transformation phase is PALEX –based technique which can be divided into two parts as syntactic-based and semantic-based as shown below in Figure 1.

DOI: 10.4018/IJOSSP.2017040101

Copyright © 2017, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.
Figure 1. Classification of code clone detection techniques

Figure 1 shows the taxonomy of clone detection techniques. The code clone detection techniques (Gautam & Saini, 2016) can be classified on the basis of two phases. The first one is a transformation in which program text is transformed into the form of stream, tokens, abstract syntax tree and graph-based and the second phase is comparison phase. Moreover, transformation phase is further assorted as internal code representation and PALEX source code representation. Internal code is again classified into text/string-based and token/lexical-based. The PALEX source code representation is divided into two parts as syntactic and semantic-based. In addition, syntactic-based is classified into two parts as tree-based as well as metric-based and syntactic-based approach is characterized into two parts as graph-based and hybrid-based (Roy et al., 2009).

In this paper, our primary aim is to detect non-trivial code-clones (different structure, but perform the same functionality) using program dependency graph (PDG). Although, existing techniques detect syntactic type clones (structurally similar) it is very difficult to detect non-trivial code clones using any of the existing static analysis techniques (Kulkarni & Metta, 2014). Besides, program dependency graph (PDG) is also a static analysis technique for detection of semantic clones (Patil et al., 2015; Tekchandani et al., 2013). A PDG encodes both control and data dependencies of each operation in a program using directed edges connecting graph vertices, edges represent data and control dependencies and vertices corresponding to statements and control predicates (Ferrante et al., 1987). The data dependence has been used to represent the relevant data flow relationships of a program while control dependence represent control flow relationship of a program which is derived from the control flow graph.

The technical contributions in this manuscript are as follows:

- Detect non-trivial code clones using static analysis technique as program dependency graph.
- To propose an approach for generating control flow graph.
Consumer Welfare and Market Structure in a Model of Competition between Open Source and Proprietary Software
www.igi-global.com/article/consumer-welfare-market-structure-model/4089?camid=4v1a

Motivation of Open Source Developers: Do License Type and Status Hierarchy Matter?
www.igi-global.com/article/motivation-open-source-developers/41949?camid=4v1a