Available online at www.sciencedirect.com

ScienceDirect

Procedia Engineering 192 (2017) 136 – 141

Procedia Engineering

www.elsevier.com/locate/procedia

TRANSCOM 2017: International scientific conference on sustainable, modern and safe transport

Current trends in source code analysis, plagiarism detection and issues of analysis big datasets

Michal Ďuračík[a], Emil Kršák[a]*, Patrik Hrkút[a]

[a]*University of Zilina, Faculty of Management Science and Informatics, Univerzitná 8215/1, 010 26 Zilina Slovakia*

Abstract

In this work, we analyze the state of the art in source code analysis area with a focus on plagiarism detection and provide a proposal for a future work in this area. Detection of plagiarism combines the detection of clones and methods for determining similarity. Nowadays, there are several approaches that can be divided into three levels. The first one is text based and uses plain text as an input. The second level is token based. The top level is model based and uses models to represent source code. These advanced algorithms (token and model based) can't work with large datasets. We believe the future belongs to the algorithms that will be able to handle large amount of source code. These algorithms should use one of model-based representations. They can be used for formation of large-scale anti-plagiarism systems. They can be used also in the area of source code optimization.

Keywords: source code analysis; plagiarism; current trends; transport; optimizations

1. Introduction

Source code analysis has existed ever since programming started. First, programming languages did not use to be very complex, and analyzing them did not require much effort. In the course of time the individual methods of analysis developed, as well as programming languages did. Currently, the computing speed of computers (especially

* Corresponding author. Tel.: +421-41-513-4050.
E-mail address: emil.Krsak@fri.uniza.sk

doi:10.1016/j.proeng.2017.06.024

CPU and the size of RAM), number of source codes, of programming languages and programmers are increasing constantly. This represents new challenges for the analysis of source codes. The trend is likely to keep continuing, and the more people will do programming, the more programming languages there will be, and analyzing them will be more and more demanding from year to year.

In his article *Source Code Analysis: A Road Map* [1] from the year 2007, David Binkley describes methods that are still current and used in the analysis of source codes. Besides them, he deals with its future. According to him, the times when the duration of an analysis has been more serious of an issue than the volume of data has are almost over, and, gradually, the volume of data is going to constitute a problem bigger than the problem of time. The algorithms used currently will have to adapt over time.

2. Source code and its analysis

A source code is a text usually written by human programmers, and except for them, also the computer as a compiler or interpreter needs to understand it. Therefore, the text contains information necessary for both, the programmer and the computer, to understand the given code. For the computer to understand a source code, the code has to meet the syntax and grammar of the given language. As far as programmers are concerned, there are often formal and informal conventions that make the source code readable. Such conventions may include naming identifiers, documentation comments, code formatting etc. Sometimes these conventions use programming languages directly, and they add them to their grammar (e. g. in the programming language Python, indent is used right in the language to define blocks). It often happens that the conventions are not required directly by the programming language but by the frameworks which are built on it. Then the names of the individual identifiers are not a matter of the programmer´s choice but they depend on the tool / environment that the programmer works in. With some languages, also documentation comments have been applied. For example PHP, which is a dynamic-type, and programmers needing types when developing their frameworks (e. g. when using Dependency Injection) used the mechanism of documentation comments. These have become standardized, and they are understood by development environments, however, also a language can process them by reflection.

When analysing a source code, this attribute needs to be considered, to select the right analysis method. Processing the structure of source code does not present a major problem since the source code has to meet the grammar of the given language, so that it is possible to compile. Processing the meaning, respectively the idea a programmer has added into the code constitutes bigger challenge, regarding the fact there are various programmers with various levels of experience. As the problem is a complex one, the current methods of source code processing do not aspire to solve everything at once but they approach the source code from a particular point of view. Some analyze the code from the programmer´s point of view, and they try to understand its meaning, while others explore its structure. We can divide these methods into three levels based on how they approach the source code. The first of them is the analysis of source code as a plain text. With the method, it is presumed the programmer complies with the conventions, and the source code contains sufficient information describing its meaning. The algorithms being used are supposed to extract just these additional information. The next level is similar to the previous one but it does not explore the meaning of the text from the programmer´s viewpoint. It does so from the viewpoint of the computer that ´sees´ the source code as a list of commands. The individual commands sequences have their meaning in the context of the language grammar, and they tell the computer what it is supposed to do. The last level is the analysis of source code model.

3. Source code analysis using the means of NLP

We consider **Natural Language Processing (NLP)** as the basic method of acquiring information from text documents. The term natural language refers to the language people use in common communication (Slovak, English ...). NLP allows to process a natural language by using computers. Normally, the process has a few phases during which the entry document is processed gradually. In the first phase terms are extracted from the source document. They are represented in a certain way, and, consequently, by means of various classification methods, they are analyzed. Extracting terms from source code differs from extracting terms from text documents mainly by the form of text tokenization. Fortunately, there are good tokenization tools for each programming language.