AJJSME

Advanced Java JavaScript Metrics Extractor
First project interim report

Anastasia Trimasova
Vyacheslav Bikbaev

February 2017

1 Problem statement

Software metrics play an important role in the software development area. Correct metrics allow to make the process of the software development more suitable for the current needs of an organization. Therefore, many authors try to evaluate the shortcomings of existing metrics and the need for new metrics especially designed for object-orientation. [3]

The goal of this project is to extract metrics from a JavaScript code. To understand what the metrics are and their importance, several articles were read by ourselves like [3, 6, 7, 5].

2 How JavaCC works

JavaCC is a parser generator and a lexical analyzer generator. Compilers and interpreters incorporate lexical analysers and parsers to decipher files containing programs. [2]

Lexical analyser breaks a sequence of input characters into *tokens* and then classify them. The parser then analyses the sequence of tokens to determine the structure of the program. If the input is not correct, according to lexical or syntactic rules of the language, the lexical analyser and parser generate error messages. JavaCC itself produces lexical analysers and parsers written in Java according to a specification that it reads in from a file with language grammar in EBNF. [4] This grammar file is named with the extension .jj by convention. [1]

JavaCC generates seven Java classes from a JavaCC specification file, each in its own file [2, 1]:

- **TokenMgrError** Sub class of Error. Throws exceptions on errors in lexical analysis.
- **ParseException** Sub class of Exception. Throws exceptions on errors detected by the parser.
- **Token** Class for representing a token. Each token is associated with a 'token kind' which represents the type of the token. A string 'image' represents the character sequence associated with the token.
- **SimpleCharStream** Implementation of the interface character stream for the lexer.
- **ParserConstants** An interface for defining the token classes used in the lexer and parser.
- **ParserTokenManager** The lexical analyzer class.

- **Parser** The parser class.

The parser gets tokens from a lexical analyser that reads input from the stream. In the case of a lexical error (an unexpected character in the input) the program will throw a TokenMgrError. The parsing error happens when the sequence of tokens does not match the parser specification and in this case the program will throw a ParseException. Finally, if input contains a sequence of tokens matching the parser specification, no exception is thrown and the program simply terminates.

3 Grammar for the JavaScript language

Fortunately, we have found 2 working JavaCC grammars for JavaScript. One of them is written for old standard of JavaScript, while other targeting one of the newest versions of JavaScript - EcmaScript 5. (Names JavaScript and EcmaScript are used interchangeably). For our project, we are using the newer grammar.
Link: EcmaScript 5 grammar

4 What we have done

First of all, we have read several articles and guides, as stated above, to understand how JavaCC works and how to make grammar work.
After this preparation stage, we have started searching for grammar. After trying several of them, we have chosen appropriate (like described in above 3).
Next step was to make the grammar work. The tricky part was to find latest version of JavaCC. Unfortunately, official web page is slightly outdated. Luckily, newest version was obtained on Maven repository. Also slightly difficult was to satisfy all dependencies of the grammar.
Later, we have written a simple wrapper class to play with parser and understand, how it works.
Further we started to implement metrics. To start with, we have chosen most simple metrics: lines of code and amount of declared functions in the source code. Obviously, counting lines of code doesn't require a JavaScript parser. However, counting functions is much more interesting case. It partialy works at the moment.
Simultaneously with implementing metrics we have written some unit-tests, which are described below.

5 First set of tests

First set of tests checks if parser parses basic language constructions correctly, like:

- For and For..In statements
- While and Do..While statements
- If statement
- Function and variable declarations

Also, we have written a battery of tests in TDD style to check if metrics works correctly. Since we have implemented only 2 metrics, our unit-tests only verify those cases.
Link: Basic Constructions Tests
Link: Metrics Tests

6 Github

Link: https://github.com/Alcovaria/AdvancedJavaMetrics

References

[1] An Introduction to JavaCC. https://www.codeproject.com/Articles/35748/An-Introduction-to-JavaCC.

[2] The JavaCC Tutorial. http://www.engr.mun.ca/~theo/JavaCC-Tutorial.

[3] Shyam R Chidamber and Chris F Kemerer. A metrics suite for object oriented design. *IEEE Transactions on software engineering*, 20(6):476–493, 1994.

[4] Viswanathan Kodaganallur. Incorporating language processing into java applications: A javacc tutorial. *IEEE software*, 21(4):70–77, 2004.

[5] Michele Marchesi, Giancarlo Succi, and Williams Laurie. Traditional and agile software engineering, chapter 24 – measuring development.

[6] Sanjay Misra and Ferid Cafer. Estimating quality of javascript. *The International Arab Journal of Information Technology*, 9(6), 2012.

[7] Di Wu, Lin Chen, Yuming Zhou, and Baowen Xu. A metrics-based comparative study on object-oriented programming languages. In *SEKE*, pages 272–277, 2015.