RefDetectorJS for Mining Refactoring Data from Version Controlled Repositories

Vincius Garcia, *Student, UFMG*

Abstract—This paper discuss the development of a system designed to detect refactorings on diferent versions of Javascript code files using the same heuristics used by RefDetector tool for Java programming language. RefDetectorJS use Objects and functions signatures to detect refactoring between two different versions of the code. The focus of the project is on Javascript modules designed to be compiled with Nodejs.

Keywords—Computer Science, Refactoring, UFMG, Javascript, RefDetector.

1 Introduction

Refactoring is a frequent and well characterized behavior that can be used to evaluate and try to understand human behavior and habits when developing computer programs. To help these studies is necessary to create tools that can detect refactoring and return useful information about it. The RefDetector tool was developed by M. Danilo Silva of Federal University of Minas Gerais (UFMG) and aim refactoring detection on programs written in the Java programming language. This project discuss the development of a second tool called RefDetectorJS that aims to detect refactoring in Javascript source files.

RefDetectorJS is coded on Javascript and uses the same heuristics of M. Danilo's RefDetector. The main approach used is to collect a set of signagures from two versions of a source file and then compare the differences to identify refactorings between the two versions. The main objective is to provide a tool for mining javascript refactoring history from Javascript version controlled repositories.

The tool is implemented in Javascript and interpreted with Nodejs. To execute the tool it expect as parameters two consecutive versions of a same Javascript source file and outputs a log of refactorings detected between the two versions.

- *M. Danilo Silva is with the Department Computer Science, Federal University of Minas Gerais, Brazil.*

October 17, 2014

2 Approach

Since Javascript have no class concept it was needed to apply some restrictions on the type of objects that should be read as classes. Those objects signatures are read with the same criteria used on Java Classes by M. Danilo's RefDetector, the other are read just as attributes (e.g. an array or a dictionary). It was also necessary to apply some restrictions to simplify the first release of the tool. For instance we use the following criteria:

- The objects for be treated as classes must be declared with a constructor function so that RefDetectorJS so that they may be instantiated with the "new" operator.
- If an object inside one other object is declared with constructor function it will be read only as an attribute for simplicity purposes.
- Functions and attibutes declared outside those objecs are not being parsed yet. This way the first release of the tool will follow the same structure used on M. Danilo's RefDetector for Java.

To compare the two files RefDetectorJS uses reflection to recover Javascript object names,

attributes and functions. Since Javascript reflection is quite limited to recover more detailed information like the set of a function parameter's names and the set of references made insde the body of a function to attributes of this same "class" it was needed to parse the source looking for the reserved words: "function", "var". To detect implicit variable declarations it was needed to look for the assign operator "=".
The signature of the objects, functions and attributes is collected as a tuple of sets and was based on Biegel et al.[1] method:

- The objects read as classes have as signature the tuple: (n, m) the name and the set of class members respectively.
- The class attributes have as signature the tuple: (c, f) where c is the class name and f is the attribute name. The type of the attribute could not be used since Javascript has dynamic typing.
- The class functions have as signature the tuple: (c, m, p, b) where c is the class name, m is the function name, p is the parameters list and b is the set of class members being accessed within the body of the function.

3 Conclusion

The tool is not fully functioning yet but soon it will make the first tests with the BETA release. It is expected that the tool will be less effective on Javascript than it is on Java since the former dynamism prevent the tool of accessing some of the desired signatures for example the attribute's and function's type can't be used. Other problem that one might expect is the consequences of a programmer using the Javascript "eval" feature, that could make unexpected behaviors on the code since it can't be parsed in a static analysis.

References

[1] Benjamin Biegel, Quinten David Soetens, Willi Hornig, Stephan Diehl, and Serge Demeyer. Comparison of similarity metrics for refactoring detection. In Proceedings of the 8th Working Conference on Mining