## Abstract

Duplication in source code is often seen as one of the most harmful types of technical debt as it increases the size of the codebase and creates implicit dependencies between fragments of code. Detecting such problems can provide valuable insight into the quality of systems and help to improve the source code. To correctly identify cloned code, contextual information should be considered, such as the type of variables and called methods.

Comparing code fragments including their contextual information introduces an optimization problem, as contextual information may be hard to retrieve. It can be ambiguous where contextual information resides and tracking it down may require to follow cross-file references. For large codebases, this can require a lot of time due to the sheer number of referenced symbols.

We propose a method to efficiently detect clones taking into account contextual information. To do this, we propose a tool that uses an AST-parsing library named JavaParser to detect clones a retrieve contextual information. Our method first parses the AST retrieved from JavaParser into a graph structure, which is then used to find clones. This graph maps the following relations for each statement in the codebase: the next statement, the previous statement, and the previous cloned statement.

# Statement-level AST-based Clone Detection in Java using Resolved Symbols

Simon Baars, Ana Oprescu

November 6, 2019

## 1 Introduction

Duplicate code fragments are often considered as symptoms of bad design [2]. They create implicit dependencies, thus increasing maintenance efforts or causing bugs in evolving software. Changing one occurrence of such a duplicated fragment may require other occurrences to be changed as well [3]. Also, duplicated code has been shown to add up to 25% of total system volume [1], which entails more code to be maintained.

A lot of tools have been proposed to detect such duplication issues [4, 7, 5]. These tools can find matching fragments of code, but do not take into account contextual information of code. An example of such contextual information is the name of used variables: many different methods with the same name can exist in a codebase. This can obstruct refactoring opportunities.

We describe a method to detect clones while taking into account such contextual information. Our experiments show us that

## 2 JavaParser

An important design decision for CloneRefactor is the usage of a library named JavaParser [6]. JavaParser is a Java library which allows parsing Java source files to an abstract syntax tree (AST). JavaParser allows to modify this AST and write the result back to Java source code. This allows us to apply refactorings to the detected problems in the source code.

Integrated into JavaParser is a library named SymbolSolver. This library allows for the resolution of symbols using JavaParser. For instance, we can use it to trace references (methods, variables, types, etc) to their declarations (these referenced identifiers are also called "symbols"). This is useful for the detection of our refactoring-oriented clone types, as they make use of the fully qualified identifiers of symbols.

To be able to trace referenced identifiers, SymbolSolver requires access to not only the analyzed Java project but also all its dependencies. This requires us to include all dependencies with the project. Along with this, SymbolSolver solves symbols in the JRE System Library (the standard libraries coming with every installation of Java) using the active Java Virtual Machine (JVM).

## 3 Conclusion

## References

[1] Magiel Bruntink, Arie Van Deursen, Remco Van Engelen, and Tom Tourwe. On the use of clone detection for identifying crosscutting concern code. *IEEE Transactions on Software Engineering*, 31(10):804–818, 2005.

[2] Martin Fowler. *Refactoring: improving the design of existing code*. Addison-Wesley Professional, second edition, 2018.

[3] J. Ostberg and S. Wagner. On automatically collectable metrics for software maintainability evaluation. In *2014 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement*, pages 32–37, 10 2014.

[4] Chanchal K Roy, James R Cordy, and Rainer Koschke. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Science of computer programming*, 74(7):470–495, 2009.

[5] Abdullah Sheneamer and Jugal Kalita. A survey of software clone detection techniques. *International Journal of Computer Applications*, 137(10):1–21, 2016.

[6] Nicholas Smith, Danny van Bruggen, and Federico Tomassetti. Javaparser, 05 2018.

[7] Jeffrey Svajlenko and Chanchal K Roy. Evaluating modern clone detection tools. In *2014 IEEE International Conference on Software Maintenance and Evolution*, pages 321–330. IEEE, 2014.