A Code Clone Detection DSL for Java

Anonymous Author(s)

Abstract

Duplication in source code is generally considered an undesirable pattern, because it unnecessarily increases system volume and is prone to cause bugs. Many studies propose definitions and techniques to detect such duplication issues. The clones detected by these definitions and techniques differ greatly. This is because there is a trade-off between completeness and number of false-positives. Additionally, clone detection may have different purposes, which may require different clone detection techniques.

We propose a DSL for clone detection, that allows for the specification of clone definitions. Using this DSL, code clone definitions can be easily expressed, whilst not having to worry about the actual clone detection algorithm and its optimization. Our DSL allows to exclude expressions, statements and declarations from clone matching. Our DSL supports line-based and AST-based clone detection approaches. The DSL can also take into account context information of expressions.

We use the DSL to express type 1, 2 and 3 clone definitions. We then use these definitions to validate the DSL over a corpus of open-source Java projects. We compare these results with established clone detection tools. We find that our DSL matches the output of the control tools, while allowing for a much wider spectrum of configurations.

Keywords clone detection, domain specific language, language engineering, meta-programming

1 Introduction

Code clones argue about the duplicate code present in the source code of software systems. An abundant number of clone detection tools and techniques have been proposed in the literature due to the many applications and benefits of clone detection [?]. Clones are most often the result of code reuse by copying and pasting existing code, among other reasons [?]. Clones are often seen as harmful [? ? ?], but can also have positive effects on system maintainability [? ? ?].

Clone detection is applied in various domains. This includes (automated) refactoring, education (plagiarism detection) [?], (legacy code) modernization [?] and maintainability analysis [?]. For these varying purposes, many clone detection tools have been proposed: a 2013 survey by Rattan et al. [?] surveyed more than 70 clone detection tools. Comparisons show large differences in recall and precisions of these tools [?].

To allow easy experimentation with code clone defitions and methods, we propose a DSL to configure clone detection.

2 Language Design

3 Results

4 Discussion

5 Conclusion

5.1 Threats to validity

PL'18, January 01–03, 2018, New York, NY, USA
2018.