Real-time management of code clones in an IDE environment

Jakob Hansen

March 1, 2022

Contents

1 Introduction

Refactoring is the process of restructuring code in order to improve the internal behavior of the code, without changing the external behavior.[2, 9] Refactoring is often done in order to eliminate "smelly" code.

A study conducted by Diego Cedrim et al.[1] has shown that while developers tend to refactor smelly code, they are rarely successful at eliminating the smells they are targetting. A large portion of refactorings even tend to make the code smellier. Therefore, automated tools to help developers make better refactorings and code analysis is an important field of research.

Duplicated code is a code smell which occurs in practically every large software project. Code clone analysis has recently become an active field of research and many tools have been developed to detect duplicated code.[3] However, few of these tools have made an impact on the industry, and few have the capability of detecting advanced types of duplicated code in a real-time IDE environment.

This thesis will present a proposal for a tool which will fill this gap for an industry viable clone detection tool. It will explore the topics of finding clones in real-time, managing code clones in real-time and in providing clone detection (and refactorings in general) in a modern IDE environment.

2 Background

2.1 Software quality

2.1.1 Software quality metrics

2.1.2 How refactoring affects software quality

2.1.3 Duplicated code

Write about what duplicated code is, how it affects software and some statistics on duplicated code (need reference)

As stated, duplicated code damages software quality in practically every large soft-

ware project. Duplicated code can lead to a plethora of anti-patterns like Shotgun-Surgery and Divergent-Change, and will often lead to an increase in technical debt for the project.[2, 99]

2.2 Code clones

We define a code snippet as a piece of software code in a larger software system. A code clone is then defined as a code snippet which is equal to or similar to another code snippet. The two code snippets are both code clones and together they form a code clone pair.

2.2.1 The clone relation

The clone relation defines a relation between code snippets where snippets which are code clones are related to each other. The clone relation is reflexive and symmetric, but not always transitive. The transitive property depends on the treshold for similarity when identifying code clones. Given

$$a \xrightarrow{clone} b \xrightarrow{clone} c$$

where a, b, c are code snippets and $\xrightarrow{clone}$ gives the clone relation, a is a clone of b, but not necessarily similar enough to be a clone of c, depending on the threshold for similarity.

2.2.2 Code clone types

Code clones are generally classified into four types.[3] These types classify code snippets as code clones with an increasing amount of leniency. Therefore Type-1 code clones are very similar, while Type-4 clones are not necessarily similar at all. However, all code clones do still have the same functionality, it is the syntactic and structual differences which distinguish the types. The set of code clones classified by a code clone type is also a subset of the next type, meaning all type-1 clones are also type-2 clones, but not vice versa.

The code clone types are defined as follows:

Type-1 clones are syntactically identical. The only differences allowed are elements without meaning, like comments and white-space.

Type-2 clones are structurally identical. Possible differences include identifiers, literals and types.

Type-3 clones are required to be structurally similar, but not equal. Differences include statements which are added, removed or modified. For this clone type one needs to determine a threshold θ which determines how structurally different snippets can be to be considered Type-3 clones.[3]

Type-4 are clones without any requirement for syntactical or structural similarity. Therefore the only requirement is having the same functionality.

2.2.3 Code clone detection

2.2.4 Code clone management

3 The way forward

References

[1] Diego Cedrim, Alessandro Garcia, Melina Mongiovi, Rohit Gheyi, Leonardo da Silva Sousa, Rafael Maiani de Mello, Baldoino Fonseca, Márcio Ribeiro, and Alexander Chávez. Understanding the impact of refactoring on smells: a longitudinal study of 23 software projects. In Eric Bodden, Wilhelm Schäfer, Arie van Deursen, and Andrea Zisman, editors, *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*, pages 465–475. ACM, 2017.

[2] Martin Fowler. *Refactoring - Improving the Design of Existing Code*. Addison Wesley object technology series. Addison-Wesley, 1999.

[3] Katsuro Inoue. *Introduction to Code Clone Analysis*, pages 3–27. Springer Singapore, Singapore, 2021.