Real-time management of code clones in an IDE environment

Jakob Hansen

March 9, 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 a highly active field of research and many tools have been developed to detect duplicated code.[3, 7] However, few of these tools have made an impact on the industry, and few have the capability of detecting advanced types of duplicated code and managing them in a real-time IDE environment.

This thesis will present a tool for industry viable clone detection and management. It will explore the topics of finding and managing clones in real-time, refactoring-oriented clone detection and providing clone management 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

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

Type-1 clones are often referred to as "exact" clones, while Type-2 and Type-3 clones are often referred to as "near-miss" clones.[7, 1]

2.2.3 Code clone detection and management

Write about different techniques which are used and how well they work

2.2.4 Clone aware development

Developers are often not aware of the creation of clones in their code. Clone aware development involves having clone management as a part of the software development process. Since code clones can be hard to keep track of and manage, tools which help developers deal with clones are useful. However, Mathias Rieger et al. claims that a problem with many detection tools is that the tools "report large amounts data that must be treated with little tool support."[5, 1]. Existing tools which partly solves this problem are presented below.

2.2.5 IDE-based clone management

There are many existing clone management tools, however the most useful tools for clone aware development are the tools which are integrated into an IDE and offer services to the programmer while developing in real-time.

The IDE-based tools which exist can be categorized as follows[6, 8]:

- *Copy-paste-clones:* This category of tools deals only with code snippets

which are copy-pasted from another location in code. These tools therefore only track clones which are created when copy-pasting, and does not use any other detection techniques. Therefore this type of tool is not suitable for detecting clones which are made accidentally, since developers are aware that they are creating clones when pasting already existing code snippets.

- *Clone detection and visualization tools:* This category of tools have more sophisticated clone detection capabilities and will detect code clones which occur accidentally.
- *Versatile clone management:* This category of tools cover tools which provide more services than the above. Services like refactoring and simultaneous editing of clones fall under this category.

There are a few existing IDE-tools which have seen success in real-time detection of clones:

- Minhaz et al. introduced a technique for performing real-time focused searches, i.e. searching only for code clones of a given code snippet. This technique can also detect Type-3 clones.[7] This technique was later used in the tool *SimEclipse*.[6] Since this tool can only detect clones of a code snippet which the developer selects, this tool is not well suited for finding accidental clones.
- Another tool, SHINOBI can detect code clones in real-time without the need of the developer to select a code snippet, however it can only detect type-1 and type-2 code clones.[4]
- The modern IDE IntelliJ has a built-in duplication detection and refactoring, it's able to detect type-1 and type-2 code clones at a method granularity and refactors by replacing one of the clones with a method call.

No tools which we are aware of have the capability of both reporting code clones without fragment-selection and reporting type-3 clones.

3 The way forward

This thesis will attempt to provide and evaluate a modern tool which provides clone management capabilities in a real-time IDE environment. The main goal

will be to create a tool which fits well into the development cycle and works in a real-time IDE environment. Areas of focus will therefore be:

- Real-time detection and management of code clones
- Code clone refactoring and detection of refactoring-oriented clones
- IDE tooling and IDE agnostic tooling like LSP[]

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.

[4] Shinji Kawaguchi, Takanobu Yamashina, Hidetake Uwano, Kyohei Fushida, Yasutaka Kamei, Masataka Nagura, and Hajimu Iida. Shinobi: A tool for automatic code clone detection in the ide. pages 313–314, 01 2009.

[5] M. Rieger, S. Ducasse, and M. Lanza. Insights into system-wide code duplication. In *11th Working Conference on Reverse Engineering*, pages 100–109, 2004.

[6] Md Sharif Uddin, Chanchal K. Roy, and Kevin A. Schneider. Towards convenient management of software clone codes in practice: An integrated approach. In *Proceedings of the 25th Annual International Conference on Computer Science and Software Engineering*, CASCON '15, page 211–220, USA, 2015. IBM Corp.

[7] Minhaz F. Zibran and Chanchal K. Roy. Ide-based real-time focused search for near-miss clones. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC '12, page 1235–1242, New York, NY, USA, 2012. Association for Computing Machinery.