PLS2 – Software Evolution

1. Introduction

This document describes implementation details for the PLS2 assignment for the Software Evolution course.

2. Goals

The project aims to discover type 1 clones (exact copy without modifications except for whitespace and comments [1]). Once the clone classes are identified a report is generated. The report will allow the developers to easily assess the clone class distribution within the project. Using the location functionality in Rascal custom locations were implemented for each clone, thus allowing a dynamic navigation and visualizations of each clone.

3. Approach

In the first step of the implementation a M3 model is computed for a Java project. Using this model all Java files are read line by line. In order to ensure that lines that contain whitespace and comment are not taken into consideration we used the comment parser implemented in PLS1. The parser is already tested and can be used to process a file line by line. Additionally we explicitly removed any spaces in the valid lines in order to reduce the size of the string, thus obtaining a slight speed increase doing string comparisons.

All the valid lines of code are stored in a map that contains the string and the location of the line. In order to preserve the order of reading and avoid rewriting an existing value inside the map we stored all the maps inside a list.

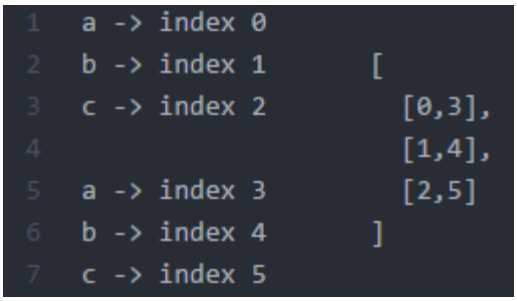

Figure 1. Index mapping for a clone pair.

In the next step the indexes position for each line that is preset at least twice are stored in separate lists. Once again in order to preserve the order the index lists are stored within a list.

In the next step we compute each clone by traversing each column from top to bottom in order to find subsequent elements. For example in Figure 1 we can easily see that we have identified a clone pair: 0, 1, 2 and 3, 4, 5.

A slightly more complicated case is presented in Figure 2 on the next page of the document.

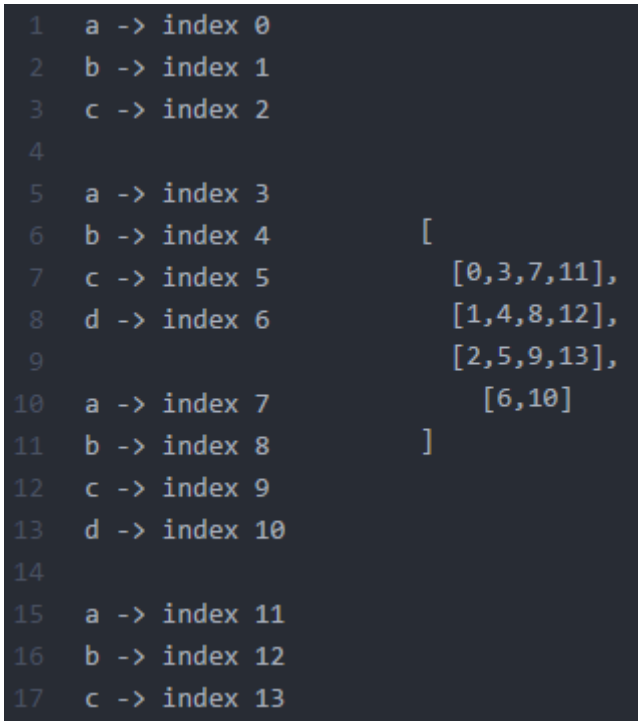

Figure 2. Index mapping for a clone pairs.

Figure 2 presents a slightly more complicated example and proves that the method removes clone classes strictly included in others. In this particular case the algorithm is able to detect that we have two clone pairs of three elements and two clone pairs of four elements.

4. Report and example results

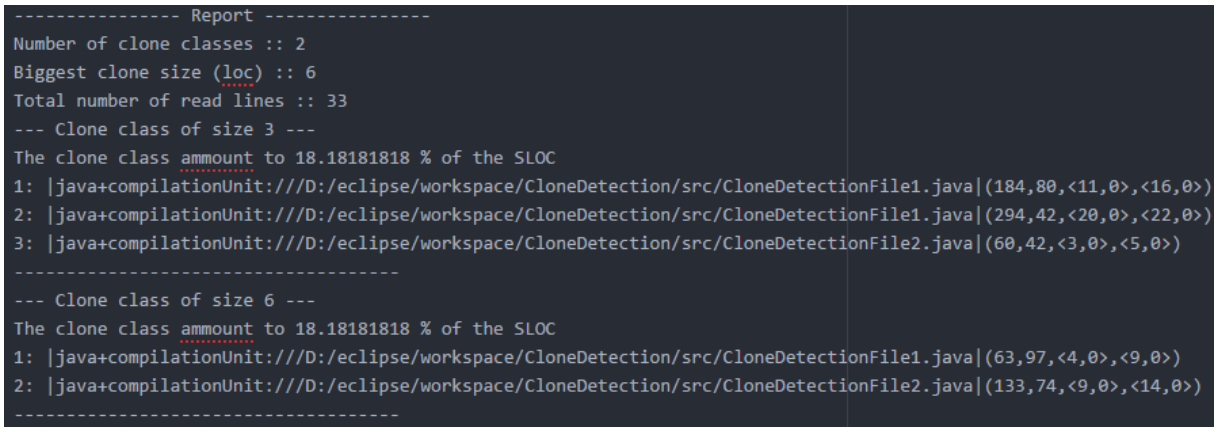

Figure 3. Example report

Figure 3 contains a report example that depicts the main elements of interest for a developer regarding code clones. We took advantage of the Rascal console inside the Eclipse IDE in order to promote easy code navigation. In order to access a clone the developer can easily left click on the location of the clone and will be immediately send to the highlighted portion of the Java file that contains the clone.

This enables a simple assessment and fix of duplicated code lines in Java projects. Figure 4 shows an example selection of a code clone

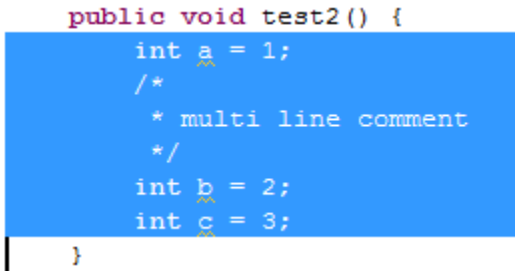

Figure 4. Selection of a code clone.

5. References

[1] - Rainer Koschke. Software Evolution, chapter 2. Identifying and Removing Software Clones, pages 15–36. Springer, 2008