## Design document Software Evolution Series 2

Guy Rombaut

Bas Meesters

## Requirements satisfied for the maintainer

The tool makes use of a lot of abstraction levels which gives the maintainer the ability to see both an overview and details in files themselves using the visualizations. The first view that is shown is an overview of the duplications in the whole project. The graph shows all the files on both axes and for every duplication found between two files (or within files) a blue dot increases in size. When you hover over the bullets they show the file names and the amount of duplications. This will give the maintainer an overview over how the duplications are distributed. Since if the bullets are in a more or less linear line the clones are mostly in the same file. If they are distributed over the whole graph this will mean that there are a lot of duplications between scattered over all the files.

Since large projects can have a lot of bullets overlapping each other it is possible to zoom in by dragging and holding the mouse to select an area. Also it is possible to make the zooming undone so you can navigate back and forth between details and overview.

When clicking a bullet it shows the code of the two files and highlights the duplications. The same code is highlighted in the same color and different colors are used between different duplications.

Next to the visualizations we provide maintainer with hard data and a report. The hard data contains the locations per clone class which are clickable using eclipse. The report shows some number such as the largest clone class, the total amount of duplicated code and the biggest clone;

## Types of clones which are detected

The clone detection tool finds type 1 clones matching sequences of statements to each other. The amount of clones found is a bit lower than the amount given in the previous series. This can be because we match on statements and not on lines of code. But it is also possible that we miss clones. Anyway we have not discovered any false positives. Another bug we did not solve yet is that the sub-clone subsuming does not seem to be entirely correct.

Type 2 clones are also discovered using the same method. Only the abstract syntax tree used is first transformed so variable names and types are all the same.

Both types can thus be found only in sequence and not at the same time. The running time for smallsql is about 1:45 minutes for type 1 and only a bit slower for type 2. So together they will be found in like four minutes.

## Explanation of the algorithm used

At first we wanted to use the algorithm of Koschke, Falke and Frenzel as described in their work called *Clone Detection Using Abstract Syntax Suffix Trees*. Since it uses the precision of tree comparison and the speed of text based algorithms. Unfortunately this seemed a bit opportunistic and we decided to just go with Baxter’s algorithm only with hashing the subtrees. This again gave problems with sequencing and finally decided to use our own view on duplication detection. The steps we do are given below.

1. Get the abstract syntax tree of the whole project.
2. For each method, constructor or initializer get all the statements recursively.
   1. This means that when a statement with block is encountered get the statements of the block.
   2. Thereafter the statement of the block is changed so it does not contain the block anymore (this so clones which are only partially in blocks can be detected).
   3. E.g. for (int x = 0; x < 5; x++) { block } will be replaced by for (int x = 0; x < 5; x++) { }, since sequences starting outside a block and ending inside a block will otherwise not be found.
3. For each of the lists made by the previous step make all sub lists possible with a minimum given length.
4. For each of the sub lists created by the previous step hash them to a map using the list of statements as a key.
   1. The values of the key-value pair will consist of a set of tuples of location, list of statements and size of that block in lines of code.
   2. Every time a key is mapped to an occupied location the value is updated with a new tuple.
   3. So for each found clone it will contain all the places where
5. The map from the previous step is filtered so only key-value pairs where the value has more than one item will remain.
6. For the remaining items it is checked if they are a sub-clone of another one and if so they are removed.
7. At the end use the all the clone pairs found for the visualization.