Software Evolution
Practical Lab Series 2- Clone Detection

Gerben van der Huizen - 10460748
Vincent Erich - 10384081

December 18, 2016

1 Introduction

The goal of this assignment is to build our own clone detection and management tools that are of help to software engineers like ourselves. We have implemented two tools: a clone detection tool that can detect Type 2 clones in a Java project, and a clone management tool that visualises the results of the clone detection tool. This document is divided into three sections. Section 2 describes the clone detection tool. Next, Section 3 describes the clone management tool. Finally, Section 4 concludes with an evaluation of the implemented tools.

2 Clone Detection Tool

The first tool that we have implemented is a clone detection tool. As described in the introduction, the clone detection tool can detect Type 2 clones in a Java project. The clone detection tool is written in pure RASCAL. We first describe the (core of the) clone detection algorithm that is used by the clone detection tool, and then we describe how the results of the clone detection tool are saved for the clone management tool.

2.1 Clone Detection Algorithm

The clone detection algorithm that we have implemented is the Basic Subtree Clone Detection (BSCD) Algorithm described in [2]. The BSCD Algorithm uses Abstract Syntax Trees (ASTs) to detect Type 2 clones in a software project. The reason that we have chosen this algorithm for our clone detection tool is that RASCAL offers great support for ASTs. The pseudocode of the BSCD Algorithm can be found in [2, Figure 1]. The next paragraph briefly describes how we have implemented the algorithm in RASCAL.

We first create all the ASTs for the source of the Java project using the RASCAL function `createAstsFromEclipseProject`. Next, these ASTs are visited and for every node we determine its sub-tree mass (the sub-tree mass is the number of nodes in a sub-tree)[1]. If the sub-tree mass of a node is greater than 10, we normalise the node and hash it to a bucket[2]. Nodes are hashed based on their normalised form: nodes that have the same normalised form (i.e., the same syntactic structure) are hashed to the same bucket. After having visited all the nodes in the ASTs, we have a collection of buckets, and in every bucket we have a number of nodes (i.e., sub-trees, including their location) that can be compared with one another. Next, we create all the possible clone pairs in a bucket (i.e., all the possible pairs of nodes). We then compute the similarity between the two nodes of all the possible clone-pairs in a bucket, and if the similarity between the two nodes is equal to 1, the corresponding clone pair is considered to be a true clone pair[3]. Finally, since we wish to recognise maximally large clone classes, clone sub-trees of clone pairs are eliminated as reportable clone classes (subsumption).

The RASCAL module `detectCloneType2` contains the implementation of the clone detection algorithm. Again, more information about the algorithm can be found in [2].

2.1.1 Results

Table 1 shows the results of applying the clone detection algorithm to the source code of the Java projects/systems `smallsql` and `hsqldb`.

Table 1: The results of applying the clone detection algorithm to the source code of the Java projects/systems `smallsql` and `hsqldb`.

Cloning statistic	**smallsql Value**	**hsqldb Value**
Percentage of duplicated lines	9%	12%
Number of unique clones	154	2046
Number of clone classes	54	833
Number of lines biggest clone	122	82
Number of clone pairs biggest clone class	66	300

[1] Every node in an AST is a sub-tree.

[2] The minimum sub-tree mass of 10 has been experimentally determined. We have experimented with a number of values for the threshold and eventually agreed upon a minimum sub-tree mass of 10.

[3] By using a similarity threshold of 1, we only take into account clone pairs of which the (normalised) nodes exactly match with each other.

2.2 Saving the Results for the Clone Management Tool

Besides writing cloning statistics to the RASCAL console (e.g., the percentage of duplicated lines, the number of clones, etc.), the clone detection tool also writes clone data to two CSV files:

- `cloneDataFiles.csv`: This CSV file contains clone data on the 'file level'. This CSV file has four columns: (1) file 1, (2) file 2, (3) the number of clone pairs between file 1 and file 2, and (4) the locations of the clones in the clone pairs between file 1 and file 2.
- `cloneDataFolders.csv`: This CSV file contains clone data on the 'folder level'. This CSV file has three columns: (1) folder 1, (2), folder 2, and (3) the number of clone pairs between folder 1 and folder 2.

The CSV files are used by the clone management tool to visualise the results of the clone detection tool. More information about the clone management tool can be found in the next section. The RASCAL module `writeToCSV` contains the implementation for writing the clone data to the CSV files.

3 Clone Management Tool

The second tool that we have implemented is the clone management tool. As described in the introduction, the clone management tool visualises the results of the clone detection tool. The goal of the clone management tool is to help maintainers of the source code find files and folders that contain (significantly) more clones than is acceptable. The clone management tool provides the user with a visualisation that shows a sortable heat map of the clone pairs between the files (and folders) in a Java project. This allows the user to quickly detect those areas of the project that contain too many clones. The identification and inspection of the clone pairs is managed using pop-up functionality, which allows the user to trace the clones back to the source code. The visualisation shows the data for the small Java project (`smallsql`) by default. The data for the large Java project (`hsqldb`) can be shown by clicking a button.

We first describe the visualisation in more detail, and then we describe the implementation details of the visualisation.

3.1 The Visualisation

The idea for the visualisation comes from [1, p. 32]; a paper that reviews several methods for visualising software clones. The paper discusses how simple visualisations such as scatter plots are widely used and have shown to be very effective at providing meaningful information to the user. Using the Javascript D3 library and the CSV data from the clone detection tool, we replicated the scatter plot visualisation. The scatter plot is a two-dimensional matrix where each cell represents the amount of detected clone pairs between two files. Both axes (dimensions) are labelled with the files that contain at least one clone.

Each cell is given a colour based on how many clone pairs it represents, so the entire matrix becomes a heat map for the clone pairs in a Java project. The visualisation shows the details of a cell by either hovering over it or clicking on it. When clicking on a cell, a separate window will open which shows the source code locations and links to the relative files of each clone pair.

The entire visualisation can be sorted based on the amount of clone pairs between files. When selecting the sort option, the cells that represent a higher amount of clone pairs will move to the top-left corner of the visualisation. The visualisation also allows data to be shown on the 'folder level' (the visualisation shows data on the 'file level' by default). If the user selects this option, a two-dimensional matrix will be shown with the amount of clone pairs between the folders in a Java project (in which case both axes [dimensions] will be labelled with the folders that contain at least one clone). These two sorting options were directly taken from the scatter plot described in [1]. However, a third sorting option was added to the visualisation. This option allows the user to change the maximum amount of clone pairs that are acceptable. For instance, if the user thinks 50 clone pairs between two files is the maximum, cells that represent a higher amount of clone pairs will receive the darkest colour in the heat map. The rest of the colours will be automatically assigned to different ranges (based on the maximum).

3.2 Implementation Details

As described in Section 2.2, the clone detection tool writes the clone data to two CSV files: a CSV file that contains the clone data on the 'file level' (`cloneDataFiles.csv`), and a CSV file that contains the clone data on the 'folder level' (`cloneDataFolders.csv`). The visualisation shows the clone data on the file level by default. The clone data on the folder level is shown by clicking the associated button.

We have used the Javascript D3 library to implement the clone management tool. The data from the CSV files is extracted using D3 functionality, and the data is stored as a matrix (array of arrays). If the visualisation has to be sorted, the matrix is simply transformed into a matrix with sorted columns and rows (the sort is based on the data of cells). Once the matrix is filled with data, the D3 components for the visualisation are created. A large window for the scatter plot visualisation is created (with options for scrolling), as well as a smaller window for the legend and the sorting buttons. During the drawing of the cells and the axes labels, the functionality (e.g., the pop-up and on-click functions) and colour of each cell is also added. The windows which show more information/details about the clone pairs between two files are generated once a cell is clicked.

4 Conclusion

The tools that we have implemented satisfy three design requirements and allow software maintainers to gain insight in the clone pairs throughout their project. Firstly, since the visualisation uses colour-coded cells, the user experiences less cognitive effort when finding problem areas (i.e., areas with a lot of clone pairs); simply looking for areas in the heat map with dark colours is enough. Secondly, the tool enables the user to find clone pairs on multiple levels/dimensions. These levels are represented by the different sorting options that the tool offers. The user can either sort by the number of clone pairs, by files or folders, or sort by the colour intensity (based on the maximum amount of clone pairs selected). Finally, the visualisation allows the user to find the actual clones from each clone pair in the code by selecting a cell and clicking on the links associated with each clone pair. Overall, the tools that we have implemented enable the maintainer to view the clone pairs in a project from different perspectives, each providing unique insight.

References

[1] Asaduzzaman, M. (2012). *Visualization and analysis of software clones.* PhD thesis, University of Saskatchewan.

[2] Baxter, I. D., Yahin, A., Moura, L., Sant'Anna, M., and Bier, L. (1998). Clone detection using abstract syntax trees. In *Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272)*, pages 368–377.