Software Evolution

Series #2

Jordan Maduro
10807411
jordanmaduro@gmail.com
Software Engineering

Cas van der Weegen
10017577
cas.vanderweegen@student.uva.nl
Software Engineering

December 17, 2017

UNIVERSITY OF AMSTERDAM

Contents

1 Summary

The Series #2 assignment consists of implementing a Clone Detection and Management framework. The overall goal of the assignment is twofold:

1. Detect Clones in a (large) body of software.
2. Visualize the detected clones in such a way that helps developers with the management of software clones.

For two software projects that were provided as part of the assignment our clone detection framework was run and results have been obtained including (but not limited to):

- The number of type #1 clones
- The number of type #2 clones
- The percentage of duplicated lines
- The number of clone classes
- The biggest clone (in lines)
- The biggest clone class.

The two projects provided are:

1. SmallSQL - a 100% pure Java database for Java desktop applications [1].
2. Hyper SQL Database - HSQLDB - 100% Java Database. [2].

The projects differ in greatly in size which makes their combination useful for testing our analysis implementation (SmallSQL is relatively small, as the name suggests, HSQLDB is quite a bit larger).

2 Clone Detection

2.1 Introduction

The first part of our submission consists of a clone detector. This clone detector uses some of the methods described in the AST based clone detection paper by Baxter[5]. Our clone detector can quickly detect and classify type 1 and type 2 clones. Type 3 clones are also detected but at the cost of 10x speed.

2.2 Framework

The clone detector consists of two parts. Firstly, the Detector module which exposes the detect method Secondly, the app module which contains the settings. The preprocessing, clone extraction, and reporting are separate functions. That allows for easy testing.

2.3 Detection

Clone detection is done using the AST. A trivial task with Rascal. We then split the AST into subtrees. During this process, we annotate each subtree with its mass, location and hash value. To reduce the number of subtrees we set a minimum threshold. The clone extraction method uses these trees to create clone classes.

2.4 Hash

Baxter does not explain nor mention which hash function he uses in his paper. The paper does mention that it should be an imperfect hash to be able to detect near-miss clones. We took an approach inspired by another article[6]. We choose to give each node an integer value, based on the first appearance. This value will then be used to represent the node within a sequence.

2.4.1 Type #1

We can quickly detect type 1 clones by comparing the AST with other members of the group. If they match, we set their relation to type 1.

2.4.2 Type #2

We assume that any member of a group has a type 2 relation with all other members of that group. The hash function captures enough syntactic information to classify type 2 clones

2.4.3 Type #3

Type 3 detection uses the similarity function presented in the Baxter paper. That is extremely slow because of the comparison of trees with all other trees. We chose 95% similarity to reduce the number of false positives. Type 3 classes are not combined with type 1 and type 2 clones.

3 Visualization

3.1 Introduction

The second part of our submissions contains a visualization framework, written in Python(3) using the TKinter package. To pass the required metrics from the clone detector to the visualization package we chose to use a modified version of the RCF (*Rich Clone Format*)[3]. The project metadata and detected clones are formatted according to this format and passed from the clone detector to the visualizer package in the JSON (*JavaScript Object Notation*) format.

3.2 Framework

The codebase of the visualizer consists of three main components, each of these components run in a separate thread and communicate using multiprocessing queues. Together they are responsible for the calculations and visualization of the metrics obtained by the clone detector.

3.2.1 Monitoring

The monitoring class is a watchdog implementation that monitors the filesystem for new files in a specific folder. When the clone detector adds files to this specific directory, they are picked up by the watchdog and passed to the handler class where they are further parsed.

3.2.2 Handler

The handler class handles the incoming files that have been detected by the monitoring class. When the handler class receives a file, its contents are parsed and converted to CloneObjects (a datastructure defined in our framework). Additionally, the handler class is also responsible for some global calculations such as the biggest clone class or total number of duplicated lines (i.e., information that always needs to be updated when new data is available). Finally, the handler class is also responsible for resetting the project when the clone detector passes a new *METADATA.json* file (which denoted the start of a new clone-detector run).

3.2.3 Interface

The interface class is a modular class responsible for drawing the user interface based on the data collected by the monitoring class and the data calculated by the handler class. We decided to make the system dynamic by allowing drop-in visualizations by passing the complete project structure to visualization classes and having the classes perform the calculations that are required for the specific type of visualization themselves.

3.3 Types

3.3.1 Treemap

A treemap is a map where the total amount and the number of duplicates can be related to the total amount of lines. Each square is a class in which one or more clones have been detected (as seen in figure 3.3.1). The surface area of a square is the percentage of source-lines that are duplicated as part of the total number of source lines. Additionally, when a square (a clone class) is clicked,

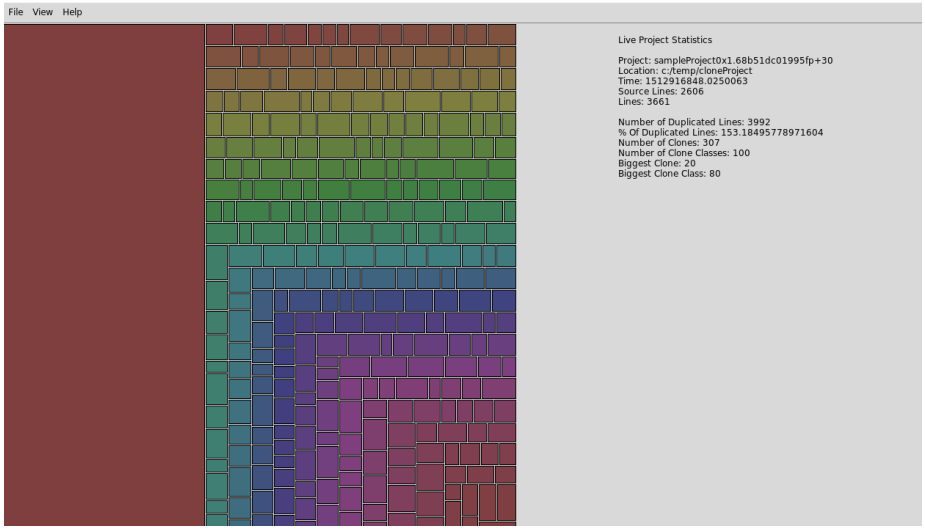

Figure 1: Example Treemap Visualization

details about the clone class are shown. The details include (but is not limited to) the following information:

- The class in which the clone was found
- The clone type (1, 2, or 3)
- The clones mass, length, and SLOC
- Its location (the file and the row, column, and offset of the detected lines)
- The duplicated fragment

An example of this screen can be found in Figure 2.

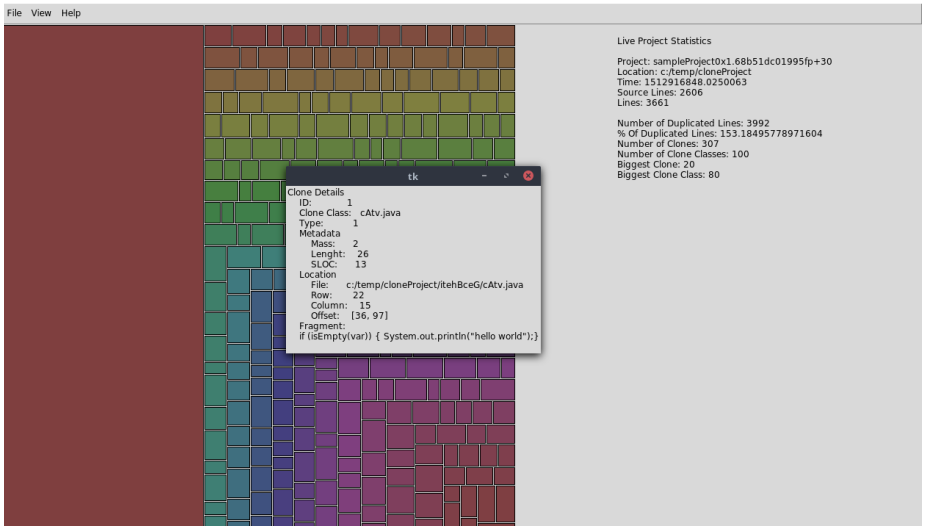

Figure 2: Example Treemap Visualization (detail)

3.3.2 Treemap (clones only)

The clones only treemap is nearly identical to the regular treemap, with the main difference being that the project source lines are not plotted in the overview, such that an overview is shown that display the relative sizes of the clones, such that clone sizes can be compared amongst each other.

3.3.3 Treemap (colors per clone)

The clones only treemap is nearly identical to the regular treemap, with the main difference being that each clone type is assigned a color, such that the distribution of clone types can be more easily seen. Like the PieChart, the following clone types have the following colors:

- Type #1 - Blue
- Type #2 - Yellow
- Type #3 - Red
- Project SLOC - Green

An example of this visualization can be seen in Figure 3

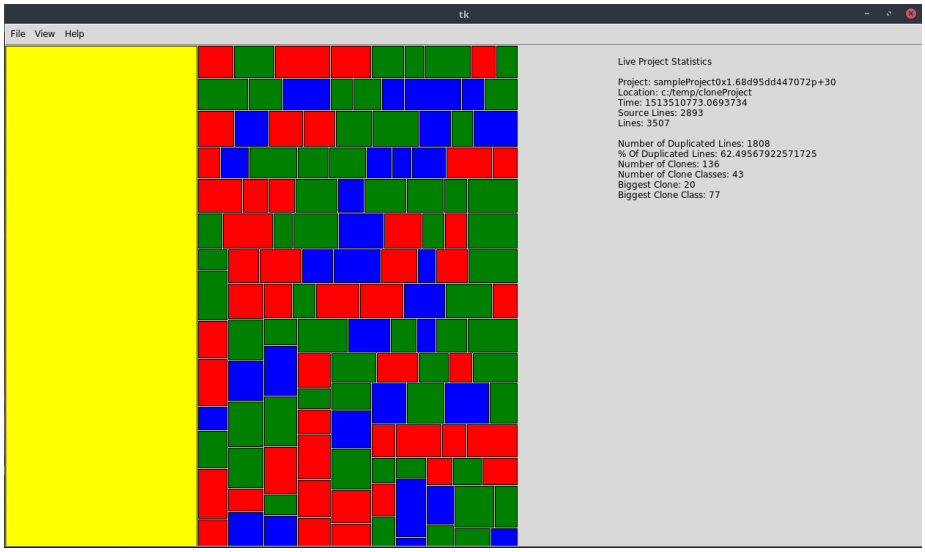

Figure 3: Treemap Colors per Clone including the relative project SLOC

3.3.4 Treemap (clones only, colors per clone)

Combination *Treemap (clones only)* and *Treemap (colors per clone)*. An example of this visualization is shown in Figure 4.

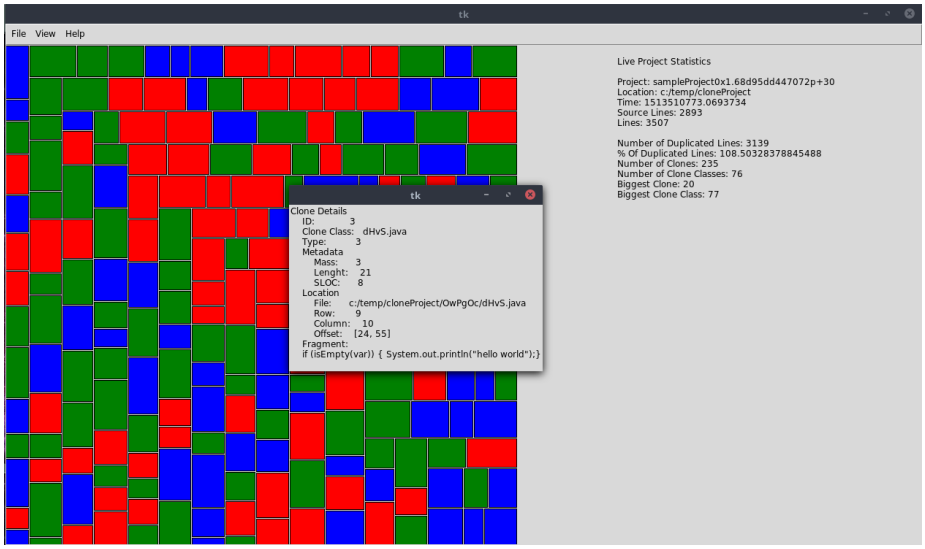

Figure 4: Example Treemap Colors per Clone

3.3.5 PieChart

The piechart visualization is a relatively simple representation of the percentage of duplicated lines which are plotted against the percentage of source code lines. The pie-chart consists of 4 area's, 3 of which represent a percentage of duplicated lines based on a clone type (again 1,2, or 3), the fourth area is the percentage of non-duplicated lines. An example of the pie-chart representation is shown in Figure 5.

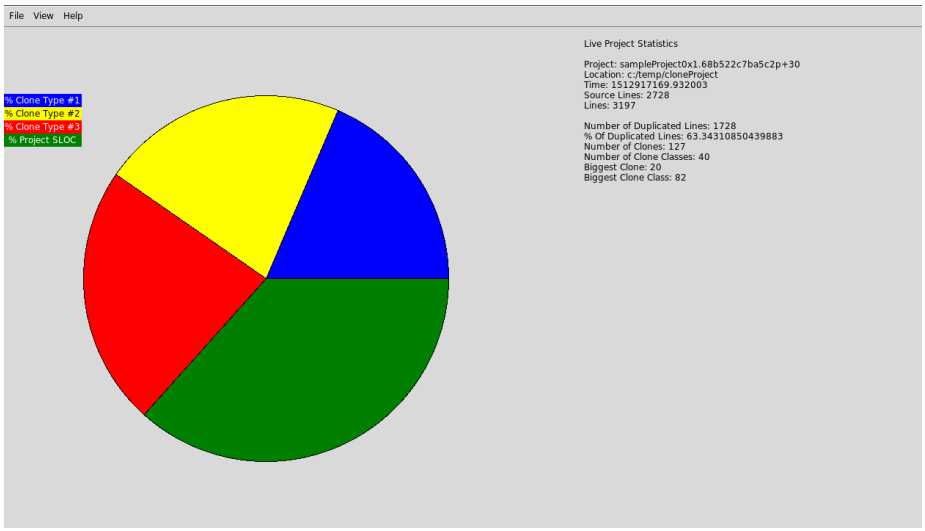

Figure 5: Example PieChart Visualization

3.3.6 Live Log

This visualization is the only visualization that is always visible in the interface. It displays several metrics that are defined in the assignment description, and are updated every time a new clone-class is received from the clone detector. The Live Log displays the following metrics:

- The Project
- Project Location
- Time
- Project Source Lines (SLOC)
- Project Lines (LOC)
- Number of Duplicated Lines
- Percentage of Duplicated Lines
- Number of Clones
- Number of Clone Classes
- Biggest Clone (in number of lines)
- Biggest Clone Class (in number of lines)

4 Installation Manual

4.1 Prerequisites

4.1.1 Detector

The detector is written in the RASCAL[4] metaprogramming language, instructions on how to install Eclipse and the RASCAL plugin can be found at: http://www.rascal-mpl.org/start/.

4.1.2 Visualizer

The visualizer has several packages that need to be installed before the package can be run. Package installation commands are provided for Fedora and Ubuntu. Other distributions and/or operating systems may work, but commands are omitted.

- Python3.6 (older versions **above** 3.0 *may* work)

  ```
  Fedora:
  > sudo dnf install python3 python3-pip

  Ubuntu:
  > sudo apt-get install python3 python3-pip
  ```

- TKinter

  ```
  Fedora:
  > sudo dnf install python3-tkinter

  Ubuntu:
  > sudo apt-get install python3-tk
  ```

- watchdog, squarify

  ```
  The 'requirements.txt' file can be found in
  $PROJECT_ROOT/Series2Visualizer/

  Fedora:
  > sudo pip3 install -r requirements.txt

  Ubuntu:
  > sudo pip3 install -r requirements.txt
  ```

4.2 Installation

The installation of both the detector and the visualizer can be done by cloning the project repository.

```
> git clone git@github.com:vdweegen/UvA-Software_Evolution.git
```

The cloned project will contain three folders:

- Series1 - First assignment
- Series2 - Second Assignment (Clone Detector)
- Series2Visualizer - Second Assignment (Visualizer)

You can decide to run the programs from the command line, or from your favorite editor. As authors we give the following suggestions editor-wise that *should* speed up development and/or execution of the project:

- Series1 → Eclipse with RASCAL support
- Series2 → Eclipse with RASCAL support
- Series2Visualizer → PyCharm

4.3 Execution

It is trivial that the visualizer is started *before* the detector is started, such that the visualizer does not miss any events generated by the detector.

4.3.1 Detector

The detector can be run by loading the Series2 project in eclipse. Open the app module in the src directory. Editing the "inputFile" variable to the target project, loading the app module in the rascal console. and running "main()".

4.3.2 Visualizer

the visualizer can be started from the terminal and has the following arguments:

```
--debug                 Run the Visualizer in DEBUG mode

Running the visualizer in debug mode starts an additional
background process which mocks the detector and can be
used for development/testing of the visualizer without
running the detector.

Defaults to FALSE when not specified.


--path <PATH>           Specify the path to be watched by the
watchdog

The path that is to be specified is RELATIVE to the current
working directory.

Defaults to './watchdir' when not specified.
```

EXAMPLES:

```
> python3 visualizer.py --path ../Series2/src/sessions

> python3 visualizer.py

> python3 visualizer.py --debug
```

5 Results

The following results were obtained using both the *small* project [1] and the *large* project [2] using a machine of the following specifications:

```
Intel(R) Core(TM) i5-3320M CPU @ 2.60GHz, 16GB RAM, 512GB SSD Storage
```

5.1 SmallSQL

```
Lines: 38423
Source Lines: 24107
Number of Duplicated Lines: 1922
Percentage of Duplicated Lines: 7.972788%
Number of Clones: 169
Number of Clone Classes: 61
Biggest Clone: 122 Lines
Biggest Clone Class: 244 Lines

Time taken to complete run: 1 minute, 4 seconds, 37 milliseconds
```

5.2 HSQLDB

```
Lines: 299727
Source Lines: 173369
Number of Duplicated Lines: 16522
Percentage of Duplicated Lines: 9.52996%
Number of Clones: 1312
Number of Clone Classes: 560
Biggest Clone: 82 Lines
Biggest Clone Class: 286 Lines

Time taken to complete run: 8 minutes, 52 seconds, 410 milliseconds
```

Bibliography

[1] Volker Berlin (2005). SmallSQL (version 0.21) [Computer software]. Berlin, Germany. http://www.smallsql.de.

[2] The HSQL Development Group (2001). Hyper SQL Database (version 2.3.1) [Computer software]. London, United Kingdom. http://www.hsqldb.org.

[3] University of Bremen, Software Engineering Group. *Rich Clone Format*. http://www.softwareclones.org/rcf.php

[4] Centrum Wiskunde & Informatica (CWI). *RASCAL*. http://www.rascal-mpl.org

[5] Baxter, Ira D., et al. "Clone detection using abstract syntax trees." Software Maintenance, 1998. Proceedings., International Conference on. IEEE, 1998.

[6] Chilowicz, Michel, Etienne Duris, and Gilles Roussel. "Syntax tree fingerprinting for source code similarity detection." Program Comprehension, 2009. ICPC'09. IEEE 17th International Conference on. IEEE, 2009.