Software Evolution
Assignment 2

Spencer, Paul 11721677

Gemistos, Vasilis 12318264

December 2018

1 Introduction

This report is about a tool for detection and visualization of clones in java code. It describes

- the algorithm
- design decisions
- clone visualization
- testing properties

The following clone type was detected:

- Type1: identical code segments, except for whitespaces and comments

The following clone type failed to be detected because of performance issues with the algorithm caused by design decisions:

- Type2: Syntactical copy, ignoring naming names types, and function identifiers.

2 Design Rationale

2.1 Type 1 detection

In summary:

- Clean lines, keep line location
- Put all combinations of 6 or more lines with locations in a map, Ignore single entries.
- Generate subsets and remove all but the largest.

- Group duplicates together (based on their cleaned string)

The advantage of this design is that the locations travel with the duplicates, thus making later display easier.

The disadvantage of this design is the factorial growth of both step 3 and step 4. This does not deal well with nearly duplicated files.

2.2 Type 2 detection

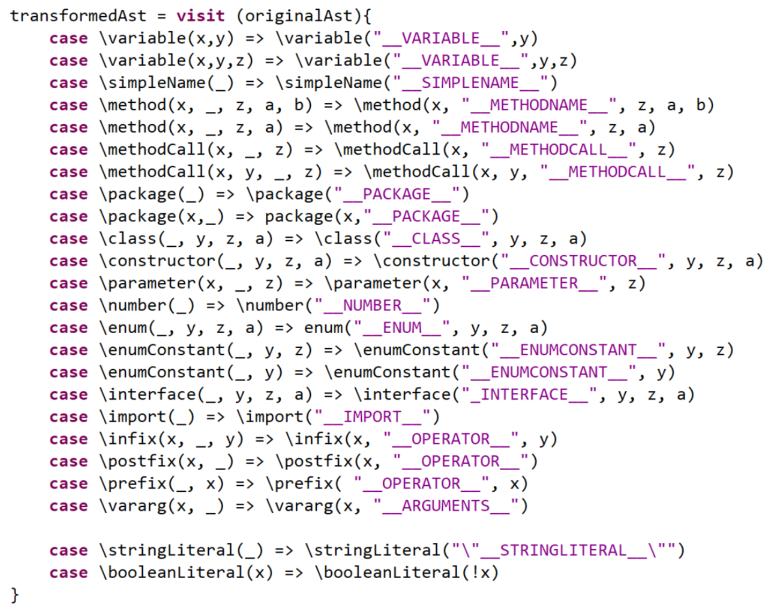
In Summary

- Normalize through AST
- Rebuild java files
- Type 1 detection

The advantage of this design was the reuse of our type 1 detection.

The disadvantage was that we never got it to work as the rebuilding of the Java files ended up being very slow. it worked for small test java files, but it did not scale with the growth of the AST.

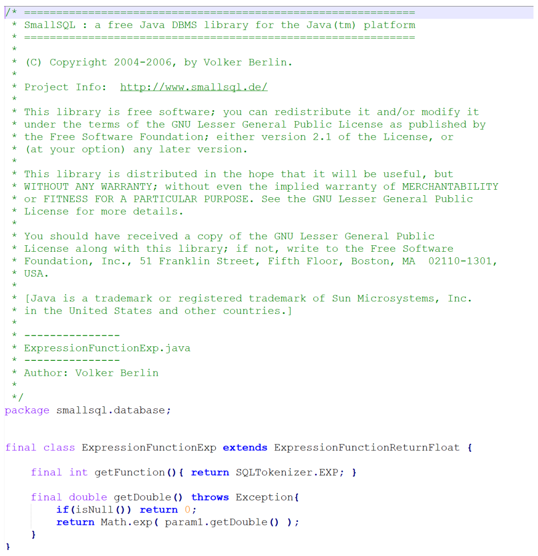
Our normalization looked like this:



```
transformedAst = visit (originalAst){
    case \variable(x,y) => \variable("__VARIABLE__",y)
    case \variable(x,y,z) => \variable("__VARIABLE__",y,z)
    case \simpleName(_) => \simpleName("__SIMPLENAME__")
    case \method(x, _, z, a, b) => \method(x, "__METHODNAME__", z, a, b)
    case \method(x, _, z, a) => \method(x, "__METHODNAME__", z, a)
    case \methodCall(x, _, z) => \methodCall(x, "__METHODCALL__", z)
    case \methodCall(x, y, _, z) => \methodCall(x, y, "__METHODCALL__", z)
    case \package(_) => \package("__PACKAGE__")
    case \package(x,_) => package(x,"__PACKAGE__")
    case \class(_, y, z, a) => \class("__CLASS__", y, z, a)
    case \constructor(_, y, z, a) => \constructor("__CONSTRUCTOR__", y, z, a)
    case \parameter(x, _, z) => \parameter(x, "__PARAMETER__", z)
    case \number(_) => \number("__NUMBER__")
    case \enum(_, y, z, a) => enum("__ENUM__", y, z, a)
    case \enumConstant(_, y, z) => \enumConstant("__ENUMCONSTANT__", y, z)
    case \enumConstant(_, y) => \enumConstant("__ENUMCONSTANT__", y)
    case \interface(_, y, z, a) => \interface("_INTERFACE__", y, z, a)
    case \import(_) => \import("__IMPORT__")
    case \infix(x, _, y) => \infix(x, "__OPERATOR__", y)
    case \postfix(x, _) => \postfix(x, "__OPERATOR__")
    case \prefix(_, x) => \prefix( "__OPERATOR__", x)
    case \vararg(x, _) => \vararg(x, "__ARGUMENTS__")

    case \stringLiteral(_) => \stringLiteral("\"__STRINGLITERAL__\"")
    case \booleanLiteral(x) => \booleanLiteral(!x)
}
```

Figure 1: Normalization code

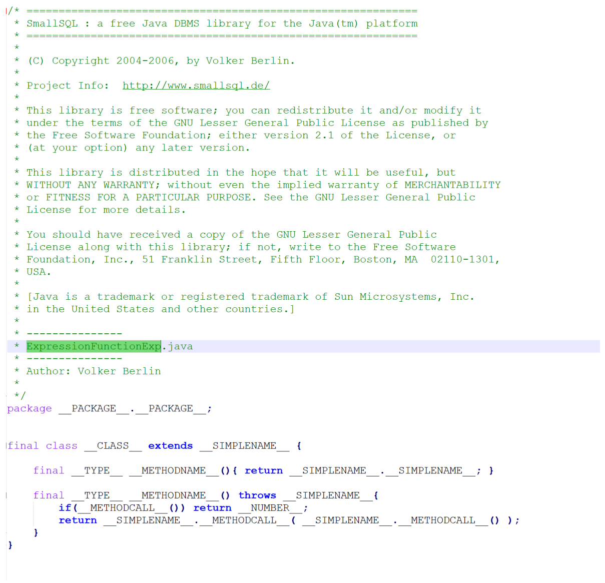
The reason we flipped the boolean rather than normalizing it was because our Java rewriter looked for differences. We never managed to rewrite modifiers. The output we were getting was turning this:



```java
/* =============================================================
 * SmallSQL : a free Java DBMS library for the Java(tm) platform
 * =============================================================
 *
 * (C) Copyright 2004-2006, by Volker Berlin.
 *
 * Project Info:  http://www.smallsql.de/
 *
 * This library is free software; you can redistribute it and/or modify it
 * under the terms of the GNU Lesser General Public License as published by
 * the Free Software Foundation; either version 2.1 of the License, or
 * (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
 * or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
 * License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301,
 * USA.
 *
 * [Java is a trademark or registered trademark of Sun Microsystems, Inc.
 * in the United States and other countries.]
 *
 * ---------------
 * ExpressionFunctionExp.java
 * ---------------
 * Author: Volker Berlin
 *
 */
package smallsql.database;


final class ExpressionFunctionExp extends ExpressionFunctionReturnFloat {

    final int getFunction(){ return SQLTokenizer.EXP; }

    final double getDouble() throws Exception{
        if(isNull()) return 0;
        return Math.exp( param1.getDouble() );
    }
}
```

Figure 2: original java code

Into this:



```java
/* =============================================================
 * SmallSQL : a free Java DBMS library for the Java(tm) platform
 * =============================================================
 *
 * (C) Copyright 2004-2006, by Volker Berlin.
 *
 * Project Info:  http://www.smallsql.de/
 *
 * This library is free software; you can redistribute it and/or modify it
 * under the terms of the GNU Lesser General Public License as published by
 * the Free Software Foundation; either version 2.1 of the License, or
 * (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
 * or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
 * License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301,
 * USA.
 *
 * [Java is a trademark or registered trademark of Sun Microsystems, Inc.
 * in the United States and other countries.]
 *
 * ---------------
 * ExpressionFunctionExp.java
 * ---------------
 * Author: Volker Berlin
 *
 */
package __PACKAGE__.__PACKAGE__;


final class __CLASS__ extends __SIMPLENAME__ {

    final __TYPE__ __METHODNAME__(){ return __SIMPLENAME__.__SIMPLENAME__; }

    final __TYPE__ __METHODNAME__() throws __SIMPLENAME__{
        if(__METHODCALL__()) return __NUMBER__;
        return __SIMPLENAME__.__METHODCALL__( __SIMPLENAME__.__METHODCALL__() );
    }
}
```

Figure 3: Normalized java code

The rewriter was incredibly slow (for smallsql it took a couple of hours). Had it had been successful the disadvantages of our type 1 detector would have been ampfied due to a reduction in unique lines, leading to a greater risk of large quantities of matched lines, meaning the implemented optimization would not have been as useful.

2.3 Output

We ran our analysis over iterations of the hsqldb project from the year 2007 to 2017. (2016 is missing through a mishap and we added 2.3.1 as an additional "2019"). From our collections, we made 4 output files required for our visualizations: For the bubble charts, we created a file containing all the required dimensions and labels, i.e. version year; package name; class name; class size; the number of duplicates per class; total duplicate lines per class; largest duplicate in class For the Sankey charts, we needed two files. First, one giving an id to each object being reported on. Our Sankey charts are unidirectional, so we had to have 2 id's for each class, one for source and one for the destination. The second file linked each of these sources and destinations by their duplicate locations. Finally, for the performance of the Sankey charts, we separated the full text of the duplicates into a separate file.

3 Maintainer

We believe that our visualization tool satisfies some of the maintainer requirements described in "Cognitive design elements to support the construction of a mental model during software exploration" paper [2]. We developed a visualization tool, that **improves comprehension**.
The time slider allows a developer to see the evolution of code over time, to see areas that are worryingly growing. Using the bubble chart it is easy to see the largest class and the duplication size of each class. Our tool makes **navigation** through the project easy, as the tool is interactive, a user can drill down into a class in the bubble chart and a Sankey diagram with the representation of the origin location of the code and the classes to which it is duplicated can be seen. The Sankey chart is also interacive. Clicking on a link between classes will display for the user the two sides of the duplicated code - including original comments and indentations.

4 Clone Visualization

For the visualization we used the plot.ly, an online platform that develops online visualization tools, using Python scientific graphing libraries. We got inspired by the paper "A practical model for measuring maintainability" [1]

At first we create a bubble chart diagram

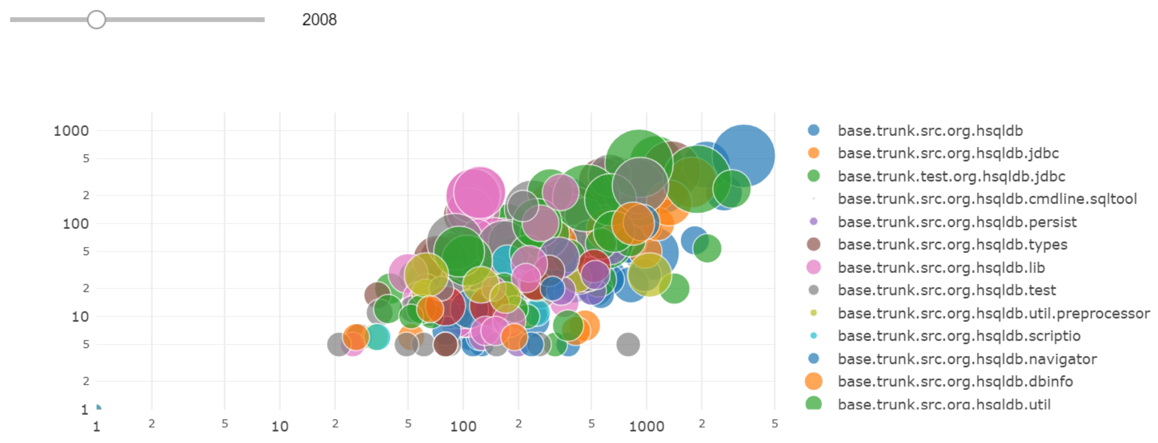


Figure 4: Bubble Chart for Duplication

In the top of the graph we have a slider that indicates each version of the project categorized by year. Sliding to the right the bubble chart shows the next year's version of the project

The X-axis in the bubble chart (Figure 4), represent the class size and the Y-axis represent the duplication size. The bubble size maps the largest duplicate, and the color is for each package as shown on the right side of Figure 4.

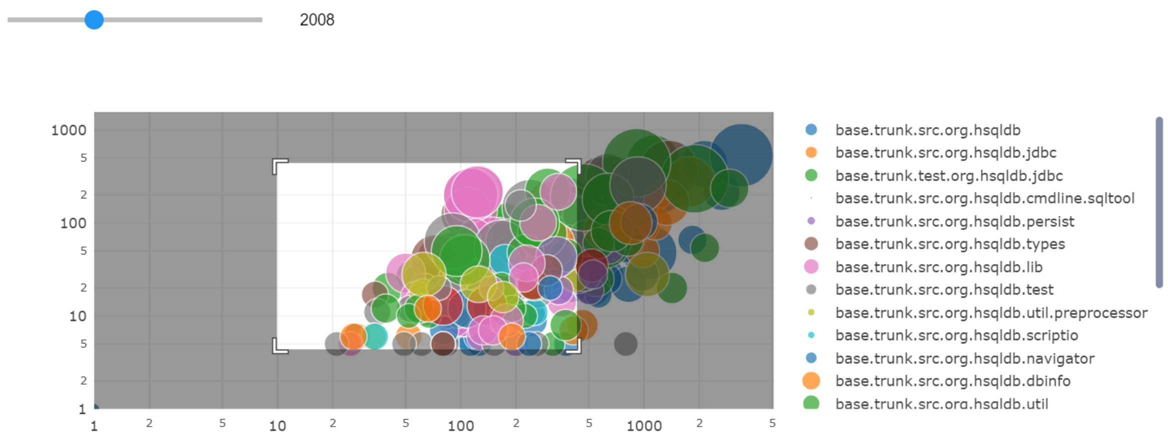

2008
1000
5
2
100
5
2
10
5
2
1
1
2
5
10
2
5
100
2
5
1000
2
5
base.trunk.src.org.hsqldb
base.trunk.src.org.hsqldb.jdbc
base.trunk.test.org.hsqldb.jdbc
base.trunk.src.org.hsqldb.cmdline.sqltool
base.trunk.src.org.hsqldb.persist
base.trunk.src.org.hsqldb.types
base.trunk.src.org.hsqldb.lib
base.trunk.src.org.hsqldb.test
base.trunk.src.org.hsqldb.util.preprocessor
base.trunk.src.org.hsqldb.scriptio
base.trunk.src.org.hsqldb.navigator
base.trunk.src.org.hsqldb.dbinfo
base.trunk.src.org.hsqldb.util

Figure 5: Zoom selection on the bubble chart

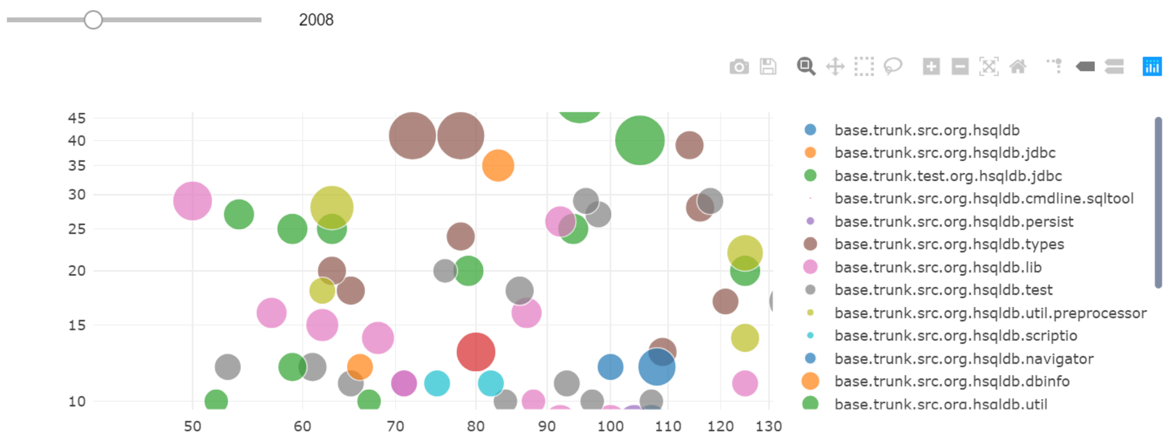


Figure 6: Zoomed bubble chart

The bubble chart is interactive and flexible, we can zoom-in in a selected area and the bubble chart show this specific area for more usability. Every bubble is clickable, and after a click, each bubble shows a Sankey diagram as shown in Figure 7.

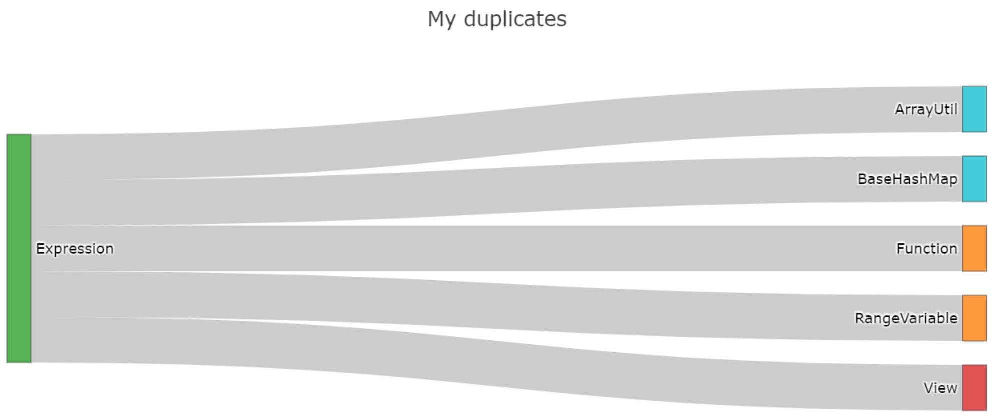
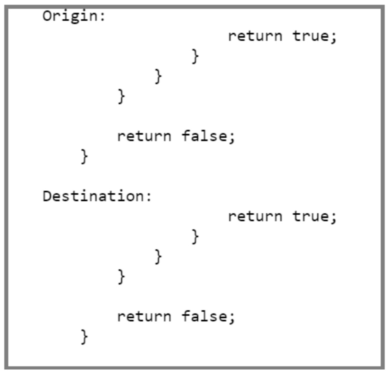


Figure 7: Sankey Diagram. Origin and destination location of duplicates

The Sankey diagram shows the location of the original code and the duplicated code of each instance. Each line is also clickable, and with click to every "pipe" Sankey present the code of origin and destination (Figure 8).



```
Origin:
                    return true;
                }
            }
        }

        return false;
    }

Destination:
                    return true;
                }
            }
        }

        return false;
    }
```

Figure 8: Sankey Diagram. Origin and destination location of duplicates

5 Issues

We had two great mistakes in our approach to this project. The first was a stubborn insistence on reusing type one detection for type two clones. It would have been better if we have worked to the strengths of Rascal, instead of trying to bend it to our will, we would have performed this task better. Secondly, we decided to use a language and framework that both of us were unfamiliar with to develop the visualization, i.e. Python and plotly. We were unaware of limitations of the framework and platform that led us to not being able to display our (excellent) visualizations in the way we wanted. In order to view them now one has to have juypiter notebooks installed.

5.1 Testing

We have a test suite for comment cleaning and type-one duplicates. However, we did not have a satisfactory solution to testing our visualization

References

[1] I. Heitlager, T. Kuipers, and J. Visser. A practical model for measuring maintainability. In *6th International Conference on the Quality of Information and Communications Technology (QUATIC 2007)*, pages 30–39, Sep. 2007.

[2] M.-A.D Storey, F.D Fracchia, and H.A Müller. Cognitive design elements to support the construction of a mental model during software exploration. *Journal of Systems and Software*, 44(3):171 – 185, 1999.