University of Amsterdam

Series 2

Report

December 11, 2024

Students:
Piotr Witek
UvAnetID 15840077

Weronika Szybińska
UvAnetID 15836134

Tutor:
Thomas van Binsbergen

1 Introduction

We decided to choose the **Frontend Route** implementing the detection of Type I clones and two visualizations:

1.1 Clone class investigator

To compare clones, check their locations and visualize how they look. It was inspired by Fig. 2.8. "Clones visualizer view in Eclipse" from Rainer Koschke 2008

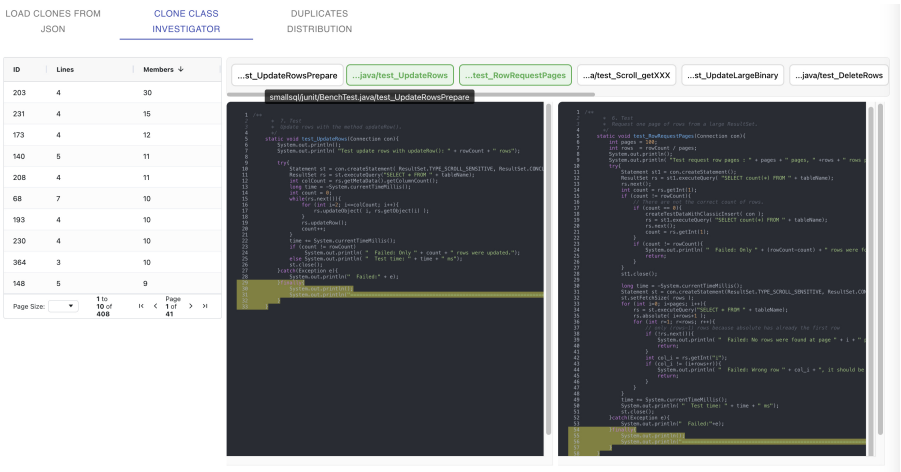

Figure 1: Clone class investigator

1.2 Duplicates distribution graph

For showing the distribution of cloned lines across the packages and files. It was inspired by Fig. 2.6. found in Rainer Koschke 2008

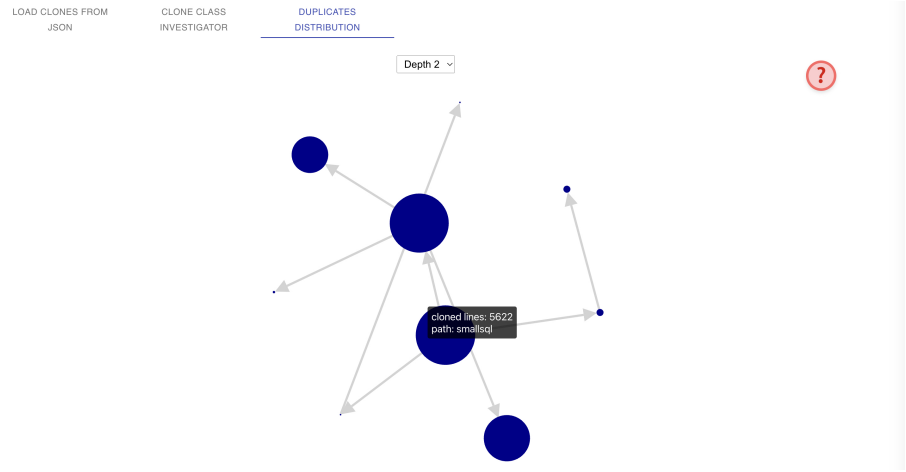

Figure 2: Duplicates distribution graph

While the Clone class investigator from the **Figure 1** is self explanatory, for second visualization we added help button.

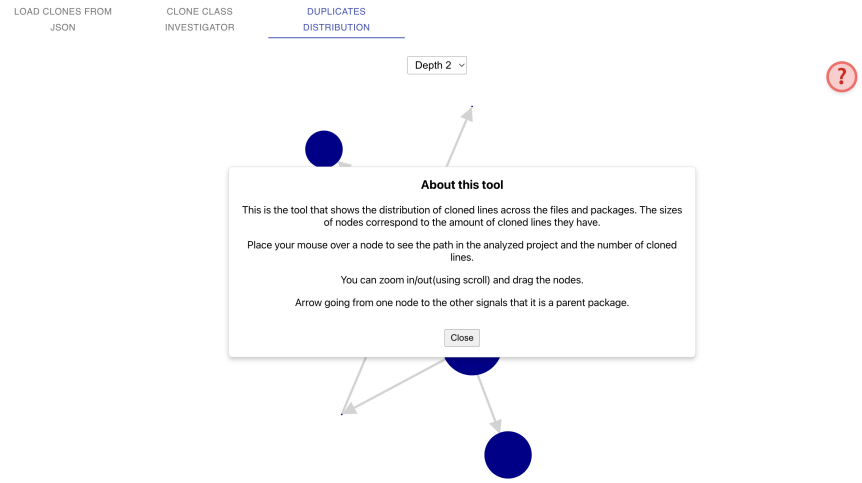

Figure 3: Duplicates distribution tab after clicking **?** button

2 What are Clones of Type I?

Type I: exact copy, ignoring whitespace and comments - **lecture slides "Clone detection; visualization; and management"** *by D. Frölich.*

3 Explanation of Clone Detection Algorithm

The algorithm can be split into the following four stages:

3.1 Data Transformation Phase

Firstly, we use the AST model to extract methods from the project and normalize them, saving the results in a hashmap. We normalize methods line by line, trimming whitespace and removing comments. This is very much Roy, Cordy, and R. Koschke 2009 *Transformation* step.

We keep track of the lines removed so that our final visualization (*Clone Class Investigator*) can correctly show the clones.

3.2 First Iteration

We start looking for clones of size 3 by using the moving window strategy. For each method, we start from line 0 in the normalized content and take the first three lines, saving them in the hashmap using the hash of those lines as the key. The value is set to 1 (count of occurrences).

Then, we move to line 1 and repeat the process for lines 1, 2, 3, and so on. If the key already exists in the map, we increment the value instead. After processing all methods, we remove records that have a value of 1, as these are not clones (because they occur only once).

3.3 Following Iterations

We extend clones of size 3 into clones of size 4. For each clone family of size 3, we attempt to include the next line.

3.3.1 Non-Extendable Clones Saved as Final

All clone locations that cannot be extended (either because they include the last line of the method or have only a single occurrence) are saved as final clones.

3.3.2 New Clones Saved to the Current Iteration

All other clone locations that occur at least twice are saved as new clones of size +1, as the result of the current iteration.

3.3.3 Finish Current Iteration and Start a New One

After processing Sections **3.3.1** and **3.3.2**, we check if any clones were found during *this iteration*.

- If none were found, we finish the search.
- If clones were found, we save *this iteration* as the *last iteration*, which will be used in the next one.

3.4 Removing Subsumption

We sort the clones in descending order by the number of lines. Then, we check all smaller clones to determine if they are contained within the larger clone. Those with content contained in a bigger clone are marked as deleted and ignored in future comparisons. This process is repeated for each subsequent clone.

4 Motivation

The final version of the algorithm underwent multiple optimizations. Initially, we treated all iterations like the *First Iteration* described in Section **3.2**. However, this approach was extremely slow for the *hsqldb* project (approximately 16 minutes), with subsumption detection consuming 90% of the time due to the sheer number of redundant clone families found.

We also failed to narrow the search domain, looking for clones of size $x + 1$ in methods that did not have clones of size x.

Our initial approach was valid for the *First Iteration*, but subsequent clones could be detected much faster. Although we wanted to avoid subsumption detection entirely, we soon realized that the algorithm still would allow producing subsumption, just not in the same method.

5 Reflection

Overall, we were able to avoid subsumption within the same file, which occurred frequently and significantly slowed us down. Using custom data types helped maintain data flow and increased readability.

We are satisfied with the final solution. After considerable effort, we were able to complete clone detection for:

- *smallsql* - in under 3 seconds
- *hsqldb* - in 2.5 minutes

Tests were conducted on a Mac M1 Pro.

6 Satisfied Maintainer Requirements

Storey, Fracchia, and Müller 1999 propose possible requirements that a maintainer may have from a software tool. In our opinion the tool we created satisfies the following:

- **Enhance top-down comprehension**: Our visualizations represent the statistics extracted from the analyzed project by the algorithm we implemented. They also help validating the solution(Clone class investigator), making life much easier for the maintainer.
- **Facilitate navigation**: The navigation bar at the top of the page is simple(to avoid the word boring) solution, that everybody is familiar with.
- **Reduce disorientation**: We decided to use react, as it is the most popular web-development tool famous for being the easiest to use on the market. To further drive this point - we did not have any react experience prior to this course, and managed just fine.

  All the application state is handled by the **GlobalStateContext.Provider** and extension of the visualizations is possible by creating new visualization component and a respective **Tab** in **index.js**.

We opted for a clear, clutter-free UI so that both maintainers and users treat it as a tool first and a website second.

7 Implementation of Visualizations and Critical Reflection

We are most satisfied with the *Clone Class Investigator*, as it provides insights and searchability over clone classes. It was our minimal requirement for a workplace-ready clone detection solution. Thanks to *AgGridReact*, the table scales well. Previously, we faced performance issues with *DataGrid* from @mui, which looked better but failed on bigger projects like *hsqldb*.

The *Duplicates Distribution* visualization, was more complicated to implement than expected. While D3 is flexible, we faced issues with label display and positioning. To reduce visual clutter, node labels were moved to tooltips, instant insights of labels for more concise and better for bigger projects tooltips.

Depth was introduced to mitigate the overwhelming number of nodes overlapping and rendering the visualization useless.

This visualization could be used to distribute (as evenly as possible)the refactoring of the codebase among the developers.

7.1 What Could Have Been Done Better

- Our algorithm ignores cloned lines in constructors and fields of the class. We deemed them not relevant enough to complicate the solution and instead focused solely on the methods. As any design choice, it could be both good and bad decision.
- The *Duplicates Distribution* could include more features, such as parent node highlighting upon clicking a node.
- D3, though powerful, proved less practical than anticipated. A different graph visualization solution might have been more efficient.

7.2 Overall

We believe the visualizations are complementary and of high quality. They could be further developed for broader applications.

References

Koschke, Rainer (2008). *Identifying and Removing Software Clones*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 15–36. ISBN: 978-3-540-76440-3. DOI: 10.1007/978-3-540-76440-3_2. URL: http://dx.doi.org/10.1007/978-3-540-76440-3_2.

Roy, C. K., J. R. Cordy, and R. Koschke (May 2009). "Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach". In: *Science of Computer Programming* 74.7, pp. 472–475. ISSN: 0167-6423. DOI: 10.1016/j.scico.2009.02.007.

Storey, M.-A. D., F. D. Fracchia, and H. A. Müller (1999). "Cognitive design elements to support the construction of a mental model during software exploration". In: *Journal of Systems and Software* 44.3, pp. 171–185. ISSN: 0164-1212. DOI: 10.1016/S0164-1212(98)10055-9. URL: http://www.sciencedirect.com/science/article/pii/S0164121298100559.