Final Project

LiveRef

236651 - Advanced Topics in Software Engineering L+T

Name: Hussein Abu Jabal

ID: 318585577

Email: Hussein.ab@campus.technion.ac.il

Name: Mahmoud Nassar

ID: 318489200

Email: mahmoud-nas@campus.technion.ac.il

The relevant code can be found in the GitHub repository:

https://github.com/AbuJabal-Hussein/LiveRef

General Description:

LiveRef is an Eclipse plug-in that provides live refactoring of Java code, by suggesting candidates for “extract method refactoring” in real time while programming.

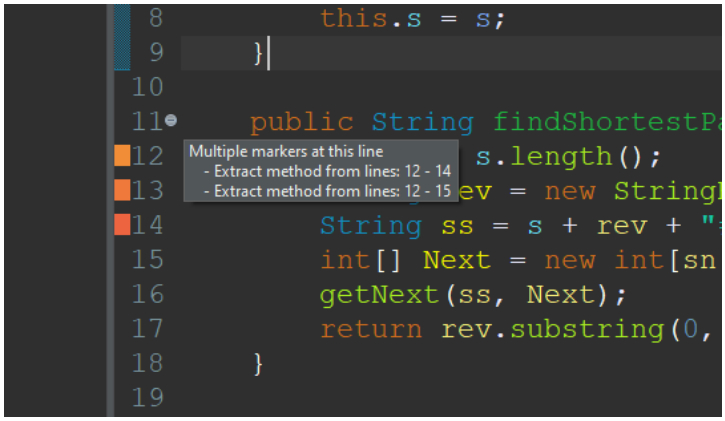
The plug-in is based on the paper *“A Live Environment to Improve the Refactoring Experience”* (the paper can be found in the GitHub repository), which relies on certain metrics.

LiveRef suggests candidates for method extraction by displaying markers in the vertical ruler of eclipse, in the left side of the start line of code of the respective refactoring candidate.

A marker shows a color from a certain range of colors, from bright green to dark red, that indicates the severity of the candidate.
To show the candidates relevant to a marker, you should move the mouse curser over the marker, so it will show the candidates sorted by the most to the least crucial refactoring candidate.
To apply the “extract method” refactoring, you should select the lines in the source code, which are indicated by the marker, and use the built-in refactor function in the Eclipse IDE (by right clicking the selection and choosing the “refactor” option).

The color gradient of the markers is:

The markers are displayed as in the following screenshot:



Every time the code changes (i.e. a file is saved and compiled by “ctr+s”, or a refactoring is done), the algorithm is executed once again, and it finds and displays new candidates.

The candidates are sorted in accordance with the following metrics, as described in the paper, in the following importance order:

1- CC (cyclomatic complexity)
   A quantitative measure of the number of linearly independent paths through a program's source code (the number of branches in the method, including If statements, For and While loops and others).
2- LOC (the number of lines of code)
3- LCOM (lack of cohesion)
   A lack of cohesion can refer to a lack of logical flow or coherence in the steps or procedures being followed, i.e., the statements aren’t related to each other in respect to the class fields that they access.

We strive to minimize each of these metrics in order to get better method extraction candidates.

The main algorithm that we used, which finds the refactoring candidates (on file change), goes as the following:

1- Candidates <- [ ]
2- For each method in the file:
    a. For each block of code (i.e., a loop or If/Else Statement body)
        i. For each moving window of sizes in range [3, 0.75 * number of statements in the block]
            1. Calculate the metrics of the window (CC, LOC, LCOM)
            2. Nodes <- nodes of the window
            3. Range <- the borders of the window
            4. Create new candidate <- Candidate(range, nodes, method, cc, lcom, loc)
            5. Candidates.add(candidate)
3- Sort the Candidates by the metrics as explained above

Code Structure:

The plugin source code is found in the folder “LiveRef”, while the tests are found in the “Tests” folder.

Both the source code and the tests are written in Java.

The setup and installing requirements are shown in the readme.md file GitHub repository.

LiveRef:

The main algorithm described above is found in the class LiveRef in “liveRef.Components.LiveRef.java”.

The folder “liveRef.Handlers” contains files of classes that are responsible for reading, parsing and analyzing the source code, and listening for changes, in addition to creating and placing the markers on the editor.

Tests:

This folder contains 10 test examples that cover most aspects of the plugin.