CS 7140 - Advanced Software Engineering

Fall 2020

Project Implementation Documentation

Team Members:

Lancius (Lance) Matthieu

Abhishek Pandya

Griffin Mosley

1. Non-Functional Requirements

1.1 Source Size Reduction

The main requirement is to reduce the source code by 5%. Using Intellij IDEA statistic plugin, the current source lines of code is 7848, the goal is to reduce this target number to 7500. The SLOC for each individual document can be seen in the Design portion of the report. Using Intellij IDEA statistic plugin, the files that have the most SLOC can be easily observed. The major files in decreasing order are as follows: TextGrid.java, Diagram.java, and DiagramShape.java. These are also the most significant files in the source code, providing the conversion from a text file (TextGrid) to a diagram (Diagram and DiagramShape). The SLOC of the files are shown below under the Code column.

File	Blank	Comment	Code
BitmapRenderer.java	88	54	354
CompositeDiagramShape.java	50	35	227
CustomShapeDefinition.java	5	19	55
Diagram.java	148	172	678
DiagramComponent.java	14	23	84
DiagramShape.java	157	151	670
DiagramText.java	32	59	101
FontMeasurer.java	27	30	144
ImageHandler.java	26	28	85
OffScreenSVGRenderer.java	25	23	90
ShapeEdge.java	43	62	152
ShapePoint.java	24	41	75
SVGBuilder.java	136	10	270
SVGRenderer.java	7	3	8

File	Blank	Comment	Code
BitmapRenderer.java	88	54	354
CompositeDiagramShape.java	50	35	227
CustomShapeDefinition.java	5	19	55
Diagram.java	148	172	678
DiagramComponent.java	14	23	84
DiagramShape.java	157	151	670
DiagramText.java	32	59	101
FontMeasurer.java	27	30	144
ImageHandler.java	26	28	85
OffScreenSVGRenderer.java	25	23	90
ShapeEdge.java	43	62	152
ShapePoint.java	24	41	75
SVGBuilder.java	136	10	270
SVGRenderer.java	7	3	8

In order to reduce SLOC, there are three ways this can be done and have been utilized. First, unused methods can be removed, to reduce confusion. Second, repeated pieces of code can be transferred into functions to provide maintainability. Finally, unpolished functions can be changed to be more proficient. All three methods have been utilized to reduce the SLOC.

2. Functional Requirements

2.1 Plugin

For the additional plugin, we have included more default color options. In DITAA, the program uses hex code values in order to change the coloring of shapes and backgrounds. We have added three new default hex values being: orange, aqua, and purple. For this, the hex codes “F60,” “0FF,” and “808” were used respectively. The shorthand hex code was used for this, where “F60” would be “FF6600” in the full code. In order to add the new color codes, the codes and the corresponding human coding was added to the list of human color codes. This is shown below.

```
humanColorCodes.put("GRE", "9D9");
humanColorCodes.put("BLU", "55B");
humanColorCodes.put("PNK", "FAA");
humanColorCodes.put("RED", "E32");
humanColorCodes.put("YEL", "FF3");
humanColorCodes.put("BLK", "000");
humanColorCodes.put("ORG", "F60");
humanColorCodes.put("AQA", "0FF");
humanColorCodes.put("PUR", "808");
```

The functionality of humanColorCodes, is just a list datatype holding the set color codes. Shown in the code, the color codes are applied to the list using the put function, adding both the human color code (“GRE,” “BLU,” etc.) and the hex code value corresponding to the value. The human color code gives a more understandable color representation than the hex code replacement. Using the replaceHumanColorCodes function, the human color code is swapped to the hex code correspondence. The character indicates that a color being set is also removed from the image. Once the hex code is applied, the shape is colored with the corresponding human color code, hex code pair.

```
public void replaceHumanColorCodes(){
        int height = getHeight();
        for(int y = 0; y < height; y++){
                String row = rows.get(y).toString();
                Iterator it = humanColorCodes.keySet().iterator();
                while(it.hasNext()){
                        String humanCode = (String) it.next();
                        String hexCode = (String) humanColorCodes.get(humanCode);
                        if(hexCode != null){
                                humanCode = "c" + humanCode;
                                hexCode = "c" + hexCode;
                                row = row.replaceAll(humanCode, hexCode);
                                rows.set(y, new StringBuilder(row));
                                row = rows.get(y).toString();
                        }
                }
        }
}
```

2.2 GUI

The implementation, as desired, allows the user to quickly select a file to be converted from ASCII to a diagram. For the implementation, Java Swing was used. The implementation was completed in three major steps: Allowing the user to select a file, run the DITAA program on the selected file, then finally apply the DITAA results to the output frame.

For the selection process, there are a couple key actions to allow this process to take place. First and foremost, a button was created to allow the user to open up the file selector. Using a listener for the button, once it’s clicked a file chooser appears allowing the user to quickly go through the files on their

computer. Using this file chooser, an action is tested to see if a file is selected. Once the file is selected, file information is obtained and is used in the next step.

The next step is applying for DITAA. In order to apply DITAA to the file, a command line execution needed to be developed. This was created through a string allowing the DITAA execution in the following order: “java -jar (ditaa.jar) (selected file) (output file).” This order is the default functionality on how to run DITAA. Using java’s jar execution and providing the jar to run and command line options. DITAA, takes in an input (ASCII file) and output file (Diagram file). To accomplish the command line execution, a process object was created using Runtime, allowing java to run the command line. The command line string as stated earlier was executed and waited on until completed. The information from the process is also displayed to the command line, showing the completion information as if you ran DITAA without the GUI.

The final step is displaying the output image. Once DITAA has been run, the output file obtained is used for the next step. For this, the image is buffered to get the contents then applied to an image icon is then applied to the frame. Once applied to the screen, the image does not directly display. In order to get the image to display, the frame needs to be refreshed to obtain the correct contents.