

CS 7140 - Advanced Software Engineering

Fall 2020

DITAA Project Documentation

Team Members:

Lancius (Lance) Matthieu

Abhishek Pandya

Griffin Mosley



Introduction	5
Requirement	6
2.1 Requirement Purpose	6
2.2 Problem Statement	6
2.3 Project Scope	7
2.4 System Purpose	7
2.5 Functional Objectives	8
2.6 Non-Functional Objectives	9
2.7 Context Model	10
2.8 The Use Case Model	11
Specification	13
3.1 Syntax of Input:	13
3.2 Interpretation of Input for Rendering:	14
3.3 DITAA GUI Specification	15
3.4 Compliance Test Specification	18
3.5 Code Reduction Specification	18
3.6 Assertions	19
Design	21
4.1 Architecture Design	21
4.2 Code Outline	21
4.3 Detailed Design	23
4.3.1 GUI	23
4.4 Preset Colors	23
4.5 Code Reduction	24
Implementation	27
5.1 Source Size Reduction	27
5.2 Plugin	27
5.3 GUI	29
Testing	30
6.1 Smoke Testing	30
6.2 White Box Testing	33
6.3 Black Box Testing	35
6.4 Stress Test	41
6.5 Acceptance Test	43
6.6 Additional Requirements	45
Conclusion	46

Reflection	47
8.1 Lance Matthieu	47
8.2 Abhishek Pandya	47
8.3 Griffin Mosley	47
References	48
Team Journals	50
10.1 Lance Matthieu	50
10.2 Abhishek Pandya	55
10.3 Griffin Mosley	60

1. Introduction

Ditaa is a small Java-written command-line utility that can transform diagrams drawn using ascii art ('drawings' containing characters that match lines like | / -) into proper graphics for bitmaps. This is best shown by the illustration below, which also demonstrates the advantages of using ditaa compared to other methods.

The scope, goals, and intent of DITAA are defined in this document. The aim of this document is to illustrate how this software is made with a full SDLC (Software Development Life Cycle) process which includes requirement, specification, design, implementation and testing.

2. Requirement

2.1 Requirement Purpose

This Requirements document is for a new conversion software that converts ascii art drawings into proper bitmap graphics. This conversion software will be titled Diagrams Through Ascii Art (DITAA). This document describes the scope, objectives, and goal of DITAA. Functional and non-functional objectives of DITAA will be covered and functional objectives will be modeled with use cases and a context diagram. The intention of this requirements document is to guide the design and implementation of DITAA in an object-oriented language.

2.2 Problem Statement

Our team is tasked with developing software that will enable the conversion of new and potentially legacy ascii art drawings into proper bitmap graphics. Ascii art are drawings that are constructed with characters that resemble lines, such as |, /, -, and).



Motivation:

There are legacy frequently asked questions (FAQs) that utilize ascii drawings and/or diagrams that could be aesthetically improved with the use of rendered graphics. Currently, these ascii art drawings would need to be recreated from scratch with a separate graphic design software (e.g. GIMP, Photoshop, Inkscape, etc.). DITAA aims to reduce the burden to update and maintain ascii art drawings into usable graphics, by directly converting the ascii art into bitmap graphics.

Vision:

In addition to the standard version of the software, the following enhancement(s) will be made to the baseline DITAA software:

- Reduce baseline codebase by a minimum of 5%
- Update baseline codebase to utilize Java 8 functionality
- Provide wider range of pre-set background colors
- Provide ability to render wider range of shapes
- Provide ability to specify texture of lines drawn (e.g. solid, dashed, dotted)

DITAA will be offered as free open source software under the GPL license with no intention of monetizing the software.

2.3 Project Scope

The scope of this project is a command line interface executable that takes ascii art drawings in a specified format (see section 5.3) and converts the drawings into proper bitmap graphics. Note: some graphics will not be rendered, such as “Display” and “Off-Page Reference”.



Examples of graphics not rendered: (Left) Display (Right) Off-Page Reference

A design for a graphical user interface (GUI) look and feel will also be developed.

Integration of DITAA into other text-based formats (e.g. HTML, DocBook, LaTeX) is not part of this project.

2.4 System Purpose

2.4.1 Users

Users who will directly benefit from the newly developed software are identified below:

End User:

Once software is complete, end users will have an offline, standalone command line interface tool able to convert legacy and/or new ascii art drawings into usable bitmap graphics.

2.4.2 Location

The software (including source code) will be available for anyone with access to GitHub. The software is intended to be utilized as a standalone command line interface tool on the end user’s local machine.

2.4.3 Responsibilities

The primary responsibilities of the software are:

- Provide end user ability to convert ascii art diagrams to bitmap graphics
- Provide end user with documentation of expected ascii art input expectations
- Provide end users with an intuitive, easy to use command line interface
- Provide end users a common -help option that specifies usage and optional flags
- Allow the end user to customize output graphics with the following:

- anti-aliasing
- graphic background color
- edge separation (default on)
- rounded corners
- shadows
- transparency
- scale
- fixed width or slope for parallelograms
- dashed lines
- point markers
- text insertion – to include bullets
- Allow end users to overwrite file if it already exists
- Allow end users to specify encoding of input ascii art drawing
- Allow end users to specify the number of spaces “tabs” are interpreted as (default 8)

Other desired features of the new software:

- Allow end users access to a wider range of pre-set background colors
- Allow end users to specify line texture (solid, dashed, dot, etc.)
- Allow end users to render more shapes (see 1.4)

2.4.4 Need

This new software is needed as current, legacy ascii art diagrams would need to be re-created from scratch via a separate graphics design software and presents maintainability concerns when updates are required. DITAA would allow a user to take advantage of newer versions by simply re-rendering their previous ascii art diagram.

2.5 Functional Objectives

2.5.1 High Priority

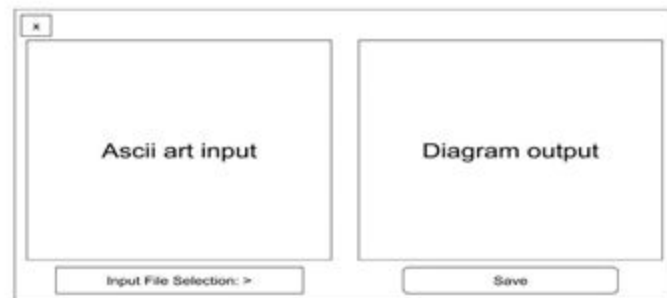
Identifier	Requirement
REQ1	The program shall provide an easy way to convert ascii art(characters that resemble lines like / - + *) to diagrams.
REQ2	The program shall allow the user to draw any design using ascii art and in return the program should give diagrams as an output from that ascii art.
REQ3	The program shall convert each ascii value to a particular design.
REQ4	The program shall provide information on which ascii value converts into which diagram.

2.5.2 Medium Priority

Identifier	Requirement
REQ5	The program shall provide the user a choice for filling the color in particular shape.
REQ6	The program shall provide the user a list of tags from which the user can render a particular diagram.

2.5.3 Low Priority

Identifier	Requirement
REQ7	The program shall provide GUI to the user, so that, user can easily convert ascii art to diagram.
REQ8	The program shall provide enhancements to improve the quality of the program. Examples are command line flags, more line conversions (double lined), and shapes.



This is an example of what the GUI might look and feel like.

2.6 Non-Functional Objectives

2.6.1 Accessibility

Users with the knowledge of java, ascii and GitHub can access this software. GitHub is the platform where the software will be available after development. Software is using java to convert ascii art into a diagram.

2.6.2 Portability

We want our software to be usable by the greatest number so it will be available on GitHub. Users can download the project without paying any amount. Users need to have java installed in their system to run the project. The project will run on any operating system.

2.6.3 Generality

This software will take input-file(ascii art) as a command line argument and will give output-file(diagram). This software will be very general. If GUI is implemented, then the interface will be very easy to use.

2.6.4 Extensibility

Due to less time in the semester we might not be able to implement the GUI (Graphical User Interface) but other functionalities will be available by the end of the semester.

2.6.5 Quality

The quality of the code reduction will be measured through the following: reliability, efficiency, and maintainability. The reliability will be how likely the software will fail, in the current state it should be very rare for the software to fail. The efficiency is the ability to keep high performance. Maintainability is the ability for the code to be adapted for future developments. The code should keep, if not improve, the state of each: reliability, efficiency, and maintainability.

2.6.5 Expectations

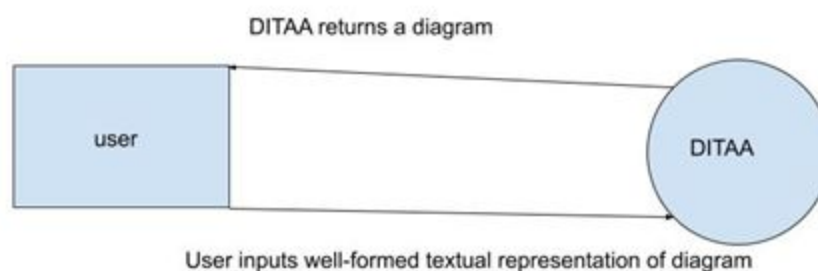
After the test, it is expected that the final software product will be free of errors, while also decreasing the overall project code size by a minimum of 5% and increasing maintainability of the codebase.

2.7 Context Model

2.7.1 Goal Statement

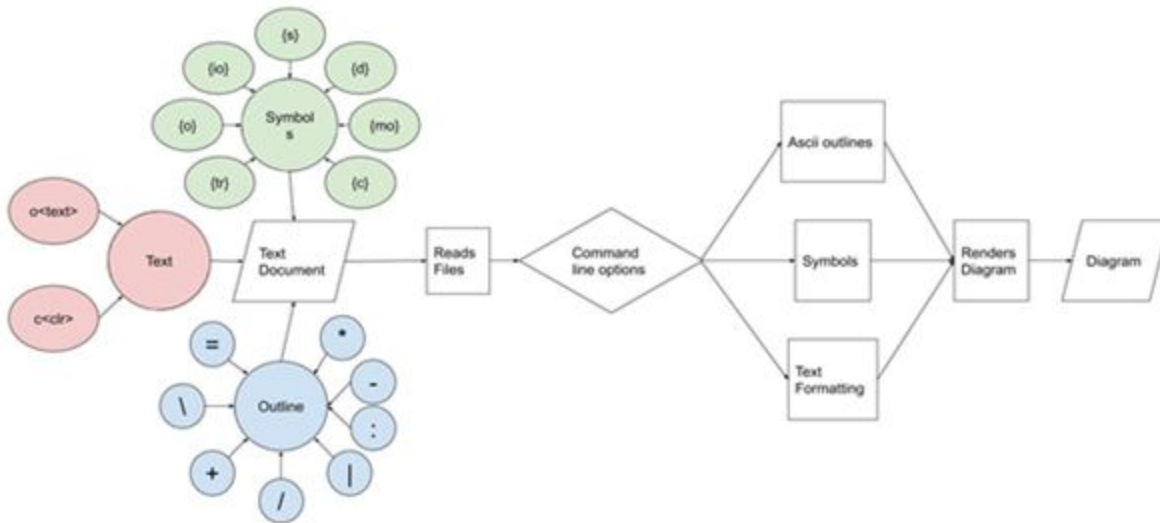
The goal of the system is to allow users to convert ascii textual representations into diagrams rendered in bitmap graphics

2.7.2 Context Diagram



2.8 The Use Case Model

2.8.1 System Use Case Diagram



2.8.2 System Use Case Description

Default

Use Case:	Default
Summary:	Default functionality of DITAA. Takes in a file of ascii characters and converts it to a diagram formed from the characters.
Basic Flow:	<ol style="list-style-type: none">1. User specifies which document as input.2. User specifies an output file.3. Reads in the input file.4. Checks and applies options from the command line.5. Creates outlines based on specified characters.6. Converts outlines surrounding symbols to corresponding formats.7. Applies text formatting options.8. Renders the diagram to the output file.
Alternative Flow:	<p>Step 2: If the user does not specify an output file, an output file with the given input name is created rendered to a .png file. Input.txt would be rendered to Input.png.</p> <p>Step 4: No command options need to be specified.</p>
Preconditions	<p>Input file must not be empty.</p> <p>Input file must be in a readable encoded file, examples: .txt or .html.</p>

Postconditions	Users can access the ascii art converted to a picture..
----------------	---

2.8.3 Text and Command Line Input Details

Outline Characters:	<ul style="list-style-type: none"> · : Creates left and right side of the outline. · - : Creates top and bottom of the outline. · / and \ : Creates rounded corners of the outline. · + : Creates squared corners of the outline. · = : Creates dotted lines on the top and bottom part of the outline. · :: Creates dotted lines on the left and right side of the outline. · * : Creates a point on the outline.
Symbols	<ul style="list-style-type: none"> · {d} : Creates shape representing a document for diagrams. · {s} : Creates shape representing a storage for diagrams. · {io} : Creates shape representing I/O for diagrams. · {mo} : Creates a shape representing manual operation for diagrams. · {c} : Creates a shape representing a choice for diagrams. · {o} : Creates an ellipse. · {tr} : Creates a trapezoid.
Text	<ul style="list-style-type: none"> · o<text> : Creates a bullet point followed by the text. · c<XXX> : Creates a background color for the outlined shape. XXX is the hex number corresponding to a color.
Options	<ul style="list-style-type: none"> · -A : Turns off anti-aliasing. · -b : Takes in color and sets background color of the image. · -d : Renders the debug grid. · -E : Prevents the default separation of shapes. · -e : Takes in encoding and specifies the encoding of the input file. · -h : Specifies the input is an HTML and produces HTML file with tags. · -o : Overwrites a file if the file is already created. · -r : Makes all corners round corners. · -S : Turns off drop-shadow effect. · -s : Takes in scale and uses it to determine the size of the rendered image. · -T : Makes the background transparent. · -t : Takes in length of tabs and uses it to change the spacing for tab characters · -v : Makes ditaa more verbose. · -w : Makes sides of shapes with fixed slopes instead of widths.

3. Specification







3.1 Syntax of Input:

The input must be a text/html file containing ASCII art derived from the grammar given below:






```
corner ::= [ "+", "/", "*" ]
tbOutline ::= [ "-", "=", "*" ]
sideOutline ::= [ "|", ":", "*" ]
arrow ::= "<" | ">", tbOutline | sideOutline | corner, "<" | ">"
top ::= <corner> <tbOutline> <corner>
bottom ::= <corner> <tbOutline> <corner>
text ::= [letters, numbers, symbols]
bullet ::= < o > <text>
tag ::= < { > [ d, s, io, o, mo, c, tr ] < } >
colorCode ::= [ RED, GRE, BLK, BLU, PINK, YEL ]
color ::= < c XXX > st XXX is a hex value | < c colorCode >
internal ::= [ color, tag, text, bullet ]
side ::= < sideOutline, white space, sideOutline > | < sideOutline, internal, sideOutline >
shape ::= < top >, < sides >, < bottom > | < arrow >, where the number of sides is arbitrary
setofShapes( eof ) ::= shape
setofShapes( s + [setofShapes ( eof ) ::= shape + setofShapes( eof )]. Here s is a shape that is added to
previous set of shapes until the end of file is met
```



3.2 Interpretation of Input for Rendering:

1. Corners:
 - a. + is interpreted as a square corner
 - b. / is interpreted as a round corner
 - c. * is interpreted as point marker
2. Outline:
 - a. - and | is interpreted as a solid line
 - b. = and : if present outline is interpreted as a dashed line
3. Arrow:
 - a. < or > at the start/end (or both) of a shape will be interpreted as an arrow
 - i. Same interpretation rules apply in 1 and 2 above for rendering
4. Shape Internals:
 - a. o immediately followed by text will be interpreted as a bullet
 - b. c immediately followed by a color code or XXX will be interpreted as the shape color (XXX is a hex code)

Color Code	Color
RED	
GRE	
BLK	
BLU	
PNK	
YEL	

c. { [option] } will be interpreted as a tag and change how shape is rendered

Option	Description	Shape
d	Document	
s	Storage	
io	Input / Output	
o	Ellipse	
mo	Manual Operation	

c	Decision	
tr	Trapezoid	

d. Text not conforming to the above will be interpreted as plain text

3.3 DITAA GUI Specification

3.3.1 Operation

The GUI that is developed for the DITAA program is intended to be simple. Below is a high-level concept of operation.

1. User starts program and is presented with a blank GUI (figure 1)
2. User selects input file (figure 2)
3. Both the input ASCII art and rendered graphic are displayed (figure 3)
4. User saves the rendered graphic to user specified location

3.3.2 Screenshots

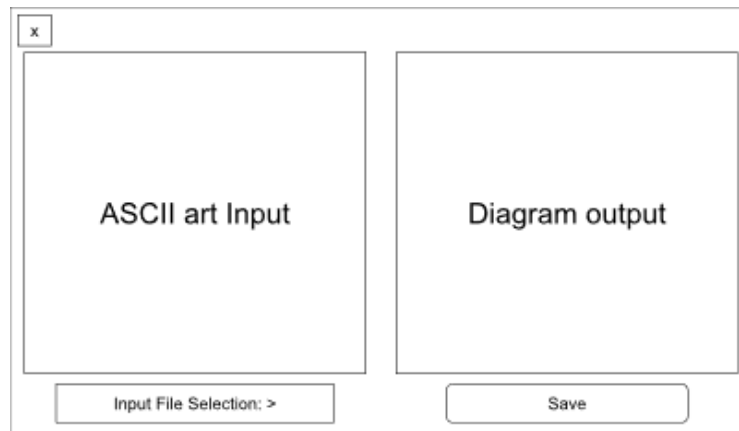


Figure 1: GUI: Blank Presentation to User

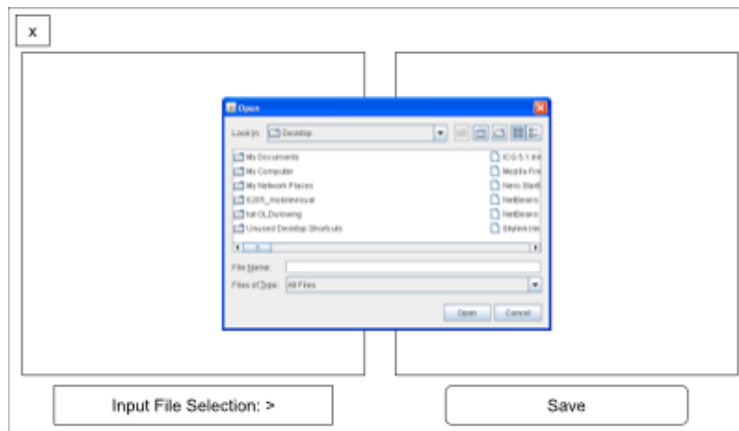


Figure 2: GUI: Select Input



Figure 3: GUI: ASCII and Bitmap Render

3.4 Compliance Test Specification

The following compliance tests will be executed to ensure the specifications described in this document are met:

Test ID	Test Scenario	Test Step	Expected Result
T001	Program is robust to invalid input	Select an input file that does not conform to ASCII art expectations	Program notifies the user and stops gracefully
T002	Program renders single graphic correctly	Use ASCII art input file that contains only one graphic	Program successfully renders graphic in output file
T003	Program renders multiple graphics correctly	Use ASCII art input file that contains multiple graphics	Program successfully renders graphics in output file
T004	Perform T002 & T003 over permutations of graphics (color, shapes, arrows, etc.)	Create and utilize multiple input files to test permutations of specification document	Program successfully renders all permutations
T005	Perform T004 utilizing the GUI	Re-use input files from T004 in GUI	GUI successfully renders both ASCII and bitmap graphic and saves output file successfully

3.5 Code Reduction Specification

3.5.1 Goal

The goal of the code reduction specification is to reduce the overall number of lines of code (LOC) by at least 5%.

3.5.2 Analysis & Metrics

The baseline code will be analyzed using an open source tool (e.g. count lines of code (CLOC)) to identify source files with the most LOC. After analysis, the identified source files will be evaluated for reduction through re-design and/or re-coding.

3.5.3 Verification

To verify the 5% code reduction has been achieved, a measure of LOC before re-design and/or re-coding and after will be conducted. Below is an example of the current LOC count of the baseline DITAA code base using the CLOC program:

```
PS F:\Users\lance\Documents\WSU\CS 7140 - Advanced Software Engineering> ..\..\..\Downloads\cloc-1.88.exe .\ditaa
182 text files.
179 unique files.
116 files ignored.

github.com/AlDanial/cloc v 1.88 T=1.56 s (75.1 files/s, 11652.8 lines/s)
-----
Language          files          blank          comment          code
-----
Java               49             1902             1978             7845
SVG                20              8                20             3062
XML               33             59              18             2243
Markdown          2              102              0              370
HTML              5              25              0              183
JSP               3              24              8              166
Ant               3              13              2              111
Clojure           1               0               0              16
CSS               1               0               0               7
-----
SUM:              117            2133            2026            14003
-----
```

Figure 4: Overall LOC count of original DITAA program

3.6 Assertions

GUI

Entry: A document that supports ascii characters (.txt, .html, etc.).

Exit: Returns a bitmap graphic.

Outline

Entry: - or | ascii character.

Exit: “-” for top and bottom of shape, “|” for left and right of shape on bitmap graphic.

Dashed Outline

Entry: = or : ascii characters

Exit: “=” for top and bottom dashed lines for shape, “:” for left and right dashed lines for shape.

Point

Entry: * ascii character.

Exit: Point created in place of a * on the shape.

Squared corners

Entry: + ascii character.

Exit: Squared corner on output bitmap graphic

Round corners

Entry: / or \ ascii character.

Exit: Rounded corner on output bitmap graphic.

Color

Entry: c###, where ### is a number 0-9.

Exit: Shape color change.

Document

Entry: {d} input inside shape.

Exit: Shape turned to document shape (torn page).

Storage

Entry: {s} input inside shape.

Exit: Shape turned to storage shape (cylinder).

Input/Output

Entry: {io} input inside shape.

Exit: Shape turned into input/output shape (rhombus).

Ellipse

Entry: {o} input inside shape.

Exit: Shape turned into an ellipse.

Manual Operation

Entry: {mo} input inside shape.

Exit: Shape turned into Manual operation shape (upside down trapezoid).

Decision

Entry: {c} input inside shape.

Exit: Shape turned into diamond.

Trapezoid

Entry: {tr} input side shape.

Exit: Shape turned into trapezoid.

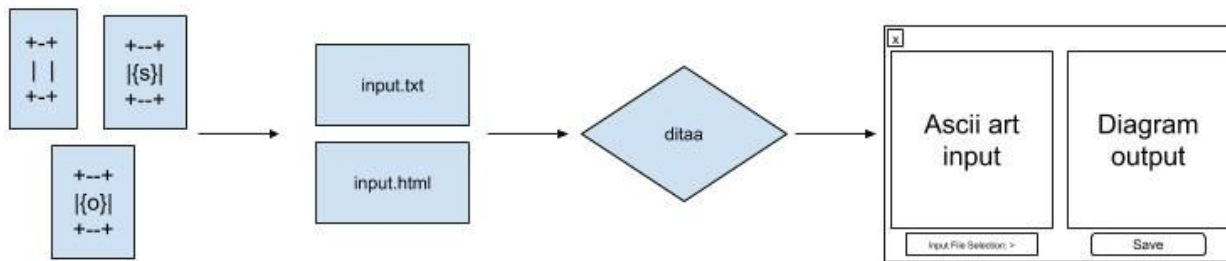
Text

Entry: o X, where X is text.

Exit: A bullet point created where “o” is used.

4. Design

4.1 Architecture Design



For traditional execution of the DITAA program, please reference the DITAA requirements and specification documentation. There are two additional features in which the team will provide to enhance the user experience with DITAA.

The first feature will be the inclusion of additional color presets, to provide the user a more intuitive experience when selecting colors for their output diagrams. The user can use these new color tags in their ASCII art instead of inputting hexadecimal color values. The second feature that will be included is a graphical user interface (GUI) for those users who do not wish to utilize the command line version of DITAA. Instead of outputting the diagram directly to an image file, the diagram will be displayed in a GUI prior to saving it to the host filesystem. The purpose of this allows for the user to know the end result of the diagram prior to getting the file saved and taking up space on the computer and make the overall program much easier to use.

Another goal is to reduce the overall amount of code by 5% or more. By reducing the overall amount of code, this will allow the code to be more accessible to new developers, be easier to maintain, and be more concise and less cluttered.

4.2 Code Outline

There are three major sections for the code: Command Line, Diagram, and Rendering.

The command line section handles the command line input when running the jar. This allows the user to set flags and define the input and output files to do the conversion. The diagram section is the main bulk of the code. This section creates the shapes from the ASCII input. For example, the `diagramShape` class creates the diagram shapes by getting the location and path of the shape defined by ASCII. Finally the rendering section applies the transition from ASCII to a diagram, putting what was originally an ASCII formatted file into a picture format (jpg for example).

4.2.1 Command Line

The command line portion handles the major flags run with the DITAA. The main two classes for this section are `CommandLineConverter.java` and `ProcessingOptions.java`.

The `CommandLineConverterClass`, creates the command line options. It provides the abbreviated form of each flag and defines what they are when the help page is run with DITAA. Once the abbreviated forms are created, the command line is traversed to determine which flags are set while also obtaining the input and output file defined. If a flag is provided, a toggle is set and applied to the `ProcessingOptions.java` object that holds the flag toggles. When the flags need to be applied, they are retrieved from the object with a getter for the flag. The flags are shown below.

```
usage: java -jar ditaa.jar <infile> [outfile] [-A] [-d] [-E] [-e
<ENCODING>] [-h] [--help] [-o] [-r] [-S] [-s <SCALE>] [-t <TABS>]
[-v]
-A,--no-antialias      Turns anti-aliasing off.
-d,--debug             Renders the debug grid over the resulting
                        image.
-E,--no-separation     Prevents the separation of common edges of
                        shapes.
-e,--encoding <ENCODING> The encoding of the input file.
-h,--html              In this case the input is an HTML file. The
                        contents of the <pre class="textdiagram"> tags
                        are rendered as diagrams and saved in the
                        images directory and a new HTML file is
                        produced with the appropriate <img> tags.
                        Prints usage help.
--help
-o,--overwrite         If the filename of the destination image
                        already exists, an alternative name is chosen.
                        If the overwrite option is selected, the image
                        file is instead overwritten.
-r,--round-corners     Causes all corners to be rendered as round
                        corners.
-S,--no-shadows        Turns off the drop-shadow effect.
-s,--scale <SCALE>     A natural number that determines the size of
                        the rendered image. The units are fractions of
                        the default size (2.5 renders 1.5 times bigger
                        than the default).
-t,--tabs <TABS>       Tabs are normally interpreted as 8 spaces but
                        it is possible to change that using this
                        option. It is not advisable to use tabs in
                        your diagrams.
-v,--verbose           Makes ditaa more verbose.
```

4.2.2 Diagram

The diagram section is the main bulk of the program. It is the main functionality to do the conversion from ASCII to diagram. The major classes here are `Diagram`, `DiagramShape`, `TextGrid`, and `CellSet`.

The `Diagram` class is the major overview of what the output becomes. It contains all the shapes that are converted from ASCII characters to diagram symbols and lines. This class contains all the information on where the shapes are located and what shapes need to be located. The `DiagramShape` class is where the shapes are created. In this class, the shapes are created by following by using the edge information for the shapes from the ASCII input file. A path is generated using the edges to create the shape according to the symbol defined in the input file. The `TextGrid` class contains the information about the input file. A grid is created from the file to be able to traverse and gather the characters for transformation. The `CellSet` class contains the information about the object. It provides a set of cells (points on the grid) that are connected to form a shape.

4.2.3 Rendering

The rendering section provides the ability to render the diagram from an ASCII file to a picture format. The major classes used for this are SVGBuilder and BitmapRenderer.

These two classes do pretty much the same thing, but render the output differently. The SVGBuilder renders the diagram in SVG format,, while the BitmapRenderer renders the diagram as a bitmap. Both classes use the diagram created by using the Diagram to render in the shapes, along with the command line options to do any last changes to the final results.

4.3 Detailed Design

4.3.1 GUI

For the GUI(Graphical User Interface), we will be using Java Swing which provides platform-independent and lightweight components. There are several packages that create a good graphical interface of the software like JFrame, JMenu, JMenuItem, JPanel, JTextField, etc.

4.4 Preset Colors

4.4.1 Data Structure - humanColorCodes

Additional preset colors will be included in a private HashMap<String, String> that is already included in the original code base in the TextGrid class located in the text package of the project. The key in the HashMap is the 3 character label of the color (e.g. BLK → Black) and the value with the hexadecimal code for that particular color. The data structure from the original code base can be seen below.

```
private static HashMap<String, String> humanColorCodes = new HashMap<>();
static {
    humanColorCodes.put( k: "GRE", v: "9D9");
    humanColorCodes.put( k: "BLU", v: "55B");
    humanColorCodes.put( k: "PNK", v: "FAA");
    humanColorCodes.put( k: "RED", v: "E32");
    humanColorCodes.put( k: "YEL", v: "FF3");
    humanColorCodes.put( k: "BLK", v: "000");
}
```

During execution, this data structure will be queried when a color tag is found during parsing of the ASCII art. The associated hexadecimal value with the found tag will be used to color the diagram when rendering.

4.5 Code Reduction

Code reduction will primarily be achieved through the refactoring and/or re-coding of the previous code base. There are a multitude of unused and incomplete functions that make the code complicated to maintain. These functions, as described below, will be removed from the production code base.

The CLOC software analysis tool was used to identify classes in the DITAA program which had the most lines of source code and this was used to prioritize the team's efforts. A breakdown of the classes and the number of lines of blanks, comments, and code can be seen below.

Core Package:

File	Blank	Comment	Code
CommandLineConverter.java	53	44	210
ConfigurationParser.java	27	22	155
ConversionOptions.java	20	75	98
DebugUtils.java	1	19	6
DocBookConverter.java	7	20	40
FileUtils.java	23	30	98
HTMLConverter.java	46	45	147
Pair.java	2	19	9
PerformanceTester.java	13	23	28
ProcessingOptions.java	42	89	112
RenderingOptions.java	28	33	71
Shape3DOrderingComparator.java	7	27	12
ShapeAreaComparator.java	7	27	12

Graphics Package:

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

Text Package:

File	Blank	Comment	Code
AbstractCell.java	16	23	84
AbstractionGrid.java	25	40	117
CellSet.java	102	132	453
GridPattern.java	38	107	188
GridPatternGroup.java	82	30	322
StringUtils.java	25	50	94
TextGrid.java	287	181	1318

A few functions that have already been identified for refactoring/re-coding/deletion will be shown below, but will be discussed in further detail in the implementation documentation. Note that the below is **NOT** all-inclusive.

4.5.1 TextGrid::replaceAll(char c1, char c2)

```
/**
 * Replace all occurrences of c1 with c2
 *
 * @param c1
 * @param c2
 */
public void replaceAll(char c1, char c2){
    int width = getWidth();
    int height = getHeight();
    for(int yi = 0; yi < height; yi++){
        for(int xi = 0; xi < width; xi++){
            char c = get(xi, yi);
            if(c == c1) set(xi, yi, c2);
        }
    }
}
```


4.5.2 TextGrid::exactlyOneNeighbourIsBoundary(Cell cell)

```
public boolean exactlyOneNeighbourIsBoundary(Cell cell) {
    int howMany = 0;
    if(isBoundary(cell.getNorth())) howMany++;
    if(isBoundary(cell.getSouth())) howMany++;
    if(isBoundary(cell.getEast())) howMany++;
    if(isBoundary(cell.getWest())) howMany++;
    return (howMany == 1);
}
```

4.5.3 TextGrid::seedFill(Cell seed, char newChar)

```
private CellSet seedFill(Cell seed, char newChar){
    CellSet cellsFilled = new CellSet();
    char oldChar = get(seed);

    if(oldChar == newChar) return cellsFilled;
    if(isOutOfBounds(seed)) return cellsFilled;

    Stack<Cell> stack = new Stack<>();

    stack.push(seed);

    while(!stack.isEmpty()){
        Cell cell = (Cell) stack.pop();

        //set(cell, newChar);
        cellsFilled.add(cell);

        Cell nCell = cell.getNorth();
        Cell sCell = cell.getSouth();
        Cell eCell = cell.getEast();
        Cell wCell = cell.getWest();

        if(get(nCell) == oldChar && !cellsFilled.contains(nCell)) stack.push(nCell);
        if(get(sCell) == oldChar && !cellsFilled.contains(sCell)) stack.push(sCell);
        if(get(eCell) == oldChar && !cellsFilled.contains(eCell)) stack.push(eCell);
        if(get(wCell) == oldChar && !cellsFilled.contains(wCell)) stack.push(wCell);
    }

    return cellsFilled;
}
```


5. Implementation

5.1 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.

5.2 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();
            }
        }
    }
}
```

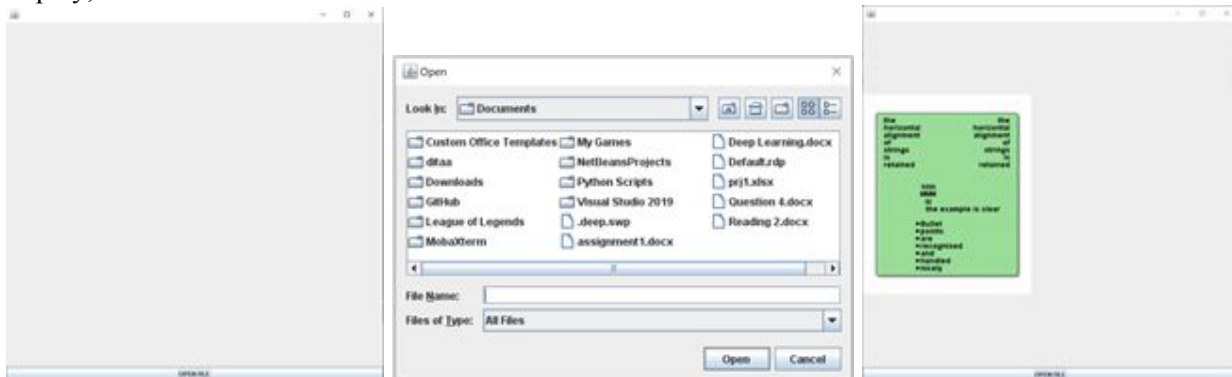
5.3 GUI

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.



6. Testing

6.1 Smoke Testing

To ensure the DITAA program is able to execute at a course level, two separate smoke tests will be run. The main purpose of this test is to verify that the program is able to execute at a course level before proceeding with more stringent tests.

The first test is to verify the DITAA program runs with no input and the appropriate help menu is displayed when run. The command that will be used for this purpose can be seen below.

```
$ java -jar ditaa.jar
```

The second test that will be run will be with a simple, verified correct, input ascii art text file (Example 6.txt). This test is to confirm that the DITAA program can run when given a correct input file and the expected output png is rendered. The command that will be used for this purpose can be seen below:

```
$ java -jar ditaa.jar "Example 6.txt"
```

6.1.1 Smoke Test #1

Smoke test #1 is the running of the DITAA program with no file input. The expected result is the output of the help menu to the console.

Smoke Test #1 Input

```
java -jar .\ditaa.jar
```

Smoke Test #1 Output

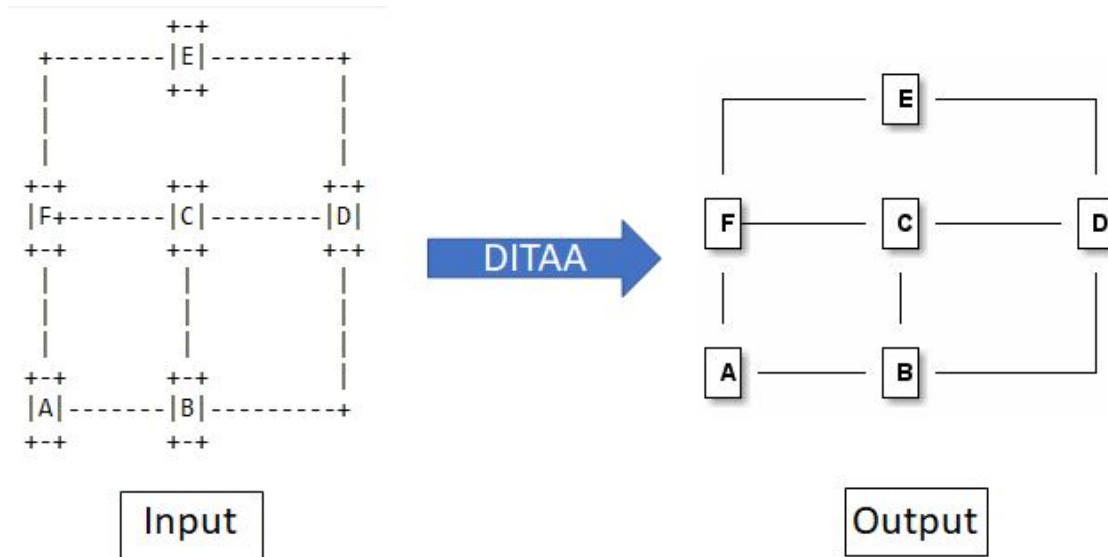
```
usage: java -jar ditaa.jar <INPFILE> [OUTFILE] [-A] [-b <BACKGROUND>] [-d]
      [-E] [-e <ENCODING>] [-h] [--help] [-o] [-r] [-S] [-s <SCALE>]
      [--svg] [--svg-font-url <FONT>] [-T] [-t <TABS>] [-v] [-W]
-A,--no-antialias      Turns anti-aliasing off.
-b,--background <BACKGROUND> The background colour of the image. The
                             format should be a six-digit hexadecimal
                             number (as in HTML, FF0000 for red). Pass
                             an eight-digit hex to define transparency.
                             This is overridden by --transparent.
-d,--debug             Renders the debug grid over the resulting
                             image.
-E,--no-separation     Prevents the separation of common edges of
                             shapes.
-e,--encoding <ENCODING> The encoding of the input file.
-h,--html              In this case the input is an HTML file.
                             The contents of the <pre
                             class="textdiagram"> tags are rendered as
                             diagrams and saved in the images directory
                             and a new HTML file is produced with the
                             appropriate <img> tags.
--help                Prints usage help.
-o,--overwrite         If the filename of the destination image
                             already exists, an alternative name is
                             chosen. If the overwrite option is
                             selected, the image file is instead
                             overwritten.
-r,--round-corners     Causes all corners to be rendered as round
                             corners.
-S,--no-shadows        Turns off the drop-shadow effect.
-s,--scale <SCALE>     A natural number that determines the size
                             of the rendered image. The units are
                             fractions of the default size (2.5 renders
                             1.5 times bigger than the default).
--svg                 Write an SVG image as destination file.
--svg-font-url <FONT> SVG font URL.
-T,--transparent       Causes the diagram to be rendered on a
                             transparent background. Overrides
                             --background.
-t,--tabs <TABS>       Tabs are normally interpreted as 8 spaces
                             but it is possible to change that using
                             this option. It is not advisable to use
                             tabs in your diagrams.
-v,--verbose           Makes ditaa more verbose.
-W,--fixed-slope       Makes sides of parallelograms and
                             trapezoids fixed slope instead of fixed
                             width.
```

Result - **SUCCESS**

The output of the DITAA program was what was expected and thus smoke test #1 is considered successful.

6.1.2 Smoke Test #2

Smoke test #2 is the running of the DITAA program with a simple, verified correct, ascii art text file. The input file contents (Example 6.txt) and the expected output can be seen below.



Smoke Test #2 Input

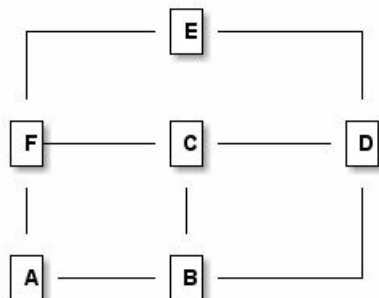
```
java -jar .\ditaa.jar 'F:\Users\lance\Downloads\Example 6.txt'
```

Smoke Test #2 Output

```
ditaa version 0.11, Copyright (C) 2004--2017 Efsthios (Stathis) Sideris

Running with options:
Reading file: F:\Users\lance\Downloads\Example 6.txt
Rendering to file: F:\Users\lance\Downloads\Example 6.png
Done in 0sec
```

Photos - Example 6.png



Result - SUCCESS

The output of the DITAA program was what was expected and thus smoke test #2 is considered successful.

6.2 White Box Testing

JUnit testing will be used to white box test 2 critical classes of the DITAA program, TextGrid and CellSet.

6.2.1 TextGrid Class

6.2.1.1 testFillContinuousAreaSquareOutside()

The goal of this test is to verify that a connected square that is read and parsed from a file is the same as when created programmatically. A text file containing a simple closed square (simple_square01.txt) is used to initially create a filled area. This simple square is filled and tested to be accurate before moving on to programmatically producing the same expected square and testing that they are equal. Below are the contents of simple_square01.txt and the JUnit code to test this case respectively.

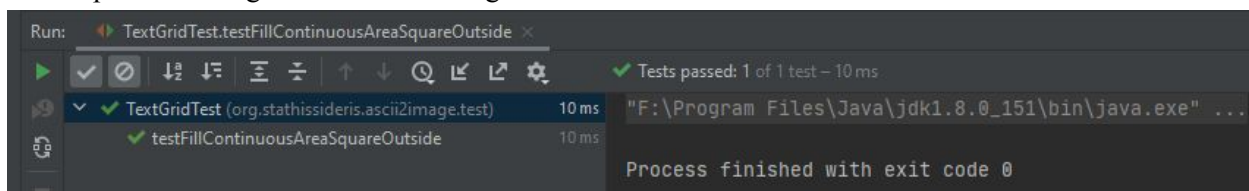


```
@Test public void testFillContinuousAreaSquareOutside() throws FileNotFoundException, IOException {
    TextGrid squareGrid;
    squareGrid = new TextGrid();
    //squareGrid.loadFrom("../..\\test-resources\\text\\simple_square01.txt");
    squareGrid.loadFrom(filename: "F:\\Users\\lance\\Documents\\WSU\\CS 7140 - Advanced Software Engineering\\" +
        "ditaa\\test-resources\\text\\simple_square01.txt");

    CellSet filledArea = squareGrid.fillContinuousArea(x: 0, y: 0, c: '*');
    int size = filledArea.size();
    assertEquals(expected: 64, size);

    CellSet expectedFilledArea = new CellSet();
    addSquareToCellSet(squareGrid, expectedFilledArea, x: 0, y: 0, width: 11, height: 2);
    addSquareToCellSet(squareGrid, expectedFilledArea, x: 0, y: 7, width: 11, height: 2);
    addSquareToCellSet(squareGrid, expectedFilledArea, x: 0, y: 2, width: 2, height: 5);
    addSquareToCellSet(squareGrid, expectedFilledArea, x: 9, y: 2, width: 2, height: 5);
    assertEquals(expectedFilledArea, filledArea);
}
```

The output of running this test case through the IntelliJ IDEA IDE can be found below.



6.2.1.2 testFindBoundariesExpandingFromSquare()

The goal of this test is to verify that the boundaries of a connected square parsed from a file is the same as a connected square created programmatically. A text file containing a simple closed square (simple_square01.txt) is used to initially create and determine the boundary size. Below are the contents of simple_square01.txt and the JUnit code to test this case respectively.



```
@Test public void testFindBoundariesExpandingFromSquare() throws FileNotFoundException, IOException {
    TextGrid grid;
    grid = new TextGrid();
    //grid.loadFrom("tests/text/simple_square01.txt");
    grid.loadFrom( filename: "F:\\Users\\lance\\Documents\\WSU\\CS 7140 - Advanced Software Engineering\\" +
        "ditaa\\test-resources\\text\\simple_square01.txt");

    CellSet wholeGridSet = new CellSet();
    addSquareToCellSet(grid, wholeGridSet, x: 0, y: 0, grid.getWidth(), grid.getHeight());

    TextGrid copyGrid = new AbstractionGrid(grid, wholeGridSet).getCopyOfInternalBuffer();
    CellSet boundaries = copyGrid.findBoundariesExpandingFrom(copyGrid.new Cell( x: 8, y: 8));
    int size = boundaries.size();

    boundaries.printAsGrid();
    assertEquals( expected: 56, size);

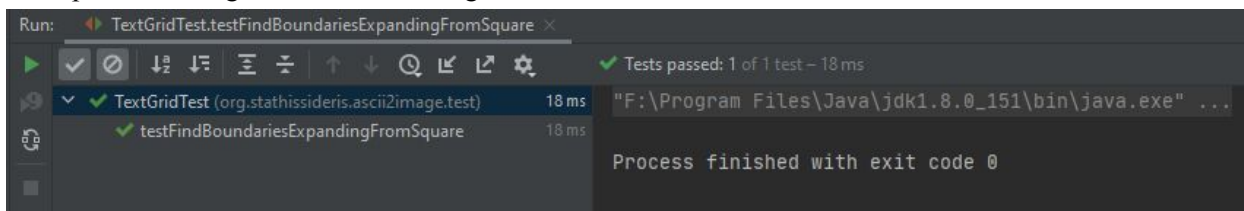
    CellSet expectedBoundaries = new CellSet();

    addSquareToCellSet(copyGrid, expectedBoundaries, x: 8, y: 7, width: 17, height: 1);
    addSquareToCellSet(copyGrid, expectedBoundaries, x: 8, y: 19, width: 17, height: 1);

    addSquareToCellSet(copyGrid, expectedBoundaries, x: 7, y: 8, width: 1, height: 11);
    addSquareToCellSet(copyGrid, expectedBoundaries, x: 25, y: 8, width: 1, height: 11);

    expectedBoundaries.printAsGrid();
    assertEquals(expectedBoundaries, boundaries);
}
```

The output of running this test case through the IntelliJ IDEA IDE can be found below.



6.2.2 CellSet Class

6.2.2.1 testContains()

The goal of this test is to verify that cells are correctly constructed and stored by the TextGrid class, to include duplicate cells. All cells are created programmatically. Below is the JUnit code to test this case. Note that the entire class is shown to include the necessary setup method for testing.

```
public class CellSetTest {

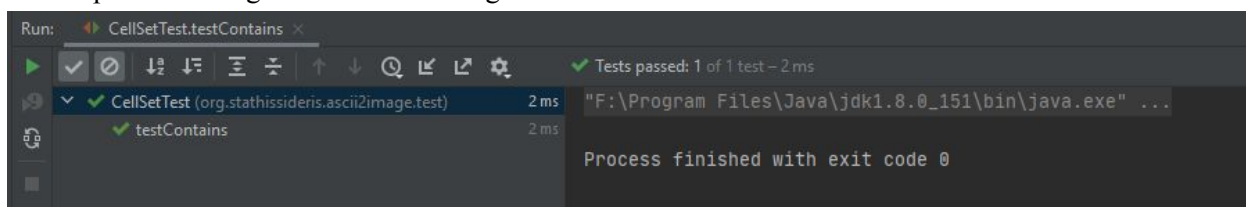
    TextGrid g = new TextGrid();
    CellSet set = new CellSet();

    @Before public void setUp() {
        set.add(g.new Cell( x: 10, y: 20));
        set.add(g.new Cell( x: 10, y: 60));
        set.add(g.new Cell( x: 10, y: 30));
        set.add(g.new Cell( x: 60, y: 20));
    }

    @Test public void testContains() {
        TextGrid.Cell cell1 = g.new Cell( x: 10, y: 20);
        TextGrid.Cell cell2 = g.new Cell( x: 10, y: 20);

        assertTrue(cell1.equals(cell2));
        assertTrue(set.contains(cell1));
    }
}
```

The output of running this test case through the IntelliJ IDEA IDE can be found below.



6.3 Black Box Testing

A number of black box tests will be performed by providing input files to the DITAA program and comparing the rendered output png, to the expected output. A number of tests are detailed below.

6.3.1 Simple Shapes

6.3.1.1 simple_square01.txt

Input:



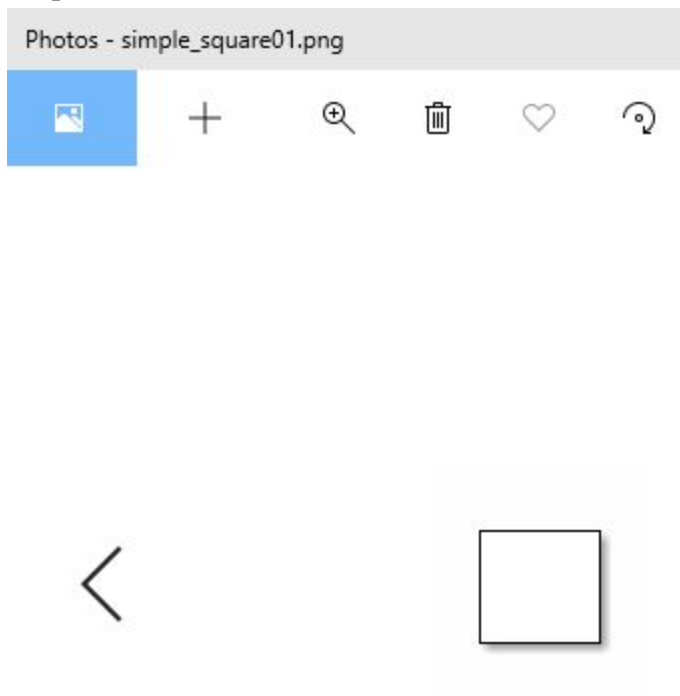
Execution:

```
java -jar .\ditaa.jar '..\..\test-resources\text\simple_square01.txt' '..\..\test_doc\results\simple_square01.png'

ditaa version 0.11, Copyright (C) 2004--2017 Efsthios (Stathis) Sideris

Running with options:
Reading file: ..\..\test-resources\text\simple_square01.txt
Rendering to file: ..\..\test_doc\results\simple_square01.png
Done in 0sec
```

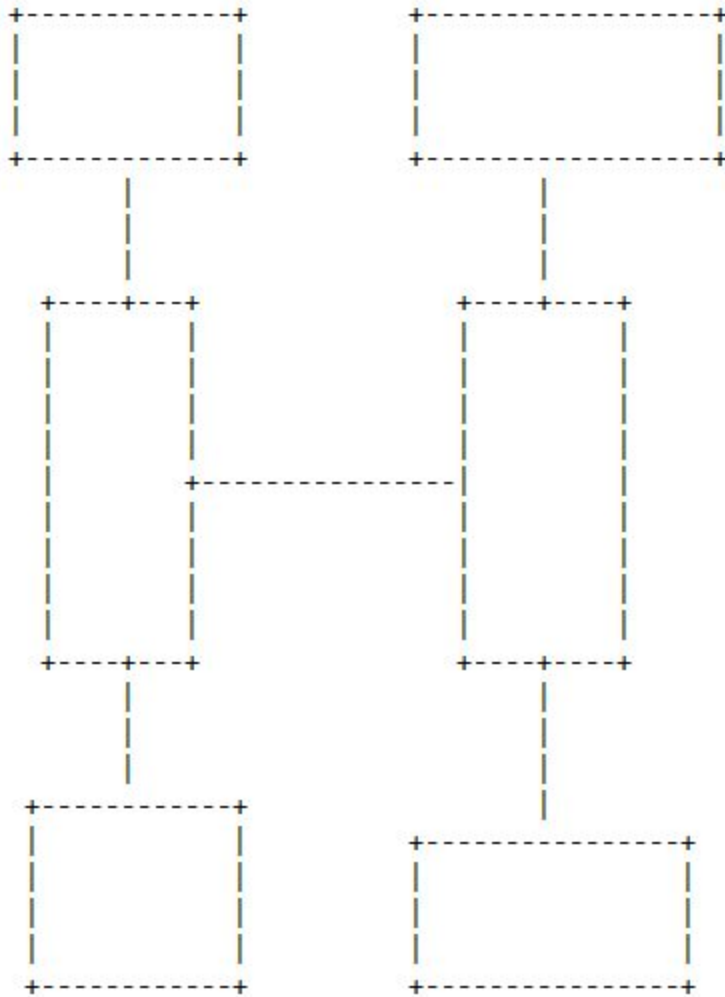
Output:



Result: **SUCCESS**

6.3.1.2 Example_7.txt

Input:



Execution:

```
java -jar .\ditaa.jar '..\..\test-resources\text\Example 7.txt' ..\..\test_doc\results\Example_7.png
```

ditaa version 0.11, Copyright (C) 2004--2017 Efsthathios (Stathis) Sideris

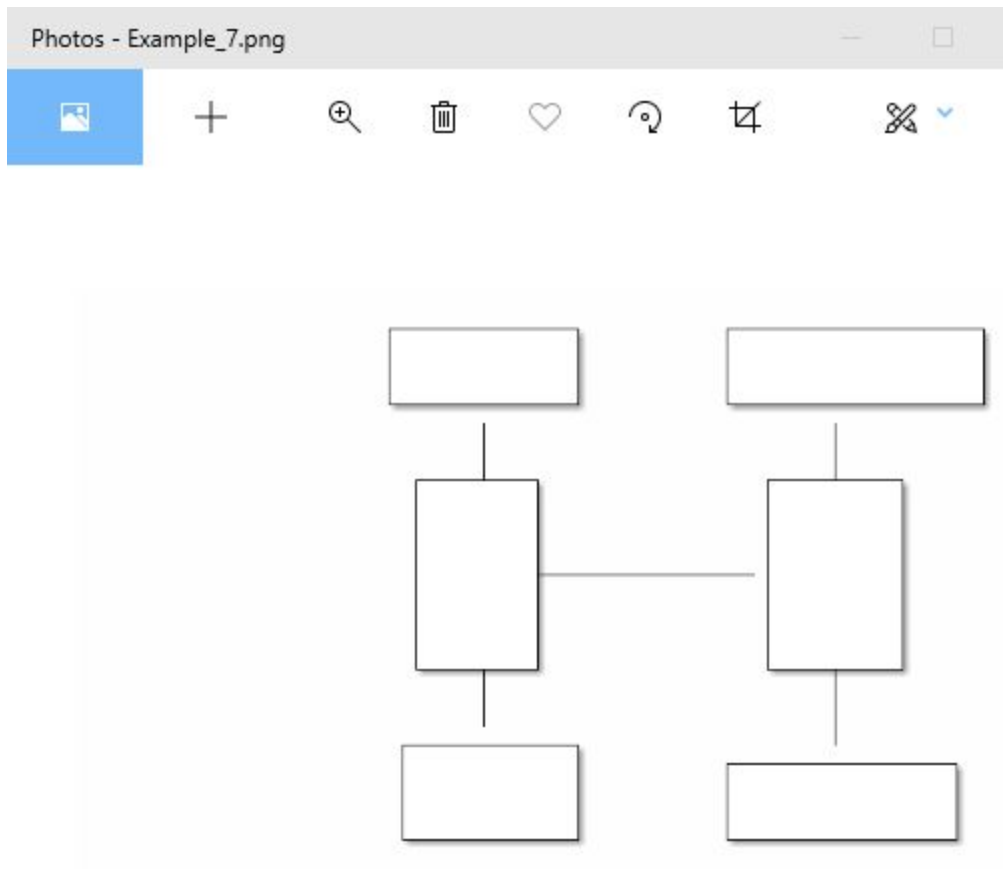
Running with options:

Reading file: ..\..\test-resources\text\Example 7.txt

Rendering to file: ..\..\..\test_doc\results\Example_7.png

Done in 1sec

Output:

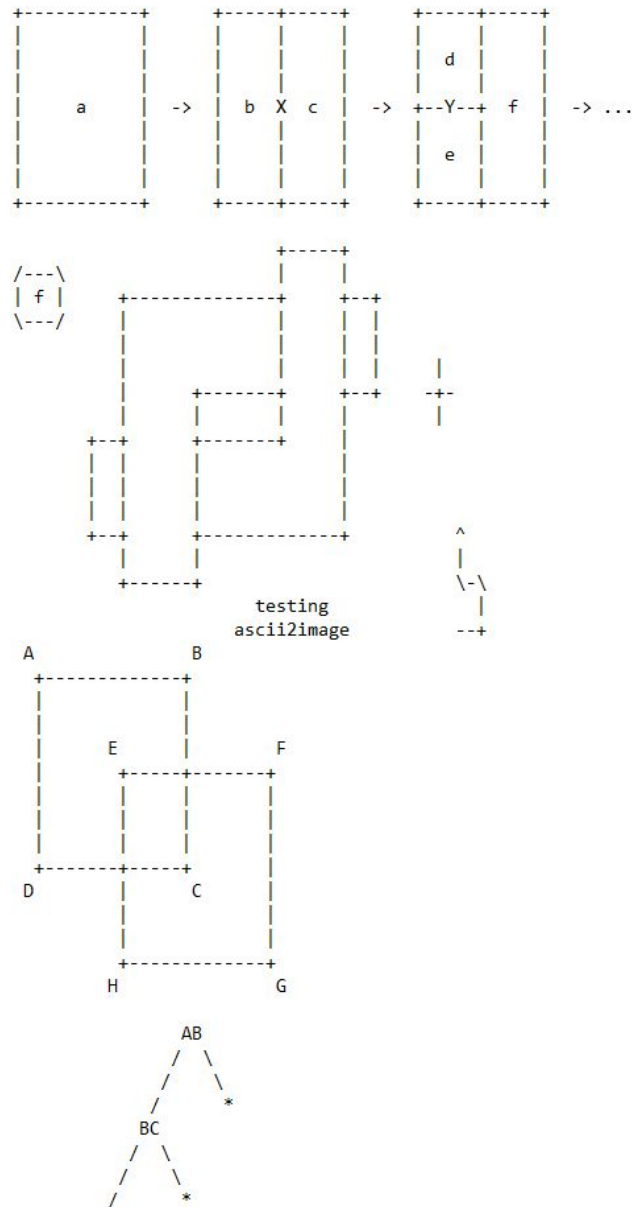


Result: SUCCESS

6.3.2 Advanced Shapes

6.3.2.1 art1.txt

Input:



Execution:

```
java -jar .\ditaa.jar '..\..\test-resources\text\art1.txt' '..\..\test_doc\results\art1.png'
```

```
ditaa version 0.11, Copyright (C) 2004--2017 Efstathios (Stathis) Sideris
```

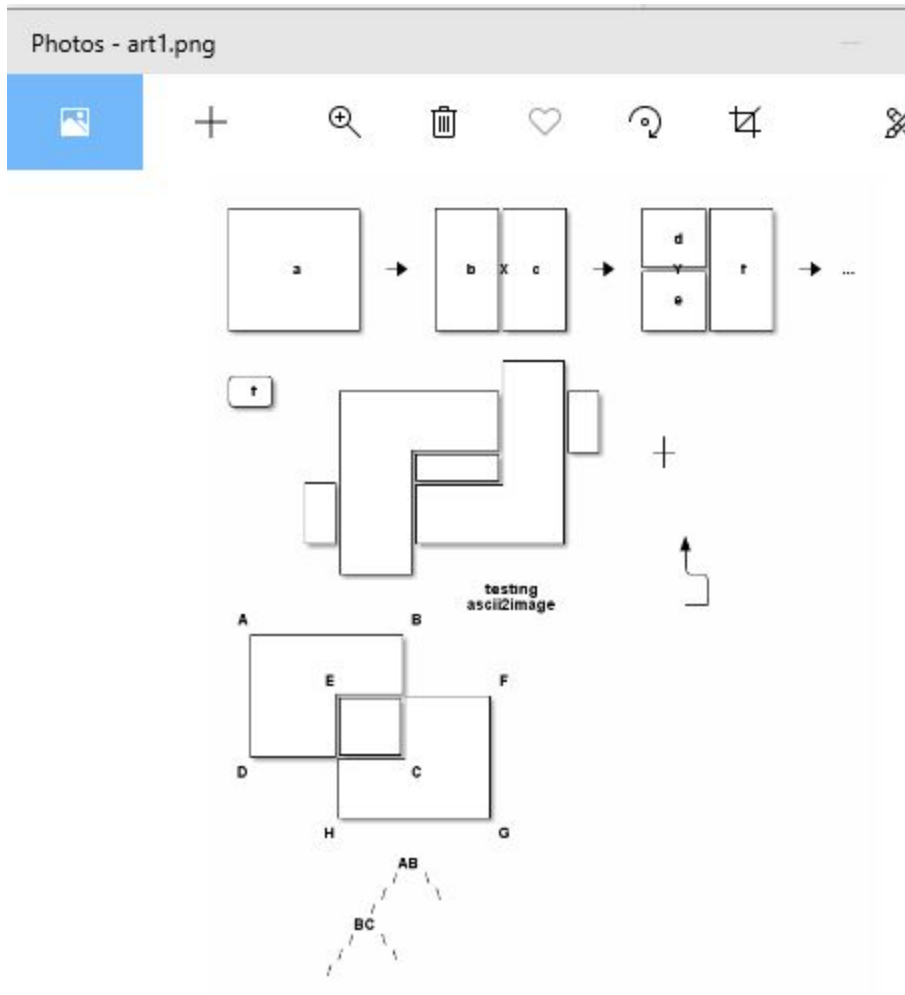
```
Running with options:
```

```
Reading file: ..\..\test-resources\text\art1.txt
```

```
Rendering to file: ..\..\test_doc\results\art1.png
```

```
Done in 1sec
```

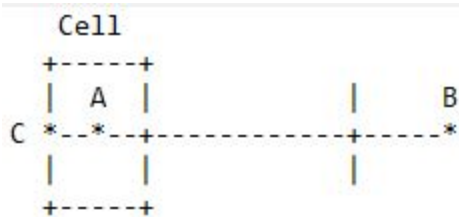
Output:



Result: **SUCCESS**

6.3.2.2 bug9_5.txt

Input:



Execution:

```
java -jar .\ditaa.jar '..\..\test-resources\text\bug9_5.txt' '..\..\test_doc\results\bug9_5.png'
```

```
ditaa version 0.11, Copyright (C) 2004--2017 Efstathios (Stathis) Sideris
Running with options:
Reading file: ..\..\test-resources\text\bug9_5.txt
Rendering to file: ..\..\test_doc\results\bug9_5.png
Done in 0sec
```

Output:



Result: **FAIL**

Discussion:

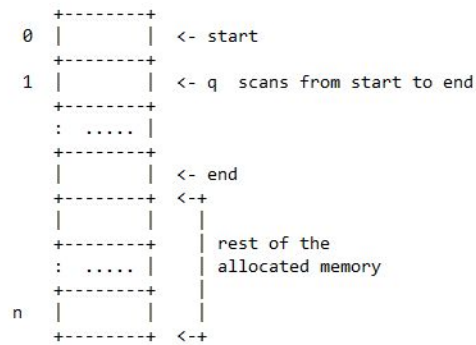
Expected output is that a single connected box would be created, not two. Initial analysis suggests that problem is occurring within Diagram::seperateCommonEdges().

6.4 Stress Test

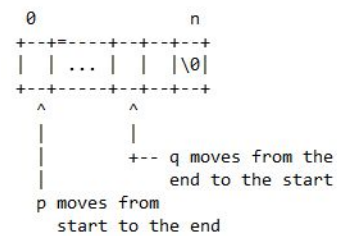
A stress test was performed on the DITAA program in order to see how it would handle a large file size. An ascii art text file was generated that had repeated content (see below) to produce a single large file size compared to the previous test. Previous tests performed would run in the single digit seconds, whereas the stress took 40 seconds on the same compute hardware.

Input (copied multiple times in a single file):

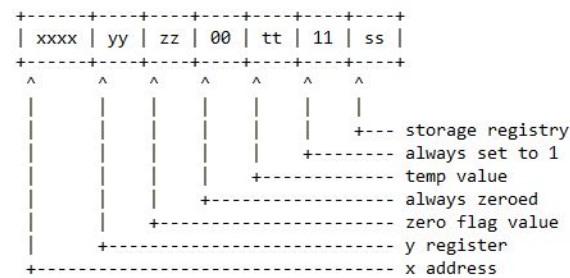
Memory:



Strings:



Sectioned:



Execution:

```
java -jar .\ditaa.jar '..\..\test-resources\text\Stress Test.txt' '..\..\test_doc\results\Stress_Test.png'
```

```
ditaa version 0.11, Copyright (C) 2004--2017 Efsthios (Stathis) Sideris
```

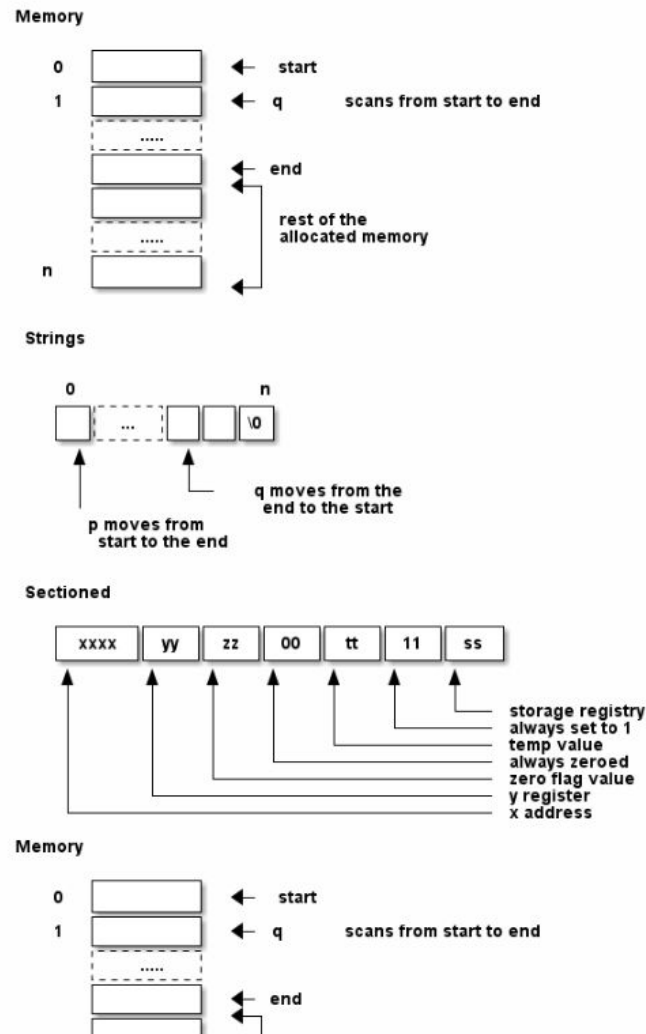
```
Running with options:
```

```
Reading file: ..\..\test-resources\text\Stress Test.txt
```

```
Rendering to file: ..\..\test_doc\results\Stress_Test.png
```

```
Done in 40sec
```

Output:



Result: **SUCCESS**

6.5 Acceptance Test

The following specifications were utilized for acceptance testing. The following is a summarization of the specification tested, which file it was tested by, and the following result. All tests must be successful for the program to be accepted for release.

Specification	File Tested By	Result
Horizontal Line {'-', '='}	art2.txt, art10.txt	Success
Vertical Line {' ', ':'}	art2.txt, art10.txt	Success
Corner Characters {'+', '/', '\'}	art2.txt, art10.txt	Success
Closed Shape	art2.txt, art10.txt	Success

Dashed Lines {‘:’, ‘=’}	art10.txt	Success
Closed Shape + Input Text	art2.txt, art10.txt	Success
Closed Shape + Color (Hex)	color_codes_hex.txt	Success
Closed Shape + Color (Predefined)	art2.txt, art10.txt	Success
Closed Shape + “{d}”	art10.txt	Success
Closed Shape + “{s}”	art10.txt	Success
Closed Shape + “{io}”	art10.txt	Success
Closed Shape + Bullet “o” + Text	art2.txt	Success
Arrowhead - Down	art2.txt	Success
Arrowhead - Right	art2.txt, art10.txt	Success
Arrowhead - Left	art2.txt	Success
Arrowhead - Right	art2.txt	Success

6.6 Additional Requirements

6.6.1 Code Reduction

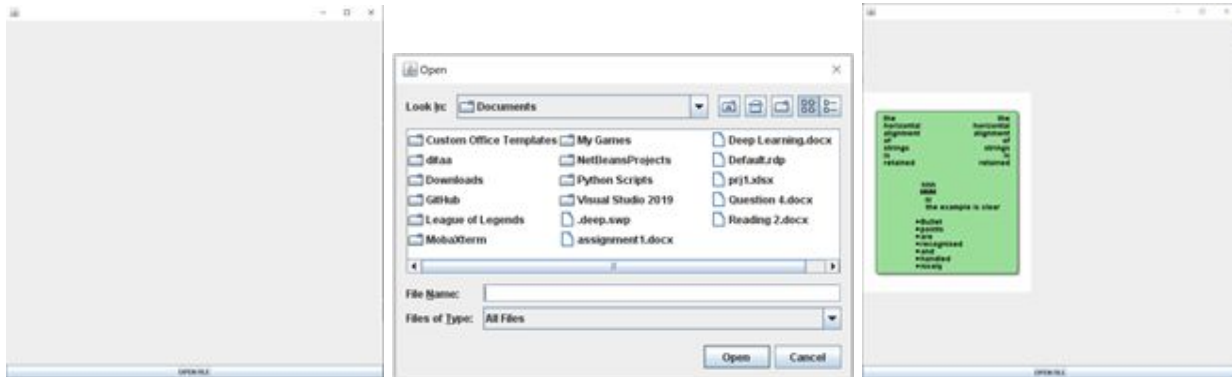
Refer to the implementation section for more information on the reduction of code size.

6.6.2 Maintainability

Refer to the implementation section for more information on code reduction and its impact on the overall maintainability of the code.

6.6.3 Graphical User Interface

A GUI was developed as a more accessible front-end for the DITAA program. Below is a screenshot of the developed GUI.



7. Conclusion

Close to the conclusion of the program, a common specification document was chosen that was similar but distinct from that which is found in chapter 3, Specification. This change in direction in the midst of development undoubtedly caused changes in our program.

If time allowed, the design (chapter 4) and implementation (chapter 5) of our program would have been updated to reflect this change in software specification. The contents of chapter 6, Testing, is the only chapter that wholly reflects the change in specification.

8. Reflection

8.1 Lance Matthieu

Creating the software documentation for an already existing project made for a unique experience. As mentioned at the start of class, having to comprehend someone else's code is a challenge that all of us will face in our life after academia. I would hope that code utilized in industry will have a rigorous software development cycle and documentation generated for this class would already exist, facilitating new developers on the team. I know that this is not always the case.

I think the biggest takeaways that I have from this exercise is what was previously discussed. I know generating the requirements, specification, design, implementation, and testing documents were meant to be learning exercises but felt that the majority of those documents were focused on what was already done. We never got to exercise our ability to generate those documents to a problem that we were introduced with. In hindsight, I think it would have been much more beneficial to generate our own requirements, specification and design to a solution that was our own.

I do want to say that although I thought the team dynamic in a remote class would be difficult, Abhishek and Griffin made it a very enjoyable experience and were very reliable. I know in a graduate course this is expected, but it is not always the case.

8.2 Abhishek Pandya

As Lance said in his reflection that it would be difficult to work in a team in this online classes situation but I did enjoy working in a team. Both teammates were so helpful that in any situation when I say I have difficulty doing any task they helped me each time.

In this course, I learned how to write documents for an existing software. In the early course I did learn about SDLC(Software Development Life Cycle) but I never applied in real life. In this, I enjoyed making document which go through each phase of SDLC like requirement, specification, design, implementation, and testing. I did not know java Swings which is used for GUI. In the implementation phase, I tried learning Swings. I also followed all the rules which needed to be followed in order to create the document.

8.3 Griffin Mosley

The project overall was a good experience. Using both a group environment and preexisting code made it unique to all other classes I have taken. The preexisting code was the major take away from the project. Being able to understand the code developed by another person is a skillset that is required for the field. This project made it so you had to fully understand the code in order to accomplish the final results. However, the down-side is that the project seemed to be a little unorganized, with every group set out to accomplish different goals at the beginning, but then put into a position where each group was limited in their creativity. The project started off looking towards developing a GUI, plugin, and providing a 5% code reduction. At the end of the project, a week before it is due, the project shifted into changing the group's goal set to meet another group's specifications if time permitted, this being only a 5% code reduction.

I felt as though consistency lacked between the groups as well. Some groups looked to accomplish what the project stated, while others cut portions out focusing solely on the 5% code reduction. Code reduction is very important, as this allows code to be maintained going forward, however the overall experience of using ditaa was lackluster. The process of running the jar, waiting for the output, then having to open up a file every time eventually got tedious and annoying to do. The GUI development made this process significantly easier to do which made testing a lot easier.

The group aspect was enjoyable as well. Each participant was active and fully involved, providing input and resources as necessary. The group as whole met several times before each portion was due and actively pulled their part. However, this aspect was not unique. A twist for future classes might be to split up the class into groups to work on different portions of the project. For example, one group works on developing a GUI, one on code reduction, and another on a plug-in. At the very end all groups would come together and attempt to piece together the project as a whole. This would make grading much more difficult, but would provide a much better experience in my opinion.

9. References

1. <https://github.com/stathissideris/ditaa>
2. http://web.cse.ohio-state.edu/~bair.41/616/Project/Example_Document/Req_Doc_Example.html
3. https://www.ece.rutgers.edu/~marsic/books/SE/book-SE_marsic.pdf
4. http://www.plcdev.com/ascii_chart
5. <https://techterms.com/definition/eps>
6. <https://www.guru99.com/conformance-testing.html>
7. <http://biorob2.epfl.ch/pages/studproj/birg66475/Project%20specifications.pdf>
8. <https://cecs.wright.edu/~pmateti/Courses/7140/Lectures/Examples/TicTacToe-JavaFX-UnRedo/tictactoe-spec.html>
9. <https://cecs.wright.edu/~pmateti/Courses/7140/Lectures/Examples/Tabulate-Eqns/>
10. <https://cecs.wright.edu/~pmateti/Courses/7140/Lectures/Examples/Condense-Num-Ranges/>
11. <https://github.com/AIDanial/cloc>
12. <http://cloc.sourceforge.net/>
13. https://en.wikipedia.org/wiki/Software_quality
14. <https://plugins.jetbrains.com/plugin/12415-statistics>

10. Team Journals

10.1 Lance Matthieu

26 August 2020:

- Posted on Pilot forum introducing myself and requesting teammates for group project

30 August 2020:

- Received responses from fellow students and sent a team e-mail organizing ongoing communications
- Team decided on utilizing discord, so I set up a group project channel and provided invitation link to team members

08 September 2020:

- Reached out to Griffin Mosely to join our current team, which he accepted as he was looking to join a team

14 September 2020:

- Joined group GitHub repository for project created by Daniel Ketterer
- Obtained access to Google Docs requirements document, set up by Griffin
- Reviewed Marsic requirements documentation example (Appendix G)
- Researched other examples of requirements documentation
 - Found example from Ohio State University that team will likely use for our requirements document
- Organized team meeting to discuss requirements document on 15 September at 1400

15 September 2020:

- Cloned ditaa repository utilizing IntelliJ IDEA IDE
- After obtaining ditaa dependencies, was able to build and run ditaa
- Team meeting: Discussed outline of Requirements documentation and handed out responsibilities
 - Lance: Introduction
 - Abhishek: Functional & Non-Functional Objectives
 - Daniel: Context Model and GUI Look & Feel
 - Griffin: Use Case Model
- Organized follow-up team meeting for 16 September 2020 at 1700
- Started Introduction of requirements document

16 September 2020:

- Completed first draft of introduction section and included in shared google drive document
 - Submitted before team meeting to allow for review beforehand
- Attended team meeting to discuss progress on requirements documentation
 - Decided GUI look & feel would be completed by Griffin (done before end of meeting)
- Reviewed team members portions of the requirements document and provided feedback
- Scheduled final team meeting on requirements documentation for 17 September 2020 at 1700
- Finished incorporating feedback to introduction portion of requirements documentation

- Added cover page and table of contents

17 September 2020:

- Attended final team meeting before turn-in of requirements documentation
 - Provided feedback on non-functional requirements section, glossary of terms, and sources
- Finalized requirements document and Abhishek took point to submit via e-mail
- Plan is to reconvene on 28 September 2020 to start discussion on specifications document

28 September 2020:

- Coordinated team meeting for 30 September to discuss specifications documentation

29 September 2020:

- Submitted requirements documentation feedback for teams 1 & 2 via pilot

30 September 2020:

- Discussed approach to specifications document
- Decided to break for a few days before deciding on path forward
- Team meeting set up for 5 October at 1700

4 October 2020:

- Researched CS 7140 slides for specification guidance
- Researched potential templates for specification document

5 October 2020:

- Team meeting to discuss outline for specifications document and agreed upon responsibilities:
 - Lance: Specifications
 - Abhishek: Introduction & Project Expectations
 - Griffin: Assertions
- Organized battle rhythm for documentation review and feedback
 - Feedback: 8 OCT
 - Draft: 9 OCT
 - Final Review: 12 OCT

7 October 2020:

- Finished research on specifications approach
 - Finalized to 3 potential approaches – plan to discuss with team at next meeting before starting

8 October 2020:

- Team meeting
 - Provided feedback on fellow teammates current content
 - Requested team feedback on selection of specification approach
 - Draft of specifications to be complete by next meeting 9 October at 1700
 - Abhishek to incorporate feedback on sections 1 & 2
 - Griffin to incorporate feedback on section 4

9 October 2020:

- Completed draft of specifications sections and incorporated into google docs draft
- Added table of contents and cover page to specifications document
- Completed other formatting tasks (e.g. Headings, font, etc.)
- Team Meeting:
 - Received feedback from other team members on current draft of specification
 - Still need to complete code reduction portion of specification
 - Griffin to complete Conclusion section
 - Next meeting organized for 12 October at 1700

12 October 2020:

- Completed code reduction section of specification and added to google doc
- Added personal journal to specification document
- Team Meeting:
 - Discussed inclusion of quality in non-formal requirements
 - Griffin added section 2.2.5 Quality
 - Added software quality link in references
 - Abhishek agreed to finalize team journal and upload final specifications document to GitHub

16 October 2020:

- Team Meeting:
 - Discussed expectations for project design document
 - Decided to take a week and become more familiar with the current code
 - Next meeting set for 23 October

18 October 2020:

- Reviewed code to determine execution flow
- Diagram & TextGrid classes seem the most important

23 October 2020:

- Team Meeting:
 - Meeting was short and unproductive due to focus on upcoming midterms
 - Decision to start the design document on enhancements and GUI
 - Abhishek to start GUI design/development
 - All members were able to briefly skim the code, but do not have enough understanding to start design document
 - Next meeting set for 29 October

29 October 2020:

- Team Meeting:
 - Members have a slightly deeper understanding of the code as compared to last meeting
 - Abhishek has started GUI design/development
 - Next meeting set for 30 October

30 October 2020:

- Team Meeting:
 - Griffin to look for software design document reference by next meeting
 - Abhishek posted screenshots of current GUI progress
 - Next meeting set for November 2

1 November 2020:

- Utilized source code analysis software tool, Count Lines of Code (CLOC), to determine overall number of lines for code reduction goal
- Ran tool over each class to identify large classes to prioritize code reduction

2 November 2020:

- Team Meeting:
 - Provided code analysis to team
 - Goal is to identify primary classes to be covered by the design document
 - Reasoning is that covering every class would be too tedious to complete by due date
 - Next meeting set for 5 November

5 November 2020:

- Team Meeting rescheduled to 10 November due to my last minute work travel

10 November 2020:

- Team Meeting:
 - Members shared what they believed to be primary classes
 - Came to a consensus on a concise list of classes to cover in the design document
 - Griffin found a design document reference, which he shared with the group
 - Design Document Assignments:
 - Me - TBD (due to being on travel for work)
 - Abhishek - Architectural Design
 - Griffin - Detailed Design
 - Next meeting set for 19 November

19 November 2020:

- Team Meeting:
 - Members shared their ideas on locations of code reduction could be easily accomplished
 - Griffin summarized the classes that would be covered by the design document
 - Overall document is still being worked
 - Next meeting set for November 24

24 November 2020:

- Team Meeting:
 - Reviewed current draft of the design document
 - Action Items:

- Me - Finalize last sections of design document
 - Abhishek - Continue developing GUI
 - Griffin - Start Implementation Document
- Discussion on plug-in implementation
- Next meeting set for 30 November

30 November 2020:

- Team Meeting:
 - Discussed notification from Dr. Mateti to focus on Team 1's specification document
 - Implementation document has been started
 - Dependent on completion of implementation
 - Testing document started
 - Dependent on completion of plug-in and code reduction to complete
 - Action Items:
 - Me - Plugin
 - Abhishek - GUI
 - Griffin - Code Reduction
 - Next meeting set for 2 December

2 December 2020:

- Team Meeting:
 - GUI implementation provided to the team
 - Functional, but known bugs still need fixed
 - Code reduction still in progress
 - Far enough along to start implementation document
 - Griffin able to implement predefined color plugin
 - Testing document draft provided
 - Few sections still need to be completed, but outline finished
 - Need a new test file for a stress test - Abhishek will provide
 - Action Items:
 - Me - Complete test document with test inputs
 - Abhishek - Provide stress test file and start final document
 - Griffin - Clean up the GUI
 - Next meeting set for 3 December

3 December 2020:

- Team Meeting:
 - GUI, Code Reduction, Implementation and Test documents complete
 - Action Items:
 - Me - Write conclusion and format the final document
 - Abhishek - Write introduction and finish compiling documents
 - All - Post journals and reflections
 - Final meeting set for 4 December

4 December 2020:

- Team Meeting:
 - Compiled all members journals
 - Last review of document
 - Lance - reformat table of contents before submission
 - Abhishek - Post final document on GitHub, Pilot discussion board, and dropbox
 - Griffin - Posted source code to official GitHub repo

10.2 Abhishek Pandya

Friday 8/28:

Responded to Lancius (Lance) Matthieu post to join the team in pilot discussion.

Monday 8/31:

Join the discord channel and meet everyone in the team.

Thursday 9/10:

Pull ditaa project from github and tried to build and run the program.

Installed ditaa.jar from the link given in the discussion forum.

Monday 9/14:

Joined group github repository, created by Daniel.

Got access of google drive folder

Tuesday 9/15:

Group meeting 2-3pm.

Discussed different parts for the requirements document.

Discussed who will complete each part.

Lance M - 1) Introduction

Abhishek - 2) Functional 3) non-functional Objectives

Daniel - 4) Context Model

Griffin - 5) Use Case Model

Wrote Functional and Non-Functional Objective for the requirement document

Wednesday 9/16:

Group meeting 5-6 pm.

Discussed any changes for specific parts.

Discussed a GUI for the project.

Discussed the additional features.

Thursday 9/17:

Group meeting 5-6 pm.

Discussed final changes.

Discussed who will submit what.

Tuesday 9/29:

Discussed setting up group meeting for specification document (done by Lance)

Homework for meeting look over the specification requirements and what is needed

Wednesday 9/30:

Group meeting 5-6 pm.

- Discussed the requirements posted on the webpage

- Discussed finding a template for the specification document

- Discussed next meeting time

- Set to Monday 10/5

Monday 10/5:

Group meeting 5-6 pm.

- Discussed the templates found

- Discussed who will completing each task

- Abhishek - Intro and expectations

- Lance - Specification

- Griffin - Assertions

- Discussed next meeting time

- Set to Thursday 10/8

Tuesday 10/6:

Completed Introduction part.

Wednesday 10/7:

Completed Expectation of Project

Thursday 10/8:

Created Specification Document in google drive shared folder

Group meeting 5-6 pm.

- Discussed parts that were done

- Discussed what needed to be added

- Discussed confusion about the specification portion and how to go about completing it.

- Discussed next meeting

- Set to Friday 10/9.

Edited my sections as per feedback from teammates

Friday 10/9:

Group meeting 5-6 pm.

- Reviewed specification portion.

- Discussed what was not completed in each portion (parts that needed ideas from others)

- Discussed measuring techniques for code reduction

Discussed remaining things needed to be completed
Lance - Code reduction portion for specifications
Griffin - Conclusion
Discussion for final meeting
Set for Monday 10/12:

Sunday 10/11:
Conclusion finished and posted in google doc

Monday 10/12:
Group meeting 5-6 pm.
Reviewed final document
Added final changes to the document.
Meeting set for 10/16 to go over design document
Uploaded the document in GitHub

Friday 10/16:
Group meeting 5-6 pm.
Looked over design expectations
Focus on getting a better grasp on how the code works
Obtain a reference for the design document report
Next meeting was set for 10/23 to review what we looked over.

Friday 10/23:
Group meeting 5-6 pm.
Not much was done, due to midterms
Most group members reviewed the code slightly, but didn't have much time to fully explore.
Design document reference had not been found, little resources.
Decided to focus design document on enhancements - GUI and plugin
Abhishek set to start working on GUI to get an idea of what the design will look like.
Goal for next meetings for rest is to find a reference document and come up with ideas to get started.
Next meeting set for 10/29 to go over progress

Monday 10/26:
Started creating GUI. So that, I can get an idea of how the GUI will look like

Thursday 10/29:
Group meeting 5-6 pm.
GUI has been started.
Code has a better understanding.
Next meeting set to 10/30 to go over progress

Friday 10/30:
Got the screenshot of GUI and posted it in the document.

Group meeting 5-6pm.

GUI screenshot posted, in good development.

Design document reference to be found over weekend - Griffin.

Next meeting is set to 11/2.

Monday 11/2:

Group meeting 5-6pm.

Lance posted ideas for the design document along with SLOC from current ditaa.

Ideas started to get major classes to provide information on the project in its current state.

Next meeting set to 11/5 - this was pushed back to 11/10 due to work travelling.

Thursday 11/10:

Group meeting 5-6pm.

Posts from group members on major classes from ditaa.

Input on each post to condense the list to be concise.

Design document reference found.

Parts separated:

Abhishek - Architectural Design

Griffin - Detailed Design

Lance - Currently out for work, assigned later

Next meeting is set for 11/19.

Thursday 11/19:

Group meeting 5-6pm.

Occurrences where code could be reduced was found.

Classes were summarized

Design document is still a work in progress.

Next meeting set to 11/24

Tuesday 11/24:

Still working on GUI implementation.

Group meeting 5-6 pm.

Design document is almost complete, just need touch ups.

Code reduction started.

GUI is still a work in progress.

Ideas of implementing the plugin.

Assignments:

Lance - Finish up Design Document

Griffin - Start on implementation Document

Abhishek - Work on GUI

Next meeting is set for 11/30.

Monday 11/30:

Group meeting 5-6 pm.

Professor commented to focus on team 1's specifications if time allowed. Decided to focus on our design as we allocated time to other areas.

Implementation document started and is a work in progress until implementation is complete.

Testing document started and is a work in progress until implementation is complete.

Assignments:

Griffin - Code reduction

Lance - Plugin

Abhishek - GUI

Next meeting is set to 12/2.

Wednesday 12/2:

Made some test cases for the testing document.

One Stress test case is implemented by me.

Group meeting 5-6 pm.

GUI implementation has been provided, has a few known issues that need to be worked out, but is functional.

Code reduction is in progress, but enough to report on for the time being.

Plugin has been implemented, new human color codes added and usable.

Implementation work in progress posted and ready for revisions once code reduction is finalized.

Testing document still a work in progress

Assignments:

Griffin - Clean up GUI

Abhishek - Provide stress tests for ditaa and work on final documents.

Lance - Work on testing documents with given test documents.

Next meeting is set to 11/3.

Thursday 12/3:

Group meeting 5-6 pm.

GUI has been finalized.

Code Reduction finalized.

Implementation document finalized.

Testing document finalized.

Assignments:

Lance, Abhishek - Work on final document.

Griffin - Provide results of working GUI and ditaa code.

All - Post journals and reflections.

Next meeting is set to 12/4.

Friday 12/4:

Wrote introduction of the final document and reflection.

Group meeting 5-6 pm.

Finalised the final document.

Posted implementation and testing document in pilot discussion.

10.3 Griffin Mosley

Monday 9/14:

Installed ditaa.jar from link given in discussion.

Pull ditaa from github.

Ran jar file with tests.

Created google drive folder for group

Tuesday 9/15:

Group meeting 2-3pm.

Discussed different parts for the requirements document.

Discussed who will complete each part.

Lance M - 1) Introduction

Abhishek - 2) Functional 3) non-functional Objectives

Daniel - 4) Context Model

Griffin - 5) Use Case Model

Wednesday 9/16:

Group meeting 5-6 pm.

Discussed any changes for specific parts.

Discussed a GUI for the project.

Discussed the additional features.

Thursday 9/17:

Group meeting 5-6 pm.

Discussed final changes.

Discussed who will submit what.

Tuesday 9/29:

Discussed setting up group meeting for specification document (done by Lance)

Homework for meeting look over the specification requirements and what is needed

Wednesday 9/30:

Group meeting 5-6 pm.

Discussed the requirements posted on the webpage

Discussed finding a template for the specification document

Discussed next meeting time

Set to Monday 10/5

Monday 10/5:

Group meeting 5-6 pm.

Discussed the templates found

Discussed who will completing each task

Abhishek - Intro and expectations

Lance - Specification
Griffin - Assertions
Discussed next meeting time
Set to Thursday 10/8

Wednesday 10/7:
Finished assertions.

Thursday 10/8:
Posted assertions in google doc for review
Group meeting 5-6 pm.
Discussed parts that were done
Discussed what needed to be added
Discussed confusion about the specification portion and how to go about completing it.
Discussed next meeting
Set to Friday 10/9.

Friday 10/9:
Group meeting 5-6 pm.
Reviewed specification portion.
Discussed what was not completed in each portion (parts that needed ideas from others)
Discussed measuring techniques for code reduction
Discussed remaining things needed to be completed
Lance - Code reduction portion for specifications
Griffin - Conclusion
Discussion for final meeting
Set for Monday 10/12:

Sunday 10/11:
Conclusion finished and posted in google doc

Monday 10/12:
Group meeting 5-6 pm.
Reviewed final document
Added final changes to the document.
Meeting set for 10/16 to go over design document

Friday 10/16:
Group meeting 5-6 pm.
Looked over design expectations
Focus on getting a better grasp on how the code works
Obtain a reference for the design document report
Next meeting was set for 10/23 to review what we looked over.

Friday 10/23:

Group meeting 5-6 pm.

Not much was done, due to midterms

Most group members reviewed the code slightly, but didn't have much time to fully explore.

Design document reference had not been found, little resources.

Decided to focus design document on enhancements - GUI and plugin

Abhishek set to start working on GUI to get an idea of what the design will look like.

Goal for next meetings for rest is to find a reference document and come up with ideas to get started.

Next meeting set for 10/29 to go over progress

Thursday 10/29:

Group meeting 5-6 pm.

GUI has been started.

Code has a better understanding.

Next meeting set to 10/30 to go over progress

Friday 10/30:

Group meeting 5-6pm.

GUI screenshot posted, in good development.

Design document reference to be found over weekend - Griffin.

Next meeting is set to 11/2.

Monday 11/2:

Group meeting 5-6pm.

Lance posted ideas for the design document along with SLOC from current ditaa.

Ideas started to get major classes to provide information on the project in its current state.

Next meeting set to 11/5 - this was pushed back to 11/10 due to work travelling.

Thursday 11/10:

Group meeting 5-6pm.

Posts from group members on major classes from ditaa.

Input on each post to condense the list to be concise.

Design document reference found.

Parts separated:

Abhishek - Architectural Design

Griffin - Detailed Design

Lance - Currently out for work, assigned later

Next meeting is set for 11/19.

Thursday 11/19:

Group meeting 5-6pm.

Occurrences where code could be reduced was found.

Classes were summarized

Design document is still a work in progress.

Next meeting set to 11/24

Tuesday 11/24:

Group meeting 5-6 pm.

Design document is almost complete, just need touch ups.

Code reduction started.

GUI is still a work in progress.

Ideas of implementing the plugin.

Assignments:

Lance - Finish up Design Document

Griffin - Start on implementation Document

Abhishek - Work on GUI

Next meeting is set for 11/30.

Monday 11/30:

Group meeting 5-6 pm.

Professor commented to focus on team 1's specifications if time allowed. Decided to focus on our design as we allocated time to other areas.

Implementation document started and is a work in progress until implementation is complete.

Testing document started and is a work in progress until implementation is complete.

Assignments:

Griffin - Code reduction

Lance - Plugin

Abhishek - GUI

Next meeting is set to 12/2.

Wednesday 12/2:

Group meeting 5-6 pm.

GUI implementation has been provided, has a few known issues that need to be worked out, but is functional.

Code reduction is in progress, but enough to report on for the time being.

Plugin has been implemented, new human color codes added and usable.

Implementation work in progress posted and ready for revisions once code reduction is finalized.

Testing document still a work in progress

Assignments:

Griffin - Clean up GUI

Abhishek - Provide stress tests for dita and work on final documents.

Lance - Work on testing documents with given test documents.

Next meeting is set to 11/3.

Thursday 12/3:

Group meeting 5-6 pm.

GUI has been finalized.

Code Reduction finalized.

Implementation document finalized.

Testing document finalized.

Assignments:

Lance, Abhishek - Work on final document.

Griffin - Provide results of working GUI and ditaa code.

All - Post journals and reflections.

Next meeting is set to 12/4.

Friday 12/4:

Group Meeting 5-6pm.

Final Document Finalized

Implementation included

Testing included

Code uploaded

Video representation of code uploaded

Final Document to be posted by Abhishek