CS 7140 - Advanced Software Engineering

Fall 2020

Project Specification Documentation

Team Members:

Lancius (Lance) Matthieu

Abhishek Pandya

Griffin Mosley

Daniel Ketterer

WRIGHT STATE UNIVERSITY

1. Introduction

The main application of this software would be to be able to convert ascii art drawings into proper bitmap graphics. This software is called Diagrams Through Ascii Art (DITAA). The purpose of this document is to describe the specifications of the DITAA program. Ascii art are drawings that are constructed with characters that resemble lines, such as |, /, -, and). Without this software users must download some external graphic design software such as Photoshop, Inkscape, etc., in order to render graphic diagrams.

2. Expectations of the Project

In this section we will summarize the requirements of the project. In the first section we present the functional requirements, and in the second one the non-functional ones.

2.1 Functional Requirements

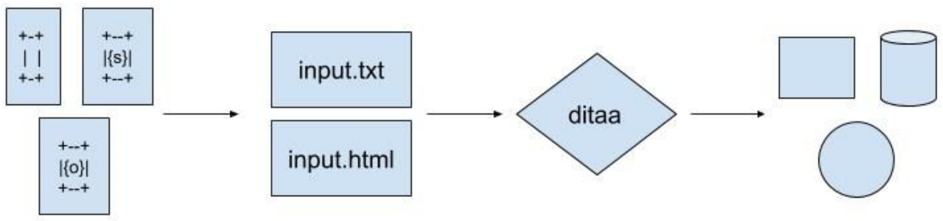
In this section we will describe the functionality of the software.

2.1.1 Convert ascii art to diagram

The user should be able to draw the diagram with the help of ascii art where the user needs to draw shapes using the special characters like | / - + *. The user will have documentation of expected inputs that the user should enter. The user should also be able to get help with the -help command that will have the information on how to use the program from the command line. The user must draw ascii art in a text file in order to get the output as a diagram.

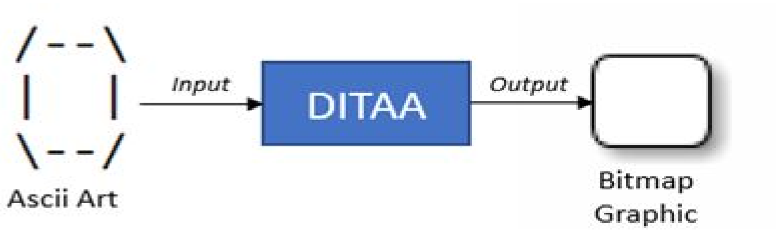
2.1.2 Build the structure of the program

The structure of the program is followed by some steps. First step is that users need to draw diagrams with special characters in a text file. Next step is the user will run the program with the text file as an input file in the command line argument and also give output file name which will contain the final output. Now, the program will take that ascii art and convert it into a diagram. Output file will contain a diagram from the user input file.



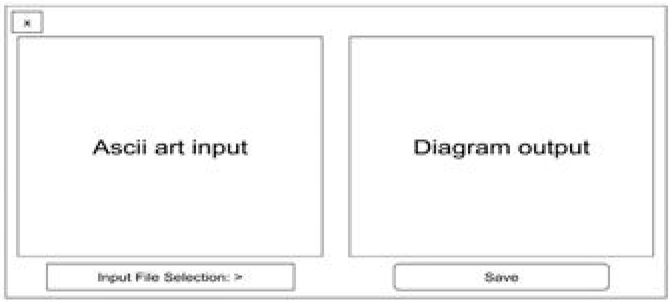
2.1.3 Convert into proper bitmap graphics

The program will have functionality to convert ascii art into bitmap graphic. All the special characters will be assigned to draw different shapes in the bitmap graphic. Program will have power to read the input file where the ascii art is present and convert those characters in the diagram. For example



2.1.4 GUI for the program

This feature will be added later on after implementing the main requirements. The GUI (Graphical User Interface) will load an input file which has ascii art and display the diagram in the output screen. Users can also save the output file if they want a copy of the diagram.



2.2 Non-Functional Requirements

2.2.1 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.

2.2.2 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.

2.2.3 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.

2.2.4 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.

2.2.5 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.

3. DITAA Specification

3.1 Syntax of Input:

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

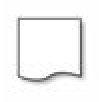
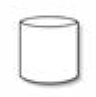
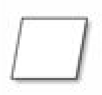
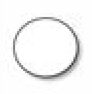
corner ::= [“+”, ”/”, “*”]
tbOutline ::= [“-“, ”=”, “*”]
sideOutline ::= [“|”, ”:”, “*”]
arrow ::= “<” | “>”, tbOutline | sideOutline | corner, “<” | “>”
top ::= <corner> <tbOutline> <corner>
bottom ::= <corner> <tbOutline> <corner>
text ::= [letters, numbers, symbols]
bullet ::= < o > <text>
tag ::= < { > [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

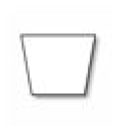
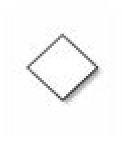
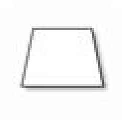
3.2 Interpretation of Input for Rendering:

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	🟥 RED
GRE	🟩 GRE
BLK	⬛ BLK
BLU	🟦 BLU
PNK	PNK
YEL	🟨 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

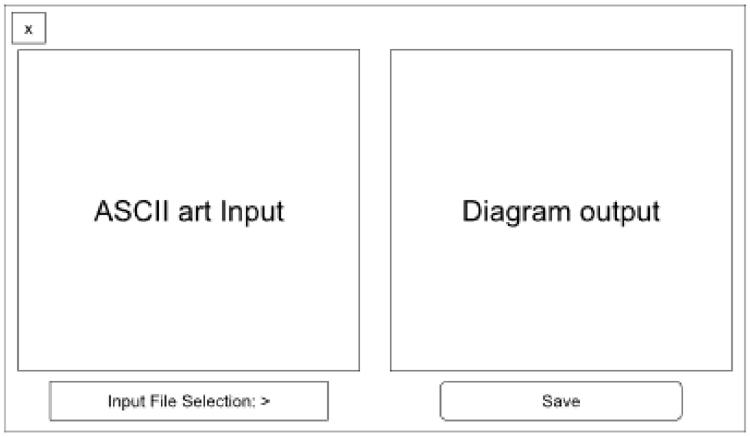


Figure 1: GUI: Blank Presentation to User

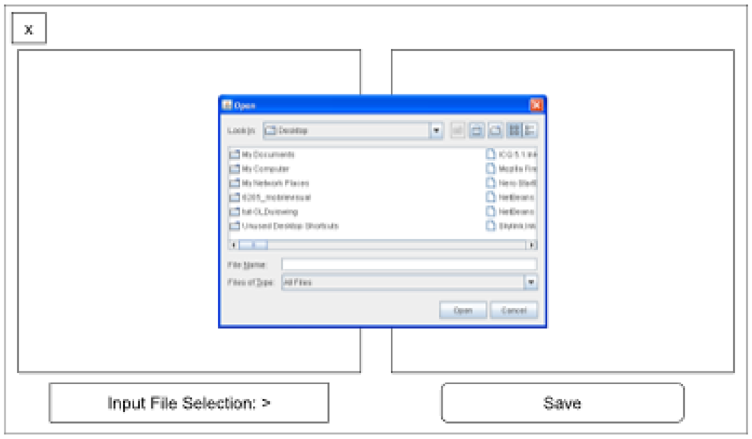
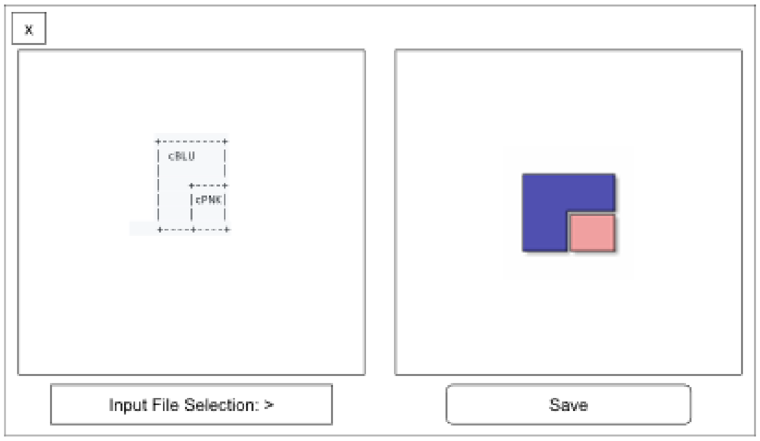


Figure 2: GUI: Select Input



```
+---------+
| cBLU    |
|         |
|    +----+
|    |cPNK|
|    |    |
+----+----+
```

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

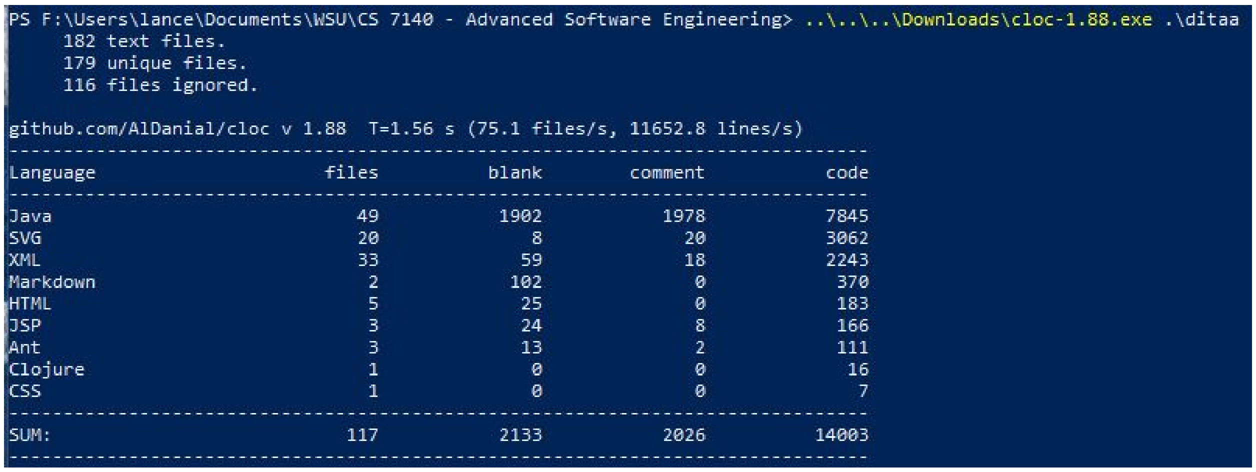
The goal of the code reduction specification is to reduce the overall number of lines of code (LOC) by at least 5%.

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

4. 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.

5. Conclusion

In this document we went into more specifics on the requirements, broken down into functional and nonfunctional requirements. We have included the features that must remain inside the project, along with detailed the changes we will make with a generalized description of what the final results will look like. We have specified our end goals being: creating a GUI, adding a few features and obtaining a 5% code reduction. The final area touched on were the assertions for the project, giving the entry and exit assertions for the code.

6. References

1. https://github.com/stathissideris/ditaa
2. https://www.guru99.com/conformance-testing.html
3. http://biorob2.epfl.ch/pages/studproj/birg66475/Project%20specifications.pdf
4. https://cecs.wright.edu/~pmateti/Courses/7140/Lectures/Examples/TicTacToe-JavaFX-UnRedo/tictactoe-spec.html
5. https://cecs.wright.edu/~pmateti/Courses/7140/Lectures/Examples/Tabulate-Eqns/
6. https://cecs.wright.edu/~pmateti/Courses/7140/Lectures/Examples/Condense-Num-Ranges/
7. https://github.com/AlDanial/cloc
8. https://en.wikipedia.org/wiki/Software_quality

7. Team Journal

29 September 2020:

- Discussion of first meeting for Specification Document, set up by Lance
    - Set to 30 September 2020 5-6 pm
- Goal set to discuss what we are going to do/write in specification document

30 September 2020:

- First meeting (Danial was not there(sick))
- Discussed how to write specification document
- Each team member will try to find some example on specification document
- Discussion of second meeting, set up by Lance
    - Set to 5 October 2020 5-6 pm
    - Goals
        - Finalize the structure of the document
        - Each team member will do what task

5 October 2020:

- Second meeting (Danial was not there(sick))
    - How the specification document will be organized
    - Who will be completing each task
        - Abhishek – 1) Introduction 2) Expectation of the project
        - Lance – 3) DITAA Specification
        - Griffin – 4) Assertions
- Discussion of third meeting, set up by Lance
    - Set to 8 October 2020 5-6 pm
    - Goals
        - Review each portion
        - Feedback on each portion by other teammate

8 October 2020:

- Third meeting (Danial was not there(sick))
    - Abhishek and Griffin has finished their task what was given
    - Lance discussed about his task (What to add and what not)
    - Each team member shared feedback on other teammate section
    - Task to be completed before next meeting
        - Abhishek – Some changes in his section (editing)
        - Lance M – 3) DITAA Specification
        - Griffin – Some changes in his section (editing)

- Discussion of forth meeting, set up by Lance
    - Set to 9 October 2020 5-6 pm
    - Goals
        - Review each portion
        - Feedback on each portion by other teammate

9 October 2020:

- Forth meeting (Danial was not there(sick))
    - All sections completed
    - Reviewed each section
    - Discussed about how we can measure code reduction
    - Task to be completed before next meeting
        - Lance M – sub section of DITAA Specification: 3.5) Code Reduction Specification
        - Griffin – Volunteered for writing 5) Conclusion
- Discussion of forth meeting, set up by Lance
    - Set to 12 October 2020 5-6 pm
    - Goals
        - Final changes
        - Review edits and things added on
        - Determine who will submit document

12 October 2020:

- Fifth meeting (Danial was not there(sick))
    - Reviewed final addition
    - Griffin added quality section in the non-functional requirements
    - Finalized document
    - Added journals
    - Uploaded the document in GitHub

8. Personal Journals

8.1 Abhishek Journal

30 September 2020:

- Group meeting 5-6 pm, organized by Lance
  - Discussed how we can write specification document
  - Decided that everyone will gather some examples of specification document from internet

5 October 2020:

- Group meeting 5-6 pm, organized by Lance
  - Discussed different parts for the specification document
  - Discussed who will complete each part,
    - Abhishek – 1) Introduction 2) Expectation of the Project
    - Lance M – 3) DITAA Specification
    - Griffin – 4) Assertions

6 October 2020:

- Completed Introduction

7 October 2020:

- Completed Expectation of Project

8 October 2020:

- Created Specification Document in google drive shared folder
- Group meeting 5-6 pm, organized by Lance
  - Got feedback from team members about my sections
  - Edited my sections as per feedback from teammates

9 October 2020:

- Group meeting 5-6 pm, organized by Lance
  - Discussed about how we can measure code reduction

12 October 2020

- Group meeting 5-6 pm, organized by Lance
  - Reviewed final changes in the document
  - Finalized the document
- Uploaded the document in GitHub

8.2 Griffin Journal

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 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.
    - Included personal journals

8.3 Lance Journal

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 to references
  - Abhishek agreed to finalize team journal and upload final specifications document to GitHub