MASTER TEST PLAN

for

Code Visualization

Version 1.0

Prepared by : **Curtice Gough**
Joshua Hartzfeld
Catherine DiResta

Submitted to : **Dr. Philip Chan**
Instructor

September 29, 2023

Contents

1 Introduction **3**

1.1 Scope . . . 3

1.2 System overview and key features . . . 3

2 Specific Tests **4**

2.1 Interfaces . . . 4

2.1.1 Graphical User Interface . . . 4

2.2 Functional requirements . . . 5

2.2.1 Source code input . . . 5

2.2.2 Traceprinter subprocess . . . 5

2.2.3 JSON trace parsing . . . 5

2.2.4 Structure visualization . . . 6

2.2.5 Animations . . . 6

2.2.6 Re-typing/re-naming . . . 6

2.2.7 Line-by-line-execution . . . 6

1 Introduction

1.1 Scope

This document defines the testing requirements and procedures for the software developed in association with the project titled "Code Visualization". Each of the test requirements described here will apply to what may be considered a finished product. For example, a test requirement defining the expected efficiency of a given feature may not apply if that feature has not yet been implemented.

1.2 System overview and key features

The purpose of this software is to provide a visual learning aid for beginner-level students of the Java programming language. Users will be able to submit simple Java code for analysis, then visually examine the execution of said code. Data structures will be represented with standardized diagrams, and the movement of data between those structures will be shown using animations in the graphical user interface.

At a high level, the key features of this software are as follows:

- Interactive GUI
  This program will feature an interactive GUI. The GUI includes a display pane used for showing diagrams to represent various data structures. In addition to showing the data structures themselves, there will also be animations for demonstrating the movement of data

- Dynamic code analysis
  Users will be able to step through source code during program execution line-by-line in a fashion reminiscent of popular debuggers such as GDB and x64dbg. This allows them to keep track of where they are in the program flow.

- User intervention
  The data structure visualization engine may occasionally make incorrect assumptions about which data structure is being represented. To solve this problem, we take inspiration from popular decompilers such as Binary Ninja and Ghidra to allow the user to rename/retype visual elements during analysis.

This information can also be found in the initial Project Plan.

2 Specific Tests

2.1 Interfaces

2.1.1 Graphical User Interface

2.1.1.1 Data Structure View

To test that the main section can correctly contain the visual representations of the source code provided, we will test each data structure through multiple code samples. This window should be able to contain the full data structure without any parts going off the screen or not being fully rendered onto the screen.

The diagram should be correctly labeled upon visualization. To test that this is done correctly, an expected output diagram will be created and the generated output will be compared. If it does not fully match the expected output the test will be considered a failure. This will be done multiple times for each data structure supported.

To test that the animations correctly show the data that is being moved or modified an expected output will be created. A trace of the code will be done by hand. All values will be compared as the system goes through each line of source code. If it does not fully match the expected output the test will be considered a failure. Multiple tests will be done for each data structure supported.

The user will be able to re-name the structure regardless of the data structure being used. This will be tested multiple times with each label for each supported data structure. If the user is unable to change the name of any of the labels the test will be considered a failure. A google form will be used to ask questions regarding any difficulties they had while attempting to re-name their structures.

The user will be able to move the diagrams around in the main structure view by clicking and dragging it to where they would be needed. If the user is unable to move the structure around then the test will be considered a failure. A Google form will be used to ask questions regarding how easily they could move the diagram and any difficulties the user encountered while attempting.

2.1.1.2 Source View

The length of the programs will be tested to ensure a reasonable limit for source code size will be implemented for the user to have the best experience. Multiple test cases will be created of varying lengths. If any are unable to compile and be properly parsed then a limit will be put into place.

The user will be able to scroll through the source code inputted into the program regardless of what part is being visualized at the time.

The program will highlight the line of code currently being visualized. Multiple tests using the same code created will be done. If the program doesn't keep up on highlighting the current line, the test will be considered a failure.

2.1.1.3 Structures List

To test that the list of data structures in the source code are being listed correctly multiple tests will be conducted. An expected output will be created for each test. If the program fails to list all data structures shown and update it as the program goes through the source code then the test will be considered a failure.

The user will be able to rename the data structures in the list. If the user is unable to change the name of any structures then the test will be considered a failure. A Google form will be used to ask the user about the ease of changing the name of the data structures.

2.2 Functional requirements

2.2.1 Source code input

The purpose of this test is simply to ensure that users are able to paste/type Java code into the Source View. The test itself involves inputting text in two ways: copy-paste; typing. Success is measured by two factors:

- Can text be entered in the input field?
- Does the text remain in the input field after clicking away/unfocusing?

2.2.2 Traceprinter subprocess

The purpose of this test is to verify whether the software successfully launches an instance of Traceprinter with the correct arguments, sends source code as input, and receives JSON output. The success of this test is dependent on the success of the test outlined in "Source code input".

Valid Java code will be entered in Source View, and the tester will click the appropriate button to begin the code trace. If valid JSON data is received by the software, we consider the test a success.

2.2.3 JSON trace parsing

The purpose of this test is to verify the correct parsing of JSON data received from the Traceprinter subprocess. The success of this test is dependent on the success of the test outlined in "Traceprinter subprocess".

To conduct this test, a simple Java program will be constructed of no more than 15 lines. If a valid Python dictionary is constructed with values that correspond with the JSON output of Traceprinter, we consider the test a success.

2.2.4 Structure visualization

The purpose of this test is to ensure that data structures are correctly detected and visualized in the GUI.

This testing procedure involves creating a simple Java program with one or more instances of supported data structures (defined in SRS), and stepping through the trace. Each data structure type has a pre-defined diagram associated with it, or a generic class diagram for unsupported types. If each object is represented using the correct diagram at each stage of the program, we consider the test a success.

2.2.5 Animations

The purpose of this test is to visually confirm that the movement of data is accurately represented in the Data Structures View during code tracing.

To perform this test, a simple Java program will be written involving two supported data structures. Data will be moved from one structure to another, as well as among the same structure. As the tester steps through the execution of the program, if each operation is correctly represented by the appropriate animation, the test is considered a success.

2.2.6 Re-typing/re-naming

The purpose of this test is to verify that users will have the ability to redefine a class/object diagram and/or rename it.

A simple Java program will be written involving a custom implementation of a tree rather than the automatically imported Tree class provided by the software. The tester will first re-name the object. Then, the tester will re-define the object as a Tree structure. If the structure is successfully renamed and visualized using the Tree diagram, we consider the test a success.

2.2.7 Line-by-line-execution

The purpose of this test is to verify the accurate representation of each program state.

To test this function, any valid Java program can be submitted for tracing. As the tester steps through the program, if all data values are accurate at each step, we consider the test a success.