Software Test Description For Logic Gate Simulator

December 10th, 2018

Jenna Westfall, Aby Prasad, Robert Rose, Yael Weiss, Bright Sun

Table of Contents:

1. Scope.

1.1 Identification

This document describes the purpose and design of a browser-based circuit simulation application. This application allows users to build and test circuits inside a web browser, mitigating the need to install specialized software for the same purpose. It will also allow for easy sharing of circuit diagrams via URLs, as well as saving and loading work. The goal of this program is to allow for a more user-friendly experience than existing logic gate editors.

1.2 System overview.

Logic Gate Simulator will be a web version of the logic gate learning tool Logisim. Like Logisim, the app will simulate basic circuitry and logic gates. The end goal is to make a simulation environment that will be easily accessible and user-friendly to students. A top priority will be to improve upon Logisim's design by making a logic gate simulator that will be more user-friendly and more accessible (easy to access without download/setup, and be able to share work with a web link).

1.3 Document overview.

This document provides a checklist for all tests associated with the features of this product. Section 2 provides all referenced documents used in the following sections. Section 3 discusses the necessary safety precautions as well as any other pre-test preparations. Section 4 goes into detail on each specific test. Section 5 displays a matrix connecting the each test case to its corresponding CSCI.

1.4 Definitions

Throughout this documents the following terms are used:

- **Logic Gates:** Refers to an elementary building block of a digital circuit. Logic gates have inputs and one output. The inputs and outputs are binary values that indicated by 1 and 0. Different types of logic gates output different results.
  Ex: AND gate, OR gate, NOT gate, XOR gate, NAND gate, NOR gate, XNOR

- **Circuits:** A collective of logic gates that use one and others as inputs. They produce a single binary output.

- **Diagram:** The entire display consisting all a logic gates and circuits constructed.

- **GUI:** Stands Graphical User Interface

- **Logisim:** A downloadable program used to simulate logic gates and circuits.

- **Wires:** Used to connect the output of one logic gate to the input of another gate thereby creating a circuit.
- **Simulate:** To have the circuits constructed display its output.

- **Save:** Stores the constructed circuit as a file

- **Load:** Retrieves a constructed circuit as a file

- **Share:** Saves the constructed circuit to a public database that allows it to accessed by URL.

2. Referenced documents.

The following documents were used in the development of this SSDD and are referenced in the proceeding sections.

2.1 Government Documents

MILSTD 498 SSDD Template: Provided template for which this document was based on. Can be accessed via the following URL: https://blackboard.umbc.edu/bbcswebdav/pid-3248655-dt-content-rid-27085504_1/courses/CMSC447_8367_FA2018/STD.html .

2.2 Non-Government Documents

Software Requirements Specification (SRS): Document listing and explaining the agreed upon requirements by the client and the developers. Can be accessed via the following URL: https://github.com/RobRoseKnows/logisim-browser .

System/Subsystem Design Description(SSDD): Document used to establish concrete design decision for the product. Can be accessed via the following URL:

https://github.com/RobRoseKnows/logisim-browser .

3. Test preparations.

3.1 PT-01 Firefox.

This test preparation will open Logic Gate Editor via the Firefox web browser. The program shall use a javascript-based framework for the user interface. To be considered having full functionality the test will display the main interface, and have a drag-and-drop features for creating the logic gates as stated in R10 of the SRS document.

3.1.1 Hardware preparation.

The hardware requirements will need to support the needed software detailed in the following section (i.e: in order to run Firefox 62, a system requires a processor of Pentium 4 or later, 512 MB of RAM (32-bit system)/ 2GB RAM (64-bit system)) . A system that is able to run the following software will in turn satisfy the needed hardware requirements.

3.1.2 Software preparation:

Program loading procedure:

1. The tester will open an instance of Firefox (62 or later).
2. The tester will enter the following URL into their search bar: “BLANK”
3. The tester will ensure that the main interface has loaded.
4. The tester will ensure that the drag-and drop functionality is working by dragging an AND gate from the left-hand drawer to the edit area. A successful output will be if the AND gate stays in the edit area.

3.1.3 Other pre test preparations.

In addition to the preceding preparations, the following requirements must also be established:

- Prior to opening the software, the system must have adequate space (100 MBS) to store the JSON save files for the save test outlined in section 4.1.
- The test will require the user to be running a computer with windows 8.1 and up (requirement R13 of the SRS - a link to the document is given in section 2).
- The test will require an internet connection to be established (requirement R14 of the SRS).

3.2 - PT-02 Microsoft Edge.

This test preparation will open Logic Gate Editor via the Microsoft Edge web browser.The program shall use a javascript-based framework for the user interface. To be considered having full functionality the test will display the main interface, and have a drag-and-drop features for creating the logic gates as stated in R10 of the SRS document.

3.2.1 Hardware preparation.

The hardware requirements will need to support the needed software detailed in the following section (i.e: in order to run Microsoft Edge, a system requires a processor 1 gigahertz (GHz) or faster, 1GB of RAM (32-bit system)/ 2GB RAM (64-bit system)) . A system that is able to run the following software will in turn satisfy the needed hardware requirements.

3.2.2 Software preparation:

Program loading procedure:

1. The tester will open an instance of Microsoft Edge (44.0 or later).
2. The tester will enter the following URL into their search bar: “BLANK”
3. The tester will ensure that the main interface has loaded.
4. The tester will ensure that the drag-and drop functionality is working by dragging an AND gate from the left-hand drawer to the edit area. A successful output will be if the AND gate stays in the edit area.

3.2.3 Other pre-test preparations.

In addition to the preceding preparations, the following requirements must also be established:

- Prior to opening the software, the system must have adequate space (100 MBs) to store the JSON save files for the save test outlined in section 4.1.
- The test will require the user to be running a computer with Windows 10 and up (requirement R13 of the SRS - a link to the document is given in section 2).
- The test will require an internet connection to be established (requirement R14 of the SRS).

3.3 - PT-03 - Chrome

This test preparation will open Logic Gate Editor via the Chrome web browser. The program shall use a javascript-based framework for the user interface. To be considered having full functionality the test will display the main interface, and have a drag-and-drop features for creating the logic gates as stated in R10 of the SRS document.

3.3.1 Hardware preparation.

The hardware requirements will need to support the needed software detailed in the following section (i.e: in order to run chrome 70, a system requires a processor of Pentium 4 or later, that's SSE2 capable) . A system that is able to run the following software will in turn satisfy the needed hardware requirements.

3.3.2 Software preparation:

Program loading procedure:

1. The tester will open an instance of Chrome (70.0 or later).
2. The tester will enter the following URL into their search bar: "BLANK"
3. The tester will ensure that the main interface has loaded.
4. The tester will ensure that the drag-and drop functionality is working by dragging an AND gate from the left-hand drawer to the edit area. A successful output will be if the AND gate stays in the edit area.

3.3.3 Other pre-test preparations.

In addition to the preceding preparations, the following requirements must also be established:

- Prior to opening the software, the system must have adequate space (100 MBS) to store the JSON save files for the save test outlined in section 4.1.
- The test will require the user to be running a computer with windows 8.1 and up (requirement R13 of the SRS - a link to the document is given in section 2).
- The test will require an internet connection to be established (requirement R14 of the SRS).

4. Test descriptions.

4.1 Test-01: Save

This test tests that a save gui is correctly pops up and saves the file when a user presses the "save" button.

4.1.1 Test Case 01: Empty File.

This test case tests whether the program properly behaves when a user attempts to store an empty diagram. Unlike saving to a server, saving a blank diagram to the user's local desktop is allowed.

4.1.1.1 Requirements addressed.

Addresses R4 (*The user shall be able to save gate files*) and R7 (*When a user hits the "save" button, the program will open a GUI that will allow exporting of data to a local desktop*).

4.1.1.2 Prerequisite conditions.

All of prerequisite mentioned in section 3 will be required for this test.

4.1.1.3 Test inputs.

The input for this test will be a blank diagram, identical to that which would exist on a fresh install of the program. The file shall be saved to the tester's desktop.

4.1.1.4 Expected test results.

The expected output is a GUI allowing the user to select a location where the file will be saved. The expected output of the GUI is a JSON file matching the circuit schema without any contents.

4.1.1.5 Criteria for evaluating results.

This test falls in the demonstration category of testing and is therefore determined by implementing the code and seeing whether it succeeds in executing everything it intended or fails. Since the user is saving data in memory to disk, measuring load times is unnecessary as it will largely vary by system setup.

4.1.1.6 Test procedure.

Step 1. The tester will start one of the supported browsers.
Step 2. The tester will navigate to the website.
Step 3. The tester shall create a new blank diagram on the website.
Step 4. The tester will click to Save the diagram.
Step 5. The tester will mark whether the GUI opened or not.
Step 6. The tester will save the test file to the desktop.
Step 7. The tester will open the file in a text editor and ensure the schema is in-tact.

4.1.1.7 Assumptions and constraints.

Basic assumptions of having a functional computer with no issues existing outside the software. This includes no external hardware or software bugs. It is assumed that the test will be run on a Windows 10 or later system.

4.1.2 Test Case 02: File with Contents

This test case tests whether the program properly behaves when a user attempts to store a diagram with something in it.

4.1.2.1 Requirements addressed.

Addresses R4 (*The user shall be able to save gate files*) and R7 (*When a user hits the "save" button, the program will open a GUI that will allow exporting of data to a local desktop*).

4.1.2.2 Prerequisite conditions.
All of prerequisite mentioned in section 3 will be required for this test.

4.1.2.3 Test inputs.

The input for this test will be a diagram with some circuits on it. The file shall be saved to the tester's desktop.

4.1.2.4 Expected test results.
The expected output is a GUI allowing the user to select a location where the file will be saved. The expected output of the GUI is a JSON file matching the circuit schema with the expected circuit saved within.

4.1.2.5 Criteria for evaluating results.
This test falls in the demonstration category of testing and is therefore determined by implementing the code and seeing whether it succeeds in executing everything it intended or fails. Since the user is saving data in memory to disk, measuring load times is unnecessary as it will largely vary by system setup.

4.1.2.6 Test procedure.
Step 1. The tester will start one of the supported browsers.
Step 2. The tester will navigate to the website.
Step 3. The tester shall create a new blank diagram on the website.
Step 4. The tester will randomly select a gate to add to the diagram.
Step 5. The tester will click to Save the diagram.
Step 6. The tester will mark whether the GUI opened or not.
Step 7. The tester will save the test file to the desktop.
Step 8. The tester will open the file in a text editor and ensure the schema is in-tact.

4.1.2.7 Assumptions and constraints.
Basic assumptions of having a functional computer with no issues existing outside the software. This includes no external hardware or software bugs. It is assumed that the test will be run on a Windows 10 or later system.

4.2 Test - 02: Load
The tests in this section are designed to test the load file functionality.

4.2.1 Test case 02-1: Empty File
Test designed to address how the system handle a empty file.

4.2.1.1: Requirements
This test shall cover the requirement that the GUI will only load the correct file type and check to see if an exception is thrown if an incompatible file is entered(R9 of SRS). This test will also measure the load time of the program(R17 of SRS).

4.2.1.2: Prerequisite Conditions
All of prerequisite mentioned in section 3 will be required for this test.

4.2.1.3 Test inputs.
The independent variable for this test includes a JSON file. The JSON file will be blank and will be loaded as the default value on the programs start up.

4.2.1.4 Expected test results.
The test will be considered a success if the browser loads a blank file that can be edited by the user all under five seconds. The test will considered a failure if the browser crashes or the file loads without having an window that can be edited.

4.2.1.5 Criteria for evaluating results.
This test falls in the demonstration category of testing and is therefore determined by implementing the code and seeing whether it succeeds in executing everything it intended or fails. Load times will also be measured when judging the program.

4.2.1.6 Test procedure.
Step 1. The tester will start one of the supported browsers.
Step 2. The tester will enter the website URL.
Step 3. The website will load a blank file as the default.
Step 4. The tester will record whether the website crashes or not.
Step 5. The user will mark down the time taken to load the file.
Step 6. If the website does not crash the user will test whether or not they can edit the main panel and record the results.

4.2.1.7 Assumptions and constraints.
Basic assumptions of having a functional computer with no issues existing outside the software. This includes no external hardware or software bugs. It is assumed that the test will be run on a Windows 10 or later system.

4.2.2 Test case 02-2: File with Content
Test designed to address how the system loads in a file with a diagram stored in it.

4.2.2.1: Requirements
This test shall cover the requirement that the GUI will only load the correct file type and check to see if an exception is thrown if an incompatible file is entered (R9 of SRS). This test will also

measure the load time of the program (R17 of SRS). The file load will contain gates in order to further test the load functionality of this program (R5 of SRS)

4.2.2.2: Prerequisite Conditions
All of prerequisite mentioned in section 3 will be required for this test. The testing of the gates (section 4.3) should have been completed in order to not conflate errors with loading with errors in the gate functionality. The save testing must also have been completed in order to have inputs for this test.

4.2.2.3 Test inputs.
The independent variable for this test includes a JSON file. The JSON file will contain a variety of diagrams in order to test the full capability of the load functionality.

4.2.2.4 Expected test results.
The test will be considered a success if the browser loads a file, with content in it, that can be edited by the user all under five seconds. The test will considered a failure if the browser crashes or the file loads without having an window that can be edited. The test is also considered a failure if the program loads a blank diagram.

4.2.2.5 Criteria for evaluating results.
This test falls in the demonstration category of testing and is therefore determined by implementing the code and seeing whether it succeeds in executing everything it intended or fails. Load times will also be measured when judging the program.

4.2.2.6 Test procedure.
Step 1. The tester will start one of the supported browsers.
Step 2. The tester will enter the website URL that is associated with the JSON FIle.
Step 3. The website will load a JSON file.
Step 4. The tester will record whether the website crashes or not.
Step 5. The tester will mark down the time taken to load the file
Step 6. The tester will compare the loaded diagram to the one saved
Step 7. If the website does not crash the tester will test whether or not they can edit the main panel and record the results.

4.2.2.7 Assumptions and constraints.
Basic assumptions of having a functional computer with no issues existing outside the software. This includes no external hardware or software bugs. It is assumed that the test will be run on a windows os system.

4.3 Test - 03: Simulate

4.3.1 Test case 03-1: Basic functionality

This test is designed to test each logic gate on an individual basis for proper functionality. It will serve as a prerequisite for tests relying on proper gate functionality.

4.3.1.1 Requirements addressed.

This test will address a portion of requirement R6 dealing with simple construction and simulation of basic logic gates and wires.

4.3.1.2 Prerequisite conditions.

All of the prerequisite conditions mentioned in section 3 will be required for this test. The user must have the required files on their local desktop.

4.3.1.3 Test inputs.

The following inputs will be of roughly the same design but with varied parameters. Each input file will have to be loaded from the local desktop. Each file will feature either 6 or 7 gates disconnected from each other, but with 1 or more isolated inputs and 1 isolated output. The four files will differ in their input parameters (i.e: "high" and "low" inputs boxes for each structure):

- Input file 1: "Test03.1.1.json." This file will feature all the gates connected to two input boxes (with the exception that NOT has only 1 input box). Both input boxes will be "low" input (i.e: "0"). The NOT gate will have only 1 input box, which will also be "low".
- Input file 2: "Test03.1.2.json." This file will feature all the gates connected to two input boxes (with the exception that NOT has only 1 input box). Both input boxes will be "high" input (i.e: "1"). The NOT gate will have only 1 input box, which will also be "high".
- Input file 3: "Test03.1.3.json." This file will feature all the gates (except NOT) connected to two input boxes. One input box will be "low" input (i.e: "0"), the other input box will be high input (i.e: "1").
- Input file 4: "Test03.1.4.json." This file will feature all the gates connected to two blank input boxes. A blank input box does not have a value.
- Input file 5: "Test03.1.5.json." This file will feature all the gates (except NOT) connected to two input boxes. One input box will be "high" input (i.e: "1"), the other input box will be a blank input box, which has no value.

4.3.1.4 Expected test results.

Here are the test results grouped by which input file they are being tested under. 0 corresponds to a "low" voltage flow result, which should display as red in the corresponding output box. 1 corresponds to a "high" voltage flow result, which should display as green in the corresponding output box. A NULL value will be invalid and be represented as a white output box.

Input file 1 ("0","0" input):

Gate	Expected Output

AND	0 (“low”)
OR	0 (“low)
NOT	1 (“high”)
NAND	1 (“high”)
XOR	0 (“low”)
NOR	1 (“high”)
XNOR	1 (“high”)

Input file 2 (“1”, “1” input):

Gate	Expected Output
AND	1 (“high”)
OR	1 (“high”)
NOT	0 (“low”)
NAND	0 (“low”)
XOR	0 (“low”)
NOR	0 (“low”)
XNOR	1 (“high”)

Input File 3 (“0”, “1” input):

Gate	Expected Output
AND	0 (“low”)
OR	1 (“high”)
NAND	1 (“high”)
XOR	1 (“high”)
NOR	0 (“low”)
XNOR	0 (“low”)

Input File 4 (NULL, NULL):

Gate	Expected Output
AND	NULL
OR	NULL
NOT	NULL
NAND	NULL
XOR	NULL
NOR	NULL
XNOR	NULL

Input file 5 (NULL, “1”):

Gate	Expected Output
AND	NULL
OR	NULL
NAND	NULL
XOR	NULL
NOR	NULL
XNOR	NULL

4.3.1.5 Criteria for evaluating results.

There are 3 values which an output can be: 0, 1, or NULL. A test will be considered a failure if any gate’s output is any value other than the value highlighted in section 4.3.4’s corresponding table. A retest will be performed after the gate(s) with errors are fixed. Only if this test is completed successfully will the next simulation test case be allowed to run.

4.3.1.6 Test procedure.

The tester will perform the following series of actions on each input file.
If the recorded data matches the data detailed in section 4.3.1.4, for all files, the test will be terminated. If one gate makes the test unsuccessful, or there is an exception during the procedure, the test in its entirety will be re-run after modifications.

Step 1. The tester will open an instance of Logic Gate Editor in a Firefox browser
Step 2. The tester will click “Load” from the interface

Step 3. The tester will navigate to one of the input files on their computer
Step 4. The tester will load the file into the program
Step 5. The tester will click the “simulate” button from the taskbar
Step 6. The tester will record the output box results

4.3.1.7 Assumptions and constraints.

This test assumes that the load functionality was implemented correctly, and that the tester is testing using Firefox on a windows OS, as detailed in section 3.1.

4.3.2 Test case 03-2: Extended functionality

This test case will ensure gates can be hooked up and extended into logic circuits by connecting wires.

4.3.2.1 Requirements addressed.

This test case addresses the requirement detailing the ability for a user to extend gates (R1 of SRS).This test case will also address the requirement allowing the user to manipulate, connect and simulate gates (R6 of SRS). This test case will also measure quality requirements R17.1, and R17.2 of the SRS, in particular, being able to simulate any combination of 4 gates, as well as a circuit being simulated in under 3 seconds.

4.3.2.2 Prerequisite conditions.

All of the prerequisite conditions mentioned in section 3 will be required for this test. The user must have the required files on their local desktop. In addition, test case 03-1 must be completed successfully.

4.3.2.3 Test inputs.

Four JSON files will be provided as inputs. Each file will feature its own characteristics, testing to see if an input can accurately be passed from gate to gate to the end output box. The multiple files feature differing input values to test:

- Input file 1: “Test03.2.1.json” will feature an AND, XOR, NOR, and a NOT gate all hooked up together. There will be 4 input boxes, a “0” and “1” hooked up to the AND gate, and then a “1” and “1” hooked up to and XOR gate. These gates will connect to a NOR gate, which is connected to a NOT gate. The connection ends with an output box.

- Input file 2: “Test03.2.2.json” will feature an AND, XOR, NOR, and a NOT gate all hooked up together. There will be 4 input boxes, a “1” and “1” hooked up to the AND gate, and then a “0” and “1” hooked up to and XOR gate. These gates will connect to a NOR gate, which is connected to a NOT gate. The connection ends with an output box.

- Input file 3: “Test03.2.3.json” will feature a OR, XNOR, NAND, and a NOT gate all hooked up together. There will be 4 input boxes, a “0” and “0” hooked up to the OR gate, and then a “0” and “0” hooked up to and XNOR gate. These gates will connect to a NAND gate, which is connected to a NOT gate. The connection ends with an output box.

- Input file 4: “Test03.2.4.json” will feature an OR, XNOR, NAND, and a NOT gate all hooked up together. There will be 4 input boxes, a “1” and “0” hooked up to the OR gate, and then a “1” and “1” hooked up to and XNOR gate. These gates will connect to a NAND gate, which is connected to a NOT gate. The connection ends with an output box.

4.3.2.4 Expected test results.

Here are the test results grouped by which input file they are being tested under. 0 corresponds to a “low” voltage flow result, which should display as red in the corresponding output box. 1 corresponds to a “high” voltage flow result, which should display as green in the corresponding output box.

Input File	Expected Result of output box
Input file 1	0 (“low”)
Input file 2	1 (“high”)
Input file 3	0 (“low”)
Input file 4	1 (“high”)

4.3.2.5 Criteria for evaluating results.

There are 3 values which an output can be: 0, 1, or NULL. A test will be considered a failure if any gate’s output is any value other than the value highlighted in section 4.3.4’s corresponding table. A retest will be performed after the gate(s) with errors are fixed.

4.3.2.6 Test procedure.

The tester will perform the following series of actions on each input file.
If the recorded data matches the data detailed in section 4.3.2.4, for all files, the test will be considered a success. If one gate makes the test unsuccessful, or there is an exception during the procedure, the test in its entirety will be re-run after modifications. If this test fails, test case 03-1 will need to be rerun after modifications have been made.

Step 1. The tester will open an instance of Logic Gate Editor in a Firefox browser
Step 2. The tester will click “Load” from the interface
Step 3. The tester will navigate to one of the input files on their computer
Step 4. The tester will load the file into the program

Step 5. The tester will click the “simulate” button from the taskbar
Step 6. The tester will record the output box results

4.3.2.7 Assumptions and constraints.

This test assumes that the load functionality was implemented correctly, and that the tester is testing using Firefox on a windows OS, as detailed in section 3.1.

4.4 Test - 04: Share

4.4.1 Communication through Database

The requirement addressed will allow shared access to database.

4.4.1.1 Requirements addressed.

Computers will communicate through shared access to a database. The program will need to test the ability for one process to share a logic gate circuit to the database and for another process to load it from the database.

4.4.1.2 Prerequisite Conditions

In order to complete this test, the program must be up and running and the database must be connected. Access to the database also must be established.

4.4.1.3 Test inputs

We will test this with creating a circuit and sending it to the database. The circuit we create will have at least one of each type of gate.

4.4.1.4 Expected results

Then, from another server, we will retrieve shared circuit from the database and continue to edit. We expect this to run error free.

4.4.1.5 Criteria For evaluating Results

If we are able to communicate with the database, then we have passed the test. We expect this test to be completely error free and as no retrieving information from the database can be detrimental to the program.

4.4.1.6 Test procedure.

Step 1. Create Program.
Step 2. Save the logic gate to the database.
Step 3. Be able to retrieve logic gates from database.

Step 4. Should be error-free.

4.4.1.7 Assumptions and constraints.
We assume that the program and database are up and running correctly.

4.4.2 Access to the Database
This requirement is regarding access to the Database.

4.4.2.1 Requirements addressed.
The database for storing projects will be obscured from the user, making direct edits to it will be impossible. Through users beta testing this requirement will be met if no user can access the database.

4.4.2.2 Prerequisite Conditions
The program must be up and running and there must be some project already existing in the database.

4.4.2.3 Test inputs
We will test this with creating a project with at least one type of each gate and saving it to the database.

4.4.2.4 Expected results
The user will try to edit the saved project through the database directly and will be unable to do so.

4.4.2.5 Criteria For evaluating Results
If we are unable to communicate directly with the database, then we have passed test. We expect this to have no errors as directly accessing the database can be detrimental to program’s security.

4.4.2.6 Test procedure.
Step 1. Create Program
Step 2. Attempt to communicate with the database directly.
Step 3. The program should not allow the user to do so.

4.4.2.7 Assumptions and constraints.
We assume that the program and database are up and running correctly.

4.4.3 Use of the Database

4.4.3.1 Requirements addressed.
The program will utilize a database for storing the components of a user’s gate. The software will be tested to return stored values and will pass this requirement if succeeds in doing so.

4.4.3.2 Prerequisite Conditions

We will create a program that is already stored in the database, in order to determine if we can retrieve it.

4.4.3.3 Test inputs
We will test this with first creating a diagram and saving it to the database. Then, we will try to load the saved components of a user’s gate from the database. The saved document will have one type of each type of gate.

4.4.3.4 Expected Results
We expect to be able to load the saved document from the database successfully and continue editing.

4.4.3.5 Criteria For evaluating Results
If we can retrieve information successfully, then it has passed the test. We expect this test to be completely error free and as not retrieving stored information from the database can be detrimental to the program.

4.4.3.6 Test procedure.
Step 1. Create Program
Step 2. Store in the database.
Step 3. Attempt to retrieve information from the database.
Step 4. Should be error-free.

4.4.3.7 Assumptions and constraints.
We assume that the program and database are up and running correctly.

4.4.4 Sharing using the URL

4.4.4.1 Requirements addressed.
The user shall be able to share gate projects using a unique URL. Demonstration The website will need to display the a usable URL otherwise this requirement has not been met.

4.4.4.2 Prerequisite Conditions

We must be able to create a diagram in a web browser.

4.4.4.3 Test inputs

To test this, we will create a project with at least one type of each gate. We also must be able to copy the URL from the web browser.

4.4.4.4 Expected Results

We expect to be able to share the URL successfully and open it in another browser with no changes.

4.4.4.5 Criteria For evaluating Results

If the URL loads successfully, than it has passed the test. We expect this test to be completely error free as sharing project via a URL is one of the defining characteristics of our project.

4.4.4.6 Test procedure.

Step 1. Create Program
Step 2. Copy and paste URL in another browser.
Step 3. Load project error-free.

4.4.4.7 Assumptions and constraints.

We assume that the program and database are up and running correctly. Also we must be able to copy and paste the URL correctly.

4.4.5 Sharing an empty diagram

4.4.5.1 Requirements addressed.

The user tries to share a blank diagram The program will issue a warning, but not stop the user from sharing a blank diagram as long as it does not violate any file type restrictions

4.4.5.2 Prerequisite Conditions

We need to create a diagram that is completely empty. The program must allow the user to do so.

4.4.5.3 Test inputs

To test this, we will load the program but not create a diagram. We must be able to load the program and copy the URL even though the diagram is empty.

4.4.5.4 Expected Results

We will try sharing the empty diagram and a warning will appear. After ignoring the warning, the diagram will be shared correctly.

4.4.5.5 Criteria For evaluating Results

If the empty diagram loads successfully than it has passed the test. This is an added functional feature of our project and should run successfully.

4.4.5.6 Test procedure.

Step 1. Create empty program.
Step 2. Attempt to share the empty diagram.
Step 3. Should run error - free.

4.4.5.7 Assumptions and constraints.

We assume that the program and database are up and running correctly and that the program allows for an empty diagram.

4.4.6 Download Speed

4.4.6.1 Requirements addressed.

Downloads times shall scale to the users internet connection speed.

4.4.6.2 Prerequisite Conditions

The program must be up and running and connected to the internet. Also, the internet connection speed must be known.

4.4.6.3 Test inputs.

We will test this requirement with downloading different projects and making sure that it scaled with the internet speed. We must be able to download projects all containing at least one of each type of gate. Also, we must calculate the download time.

4.4.6.4 Expected Results

We expect the downloading speed to scale with the internet speed.

4.4.6.5 Criteria For evaluating Results

If the download speeds scale with the internet speed, it has passed the test. We expect this test to be completely error free as not scaling with internet speed can be a real deterrent to use the project.

4.4.6.6 Test procedure.

Step 1. Download a project.
Step 2. Compare with current internet speed.

4.4.6.7 Assumptions and constraints.

We assume that the program and database are up and running correctly.

4.5 Test - 05: Help Module

This test will ensure that the help module built-in to the program is fully functioning.

4.5.1 Requirements addressed.

This test addresses requirement R19, which ensures the functionality of a help pop-up module.

4.5.2 Prerequisite conditions.

The test environment should not matter at all for this, since the help menu is an internal module. But for safety, it will be conducted on both the Firefox browser as well the Google Chrome browser. The details for this prerequisite condition are found in section 3.

4.5.3 Test inputs.

There is no given test input for this test. Following the proper procedure listed in section 4.5.6 will serve as an input.

4.5.4 Expected test results.

In a successful test result, the help menu should be opened without causing error.

4.5.5 Criteria for evaluating results.

A successful result will be the help module opening without error and displaying all of its intended message in an aesthetically pleasing manner. There will be a file named “Test05.1.html” which will provide a reference for how the help module should look.

4.5.6 Test procedure.

Step 1. The test application will be loaded into the FIrefox Browser (version outlined in section 3.1)
Step 2. The tester will navigate to the Logic Gate Editor.
Step 3. The tester will click on the “Help” button located on the taskbar.
Step 4. The tester will record whether the module loads or not, and if it displays it’s message properly and fully.

4.5.7 Assumptions and constraints.

This test assumes that the prerequisites in section 3.1 have been fulfilled.

4.6 Test - 06: Browser Crash

This test will ensure that in the event of a browser crash, the previously constructed diagram will be saved and be able to be restored via cache the next time the user loads the program in their browser.

4.6.1 Requirements addressed.

This section address requirement R 9.3 of the SRS. Test listed below are used to address browser crashes and the systems in place to handle them.

4.6.2 Prerequisite conditions.

The test will be conducted on both the Firefox browser as well the Google Chrome browser. The details for this prerequisite condition are found in section 3.

4.6.3 Test inputs.

There will be a modified version of the main source file for the program. This file will replace the main file, “index.html.” The file will be named “index2.html”, but the tester will have to remove the “2” in the filename to ensure that github pages uses that file instead of the original. This test file will add a button labeled “crash” that will start an infinite loop, crashing the browser.

4.6.4 Expected test results

This test is an example of a demonstration test. The test would be considered a success if the data is cached and the same permutation of the gate is saved. The test is considered a failure if the data is lost with no way to retrieve.

4.6.5 Criteria for evaluating results.

If, after the user navigates back to the Logic Gate Editor, the program restores the previously created diagram and the data stored, then the test will be considered successful.

4.6.6 Test procedure.

The tested application will be loaded and a simple circuit will be created. Then, without saving the circuit into a stable file, the program will be intentionally crashed with each of the crash conditions in the test cases, after an automatic cache. After a five minute period, the tested application is restarted and the circuit.

Step 1. The modified application will be loaded into a browser
Step 2. A simple circuit will be created inside the test application
Step 3. The tester will push the “crash” button
Step 3. Wait for an auto caching to occur
Step 4. The test application will be crashed
Step 5. A waiting period of 5 minutes will occur

Step 6. The test application will be loaded again into the browser
Step 7. The option to restore the previous session will be chosen when prompted

4.6.7 Assumptions and constraints.

This test assumes that the prerequisites in section 3.1 have been fulfilled, as well as that the test will be carried out on the same computer and browser throughout the test.

5. Requirements traceability

Test Case #	Requirement(s)	SRS Section	SSDD Section
01-1	R4, R7	3.2	4.1
01-2	R4, R7	3.2	4.1
02-1	R9, R17	3.3, 3.11	4.1
02-2	R5, R9, R17	3.2, 3.3, 3.11	4.1
03-1	R6	3.2	3.1,4.1
03-2	R1, R6, R17	3.2,3.11	4.1
04	R16, R12, R11, R3, R9.1, R17.4		4.4
05	R19	3.13	4.1
06	R9.3	3.3	4.2