Software Test Documentation: Test Plan

for

A GUI-based Software Package for the Simulation of Queuing Networks

Prepared by

Fatma Erem Aksoy – 2315075
Ece Erseven – 2385383
Ceazar Hatokah – 2456119

Middle East Technical University Northern Cyprus Campus

Computer Engineering

Supervised by Enver Ever

05/09/2023

Contents

1 Introduction

This report presents the test plan of the GUI-based Software Package for the Simulation of Queueing Networks. The software product is available as a web application and builds queue simulations by supplying a GUI interface to the user. For instance, GUI allows users to drag server, queue, and connection from the toolbox and drop on the canvas. The dropped items (server and queue) have clickable functionality. The user is asked to enter the number of entities, lambda (average arrival rate) for the queue component, and mue(average service rate) for the server component when clicking on it. While taking inputs from the user, the software also checks if the input parameters are all provided and in appropriate ranges. After providing all parameters, the user clicks on the run button, and the simulation is generated.

2 Test items

This software product has three components to be tested GUI, simulation code, and the integration of GUI and simulation. The first test item, GUI, has a toolbox menu, and the user can drag and drop server, queue, and connection components from the menu. The server and queue dropped on the canvas should be clickable so the user can enter the input values by clicking on the server and queue. GUI should accept input parameters lambda, mue, and the number of entities only in specific ranges. The GUI should be tested to perform the drag-and-drop, clickable functions and take valid inputs. The second component to be tested is the simulation code. The web application can generate M/M/1 and M/M/1/L simulations so far. M/M/1 simulation code takes the number of entities, lambda, and mue values as inputs, calculates inter-arrival time, service start time, service time, finish time, total waiting time, mql list(mql /finish time), and plots the graph accordingly. M/M/1/L uses the same inputs as M/M/1 but also has a limit value for the queue size provided by the user. It calculates traffic intensity, average number of customers, average system time, number of customers in the queue, and queue time and plots the related graph. Also, MMCL simulation will be implemented as a future feature of the web application; it will generate queue simulation for multiple server multiple queues. The code will be updated to take an array of input values, perform the calculations, and plot the graph. The simulation codes need to be tested for the validity of calculations and consistency of outputs. The third component that needs to be tested is the integration of the GUI and simulation. The inputs are taken from the user by GUI, and the simulation code uses inputs to calculate statistics. Integration is necessary to achieve the connection between the GUI and the simulation code. Testing

integration is essential because the software product can only generate simulations with integration.

3 Features to be tested

GUI has three features to be tested: input validation, drag and drop feature, and clickable feature.

1. Validate that the GUI checks for input validation:

1.1.Validate that the mue value should be a positive number.

1.2. Validate that the lambda value and number of entities should be positive numbers.

1.3. Validate that number of entities should be greater than 1024.

1.4. Validate that lambda should be less than or equal to the mue value.

1.5. Validate that there is no missing parameter.

2. Validate that the user should be able to drag and drop items from the toolbox menu:

2.1.Validate that dragDrop() function checks for the dragged item's type(server or queue)

2.2.Validate that dragDrop() function creates a copy node and append it on the canvas to perform the drop operation.

3. Validate that the clickable feature of the server and queue is working correctly and user can enter input values by using the prompt window:

3.1. Validate that openQueueForm() is called when the user clicks on the queue item.

3.1.1. Validate that openQueueForm() function creates a new hidden element with a value attribute that is assigned to "lambda" and a name attribute set to "queue+queueCount"(i.e., queue1)

3.1.2 Validate that the queueCount value is incremented.

3.2. Validate that openServerForm() is called when the user clicks on the server item.

3.2.1.Validate that openServerForm() function creates a new hidden element with a value attribute that is assigned to "server" and a name attribute set to "server+serverCount"(i.e., server1)

3.2.2 Validate that the serverCount value is incremented.

Simulation has two features that need to be tested; M/M/1 simulation code and M/M/1/L simulation code :

1. For M/M/1 simulation code, validate that calculate_mql() function performs the calculations correctly.

1.1. Validate that the plotLineGraph() function plots the graph.

2. For M/M/1/L simulation, validate that fun(M, lam) function plots the corresponding graph.

Integration of the GUI and simulation has two features that need to be tested:

1. Validate that the backend function get_inputs() can access the inputs taken by the prompt window.

2. Validate that each web page of the application is integrated as routes by using Flask's @app.route() so that the user can switch pages using the web application's navigation bar.

4 Features not to be tested

GUI also has a home page, help page, and contact page; however, those pages are related to the front-end side and don't have any critical functionality that affects the simulation. Therefore the corresponding HTML codes with mentioned pages don't need to be tested.

5 Approach

5.1 Unit Testing

With unit testing, we aim to test the process by running multiple tests on each function with different parameters and checking if we get the expected outputs and correct error checks. The expected values will depend on the input values the user enter but it can be considered in terms of accuracy. There are provided equations that are used in this areas to calculate all the statistics we measure in this project and we try to get similar values to those values with our program. We plot graphs for these values so we can see how accurate/close the values we get from these staistics are to the optimasl ones used in the industry. So we will have both the equations and the values we get from the execution of the program in our simulation codes and we can easily see the difference in the graphs plotted to see if the program work as efficient as we want. As we use Python language to do the statistical calculations for the simulation part, we plan to use PyUnit, which is a built-in unit testing framework for Python, to perform unit testing. We have two different functions to test so far; one is for the MM1 statistics, and the other one is for the MM1L, which has some other outputs to test.

5.2 Integration Testing

We plan to use the bottom-up technique for integration testing, which requires testing the lower-level modules first and then performing the higher-level testing. The modules we have so far for the simulation part are the MM1 graph and MM1L graph. We already checked and verified that the graphs alone (without integration) work as expected without any problem and

for the integration part of these graphs, we will run the program after doing the integration and check again if the graphs plot anything or they give any type of error. We will do the testing manually as we need to see where the exact problem is (if there is) considering we have many modules to connect with each other. We will make sure that the issues may occur are accompanied by component-specific error messages that we add to test the integration of each feature possible. The files to plot these graphs are tested individually to see if the values entered are taken correctly and the graphs are displayed on the webpage successfully after entering the inputs. For the GUI part, we will be testing different web pages and their interactions with the main Python file we execute. To do this, we first check the connection of each page with the main Python file separately; then, we test the connections between all the pages to perform the finalized integration testing.

5.3 System Testing

We test the whole system for the system testing to see if all the files work successfully in harmony. Here, we will perform functional and non-functional testing. This testing method can verify the system's overall functionality and improve the system quality based on the result. To perform the system testing, we will test all the web pages to see if they are connected successfully and if the output graphs are plotted as required with the desired outputs. We will use scenario-based system testing, and to do that, we will split different scenarios between group members to test more scenarios more efficiently. After each of them tests the assigned part of the program, the person responsible for the integration part will fix all the problems and ensure no further problems. For the integration part of the clickable feature, we will check if the values entered are saved in the Python code, and if they did, then we ensure that they are correct and sent to the other HTML and js files for graph plotting without any issue. We also test if the user could pick between the simulation types as they will have three options, and we have two options in the current project. We will check whether the selected graph among those two different types of graphs is plotted correctly according to the user's choice. As we test the entire program, we will also be able to do the necessary validation checks for the whole system and errors in case of any wrong/missing input.

5.4 Performance Testing

We perform performance testing on the simulation part of our project to measure speed, accuracy, and stability and will use stress testing for this purpose. With stress testing, we see how the program manages high traffic under an intense workload so that we can eliminate

performance bottlenecks. We will do it by entering one of the inputs that specify the number of entities in the system as very high, very low (the minimum amount that can be entered in this case, 1024), and a value between these two; so that we can see how the measurements mentioned are affected by this change in the number of entities in the program. For it to pass the stress test, the system's response time calculated in the graph needs to be minimized as much as possible. We will only test the amount of time spent evaluating the output and processing speed, and we will not have any other item to test as we do not have any database. To do this testing, we will execute our code with different values, and by increasing it every time, we will monitor how the system handles this.

5.5 User Testing

We plan to perform user testing for the project through a survey. We will create an online survey that allows participants to try the website for the simulation and give feedback for specific criteria. The questions will mostly be related to the GUI part of the project as users will only be able to interact with the user interface, webpages. The audience will be people with the necessary background knowledge about simulation tools and networks in this area.

6 Item pass/fail criteria

For the input validation for any type of queue, if the input check has been validated and an output has been plotted, in addition to the plotting, we need to check for the formulas of the queues, to make sure that we are using the correct formulas that will produce the required results,if all of that has been checked and validated, then the test has been successful,to have the best accuracy of results for MM1 queue, we check if the graph is oscillating which hints that the simulation is working successfully and giving the best results, otherwise input values are not correct and user needs to enter again.

For the drag and drop, for any desired element that user wants to drag, the same element should be dragged onto the canvas, if a user wants to drag a queue, a single queue should be dragged into the canvas, otherwise the drag and drop fails, in addition to that, the drag and drop feature should also deal with dragging multiple queues and servers, if a user wants to drag multiple queues, multiple queues should be added as well to the canvas. The type and

quantity of the desired element thata are in the canvas should be exactly same as the number of times and the type of element that the user has dragged from the tool box.

For the clickable feature, when an element is clicked, a window with an appropriate input form should appear, depending on the type of queue and server connection, if it's a single server-queue connection, then an input form for the single server/queue connection should appear, and not the input form that contains input for multiple server/queue connection case, otherwise the clickable feature fails, since it gives incorrect input data for the type of connection that the user wants.

For our project, we want it to be done with the best performance, so we will use stress testing strategy. We want the system to be fast, every function of the project system should take less than a second to start executing when the user picks a specific funcionality, in addition to that, we want the results to be very accurate, so error rate margins should be in the decimals fro the simulation results, simulation time has to be accurate with the expected results, but there are no specific limitions to it in terms of how long it should take.

7 Schedule

7.1 Milestones and Tasks

ID	Milestone	Date
M1	Unit testing of creating different test cases for queue formulas	Week 4
M2	Integration Testing: creating different test cases for plotting graphs	Week 6
M3	Integration Testing: Successful connection interaction of each single page of the simulation (Main Page, Help Page, Simulation Pages, Contact Page, etc.)	Week 6
M4	System Testing: Testing whole functionality of the code simulation + GUI using scenario-based strategy	Week 9
M5	Performance testing by applying stress testing strategy	Week 10

Table 1: Milestones Table

ID	Task	Duration	Task Dependency	Responsible Member
T1	Testing Functionality of the formulas of each queue in the code by creating different test cases which include many different input value scenarios	1 week		Ceazar

T2	Integration Testing (Plotting Graphs) to validate the plot and the entered input values for the graphs by entering multiple input cases and checking whether the plot is correctly displayed based on the given inputs	1 week	(M1)	Erem
T3	Integration Testing (Web pages) to check the functionality of each web page separately with applying multiple cases for the functionalities of each web page and then checking their interaction with each other by applying multiple functions from each web page together	2 week	T2 (M2)	Ece, Erem
T4	System Testing to check if the functionalities of the web pages along with the simulation code results are operating successfully without any errors, unwanted output	1 week	T3	All members
T5	Performance Testing to get the best performance in terms of efficiency, speed, accuracy by applying stress testing which contains heavy weight values for the test cases, (ex. Huge arrival rate)	1 week		All members

Table 2: Tasks Table

7.2 Gantt Chart

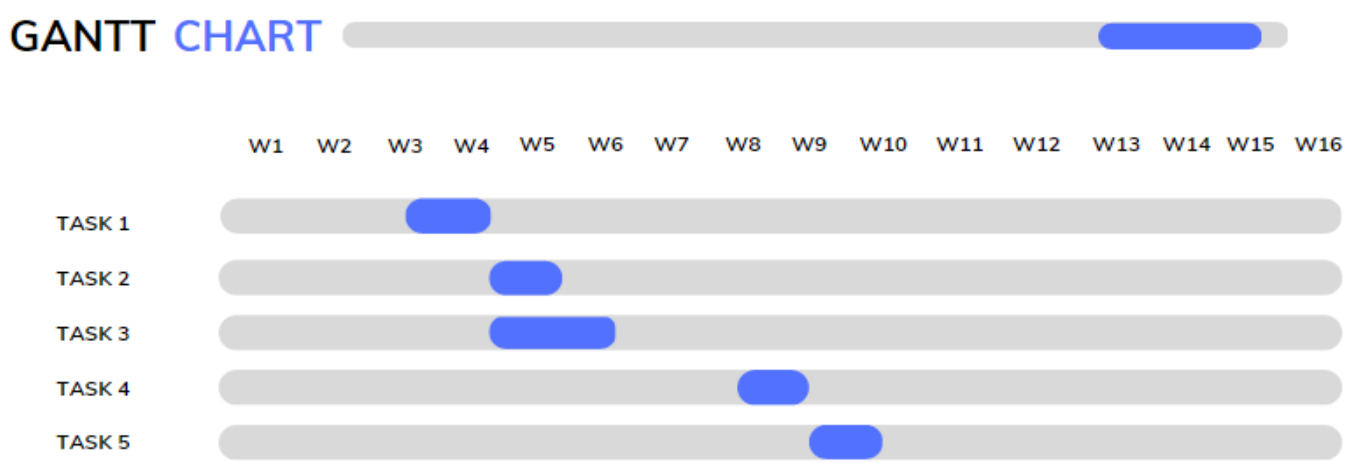

Figure 1: Gantt Chart

8 References

System testing. (2022, June 9). GeeksforGeeks. https://www.geeksforgeeks.org/system-testing/

Terra, J. (2022, July 27). *What is integration testing? Examples, challenges, and approaches | Simplilearn*. Simplilearn.com. https://www.simplilearn.com/what-is-integration-testing-examples-challenges-approaches-article#:~:text=Integration%20testing%20is%20known%20as,they%20work%20together%20as%20designed

Top 10 performance testing tools | Load testing tools guide | Edureka. (2022, March 29). Edureka. https://www.edureka.co/blog/performance-testing-tools/

Unit testing | Software testing. (2022, June 7). GeeksforGeeks. https://www.geeksforgeeks.org/unit-testing-software-testing/

Unittest — Unit testing framework. (n.d.). Python documentation. https://docs.python.org/3/library/unittest.html