TEST PLAN

Simulated computing system

January 7, 2018

António Pedro Araújo Fraga

Student ID: 279654

Cranfield University

M.Sc. in Software Engineering for Technical Computing

Contents

Test Plan Identifier

Simulated Computing System, Fraga

Product Overview

This system will be developed under two modules, **Software Testing and Quality Assurance** and **Requirements Analysis and System Design** at **Cranfield University**.

It is supposed to simulate a computer system which runs jobs of several sizes. The jobs last between **one second** and **sixty four hours**, assigning them with a category of **Short**, **Medium**, **Large** or **Huge**. The system shall be capable of scheduling a job running time on a basis of a **First Come, First Served** methodology, analysing what is the correct time to run a job every time a submission is made. The simulation must generate a set of informations regarding the pretended outputs. These outputs are dependent of inputs values that will be established in the beginning of the simulation.

The application is going to be used by the IT department, modelling the behaviour of a real computing platform and exploring alternative accounting strategies.

The system shall attend functional and non-functional requirements. Most of functional requirements shall be tested at the **Unit & Integration level**, whereas non-functional requirements shall be tested at a higher level.

Testing Synopsis

Items to be tested

- Coding standards
- Compatibility
- Functional

- Reliability

System Requirements

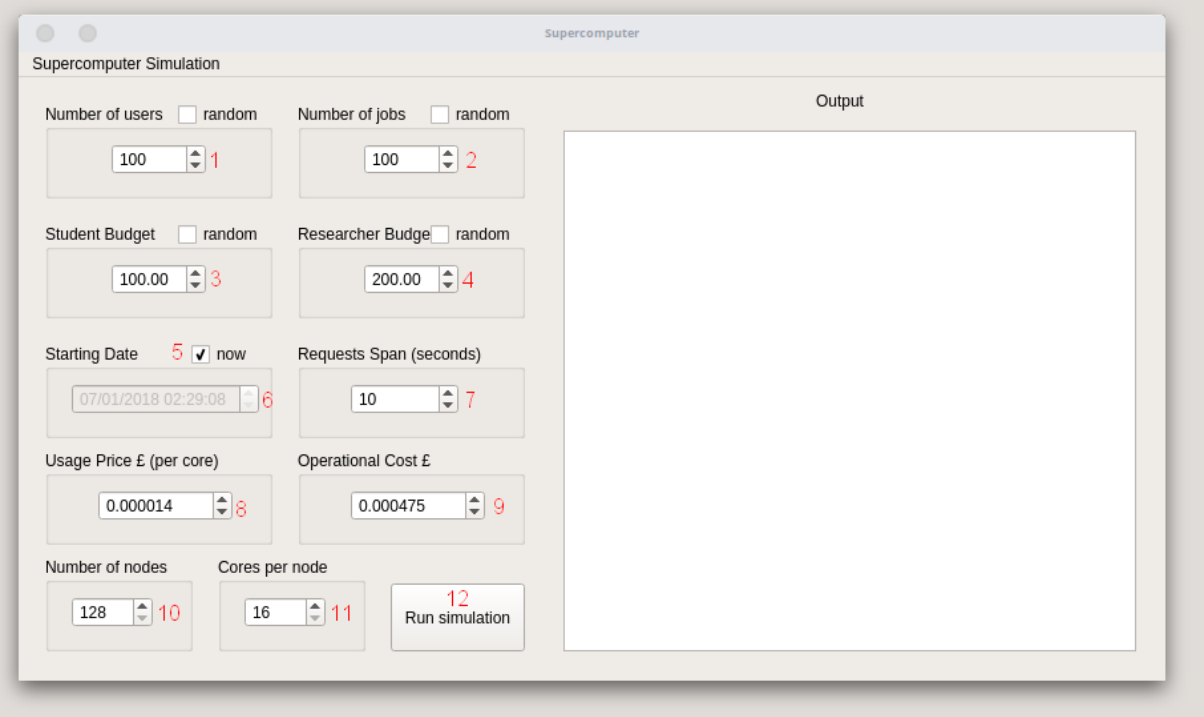
These requirements are specified in the **System Requirements Specification** document.

References & Standards

This test plan is based on the **IEEE 829 format**, regarding to a test plan outline.

Types of Testing

Acceptance Testing



- **AT01** A user is able to change the number of simulated users by changing the spin box labelled with **1**.
- **AT02** A user is able to change the number of simulated jobs by changing the spin box labelled with **2**.
- **AT03** A user is able to change the student budget by changing the spin box labelled with **3**. This spin box supports two decimal places.
- **AT04** A user is able to change the researcher budget by changing the spin box labelled with **4**. This spin box supports two decimal places.
- **AT05** A user is able to alternate between the present and a different date by hitting the checkbox labelled with **5**. If this checkbox is unchecked, the user is able to define a date by changing the date picker value labelled with **6**.

- **AT06** A user is able to change the requests span value by changing the spin box labelled with **7**.
- **AT07** A user is able to change the usage price per core value by changing the spin box labelled with **8**. This spin box supports six decimal places.
- **AT08** A user is able to change the operational cost per core value by changing the spin box labelled with **9**. This spin box supports six decimal places.
- **AT09** A user is able to change the number of nodes by changing the spin box labelled with **10**.
- **AT10** A user is able to change the number of cores by changing the spin box labelled with **11**.
- **AT11** A user is able to run the simulation by pressing the button labelled with **12**. The user is able to observe the simulation output in the "Output box".

Functional & System Level Tests

Every functional requirement will be tested with **Unit Tests**. And since the code will be developed with **C++**, **Catch**, which is an Unit Tests framework for C++, will be used. **Functional Tests** are the tests that will cover every single module individually. They will assure that every **single** part of the system is working as it should be. These tests will cover specific features.

System Level Tests will be written to test the entire system. The system, which accepts a set of inputs, will have to return a string as **Output**. This string will contain information like the **resulting price paid by the users** and the **economic balance** of the centre. This output will be tested with **regular expressions**, assuring there's a match between the actual output and the expected output. Avoiding, for instance, results like **NAN (Not A Number)** instead of an expected **floating point** value.

Code coverage will be generated by a specific tool as well. Further information will be given in the **Code Coverage** section.

Boundary Tests

This kind of tests will included in **Unit Testing**. All boundaries are respected, so that the input values are always valid, generating valid outputs as well.

Integration & Regression Tests

Integration Tests are often useful when a project is developed by a team, having different engineers developing different modules. This project will be developed in one machine, therefore testing the code on this machine can be seen as an **Integration Test**. However, a technology will run along with the **version control** system **(Travis CI)**. This technology will be responsible to run the **Unit Tests** every time there's a **merge** between repository branches.

These repository branches, as stated in the **System Requirements Specification**, will be created every time a new **requirement** is being developed. But the version control system, will always have **two** main branches.

- deployment - which will contain a version control of every accepted release.
- development - which will contain a version control of every pre-release (to be accepted)

When running **Unit Tests** on these branches, it will be assured that the system is running properly.

Compatibility Tests

The tests and program will run in different machines running different operating systems. Every test and program execution will have to produce the correct results:

- **CPU**: Intel Core i7-4750HQ, up to 3.2 GHz. **Memory**: 8GB **OS**: UNIX Elementary OS
- **CPU**: Intel Core i7-4510U, up to 2.6 GHz. **Memory**: 8GB **OS**: UNIX Arch Linux
- **CPU**: AMD A8-4555M APU with Radeon, 1.6 GHz. **Memory**: 8GB **OS**: Microsoft Windows 10
- **CPU**: Intel Pentium P6000, 1.87 GHz. **Memory**: 4GB **OS**: Microsoft Windows 10

Code Coverage

Code Coverage will be generated by **lcov**. This tool is able to generate an **html** file with all the information about line coverage. It will indicate how many times a certain line will be executed, including the **percentage** of coverage in each file as well.

Test Schedule & Completion Criteria

The tests shall be written whenever a requirement development is done. This will ensure that merge between branches include the new testing software along with the new functionality. Therefore, this measure will result in good quality **Integration Tests**.

Item Pass/Fail Criteria

Every feature must be tested and approved. Unit Tests shall achieve total code coverage.

Test Plan Review

This document will be reviewed by the **head of the IT department**. It will be assured that this document follows the **IEEE 829 format** and that it covers every bullet point of the testing procedure.