Super Computer Simulator

REASD and STQA Combined Assignment Report

M.Sc. in Software Engineering for Technical Computing

Aman Makkar

S321839

Muneeb Khan

S273025

6th January 2020

Contents

Abstract

This report has been compiled on the work carried out to design, develop, test and implement a Simulator for a Super Computer. The simulator will be used by the IT team of a University to evaluate the policies of a job submission system, for accounting and job control purposes. A design pattern will be agreed for the source code, a SRS document will be compiled for the Functional and non-Functional Requirements and a Test Plan will be compiled for the “White” and “Black” box testing methods.

Nomenclature

CPU	Central Processing Unit
SRS Specification	Software Requirements
GPU	Graphics Processing Unit
UML	Unified Modelling Language

1. Introduction

The purpose of this report is to provide the process used within Software Engineering to design, develop, test and implement a software. A simulator has been developed in C++ to help the IT team of a University to be able evaluate a job submission system. A breakdown of the user requirements has been provided in the SRS document below. Software Engineering is an important aspect within the modern world, with almost all the devices now requiring a software i.e. from mobile phones to computer games. As all these devices are connected to the internet, further software development is required to provide apps and games (Gamma, Helm, Johnson, & Vlissides, 1994).

However, software is very different to any hardware. Hardware starts to degrade over time and has a standard lifecycle that can be demonstrated via various tests, which is not possible for software. The software lifecycle depends on how long the hardware can be used for and therefore, testing a software product requires a different approach to testing any hardware (Sommerville, 2016).

There are various industry standards for developing software, however this does differ depending on the industry. For example, a software used in the Aerospace or Automotive industry would go through much more rigorous testing and is highly regulated whereas, an app developed for a smart phone would not require the same level of testing or regulations to be able to be released to the market.

2. Project Plan

Simulator for Super Computer

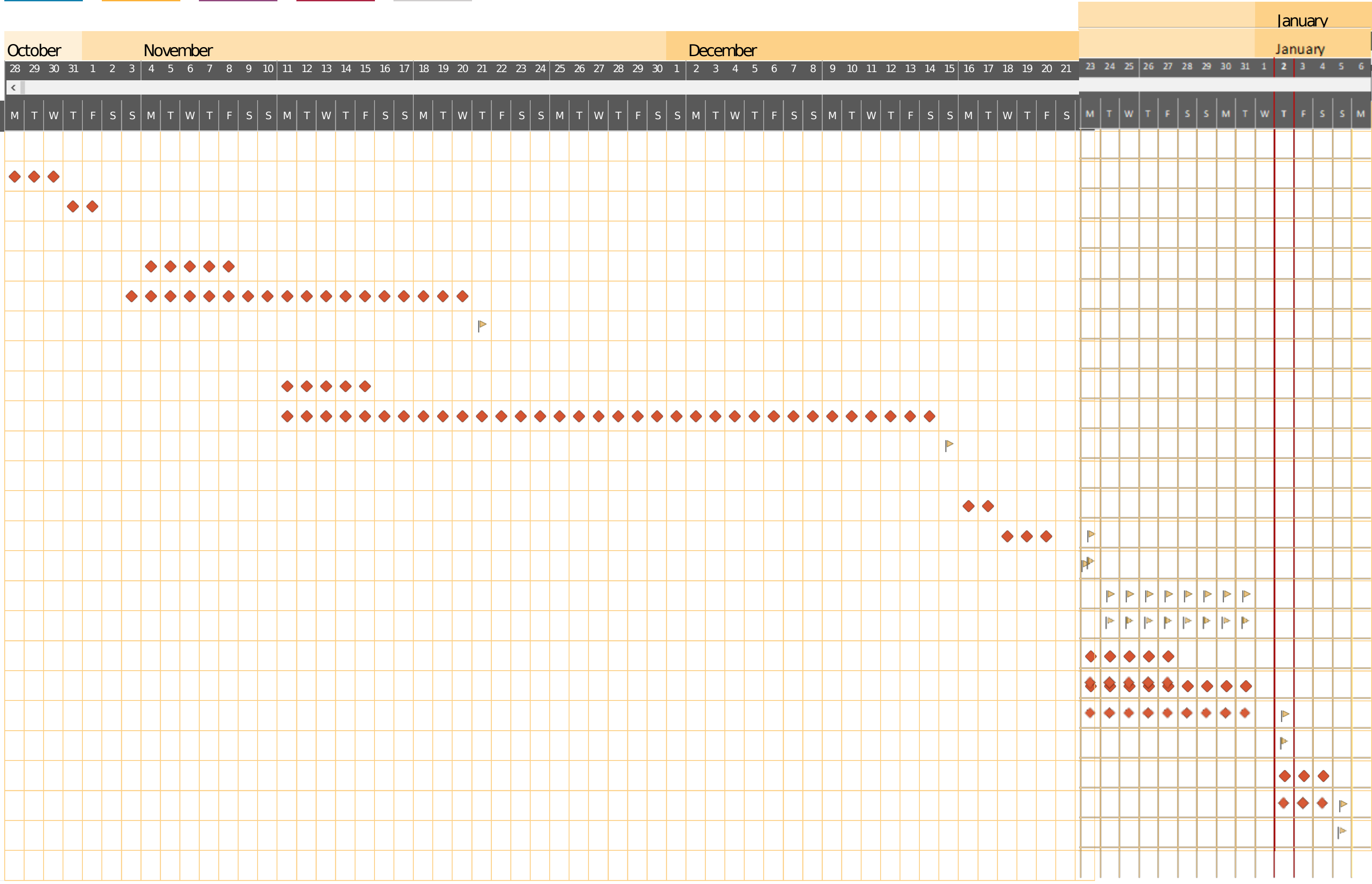
Cranfield University
Aman Makkar and Muneeb Khan

Project Start Date:	28/10/2019
Scrolling Increment:	0

Legend:

On Track | Low Risk | Med Risk | High Risk | Unassigned

Milestone Description	Category	Assigned To	Progress	Start	No. Days
Project Start					
Scope	Goal	Aman and Muneeb	100%	28/10/2019	3
Agree Tasks	Goal	Aman and Muneeb	100%	31/10/2019	2
Requirements					
non-Functional Requirements	Goal	Aman	100%	04/11/2019	5
Work on SRS Document	Goal	Aman	100%	03/11/2019	18
Complete SRS Document	Milestone	Aman	100%	21/11/2019	1
Source Code					
Initial Design	Goal	Muneeb	100%	11/11/2019	5
Source Code - WiP	Goal	Muneeb	100%	11/11/2019	34
Source Code - Completion	Milestone	Muneeb	100%	15/12/2019	1
Test Plan					
Review Code	Goal	Aman	100%	16/12/2019	2
Test Plan - WiP	Goal	Aman	100%	18/12/2019	3
Test Plan Completion	Milestone	Aman	100%	22/12/2019	2
Testing					
Carry out Tests	Milestone	Aman and Muneeb	75%	24/12/2019	8
Report					
Gather all the Documents	Goal	Aman	100%	23/12/2019	5
Write the Report	Goal	Aman	100%	23/12/2019	9
Complete the Report	Milestone	Aman	100%	02/01/2020	1
Final Submission					
Review the Report	Goal	Aman and Muneeb	100%	02/01/2020	3
Submit the Report	Milestone	Aman and Muneeb	100%	05/01/2020	1



This page has been intentionally left blank

3. Software Requirement Specification

This document specifies the requirements for a simulation system required by an IT department for evaluation purposes.

3.1. Purpose

This document has been compiled to carry out Software requirement analysis and to clarify the specifications of a simulation system that are required by the IT department of a University. It provides a breakdown of the system and user requirements and the work that will be carried out to implement the simulation system.

This document consists of the following sections:

- **Chapter 3: Introduction**
  Identifies the purpose of this document and the context of the software to which this requirements document is written.
- **Chapter 4: Project Description**
  Breakdown of the project description and requirements.
- **Chapter 5: Simulator Information**
  Provides a breakdown of the system and user requirements.
- **Chapter 6: System Features**
  Provides functional requirements
- **Chapter 7: Non-Functional Requirements**
- **Chapter 8: Other Requirements**

3.2. Document Conventions

The use of "shall", "should", "must", "will" and "may" within the SRS observe the following rules:

- The word SHALL in the text denotes a mandatory requirement of the SRS. Departure from such a requirement is not permissible without formal agreement.
- The word SHOULD in the text denotes a recommendation or advice on implementing such a requirement of the document. Such recommendations or advice are expected to be followed unless good reasons are formally stated and accepted for not doing so.
- The word MUST in the text is used for legislative or regulatory requirements (e.g. Health and Safety) and shall be complied with.
- The word WILL in the text denotes a provision or service or an intention in connection with a requirement of the SRS.
- The word MAY in the text denotes a permissible practice or action. It does not express a requirement of the SRS.

3.3. Intended Audience and Reading Suggestions

This document is intended for the developer who will be writing the code for the simulation as well as the professor who will be assessing the work.

3.4. Product Scope

The purpose of this simulation is for an IT department in a University to be able to examine and evaluate the policies that will be set in place for job control and accounting purposes. The simulation will take into account a total of a 128 nodes with 16 processor cores per node and 8 of these nodes will be equipped with a GPU. The simulated users will include the IT support team, Researchers and Students.

3.5. References

Shown at the end of the report under “Bibliography”.

4. Product Description

4.1. Product Perspective

This is a new simulator, however some of the code that currently exists on the Cranfield Blackboard has been used. The simulator will include the following:

- **Booking details:**
  The simulator shall keep booking details, such as the size of the job, time required to complete the job and the total cost of the job.

- **User information:**
  It shall include the Customer information (Student, Researcher or IT team member). This will be used to keep a record of the customer and if they have enough credits left to be able to be able to carry out the job.

- **Reservation information:**
  It shall store the reservation information to keep track of the available and booked slots of time.

4.2. Product Functions

- Number of Jobs processed per week
- Machine hours consumed
- Utilisation ratio
- Price paid by the users
- Average wait time in the queue
- Turnaround time ratio
- Economic balance of the centre

4.3. User Classes and Characteristics

Users of the simulation shall be able to define the parameters of each run of the simulator, such as the number of users, distribution in classes and budgets. A member of the IT department of the University as well as the professor assessing the work will specifically use this simulation.

4.4. Operating Environment

Operating environment for the simulator is as follows:

- Operating System: Windows or Linux
- Platform: Crescent
- Programming Language: C++

4.5. Design and Implementation Constraints

The programming language this simulation has been written in is C++, it has to be able to be compiled on the Crescent computer. Although this simulation will only need to run on Crescent for the purposes of this assignment, Windows and Visual Studio can be used for the implementation.

4.6. Assumptions and Dependencies

Find some assumptions for the simulation below:

- Each student will have 1000 credits
- Each Researcher will have 1000 credits
- Each Group will have 1000 credits (Students and Researchers), regardless of the size of the Group
- Each Group shall have a unique ID number
- Students and Researchers are not part of a same group
- Use of each node would cost two credits
- Use of nodes equipped with GUI would cost 3 credits
- External Interface Requirements

4.7. User Interfaces

- Front end Software: Visual Studio 2019 or notepad (to be able to run in the command window)
- Back end Software: C++

4.8. Hardware Interfaces

- Windows or Linux
- Crescent

4.9. Software Interfaces

Software Used	**Description**
Windows	Windows has been used to build the simulation, however it will also be able to compile on Linux/Crescent and will be tested
Visual Studio (C++)	Visual Studio has been used to write the C++ code, one of the most well-known compilers for the C/C++ language

5. Simulator Information

5.1. Simulated Users

The simulator has been designed based on the information provided that are defined below:

- There shall be three kinds of simulated Users:
  - IT Support Department
  - Researcher
  - Student

Each simulated user shall have a budget, the simulated user shall be able to make bookings based on the amount of credit available to that user.

5.2. Classes

The Researcher class:

- A certain amount of resources must be available, based on the available credits.
- Each individual within the group shall also have their own credits to be able to carry out individual research.
- The researcher shall be able to make, amend and cancel bookings.

The Student class:

- The student shall be a part of a group dependent on the module that requires the resource.
- The student shall only be able to make bookings depending on the available credits.
- The student shall be able to make, amend and cancel bookings.

The IT support class:

- Shall have access to both Researcher and Student class.
- Shall have admin rights to be able to make changes to bookings.

The Curriculum Class:

- Shall have access to the student class.
- Shall have course information, such as the subject of the MSc or PhD.

The Scheduler Class:

- Shall have access to all the classes to be able to schedule a booking, to determine if a booking slot is available.
- Shall be able to determine which simulated user is making a booking.

The Job Class:

- Shall be able to determine if the job falls within short, medium, interactive, Large or Huge category.

The Request Class:

- Shall have access to the Job class to be able to determine the type of job request.

The Machine Class:

- The Machine class shall have access to the Job and Request classes.
- It shall be able to determine the processors and nodes available for the simulator.

The User Class:

- Shall have access to the Job and Request classes.

The Unknown Job Class:

- Shall call up an error message if the User enters the job type that is not recognised the simulator.

5.3. Required Nodes and Jobs

The simulation shall include 128 nodes with 16 processors per node. Eight of these nodes must be equipped with GPU for computation purposes. It shall also have storage capability and include a storage device.

- There shall be two kinds of nodes:
  - "Traditional" – shared-memory multicore CPUs
  - "Accelerated" – with GPUs

- There shall be five kind of jobs available:
  - Short job – Takes up to 2 nodes and no more than 1 hour to complete. 10% of the machine must be reserved for short jobs.
  - Medium job – Takes up to 10% of total available nodes (~13 nodes) and no more than 8 hours to complete. 30% of the machine must be reserved for medium jobs.
  - Large job – Takes up to 50% of total available nodes (64 nodes) and requires up to 16 hours to complete.
  - GPU – Any jobs that do require the GPU capability.
  - Huge job – May require all the available power of the machine and will only be allowed to run between 5pm on Friday to 9am on Monday. The above 4 jobs cannot be booked to run during this time.

6. System Features

The simulation system maintains information on bookings, amount of time required to run a job, available slots and user information. It is of medium priority, as although it is not safety critical, the research being carried out on the super computer can be ground breaking and can provide clearer insight into the subject of the research.

6.1. Simulator

6.1.1. Types of Simulated Users

This simulation tool shall have three types of users:

- IT support staff
- Researchers
- Students

This feature is of high priority, as the purpose of the IT department running this simulation is their need to understand how the system would respond once implemented for the super computer based on the type of user making the booking.

6.1.2. Functional Requirements

The system shall allow the user to carry out the following requirements:

REQ-1: It shall match job requests to resources, high priority.
REQ-2: It shall be able to calculate the amount of machine/node hours at the end of the week. High priority.
REQ-3: The utilisation ratio, medium priority.
REQ-4: Resulting price paid by the users, medium priority.
REQ-5: Average wait time in each queue, high priority.
REQ-6: Average turnaround time ratio, medium priority.
REQ-7: Economic balance of the centre, high priority.

6.2. Scheduler

6.2.1. Description and Priority

This feature is required to be able to schedule the jobs, this schedule will be first in first out (FIFO) and will run the queues. High priority.

6.2.2. Functional Requirements

For the purposes of this simulation, the functional requirements are as follows:

REQ-1: Shall be able to maintain queues, high priority.

REQ-2: Should be FIFO (first in first out), medium priority.

REQ-3: The interface shall be designed so other algorithms can be tested. High priority.

6.3. Input

6.3.1. Description and Priority

The user shall be able to input information to run the simulation or read text from a text file, high priority.

6.3.2. Functional Requirements

REQ-1: The user shall be able to input the total number of simulated users, high priority.

REQ-2: The user shall be able to allocate a budget to each simulated user, medium priority.

6.4. Output

6.4.1. Description and Priority

The system requires an output in order for the results to be monitored and compared, high priority.

6.4.2. Functional Requirements

REQ-1: The system shall display the users currently in the queue, medium priority.

REQ-2: The system shall be able to display the output from the simulation tool, high priority.

7. Non-Functional Requirements

7.1. Performance Requirements

- The performance of the simulation should be as such that it is able to measure the amount of time it takes to run and should not take a lot of memory to run.
- It shall not run in real time.
- The system should be able to do exception handling.

7.2. Software Quality Attributes

- The system shall be portable to be able to run on various machines and operating systems. It shall be able to be compiled and executed on Crescent (Linux) as well as Windows. This would be relatively simple to use as long as the user understand how to use Linux on crescent.

- The source code shall be clean and with clear comments for maintainability purposes. This would require extra vigilance when writing the code, but it is of high importance.
- The simulator must be reliable and correct, as this would lead to a system that will be used for the Super computer in the future.

7.3. Business Rules

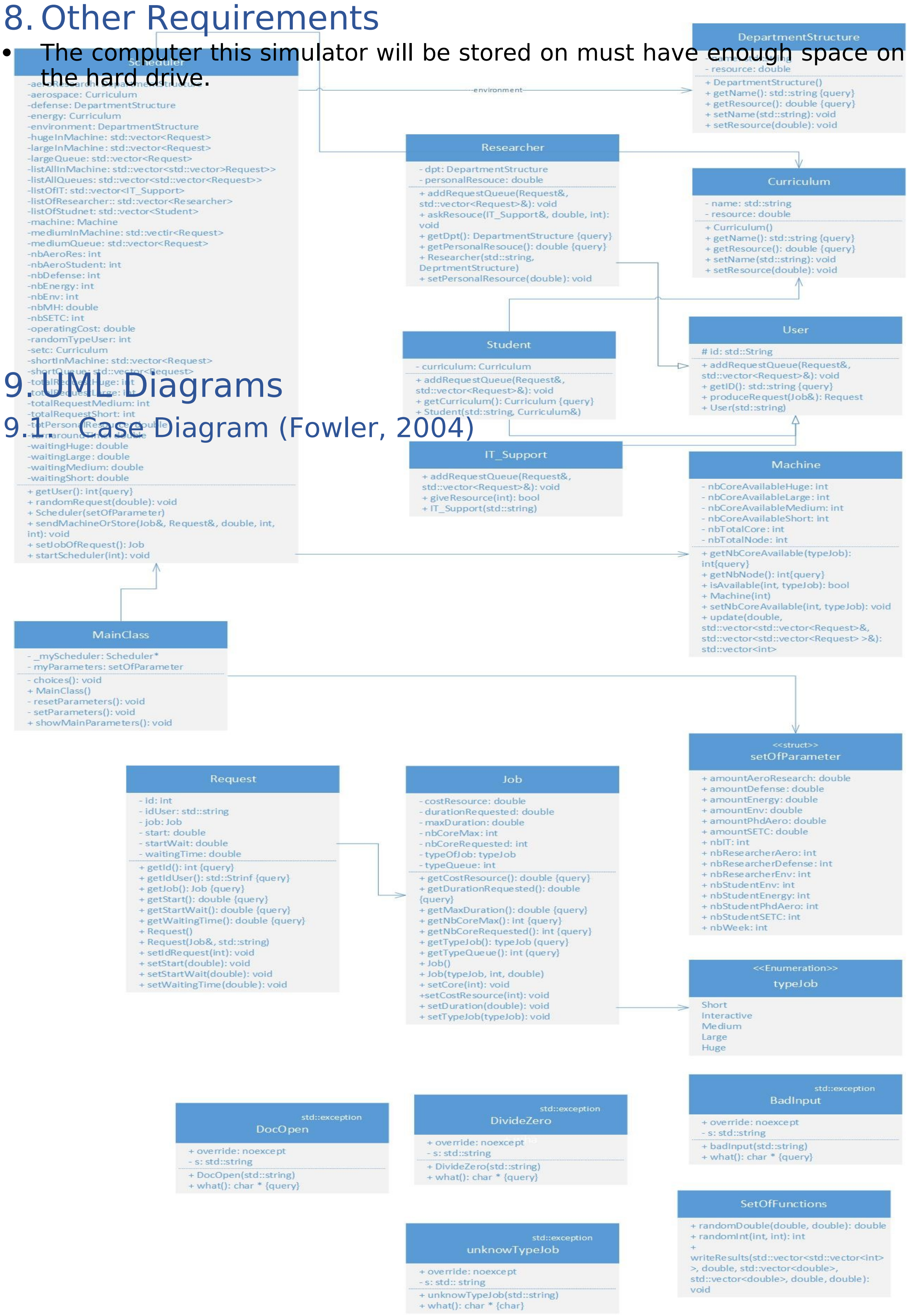
- As this simulation is specifically designed for the IT department, only the IT department will have access to using this simulator.

8. Other Requirements

- The computer this simulator will be stored on must have enough space on the hard drive.

9. UML Diagrams

9.1. Case Diagram (Fowler, 2004)



Scheduler
-aeroResearch: DepartmentStructure
-aerospace: Curriculum
-defense: DepartmentStructure
-energy: Curriculum
-environment: DepartmentStructure
-hugeInMachine: std::vector<Request>
-largeInMachine: std::vector<Request>
-largeQueue: std::vector<Request>
-listAllInMachine: std::vector<std::vector>Request>>
-listAllQueues: std::vector<std::vector<Request>>
-listOfIT: std::vector<IT_Support>
-listOfResearcher:: std::vector<Researcher>
-listOfStudnet: std::vector<Student>
-machine: Machine
-mediumInMachine: std::vectir<Request>
-mediumQueue: std::vector<Request>
-nbAeroRes: int
-nbAeroStudent: int
-nbDefense: int
-nbEnergy: int
-nbEnv: int
-nbMH: double
-nbSETC: int
-operatingCost: double
-randomTypeUser: int
-setc: Curriculum
-shortInMachine: std::vector<Request>
-shortQueue: std::vector<Request>
-totalRequestHuge: int
-totalRequestLarge: int
-totalRequestMedium: int
-totalRequestShort: int
-totPersonalResource: double
-turnaroundTime: double
-waitingHuge: double
-waitingLarge: double
-waitingMedium: double
-waitingShort: double
+ getUser(): int{query}
+ randomRequest(double): void
+ Scheduler(setOfParameter)
+ sendMachineOrStore(Job&, Request&, double, int, int): void
+ setJobOfRequest(): Job
+ startScheduler(int): void
environment
DepartmentStructure
- name: std::string
- resource: double
+ DepartmentStructure()
+ getName(): std::string {query}
+ getResource(): double {query}
+ setName(std::string): void
+ setResource(double): void
Researcher
- dpt: DepartmentStructure
- personalResouce: double
+ addRequestQueue(Request&, std::vector<Request>&): void
+ askResouce(IT_Support&, double, int): void
+ getDpt(): DepartmentStructure {query}
+ getPersonalResource(): double {query}
+ Researcher(std::string, DeprtmentStructure)
+ setPersonalResource(double): void
Curriculum
- name: std::string
- resource: double
+ Curriculum()
+ getName(): std::string {query}
+ getResource(): double {query}
+ setName(std::string): void
+ setResource(double): void
Student
- curriculum: Curriculum
+ addRequestQueue(Request&, std::vector<Request>&): void
+ getCurriculum(): Curriculum {query}
+ Student(std::string, Curriculum&)
User
# id: std::String
+ addRequestQueue(Request&, std::vector<Request>&): void
+ getID(): std::string {query}
+ produceRequest(Job&): Request
+ User(std::string)
IT_Support
+ addRequestQueue(Request&, std::vector<Request>&): void
+ giveResource(int): bool
+ IT_Support(std::string)
Machine
- nbCoreAvailableHuge: int
- nbCoreAvailableLarge: int
- nbCoreAvailableMedium: int
- nbCoreAvailableShort: int
- nbTotalCore: int
- nbTotalNode: int
+ getNbCoreAvailable(typeJob): int(query)
+ getNbNode(): int{query}
+ isAvailable(int, typeJob): bool
+ Machine(int)
+ setNbCoreAvailable(int, typeJob): void
+ update(double, std::vector<std::vector<Request>&, std::vector<std::vector<Request> >&): std::vector<int>
MainClass
- _myScheduler: Scheduler*
- myParameters: setOfParameter
- choices(): void
+ MainClass()
- resetParameters(): void
- setParameters(): void
+ showMainParameters(): void
<<struct>>
setOfParameter
+ amountAeroResearch: double
+ amountDefense: double
+ amountEnergy: double
+ amountEnv: double
+ amountPhdAero: double
+ amountSETC: double
+ nbIT: int
+ nbResearcherAero: int
+ nbResearcherDefense: int
+ nbResearcherEnv: int
+ nbStudentEnv: int
+ nbStudentEnergy: int
+ nbStudentPhdAero: int
+ nbStudentSETC: int
+ nbWeek: int
Request
- id: int
- idUser: std::string
- job: Job
- start: double
- startWait: double
- waitingTime: double
+ getId(): int {query}
+ getIdUser(): std::Strinf {query}
+ getJob(): Job {query}
+ getStart(): double {query}
+ getStartWait(): double {query}
+ getWaitingTime(): double {query}
+ Request()
+ Request(Job&, std::string)
+ setIdRequest(int): void
+ setStart(double): void
+ setStartWait(double): void
+ setWaitingTime(double): void
Job
- costResource: double
- durationRequested: double
- maxDuration: double
- nbCoreMax: int
- nbCoreRequested: int
- typeOfJob: typeJob
- typeQueue: int
+ getCostResource(): double {query}
+ getDurationRequested(): double {query}
+ getMaxDuration(): double {query}
+ getNbCoreMax(): int {query}
+ getNbCoreRequested(): int {query}
+ getTypeJob(): typeJob {query}
+ getTypeQueue(): int {query}
+ Job()
+ Job(typeJob, int, double)
+ setCore(int): void
+setCostResource(int): void
+ setDuration(double): void
+ setTypeJob(typeJob): void
<<Enumeration>>
typeJob
Short
Interactive
Medium
Large
Huge
std::exception
BadInput
+ override: noexcept
- s: std::string
+ badInput(std::string)
+ what(): char * {query}
std::exception
DocOpen
+ override: noexcept
- s: std::string
+ DocOpen(std::string)
+ what(): char * {query}
std::exception
DivideZero
+ override: noexcept
- s: std::string
+ DivideZero(std::string)
+ what(): char * {query}
SetOfFunctions
+ randomDouble(double, double): double
+ randomInt(int, int): int
+ writeResults(std::vector<std::vector<int>>, double, std::vector<double>, std::vector<double>, double, double): void
std::exception
unknowTypeJob
+ override: noexcept
- s: std:: string
+ unknowTypeJob(std::string)
+ what(): char * {char}

fig 1: Class Diagram

The Unified Modelling Language (UML), is a standard that is used industry wide specifically for the process of system modelling. The purpose of the UML is for developers to carry out the design work and think about the software requirements before writing the source code (Fowler, 2004).

10. Test Plan Overview

This test plan has been created for the Simulator of a University supercomputer, the purpose of the document is to plan, organise, execute and manage the testing of this project. The simulator is for an IT department in a University to be able to examine and evaluate the policies that will be set in place for job control and accounting purposes. The simulation will take into account a total of a 128 nodes with 16 processor cores per node and 8 of these nodes will be equipped with a GPU. The simulated users will include the IT support team, Researchers and Students.

11. Testing Synopsis (Langr, 2013)

11.1. Parts to be tested

- Simulation Tool
- Scheduler
- Input
- Output
- Performance testing
- Integration testing
- System testing
- Configuration and Compatibility testing
- Code Coverage

11.2. Systems Requirement

As this project is for a simulator that will be used by the IT department in a University, it does not require any expensive testing hardware. The simulator will be tested in the following environments:

- Standard off the shelf Alienware Laptop
  Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz, 2801 Mhz, 4 Core(s), 8 Logical Processor(s). Running Windows 10 operating system on a 64 bit system.
- Standard computer at Cranfield University – Crescent system running Linux Red hat operating system.

11.3. Standard/ Reference material

- The simulator will be developed on C++17 standard, Windows 10 operating system using Visual basic community edition 2019.

12. Types of Testing

12.1. Acceptance testing

Smoke testing will be carried out to set out the criteria. The purpose of carrying out a smoke test is to determine if further testing would be relevant at this stage, as the smoke tests determines if the software works at a high level (S, 2018) . The following parts will be tested:

- Simulator compiles on the computer it was developed.
- Simulator compiles on the partner's computer.
- Simulator compiles on a computer at the University.
- Simulator compiles on Crescent.
- Functional test will be carried out to test that all the function provide the expected results.

12.2. Feature Level Testing

12.2.1 Simulator

- SRS document, section 4.1.2: "It shall match job requests to resources."
- SRS document, section 4.1.2: "It shall be able to calculate the amount of machine/node hours at the end of the week."
- SRS document, section 4.1.2: "The utilisation ratio."
- SRS document, section 4.1.2: "Resulting price paid by the users."
- SRS document, section 4.1.2: "Average wait time in each queue."
- SRS document, section 4.1.2: "Average turnaround time ratio."
- SRS document, section 4.1.2: "Economic balance of the centre."

12.2.2. Scheduler

- SRS document, section 4.2.2, "Shall be able to maintain queues."
- SRS document, section 4.2.2: "Should be FIFO."
- SRS document, section 4.2.2: "The interface shall be designed so other algorithms can be tested."

12.2.3. Input

- SRS document, section 4.3.2: "The user shall be able to input the total number of simulated users."

- SRS document, section 4.3.2: “The user shall be able to allocate a budget to each simulated user.”

12.2.4. Output

- SRS document, section 4.4.2: “The system shall display the users currently in the queue.”
- SRS document, section 4.4.2: “The system shall be able to display the output from the simulation tool.”

12.2.5. Integration testing

All the classes that can logically be grouped together will be tested, the following provides a breakdown, (Approach & Integration, 2019):

Test Case Objective	Test Case Description	Expected Result
Each class to be tested	All the classes will be built and tested individually	To provide the amount of lines that run per class
Check the link between User and the IT support, Student and Researcher classes one at a time	Enter the user information, to check if it recognises all the different users	The User class recognises IT support, Student and Researcher classes as simulated users
Check the link between the Simulator class and Scheduler class	Run the Simulator class with the Scheduler class	FIFO system works and provides the queuing times of the simulated users
Check the link between the jobs class and simulator class	Enter the job information to check if the simulator class recognises all the various jobs	If correct information is inserted, the simulator should be able to recognise the job
Exception handling	Put in incorrect information	The simulator calls up an error, such as letting a user book a slot even if they don’t have enough credits.
All the functional requirements, mentioned in section 3.2	Check all the functional requirements one at a time	The simulator meets all the functional requirements
Enter user credit info that is not currency specific	Enter the used credit with a currency sign	Should call up an error as the credits should not be currency specific

In order to carry out the testing above, the Junit, Google primer or Visual studio test framework will be used as these will run automated tests to provide a breakdown of the amount of lines that are run during Unit and Integration testing.

12.2.6. System Testing (Guru, 2019)

Test Case Objective	Test Case Description	Expected Result
Usability test	The simulator will be run with necessary required	Purpose is to see if the simulator is user friendly

	inputs	
Recovery testing	The computer will be shut down in the middle of the code being compiled	To see if the program is affected and recoverable
Migration testing	Simulator will be run on various computers and operating systems	It is expected that the code will compile on all the various machines and operating systems

12.2.7. Performance Testing

- SRS document, section 5.1: "The performance of the simulation should be as such that it is able to measure the amount of time it takes to run and should not take a lot of memory to run."
- SRS document, section 5.1: "It shall not run in real time, as the time will be measured using a simple a counter."
- SRS document, section 5.1: "The system should be able to do exception handling."

12.3. Configuration and Compatibility Testing

- The simulator will be tested on various machines.
- It will be tested to make sure it compiles on Windows, Visual code as well as the Crescent computer at Cranfield University, i.e. Linux.

12.4. Code Coverage

- Unit test and Integration test results will be able to provide information on the code coverage and how many lines of the simulator have run in terms of percentage.

13. State Transition Testing

13.1. State Transition Diagram

This software requires input from the user to display the results. The software has standard parameters that are already set, and the simulation can be run with standard parameters. There is also a possibility to change the parameters manually by the user, however for testing purposes the simulation will be tested with standard parameters already set in the software.

There are two inputs required from the user in order to run the Simulator:

2 required inputs	**Output**
1) Change in parameters	Displays set parameters
2) Run the simulation	Runs the simulator and saves the result

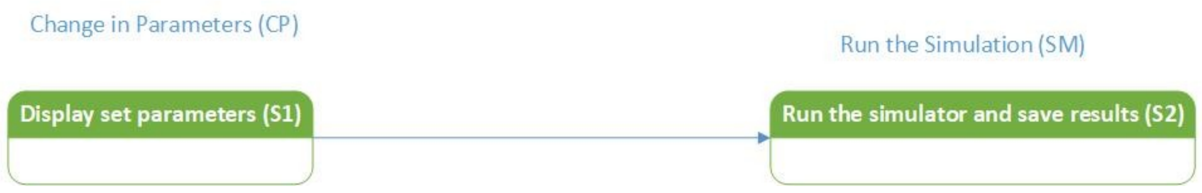

Change in Parameters (CP)
Run the Simulation (SM)
Display set parameters (S1)
Run the simulator and save results (S2)

Fig 2: State Transition Diagram

13.2. State Transition Tree.

The state transition tree for the above diagram is the following:

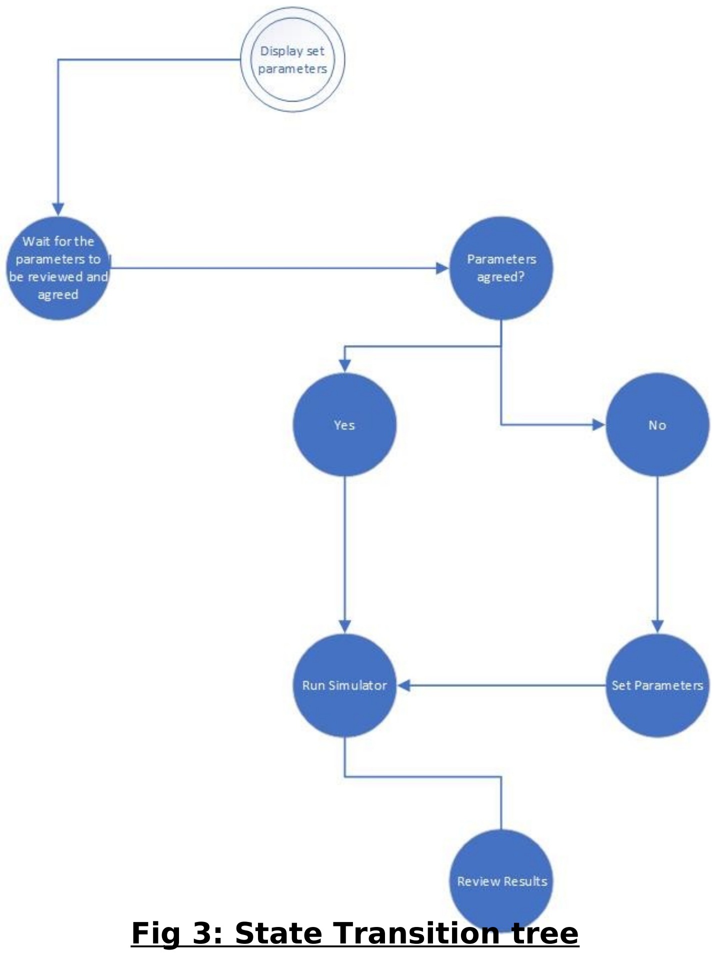


Fig 3: State Transition tree

The above method has been followed to carry out black box testing on the simulator.

14. Test Schedules and Resources

The schedule provided below is tentative and will be worked towards to be met:

- This document will be completed by the 23rd of December 2019.
- The test execution is not expected to last more than a week and will commence once the test plans have been completed. The current tentative date for the test execution to begin is the 24th of December 2019.
- A project plan has been put together in MS Excel that is shared at the beginning of this report.

- Time constraints can affect the completion and the quality of the software.

15.Test Phases and Completion Criteria

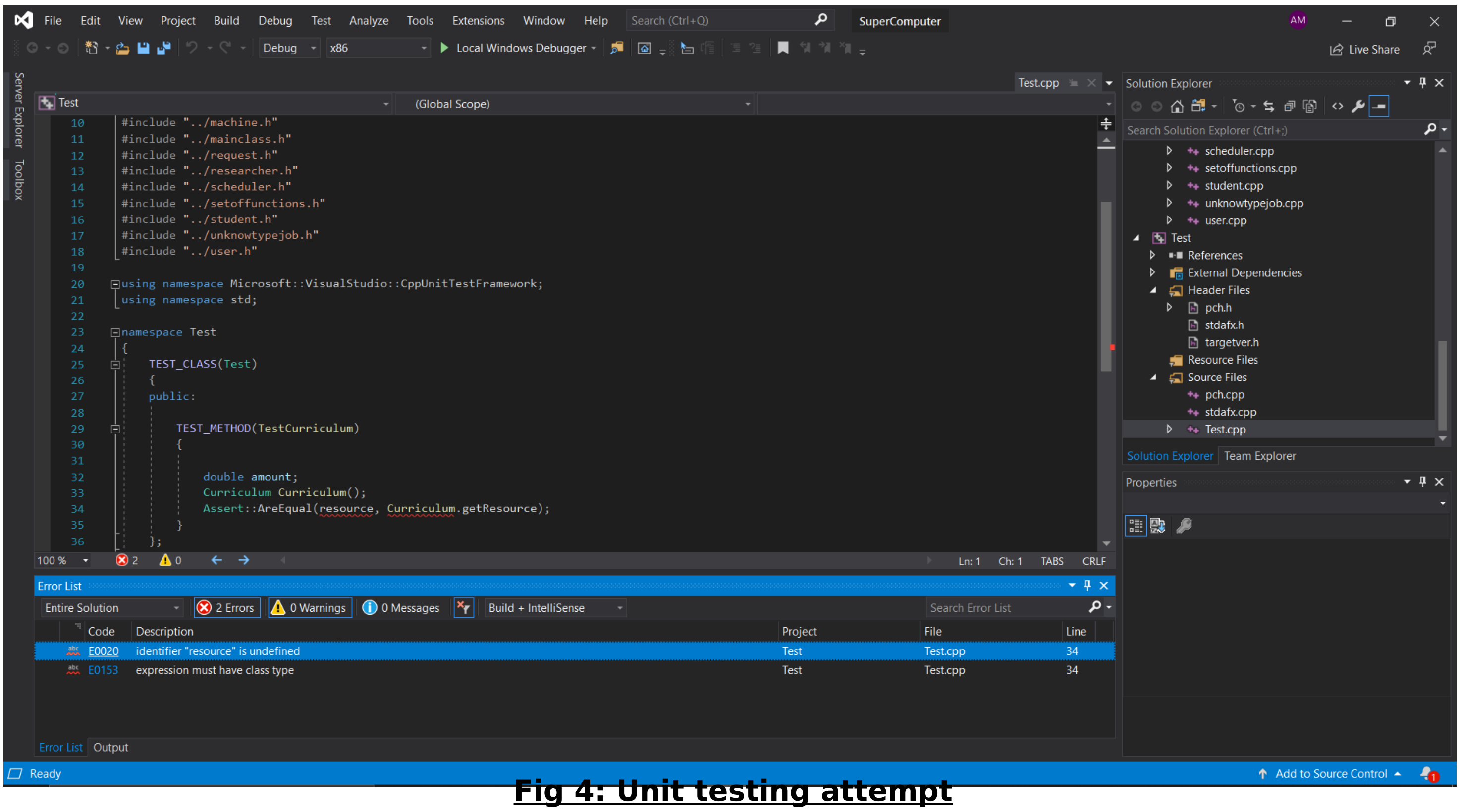
STLC Stage	Entry Criteria	Activity	Exit Criteria	Deliverables
Requirements Analysis	Requirements documents available.	Analyse the requirement and determine functional and non functional requirements that are module specific Identify types of tests to be performed	Mutual agreement between the partners	SRS document
Test Planning	Requirements documents	Analyse testing approaches Finalise the testing approach that is suitable Gather data for test plan Select a tool for testing	A test plan that has been agreed between the partners.	Test plan
Test Case Development	Requirements documents	Create test cases	Review and agree the test cases between the partners	Test cases
Test Environment Set up	Environment set up plan	Understand the environment requirements Perform smoke test	The environment set up works as expected Smoke test provides expected results	Smoke test results Environment set up for testing
Test Execution	Test plan and test cases Test Environment is kept ready	Run all the tests as per the plan Any failed tests should be logged and	All the tests in the plan have been executed	Reports with defects and coverage

	Unit testing report is available	documents provided		

16.Discussion

Following on from various steps that have been followed above to the completion of this project, it should be noted that it has not been possible to carry out unit testing due to time constraints. “Black box” testing has been carried out as per the test cases mentioned above in the test plan. The black box testing has been carried out using the “Big Bang” approach (Rossel, 2017), where various classes have been tested together to get the results.

Carrying out the work above has also led to further understanding that in some cases it is not possible to carry out unit testing based on the complexity of the code. As the standard Visual studio testing framework was used to carry out the unit testing, it constantly shows up an error and due to time constraints, it has not been possible to resolve the issue. A screenshot is provided below:



```
#include "../machine.h"
#include "../mainclass.h"
#include "../request.h"
#include "../researcher.h"
#include "../scheduler.h"
#include "../setoffunctions.h"
#include "../student.h"
#include "../unknowtypejob.h"
#include "../user.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;
using namespace std;

namespace Test
{
    TEST_CLASS(Test)
    {
    public:

        TEST_METHOD(TestCurriculum)
        {

            double amount;
            Curriculum Curriculum();
            Assert::AreEqual(resource, Curriculum.getResource);
        }
    };
```

Error List — Entire Solution — 2 Errors — 0 Warnings — 0 Messages — Build + IntelliSense

Code	Description	Project	File	Line
E0020	identifier "resource" is undefined	Test	Test.cpp	34
E0153	expression must have class type	Test	Test.cpp	34

Fig 4: Unit testing attempt

Running the code provides the required results as per the user requirements. The simulator provides the results for the following requirements:

- The number of jobs processed in each queue.
- The number of node-hours consumed by the machine.

- Waiting time per queue.
- Average turnaround time.
- The economic balance of the centre.
- Price paid by the users.

Furthermore, the simulator has been tested to make sure that it runs on Windows using Visual Studio 2019 and Makefiles have also been created and tested on Linux (Crescent).

The aid of UML diagrams has also been used for the software development process, however in this instance the Class diagram was built after the source code had been completed rather than during the design phase. In the future, the class diagram will be completed during the design phase so the source code can be written that follows the design stated in the Class diagram.

The future work to be carried out on this project would be to design the source code in such a way that unit testing is possible to carry out. Although, the simulator currently provides the required results and meets the user requirements, it would be difficult to manage, and without unit testing it is not possible to determine the code coverage and get to the core understanding of each class.

17.Conclusion

The work carried out above has followed the industry standard as per the IEEE SRS document template and the test plan documents, the steps followed are as follows:

- Review the project and assign tasks.
- Make and follow a project plan.
- Carry out the design work for the project.
- SRS document.
- Source code.
- Test plan.
- Testing.

A thorough amount of work has been undertaken to design and write the source code for the Simulator. Although no unit testing has been carried out, system and integration testing has been carried out to determine that the simulator meets the User requirements. The test results are saved in a .CSV file that can reviewed by the IT team to evaluate the policies of the batch submission system, with job control and accounting.

The amount of work required to design, test and implement is substantial as can be seen from the documents and source code that has been completed for this report. Following a project plan is of utter importance to keep the project within the required timeline and to meet the required standards.

Bibliography

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. M. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional.

Sommerville, I. (2016). Software engineering. Pearson.

Approach, I., & Integration, B. (2019). Integration Testing: What is, Types, Top Down &amp; Bottom Up Example. 1–12. https://www.guru99.com/integration-testing.html#1

Fowler, M. (2004). UML Distilled: A Brief Guide to the Standard Object Modeling Language. Pearson Paravia Bruno Mondad, 175. https://doi.org/10.1109/MS.2005.81

Guru. (2019). What is System Testing? Types &amp; Definition with Example. 2019. https://www.guru99.com/system-testing.html

Langr, J. (2013). Modern C++ Programming with Test-Driven Development. In Journal of Chemical Information and Modeling (Vol. 1). Pragmatic Bookshelf. https://pragprog.com/book/lotdd/modern-c-programming-with-test-driven-development

Rossel, S. (2017). Continuous Integration, Delivery, and Deployment. https://learning.oreilly.com/library/view/continuous-integration-delivery/9781787286610/

S, S. (2018). What is User Acceptance Testing (UAT): A Complete Guide. Software Testing Help. https://www.softwaretestinghelp.com/what-is-user-acceptance-testing-uat/