NOTTINGHAM TRENT UNIVERSITY

School of Science and Technology

COURSEWORK ASSESSMENT SPECIFICATION (UG)

In this specification, you will find information about:

MODULE CODE	SOFT20091
MODULE TITLE	Software Design and Implementation 2
MODULE LEADER	Pedro Machado
TUTOR(S)	David Adama Salim Maaji Kayode Owa
COURSEWORK TITLE	SDI2 Portfolio
LEARNING OUTCOMES ASSESSED	All
CONTRIBUTION TO ELEMENT	100% of the module, element 1
DATE SET	04th of November 2019
DATE OF SUBMISSIION -	23:59 of the 19th of January 2020 (Project Manager) 23:59 of the 9th of February 2020 (Software Architect) 23:59 of the 1st of March 2020 (Software Developer) 23:59 of the 29th of March 2020 (Software Tester) 23:59 of the 26th April 2020 (Project Manager)
METHOD OF SUBMISSION	Dropbox via NOW
DATES OF FEEDBACK	20th or 27th of January 2020 10th or 17th of February 2020 2nd or 9th of March 2020 30th of March or 6th of April 2020

	15th of May 2019
METHOD OF FEEDBACK	SDI2 Portfolio

Work handed in up to five working days late will be given a maximum Grade of Low 3rd whilst work that arrives more than five working days will be given a mark of zero.
Work will only be accepted beyond the five working day deadline if satisfactory evidence, for example, an NEC is provided. Any issues requiring NEC
https://ntu.ac.uk/current_students/resources/student_handbook/appeals/index.html

The University views **plagiarism and collusion** as serious academic irregularities and there are a number of different penalties which may be applied to such offences. The **Student Handbook** has a section on Academic Irregularities, which outlines the penalties and states that **plagiarism** includes:
'The incorporation of material (**including text, graph, diagrams, videos etc.**) derived from the work (published or unpublished) of another, by unacknowledged quotation, paraphrased imitation or other device in any work submitted for progression towards or for the completion of an award, which in any way suggests that it is the student's own original work. Such work may include printed material in textbooks, journals and material accessible electronically for example from web pages.'
Whereas **collusion** includes:
“Unauthorised and unacknowledged copying or use of material prepared by another person for use in submitted work. This may be with or without their consent or agreement to the copying or use of their work.”
If copied with the agreement of the other candidate both parties are considered guilty of Academic Irregularity.

Penalties for Academic irregularities range from capped marks and zero marks to dismissal from the course and termination of studies.

To ensure that you are not accused of plagiarism, look at the sections on **Plagiarism Support** and **Turnitin** support.

I. Assessment Requirements

The assignment is a group coursework (with some individual tasks, see Section 1 and 2). Given the problem scenario in Section 2, the group is required to produce a collective report of the designed and implemented software application. A software demo is required and will be organized during the usual timetabled lab sessions in Teaching Week 40 and 41 (week starting Monday 27/04/20 and 04/05/20). The portfolio (one per group) **MUST** be submitted electronically through NOW Dropbox by Sunday 26th April 2020 23:59 (Teaching Week 39):

- Students **MUST** use the following convention for naming their folders and files:'Group_X_SDI_Report' (for example, Group_X wher X is name of the group –report for the assignment of the module SOFT20091). Students are also advised to add the same information to the header section of your submitted document.
- All files **MUST** be submitted in a single, main folder per group.
- The zipped file **MUST** have the following structure:
  - SRC containing all the code files (mandatory);
  - TESTS with any test dataset/database or readme file with link to it (mandatory);
  - BUILD containing the build files:
  - RESULTS with results material/images of the software.
  - DOCUMENTATION containing your report in the PDF format.
- The code **MUST** also be stored in a git repository. Please refer to Lab 2 to learn how to setup your git repository. Students are encouraged to add links to the main git repository in the report. This will enable you to recover quickly should the system fail and allow you to backtrack if your development goes astray.
- Ensure that the work submitted will execute on any University computers.
- Keep evidence of the submission of your assignment, and a copy of your assignment in case of the unlikely event of any loss.

Special Instructions “Ground rules” for GroupWork:

- Students should enrol online via the learning room until 31st of October 2019. After that students will be automatically allocated to groups of 4.
- Students might be allowed to swap groups under extenuating circumstances which MUST be presented before the 13th of December 2019.
- Managing group work is part of the assessment and a brief paragraph of the report should summarise this experience (in Appendix) you could highlight when you apply for placements or job positions. Students are advised to document their group activities or keep evidence of them.
- They could include (some of) them in an Appendix into the report.
- Please upload a single text file named by your group name and listing your group members (names and studentIds) as content to Dropbox on NOW.
- Changes on memberships of work groups will not be allowed AFTER the week commencing in 13th December 2019). Anyone not having a group by then will be randomly assigned to a group, so please do seek your group mates ASAP. Only one of the team members should submit the group work in time by uploading the folder with all the components into NOW (see also above-mentioned explanations).
- The group work has group-based tasks as well as individual tasks. For each group, all students will have the same mark for the group on the group-based tasks, unless the members have poor contributions.
- The individual tasks would be marked for the member who completed them. Each group member will have to submit his deliverable(s) by the due dates.

For how the mark is determined by the contribution, please refer to the form.

II. Assessment Scenario/Problem

Artificial Intelligence (AI) is a hot research/innovation topic and aims to replicate some human capabilities in machines. AI is being applied in many research/innovations areas including Computer Vision. Convolutional Neural Networks (CNN) are being used for performing online/offline object classification. CNNs, like other types of Neural Networks and Machine Learning algorithms, MUST be trained on a dataset. Datasets are a collection of data, with the same characteristics as the data that will be used for classification, that needs to be annotated (i.e. identify the region of interest) using a labelling tool.

For this assignment, you are expected to implement a labelling application (software) using C++ for annotating Datasets for being used by CNNs. You are not expected to implement or use a CNN for this assignment; your implemented application just needs to provide the appropriate labelling functionality. The label application MUST include the following functionalities:

1. Have a simple GUI (see Figure 1) where the user can specify the following:
    a. The target folder containing the photos using buttons for browse the folder available in the Operating System;
    b. A pane listing the **compatible image files** (e.g. *.jpg, *.png) available in that folder and with the following sorting options:
        i. Sort descending by file name;
        ii. Sort ascending by file name;
        iii. Sort descending by file date;
        iv. Sort ascending by file date;

       Please use the sorting algorithms during the 1st term labs.
    c. Classes selector with a browse button so that the user can navigate through the folders and select the class files identified which are plain text files with the extension "*.names" where each line corresponds to a class; Classes are groups of people or objects (e.g. car, dog, cat, person, etc.)
    d. All the classes should be listed in a classes pane with the possibility to sort ascending or descending; The file line number MUST be preserved because the order cannot be changed in the classes file.
    e. Users should be able to add and remove classes which MUST be appended to the classes file.
    f. The user MUST be able to select and use one of the following shapes options:
        i. Triangle
        ii. Square/rectangle;
        iii. Trapezium;
        iv. Polygon (with up to 8 points);
    g. The user MUST only use the provided shapes for annotating the given images. The user must be able to select any image and draw the shape on the top of the image. Only the borders should be visible (no fill).
    h. The annotations (shapes) should be displayed on the image;
    i. Annotations file and filename selector with the following options:
        i. Open and load annotations file (*.annotations)
        ii. Save the annotation file. A warning should be displayed to the user and ask the user to confirm the overwrite of the file if a file already exists.
        iii. Change the name of an existing file.
        iv. The annotations files MUST follow the hierarchical data format 5 (HDF5[1]) standard;
        v. The following data MUST be stored in each annotation file:
            1. Number of annotated images;

               For each Image
            2. Image file name;

[1] Available online, https://portal.hdfgroup.org/display/HDF5/HDF5, last accessed 11/10/2019

    3. Number of shapes per image;
       For each shape
    4. Shape type;
    5. Point_1 (x,y);
    6. Point_2 (x,y);
    7. Point_n (x,y);

j. The selected image should be displayed[2] in the image pane;

k. The user should be able to perform the following shape operation using the mouse:
    1. Increase the size;
    2. Move the vertices of polygons;
    3. Delete shape;
    4. Copy and paste shape;
    5. Visualise the name of the class on the top of the shape.

l. The application should automatically save the annotation file every minute; The autosave MUST be done using threads;

m. MUST use the data structures developed during the 1st term labs for storing data in memory;

n. Groups MUST use a sort and a search algorithm, implemented in the 1st terms labs;

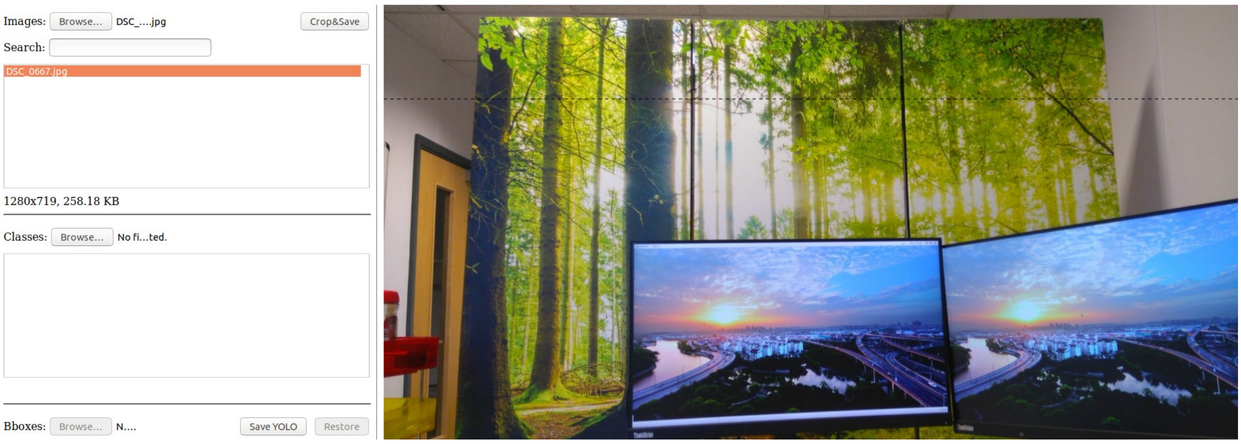

Figure 1: Example of a simple GUI

1. Tasks

Design

The team is composed of a project manager (member A), software architect (member B), software developer (member C) and software tester (member D). Each team should define a role for each member (i.e. A – D). All the members must have equivalent workloads and all the decisions should be done in a democratic way. Lab tutors will have the quality vote if no consensus is obtained about particular aspects of the design, implementation or testing.

Implementation

You are expected to implement the above software system using C++. The final application MUST meet the following requirements:

- at least one of the data structures (e.g. queues/stacks/graphs/trees) proposed in lectures;
- at least one of the sorting or searching algorithms studied during the lectures;
- at least one place where error and exceptions handling have been used;

[2] Please use OpenCV, https://opencv.org/, tools for working with images.

- at least one class where private member functions for common functionality that should not be exposed in the public interface;
- at least a simple GUI interface that allows user to interact with the software system;
- appropriate testing cases and testing library;

Data needs to be stored permanently in ASCII (text) files on the local hard drive, hence functions for loading data from files and for saving data from memory into a file are required. The files could be comma separated values (CSV).

The software should work as expected, i.e., if the software does not work then the mark could be reduced even if the individual task is completed well.

The classes need to reflect on the chosen architecture style, e.g., if MVC (recommended) is chosen, then it is expected to have classes for Controller, View and Model (including any sub classes).

Concurrent programming should be used when appropriate.

2. Deliverables

The project manager is responsible for submitting a project plan, the software architect is responsible for submitting a project design, the software developer is responsible for submitting a reference manual, the software tester the test plan and all the members must submit the final report. The five deliverables must be submitted electronically via Dropbox on NOW.

Deliverables submissions and deadlines:

1. Project plan must be submitted by the 19th of January 2020 23:59 via the NOW dropbox. The project plan MUST include the requirements lists using the table below and Gantt chart. All the members should contribute. **The Project Manager is responsible for preparing, submitting and presenting the project plan. The Project Manager is responsible for presenting the project plan in the lab running on the 20th or 27th of January 2020.** The feedback will be given in class by the Lab tutor immediately after the presentation.

Requirement	Description	Implications	Tasks

2. Project design must be submitted by the 9th of February 2020 23:59. The project design MUST include the following UML diagrams:
Use case diagram(s) and provide an explanation.
Class Diagram(s) and provide an explanation.
Sequence Diagram(s) and provide an explanation.
State Diagram(s) and provide an explanation.
Component Diagram(s) and provide an explanation.

The project degn MUST also include a list of the libraries and tools that will be used and the main GUI mock-up. **The Software Architect is responsible for preparing, submitting and presenting the project design. The Software Architect is responsible for presenting the project design in the lab running on the 10th or 17th of February 2020.** The feedback will be given in class by the Lab tutor.

3. Reference manual must be submitted by the 1st of March 2020 23:59. The reference diagram MUST include the detailed documentation of all the code developed and explaining what other implementations need to be done. DOXYGEN (or equivalent tool) SHOULD be used for generating the reference manual. Please include a code contribution guide explaining how the git commits are done and how often, the adopted coding rules, etc. **The Software Developer is responsible for preparing, submitting and presenting the reference manual. The Software Developer is responsible for presenting the reference manual in the lab running on the 2nd or 9h of March 2020.** The feedback will be given in class by the Lab tutor immediately after the presentation.

4. Test Plan must be submitted by the 29th of March 2020 23:59. The test plan MUST include unit, integration, functional and acceptance tests. **The Software Tester is responsible for preparing,**

submitting and presenting the test plan. The Software Tester is responsible for presenting the test plan in the lab running on the 30th of March or 6th of April 2020. The feedback will be given in class by the Lab tutor immediately after the presentation.

Table 1: Test scenarios description

ID		Description:	
Test type	Quantity/Quality	Success criteria:	
Number of attempts:		Comments:	
List of equipment / requirements			
Setup instructions			
Failure correction procedure			
Engineer(s)/Technician(s)			
Individual results:			

5. Final report must be submitted until Sunday 26th April 2020 23:59. All elements MUST contribute in the preparation of the final report. The **project manager** is responsible for the submission of the final report. Development report (individual tasks are marked as *, otherwise, it is a group-based task). You will deliver a structured report. The report **MUST** include the following structure:
    a. Cover page identifying the group members and the lab tutor;
    b. Abstract;
    c. Revision history;
    d. List of contents;
    e. List of tables;
    f. List of Images;
    g. Introduction;
    h. Background research – provide an overview of the external tools and libraries that were used to implement your application (e.g. OpenCV, QT designer, OpenGL, etc.).
    i. Design including the following information:
        a. list of requirements and restrictions using the SWOT[3] or MoSCoW analysis;
        b. list of tasks;
        c. time plans;
        d. assumptions;
        e. adopted coding standards;
        f. other relevant information.
    j. Implementation details. Please DO NOT add any code in your report. Use instead the UML diagrams and pseudo-code. Please explain all the diagrams. Note that the figure's caption is shown at the bottom of the figure.
    k. All the test Scenarios **MUST** be specified using Table 2. Note that the tables caption is shown at the top of the table.

Table 2: Test scenarios description

ID		Description:	
Test type	Quantity/Quality	Success criteria:	
Number of attempts:		Comments:	
List of equipment / requirements			
Setup instructions			
Failure correction procedure			

[3] Available online

Engineer(s)/Technician(s)			
Individual results:			
Test Date		Result	

l. Please provide details about each of the unit, integration, functional and acceptance tests.
m. Results
    a. Include screenshots of your application and EXPLAIN every image. All the figures MUST contain captions and be mentioned in the text. Note that the figure's caption is shown at the bottom of the figure.
    b. Include YouTube links (one per each test) as footnotes using the following format "Available online, <url>, last accessed <date>".
n. Conclusions and Future work
o. References List using the Harvard citations style.
p. Individual contributions per member.
q. Overall reflection of the work done
r. Appendix (include the reference manual)

Please include the following information:

1. A table of contents page and identifying who has contributed to which individual tasks.
2. A general description of the system.
3. Use case diagram(s) and provide an explanation.
4. Class Diagram(s) and provide an explanation.
5. Sequence Diagram(s) and provide an explanation.
6. State Diagram(s) and provide an explanation.
7. Component Diagram(s) and provide an explanation.
8. An explanation of any design pattern used.
9. Code contribution's guide;
10. An explanation of the planned architecture and the reason of the choices according to ATAM (follow step 4 and 5, i.e., identify possible architecture styles and choose one with respect to the identified utility tree, you need to explain the reason).
11. An explanation of any C++ library used.
12. *An explanation of the internal data structures used and the reason of the choices.
13. An explanation of the search or sorting algorithm used (and concurrent programming, if any) and the reason of the choices. Explain how the algorithm will work in the system with detailed steps.
14. Examples (screen shots) of user interface, this can be either console based or graphic based.
15. An explanation about the software testing process and metrics.
16. A user manual and instruction of the software.
17. Discussion and conclusion about your results (reflection on testing approach, reflection on performance such as computational efficiency, reliability, security, portability, maintainability, scalability, etc. design of system complexity using e.g. big O- notation).
18. Appendix (summary of group experience, the reference manual containing documentation for ALL the functions and classes implemented, and the test tables fully populated).

Important Note: Some of the tasks above are individual tasks (marked as *): for task 4-15, each member needs the tasks assigned to them (it is up to you to decide who is A, B, C and D). For tasks 19 – 20, each member needs to pick one tasks to do (again, it is up to you to decide who does which). Failure to implement the individual tasks would result in the corresponding member's mark to be ZERO for that corresponding section. So, for each member, the individual tasks would be the assigned tasks from tasks 4-15 and one from tasks 19-20. Please indicate on the table of content page about who contribute to which individual tasks.

2.1. Source code

The source code must be stored and maintained on a git repository. The source code must be documented using documentation tolls (e.g. Doxygen).

2.2. Contribution Form

Download the form from NOW and completed it, this needs to be submitted by each member individually.

III. Assessment Criteria

The grade table by the end of this document will be used for marking. Note that the sectional and overall grade will be determined by the application of the Criteria Grid (see next page).
You will be asked to discuss and demonstrate your assignment at a viva after your assignment. The demo requires you, as a group (you could either elect one person to be the presenter for the general group-based part), to demonstrate all functionalities mentioned in the case study. The tasks 9-11 needs to be explained and demonstrated by the group member who completed them. You only need to demo the implementation not the design. (Failure to attend the Demo will result in a cap of final mark as Marginal Fail)

General Criteria Grid

Grade %	***Class***
0	Zero
1 - 29	Low fail
30 - 34	Mid fail
35 - 39	Marginal fail
40 – 43	Low Third
44 – 46	Mid Third
47 - 49	High Third
50 - 53	Low 2.2.
54 - 56	Mid 2.2.
57 - 59	High 2.2.
60 - 63	Low 2.1.
64 - 66	Mid 2.1.
67 - 69	High 2.1.
70 - 79	Low First
80 - 89	Mid First
90 - 94	High First
95 - 100	Exceptional First

IV. Feedback Opportunities

Formative (Whilst you're working on the coursework)

You will be given the opportunity to book appointments to discuss the assessment outside of class time. Feedback will be given after the submission of the deliverables 1 to 4.

Summative (After you've submitted the coursework)

You will receive specific feedback regarding your coursework submission together with your awarded grade when it is returned to you. Clearly, feedback provided with your coursework is only for developmental purposes so that you can improve for the next assessment or subject-related module.

V. Resources that may be useful

Referencing styles please use Harvard as detailed here
Guidance for presentations as detailed here and think about what lectures you have liked and why Guide to planning your time here and an automated planner here
Source control using GitHub VCS here
Workflow management using Slack here

VI. Moderation

The Moderation Process
All assessments will be marked by two members of the Module team.

VII. Aspects for Professional Development

The assignment has the benefit of allowing you to practise both design and implementation, as well as their transition in the defined scenarios. You will learn how to communicate efficiently with other team members, create and maintain technical documentation, how to use git based distributed version control system. All of those technical skills are normally requested by Software Development Companies.

Detailed Criteria Grid

Class/grade Assessment criteria	Distinction (Excellent)				2.1. (Very Good)			2.2. (Good)			Pass (Fair)			Fail (insufficient)			Zero
	*Exceptional First	High First	Mid First	Low First	High 2.1.	Mid 2.1.	Low 2.1.	High 2.2.	Mid 2.2.	Low 2.2.	High Third	Mid Third	Low Third	Marginal fail	Mid Fail	Low Fail	zero
Project Plan: - Requirements list - Project Risks - Mitigation plan - Use of monitoring tools - Gantt Chart **Responsible:** Project Manager	**All the relevant requirements have been identified.** Every task is correlated to one or more requirements. The Gantt Chart includes the identified tasks, deliverables and optional milestones. The project plan includes the use of tools for monitoring the workflow (such as slack). All the relevant project risks have been identified and a reliable mitigation plan was developed.	**Most of the relevant requirements have been identified.** Most tasks are correlated to one or more requirements. The Gantt Chart includes the identified tasks, deliverables and optional milestones. The project plan includes the use of tools for monitoring the workflow (such as slack). Most of the relevant project risks have been identified and a reliable mitigation plan was developed.			**Relevant requirements have been identified.** Tasks are correlated to one or more requirements. The Gantt Chart includes the identified tasks, deliverables and optional milestones. The project plan includes the use of tools for monitoring the workflow (such as slack). Relevant project risks have been identified and a mitigation plan was developed.			**Some requirements have been identified.** Some tasks are correlated to the requirements. The Gantt Chart includes the majority of the identified tasks, deliverables and optional milestones. The project plan includes the use of tools for monitoring the workflow (such as slack). Some project risks have been identified and a mitigation plan was developed.			**A few requirements have been identified.** Some tasks are correlated to the requirements. The Gantt Chart includes some of the identified tasks, deliverables and optional milestones. The project plan includes the use of tools for monitoring the workflow (such as slack). A few project risks have been identified and an acceptable mitigation plan was developed.			**Up to 4 relevant requirements have been identified.** The tasks are vague and not correlated to the requirements. The Gantt Chart is limited or doesn't exists. The project plan doesn't include the use of tools for monitoring the workflow (such as slack). Up to 4 relevant project risks have been identified and the mitigation plan is very limited or doesn't exists.			
Software architecture - Use Case diagrams - Sequence diagrams - State diagrams - Class diagrams. - Cohesion and Coupling Associations - Operations and attributes identified. - Use of UML notation. - Component Diagram - Deployment Diagram - Design patterns and/or MVC - ATAM design **Responsible:** Software architect	**A thoroughly convincing design communicated well through the use of all model diagrams.** Decisions made during design insight about all the requirements. Some use cases beyond the specs were defined. **The diagrams show an exceptional grasp of the requirements and an understanding of the problem**. Critical evaluation of key needs for extra functionality of the project.	**An excellent design communicated well through the use of all model diagrams. Decisions made during design show insight about all the requirements** **The diagrams show an extensive grasp of the requirements and an understanding of the problem.** The diagrams are consistent with the others. Excellent ATAM design. Design pattern/MVC/other architecture styles used.			**Reasonably good design but may require multiple iterations for implementation.** Sufficient detail in use case, sequence and/or state flows and alternative flows added**.** **The diagrams demonstrate the ability to grasp the requirements.** The diagrams are consistent with the others. Good ATAM design.			**Sufficient design but may be difficult in the implementation.** Good detail in use case, sequence and/or state flows and alternative flows added. Some mistakes on the notations can be easily detected. **The diagrams express a good result of the design**. Decisions made on the design show understanding of all the requirements. Some inconsistencies and mistake of notations can be seen. No or weak ATAM design.			**A number, though not all, requirements have been identified.** Use of UML for identified cases, sequence and state. Some flow diagrams may be missing or incomplete. **The diagrams generally communicate result of the design**. Decisions made on the design show understanding of most of the requirements. Some inconsistencies can be seen. No or weak ATAM design.			**Major mismatch between requirements and design diagrams**. Shows a lack of understanding of the required methodology. Lack of understanding of UML notations. **The diagrams incomplete or inappropriate**. Misunderstanding of some of key requirements. No diagram consistency at all. Lack of understanding of UML notations.			
Implementation: - Functionality - Architecture (use of classes to realise MVC) - Data Structure (linked list, stacks, queue, binary search tree) - Sorting and search algorithms - Exception handling - Access modifiers - User interface - Use of VCS tools (i.e. git) **Responsible:** Software Developer	**Exceptional breadth and depth of knowledge and understanding of C++ programming.** Program meets all the required functionality and much more; well-implemented data structure and good exception handling/access modifier; Excellent user interface and its instruction. Use of git for storing the source code, the wiki for documenting the source code, git issues management, git project management and Dev. Ops. And git project management.	**Excellent knowledge and understanding of C++ programming.** Program includes more functionality than was specified, to provide new insights; reasonably implemented data structure and good exception handling/access modifier; Detailed user interface and its instruction. MVC/other architecture styles used. Use of git for storing the source code, the wiki for documenting the source code, git issues management, git project management and Dev. Ops.			**Very good knowledge and understanding of C++ programming.** Some functionality beyond the required range, although not all specified functionality may be provided. Implementation of data structure but with limited functions, simple classes throughout, no public attributes; Search/sort algorithm implemented. Good user interface and its instruction. Use of git for storing the source code, the wiki for documenting the source code and git issues management.			**Good (some are very good) knowledge and understanding of C++ programming** Most required program functionality provided; classes throughout with private attributes. Use of advanced implemented data structure; good throw and catch blocks implemented. Search/sort methods might have some flaw. Weak but acceptable user interface and its instruction. Use of git for storing the source code and the wiki for documenting the source code,			**Good knowledge and understanding of C++ programming** Most required program functionality provided; classes throughout but with public attributes. Use of simply implemented data structure; some throw and catch blocks implemented. Search/sort methods might be missing. Weak user interface and its instruction. Use of git for storing the source code.			**Insufficient knowledge and understanding of C++ programming.** Understanding is typically at the most basic level with program being uncompilable, or compiles but does nothing; fails to address any of the functionality required by the specification. No classes implemented. Arrays used as data structure. No search/sort functionality implemented. Failed to use git for storing source code.			
Testing:	**Exceptional evaluation**	**Excellent evaluation of the program.** The			**Very good evaluation of the**			**Good (some are very good)**			**Good evaluation of the**			**No Test plan and workarounds**			

- Test Plan - Results and screenshots - Workarounds. - Use of debugging tools (e.g. gdb) - Use of Tests framework. - Use of DevOp tools (e.g. Jenkins or Travis) - Test report **Responsible:** Software Tester	**of the program.** Evidence of extensive and appropriate critical evaluation of program with well documented workarounds. Use of a testing framework and/or DevOp for continuing test development.	evaluation plan is well documented as well as the results and workarounds if performed. Use of a testing framework and/or DevOp for continuing test development	**program**. Test plan and results are well documented, however not all workarounds were implemented. Use of a testing framework and/or DevOp for continuing test development.	**evaluation of the program**. Most test cases are included in the test plan, some attempt to do workaround but may contain flaws. Use of a testing framework and/or DevOp for continuing test development.	**program**. Some test cases may not be included in the test plan, however, a few workarounds were implemented and documented. Use of a testing framework and/or DevOp for continuing test development.	**presented**. Very weak technical and practical competence hampers ability to report achievement of outcomes. No use a testing framework nor DevOp for continuing test development.	
Demo: - Provided evidences that the individual tasks have been completed. - Provided evidences that understands the source code. - Provided evidences of authorship **Responsible:** All	**Excellent understanding of the code and the library involved.** The delivery comes across as a cohesive, confident and well-orchestrated to maintain audience interest and engagement. Completion of all individual tasks.	**Clear comprehension of the code has been demonstrated.** Nearly all questions are covered in the code. Completion of all individual tasks	**Some detailed discussion of the code is attempted.** Most of the functions demonstrated although some may be missing. Completion of all individual tasks	**Demo runs generally smooth and well organised.** Have explanation for the key functions, but lack of details. Completion of all individual tasks	**Demo runs generally smooth.** But some explanation are unclear and key functions are missing. Failed to complete ONE of the individual tasks.	**Demo run badly**. Very limited evidence of understanding on the code or the code itself is too simple to be explained in depth. Failed to complete TWO or more individual tasks	
Others: - Consistency between design and implementation, w.r.t. requirements - Use of concurrent programming. - Use of slack for managing the workflow **Responsible:** All	**Design and implementation are perfectly consistent.** Every significant design decision is reflected and implemented. Use of slack for managing the work flow. Use of concurrent programming.	**Design and implementation are generally consistent.** Nearly all design decisions are reflected in the code. Use of slack for managing the work flow. Use of concurrent programming.	**Design and implementation are generally consistent.** There may be some small amount of mismatch but can only be noticed by carefully investigating. Use of slack for managing the work flow. Use of concurrent programming.	**Design and implementation have inconsistencies but not very easy to detect.** Limited classes are missing, or a small set of associations are mismatched. Use of slack for managing the work flow. Use of concurrent programming.	**Design and implementation have something in common.** But some classes are missing, or some associations are mismatched. Use of slack for managing the work flow. Use of concurrent programming.	**Design and implementation are generally inconsistent**. Very limited connection can be found between diagrams and the classes. Failed to use of slack for managing the work flow. No concurrent programming was used.	

NOTTINGHAM
TRENT UNIVERSITY