NOTTINGHAM TRENT UNIVERSITY

School of Science and Technology

COURSEWORK ASSESSMENT ELEMENT

MODULE CODE: **SOFT20091**

MODULE TITLE: **Software Design and Implementation**

MODULE LEADER: Dr. Tao Chen

COMPONENT: 1 of 1

TITLE: SDI Portfolio

LEARNING OUTCOMES

ASSESSED: **All**

WEIGHTING: 100% of the total coursework mark

100% of the overall module mark

DISTRIBUTION

DATE: **Tuesday January 16th (Teaching week 25)**

SUBMISISON

DATE: **Monday April 30th (Teaching week 40)**

SUBMISISON

METHOD: **Electronic hand-in via Dropbox**

NOTE: The usual University penalties apply for late submission and plagiarism. Please consult your student handbook for further details.

I. Assessment Requirements

Given the problem scenario in Section II, produce a report and a software program, and present a software demo. Software demos will be organized during the usual timetabled lab sessions in Teaching Week 40 and 41 (week starting Monday 30/04/18 and 07/05/18). The portfolio (one per group) should be submitted electronically through *NOW Dropbox by Monday 30th April 2018 23:55 (Teaching Week 40)*:

- Students should use the following convention for naming their folders and files:

'Group_X_SDI_Report' (for example, Group_X - X=name of the group – report for the assignment of the module SOFT20091). You are also advised to add this information to the header section of your submitted document.

- All files must be submitted in a single, main folder per group. The folder should include your report (doc or pdf file) and subfolders such as:

- - CODE containing all the code files (mandatory);
- - TEST with any test dataset/database or readme file with link to it (mandatory);
- - RESULTS with results material/images of the software (optional, but containing this will serve as strong evidence of the quality of your design and implementation).

The assessment contributes to all learning outcomes as indicated.

Take regular backups of your work. This will enable you to recover quickly should the system fail and also allow you to backtrack if your development goes astray.

Ensure that the work submitted will execute on 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 Group Work:*

Permissible group size: 4 to 5 students. Why?

- Because individual projects have to be 'small scale' and so do not provide scope to show much analysis, design and its reflection on implementation.
- Engineering software system is often done as a team, and therefore it is important to practice the ability of collaborations.
- Individual assessment on programming skill has already been done as part of term 1 (although the mark does not directly contribute to the final mark of the module)

Managing group work is part of the assessment and a brief paragraph of the report should summarize 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 student Ids) as content to Dropbox on NOW. Changes on memberships of work groups will not be allowed AFTER Teaching Week 29 (the week of 13th Feb 2018). Any one not having a group by then will be randomly assigned to a group, so please do seek your group mates ASAP.

In each group, one person should volunteer to be responsible for submitting the group work in time by uploading the folder with all the components into NOW (see also above- mentioned explanations).

For each group, all students will have the same mark for the group, unless the members have different results for their in-class test in term 1 (Pass/Fail); or there is strong evidence that one student had not contributed to the group project and this student having poor attendance of the module.

Each group member will need to submit a contribution form individually. The form would be used to assess the contribution of each member in the group, and amend the mark for each individual when necessary.

II. Assessment Scenario/Problem

You are a newly recruited analyst programmer, working for a major IT company. This company own information about film projects, such as "E.T." and the "Fast and the Furious" series.

Now, you are expect to design and implement a new software system for a client called TrekStar Pictures. In a meeting with a TrekStar Pictures representative, a number of requirements have been discussed and you have identified the following:

Every time the production of a new film begins, a new project is created. This project has all the relevant information regarding the production of the film. Each new project has a title, summary, genre, release date, list of filming locations, language, runtime, keywords and a weekly ticket sale (theatrical weekly box office).

TrekStar pictures launches a project with a series of its materials for retail. These comprise single-sided DVDs, double-sided DVDs, combo box sets (containing two or three DVDs), VHS (on old projects) and Blu-rays. All of them contain features such as identification number, title (could be different from project/film title), format, audio format (Dolby, Dolby digital, MPEG-1, PCM or DTS), run time, language, retail price, subtitles, frame aspect (wide screen). However, each of them has different packaging specifications. A single or double DVD will have a plastic box packaging, while a combo box set would have a cardboard box. The Blu-ray would also have a plastic box as the DVD, but with a different size. A VHS can come in a plastic or cardboard package.

DVDs and Blu-ray can have multiple language tracks and subtitles in different languages, compared to VHS that can only have one subtitle and one audio track. In addition, DVDs and Blu-rays may contain bonus features (additional material such as short films or director comments of the production). For double-sided DVDs, it is important to know what contents are in one side, and what in the other (chapters, bonus features and languages).

From a second meeting with a TrekStar Pictures representative, it has been clarified that in addition to the project details and associated materials of the original brief, the system should handle the following information and business rules:

- Projects that are currently under production are considered "unreleased" projects (these wouldn't have any associated materials). The system should not allow the user to input any theatrical weekly box office information nor the addition of materials. Projects related to films that are currently showing on cinemas are identified as "now playing". The system should allow the user to input theatrical weekly box office information but should not allow at this point the

addition of materials. Projects related to films that are no longer at cinemas are identified as "released". Only these type of projects will allow the addition of material information.

- Each project has a crew. This crew consists of all the people involved in the production of the film such as the producer, director, writer, cast (set of actors), editor, production designer, set decorator and costume designer.

Functionalities of the New Software System

• From a cold start, existing project and material details must be loaded from a file. When the system is shut-down, the data needs to be stored persistently.

• Project and Material Creation function – this must allow the addition of new projects and materials to the database (file) with the following rules:

- o The system must not allow to add new materials to projects that are "unreleased" or "now playing".
- o Only for those projects being created with a "now playing" status, the system should allow to add a set of weekly box office figures during creation.
- o The system should allow the creation of more than one project with the same name.

The new projects and materials do not to be written directly to the file every time you add a new one. They can be inserted into a data structure (where you currently handle your existing projects/materials) and just make the save to the file during system shut-down.

• Existing Projects and Materials Update/Removal – the system must allow for any changes in the existing projects and materials following the business rules stated previously. The deletion of an existing project should remove all the material information associated with it.

• Catalogue browsing – must allow bi-directional sequential browsing of all projects; shall allow interactive project and materials search with the following rules:

- o A search by project title will allow the user to see main project detail information as well as summarised information of materials (e.g. "is currently available on DVD and VHS").
- o Provide the user with the option to view the materials details when searching by project title. When displaying double-sided DVDs, the view should show details of the contents per side. The same with combo box sets, the view should display details of every DVD in the combo box (single and double-sided DVDs).
- o A search by actor should return all project titles where that actor was part of the cast. The system may allow more flexible search with multiple fields. For example, search only for all the double-sided DVDs available of a specific project title.

• Maintenance Mode – a number of Utility functions which can either be integrated or run as command line applications, to:

- o Add/remove projects and materials to the database.
- o Raise daily reports on (may use Logger class)
  - ▪ New projects/materials added

- Projects with certain amount of total box office earnings

1. Tasks

Analysis and Design

The project should contain the following design and analysis for the WHOLE software systems:

*A Use case diagram showing use cases for all the requirements.

* A Use case description for each use case.

* The Class diagram

-Include all attributes, operations and associations derived from the above analysis.

* A Sequence diagram for a use case.

* State machine diagram for a class. Transitions should be consistent with the related sequence diagram.

* Component diagrams.

* Deployment diagrams.

* Any design patterns

* Discussion on Architecture Style and justify the reason 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)

Implementation

You are expected to implement the above software system using C++, in particular, you are expected to have:

- at least one of the data structures (e.g. queues/stacks/graphs/trees) proposed in lectures (This should be implemented by yourself rather than based on other libraries);
- at least one of the sorting or searching algorithms studied during the lectures (This should be implemented by yourself rather than based on other libraries);
- at least one place where error and exceptions handling have been used;
- at least one class where private member functions for common functionality that should not be exposed in the public interface;
- at least a console 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 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)

Below there are some optional implementations to be considered, they are regarded as bonus for the assignment:

- Concurrent programming where appropriate;
- GUI for the software system;

2. Deliverables

The submission is done electronically via Dropbox on NOW. The contribution form needs to be completed and submitted individually. For the coursework, only one member of the group will need to submit on behalf of the whole group. For this, you are expected to submit two parts: a development report and a set of source code:

4.1. Development report

You will deliver a structured report that should contain the following sections:

1. A cover page (showing the full name and student ID of all members)
2. A table of contents page
3. A general description of the system
4. A description of the followed Software Development Life-Cycle methodology. Make links to the state of the art to back your claims/assumptions/choices;
5. Illustrations of Use Case Diagram, Class Diagram, Sequence Diagram (one diagram), State Diagram (one diagram), Component Diagram and Deployment Diagram
6. An explanation of any design pattern used
7. 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)
8. An explanation of any C++ library used.
9. An explanation of the internal data structures used and the reason of the choices.
10. 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.
11. An explanation about the software testing process and metrics;
12. Discussion about your results (reflection on testing approach, reflection on performance such as computational efficiency, reliability, security, portability, maintainability, scalability, etc. analysis of system complexity using e.g. big O- notation);
13. A description of the file format(s) used – e.g., ASCII text files, so that they can be read easily by other applications.
14. A user manual and instruction of the software. (regardless if your software is based on GUI or console interface)
15. Conclusions (reflection on the adopted methods and alternatives, reflection on Professional, Social, Ethical, and Legal aspects, reflection on possible improvements).
16. Appendix (summary of group experience and some of the records on group activities)

4.2. Source code

Debugged source code, in C++ should be structured and commented.

4.3. Contribution Form

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

4.4. *Demo*

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 or each of you can present part of it), to demonstrate all functionalities mentioned in the case study, and discuss them with respect to all the points in the implementation task, except "*appropriate testing cases and testing library*". (Failure to attend the Demo will result in a cap of final mark as 48%, 3 rd D)

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).

IV. Feedback Opportunities

Formative (Whilst you are working on the coursework)

Support will be available in the labs or by email.

Summative (After you have submitted your coursework)

You will receive specific feedback regarding your coursework submission together with your awarded mark 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 subject-related module or for future projects that you may be involved with.

The Nottingham Trent University
School of Science and Technology

Software Design and Implementation – SOFT20091

Group Name:
Student Members:
Student Numbers:

Section	***Grade %/Note***
The Software Development Life-Cycle Methodology	
Use case diagram	
Class diagram	
A Sequence diagram for a use case	
State machine diagram for a class	
Component diagram	
Deployment diagram	
Any design patterns	
An explanation of the planned architecture and the reason of the choices according to ATAM	
At least one of the data structures (e.g. queues/stacks/graphs/trees) proposed in lectures; (involve in demo)	
At least one of the sorting or searching algorithms studied during the lectures (involve in demo)	
At least one place where error and exceptions handling have been used (involve in demo)	
At least one class where private member functions for common functionality that should not be exposed in the public interface (involve in demo)	
At least a console interface that allows user to interact with the software system (involve in demo)	
Appropriate testing cases and testing library	
Overall demo	
Others (e.g., if requirements are met, consistency, if design is reflected on codes, etc)	
Final Grade	

Further Comments:

General Criteria Grid

Grade %	***Class***
0	Zero
1 - 29	Low fail
30 - 44	Mid fail
45 - 49	Marginal fail
50 - 53	Low Pass
54 - 56	Mid pass
57 - 59	High pass
60 - 63	Low Comm
64 - 66	Mid comm
67 - 69	High comm
70 - 79	Low Dist
80 - 89	Mid dist
90 - 94	High dist
95 - 100	Exceptional dist

Detailed Criteria Grid

Class/grade Assessment criteria	Distinction (Excellent)				Commendation (Very Good)			Pass (Good)			Fail (insufficient)			Zero	Grade/ Comment
					High Comm.	Mid Comm.	Low Comm.	High Pass	Mid Pass	Low Pass	Marginal fail	Mid Fail	Low Fail	zero	
	***Exceptional Dist.**	High Dist.	Mid Dist.	Low Dist.											
Analysis: **-** SDLC explanation - Expected use cases and descriptions present. - Use cases reflect functionality. - Identification of actors. - Use of UML notation. - Case flows - Sequence diagrams - State diagrams	**A thoroughly convincing analysis communicated well through the use of all model diagrams**. Decisions made during analysis insight about all the requirements. Some use cases beyond the specs were defined.	**An excellent analysis communicated well through the use of all model diagrams**. Decisions made during analysis show insight about all the requirements			**Reasonably good analysis**, but may require multiple iterations for implementation. Sufficient detail in use case, sequence and/or state flows and alternative flows added.			**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.			**Major mismatch between requirements and analysis diagrams**. Shows a lack of understanding of the required methodology. Lack of understanding of UML notations.				
Design: - Class diagrams. - Associations - Operations and attributes identified. - Use of UML notation. - Component Diagram - Deployment Diagram - Design patterns and/or MVC - ATAM analysis	**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.	**The diagrams show an extensive grasp of the requirements and an understanding of the problem.** The diagrams are consistent with the others. Excellent ATAM analysis. Design pattern/MVC/other architecture styles used.			**The diagrams demonstrate the ability to grasp the requirements.** The diagrams are consistent with the others. Good ATAM analysis.			**The diagrams generally communicate result of the analysis**. Decisions made on the design show understanding of most of the requirements. Some inconsistencies can be seen. No or weak ATAM analysis.			**The diagrams incomplete or inappropriate**. Misunderstanding of some of key requirements. No diagram consistency at all. Lack of understanding of UML notations.				
Implementation: - Functionality	**Exceptional breadth and depth of**	**Excellent knowledge and understanding of C++ programming.** Program			**Very good knowledge and understanding of C++ programming.**			**Good knowledge and understanding of C++**			**Insufficient knowledge and understanding of C++**				

- 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	**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.	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.	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.	**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.	**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.		
Testing: - Test Plan - Results and screenshots - Workarounds.	**Exceptional evaluation of the program.** Evidence of extensive and appropriate critical evaluation of program with well documented workarounds.	**Excellent evaluation of the program.** The evaluation plan is well documented as well as the results and workarounds if performed.	**Very good evaluation of the program**. Test plan and results are well documented, however not all workarounds were implemented.	**Good evaluation of the program**. Some test cases may not be included in the test plan, however, a few workarounds were implemented and documented.	**No Test plan and workarounds presented**. Very weak technical and practical competence hampers ability to report achievement of outcomes.		
Demo:	**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.	**Clear comprehension of the code has been demonstrated.** Nearly all questions are covered in the code.	**Some detailed discussion of the code is attempted.** Most of the functions demonstrated although some may be missing.	**Demo runs generally smooth.** But, some explanation are unclear and key functions are missing.	**Demo run badly**. Very limited evidence of understanding on the code or the code itself is too simple to be explained in depth.		
Others: - Consistency between design and implementation, w.r.t. requirements	**Design and implementation are perfectly consistent.** Every significant design decisions are reflected and implemented.	**Design and implementation are generally consistent.** Nearly all design decisions are reflected in the code.	**Design and implementation are generally consistent.** There may be some small amount of mismatch but can only be noticed by carefully investigating.	**Design and implementation have something in common.** But, some classes are missing or some associations are mismatched.	**Design and implementation are generally inconsistent**. Very limited connection can be found between diagrams and the classes.		