Assessment Cover Sheet

This Assessment Cover Sheet is only to be attached to hard copy submission of assessments.

ASSESSMENT DETAILS

Unit title	Object-Oriented Programming	Tutorial /Lab Group		Office use only
Unit code	COS20007	Due date	4th 12/2022	
Name of lecturer/tutor	Ms Fu SweeTee			
Assignment title	Pass Task 14: Learning Report Summary			Faculty or school date stamp

STUDENT(S) DETAILS

	Student Name(s)	Student ID Number(s)
(1)	Billy Ndegwah Micah	102764311
(2)		
(3)		
(4)		
(5)		
(6)		

DECLARATION AND STATEMENT OF AUTHORSHIP

1. I/we have not impersonated, or allowed myself/ourselves to be impersonated by any person for the purposes of this assessment.
2. This assessment is my/our original work and no part of it has been copied from any other source except where due acknowledgement is made.
3. No part of this assessment has been written for me/us by any other person except where such collaboration has been authorised by the lecturer/tutor concerned.
4. I/we have not previously submitted this work for this or any other course/unit.
5. I/we give permission for my/our assessment response to be reproduced, communicated, compared and archived for plagiarism detection, benchmarking or educational purposes.

I/we understand that:

6. Plagiarism is the presentation of the work, idea or creation of another person as though it is your own. It is a form of cheating and is a very serious academic offence that may lead to exclusion from the University. Plagiarised material can be drawn from, and presented in, written, graphic and visual form, including electronic data and oral presentations. Plagiarism occurs when the origin of the material used is not appropriately cited.

Student signature/s

I/we declare that I/we have read and understood the declaration and statement of authorship.

(1)	Billy Ndegwah Micah [illegible]	(4)	
(2)		(5)	
(3)		(6)	

Learning Summary Report

	Pass	Credit	Distinction	High Distinction
Self-Assessment (please tick)	✓	✓		

Self-assessment Statement

	Included (please tick)
Learning Summary Report (This document)	✓
Assessed Pass Tasks	✓
Semester Test	✓

Minimum Pass Checklist

	Included (please tick)
All of the Pass requirements	✓
Evidence of credit tasks that have been completed	✓

Minimum Credit Checklist, in addition to Pass Checklist

	Included (please tick)
All of the Credit requirements	✓
Code for your program that demonstrates good OO design *including the use of polymorphism, implementing an interface and managing collections of objects*.	✓
UML class diagrams and UML sequence diagrams that communicate the design your custom program	✓
Description of what your program does and screenshots of your program in action	✓

Minimum Distinction Checklist, in addition to Credit Checklist

	Included (please tick)
All of the Distinction requirements	
Research report and associated pieces	

Minimum High Distinction Checklist, in addition to Distinction Checklist

Introduction

This section summarises what I learned about Object Oriented Programming. It includes a justification of the assessment pieces included in the portfolio, details of the coverage of the unit learning outcomes, and a reflection on my learning.

Overview of Pieces Included

Briefly describe what you have included in each section of the report.

Pass Task 7 – Test Driven Development using NUnit Framework in C#

Pass Task 8 – XML Documetation facilitated by the VSCode XML extension.

All projects from henceforth required the good XML documentation practice.

Pass Task 10 – Generic Collections and Object Indexers

Pass Task 12 – Inheritance & Polymorphism

Pass Task 13 – Conceptual UML Modelling and Implementation

Credit Task 1 – System Modelling

Credit Task 2 – (C++) Default Constructor & Destructors, Making Deep Copies

Coverage of the Intended Learning Outcomes

This section outlines how the pieces I have included demonstrate the depth of my understanding in relation to each of the unit's intended learning outcomes.

ILO 1: Object Oriented Principles

Object Oriented programming allows programmers to have an elevated view of the program in terms of objects and how they interact in the program. Below are the four principles of ObjectOriented Programming

a) Encapsulation - This principle allows object to contain data members (known as fields) and behavioural members (known as methods). The fields contain information which belong to the object being implemented in the context of its application, whereas the methods are written as functions that allow either accessing (known as getters) or mutating the fields (known as setters) or simply an action being performed by the object. For example, in making a game of cards, a card would be an object and contain the fields Rank (Spades, Diamonds etc.) and number, it would also contain the method showCard() and to allow the user to see what card their dealing.

b) Abstraction - In cases where object are related and share structural implementation, this principle allows an abstract description of the group of objects which would thereafter be used as a boilerplate class as descriptions get more specific. Abstraction requires that object implementing the interface it describes, inherit (inheritance explained next) from the abstract class in order to add specificity relative to the context of application.

c) Inheritance - This principle allows objects to share implementations through a parent-child relationship, the parent being the class providing a base implementation and the child being the class accessing, overriding, or extending the parent's implementation. This relationship can be extended to

further than just the child class where other classes get to inherit from the child class. There's no limit to the number of classes that can inherit from a single class.

d) Polymorphism - As object inherit from some base class, they might need different implementations of the same functions as it requires in their given context of application. This principle allows derived classes to override an implementation that suits its application case. Polymorphism applies to functions and not fields.

Pass Task 12, 13 & Credit Task 1, 2 – All these implementations heeded the principles of Object-Oriented Programming.

ILO 2: Language Syntax

We used C# Programming Language (running on .NET 6 Core), which is an object oriented programming Language, to implement Object Oriented Programs. C# contains a vast library bult into it that allows utlitites like writing to console, use of collections and even performing arithmetic. This class is known as the Base Class Library (BCL) and have to be included at the top of the file before utilising any functionality from them.

In C#, to include a library in a project (or file) we simply use the keyword 'using' followed by the name of the library ('using' <nameOfLibrary>).

We also used C++ which also does contain it's own in built libraries that support operations and algorithms too (iostream, cmath, algorithms,). In C++ class definitions are written out in header files (identifiable by their .h or .hpp extension) whereas their implementations are written in source files (which have a .cpp or .cc extension)

Pass Task 7, 8, 10 & Credit Task 2 – Explored the use of the BCL (Base Class Library in C#).

ILO 3: Writing Programs

The process of architecting software needs to be rigorous and as the programmer, one has to ascertain that it serves the intended purpose. In order to do that, the software design has to be constructed using UML charts which show the object interaction at a higher level without needing any code. This makes correcting an idea much easier since the code hasn't yet been written for it. After confirming that the overall architecture is feasible and produces the intended solution, the objects are constructed using the Object-Oriented fashion of programming.

As each class gets written, Test-Driven development requires that the tests get written as the classes get written which makes testing a single or small set of classes much easier and allows an early realisation of faults and bugs. In C# these tests can be done using the NUnit Framework library. This library allows writing simple unit tests that can narrow their focus to one aspect of the program (or a single unit of the program hence the name unit tests).

In C++, I used GoogleTest, to write out similar tests for the program. CMake is like a programming language that allows the programmer to specify how build files should be run in a program. It provides the capability to run units using a library called CTest which allows carrying out unit tests using GoogleTest. Much like the unit tests in C# GoogleTest on CMake, test single units of a program before it scales.

Since these test only test a part of the program it becomes much easier and less time consuming to locate the issues and bugs in the application.

The NUnit framework and GoogleTests with CMake can both be used with VSCode (Visual Studio Code IDE).

Pass Task 10, 12, 13 & Credit Task 1, 2 – Involved defining UML diagrams and thereafter explanations writing out the implementations which contained unit tests.

ILO 4: Object Oriented Design

To manage complexity, it's often best to use Object oriented software design in setting up the application's ideation. After constructing a UML diagram that shows how the objects should be oriented and related to each other, documenting these interactions in a comprehensive manner helps communicate the part of the program that the UML doesn't show, namely, what function calls should facilitate an implementation within a class.

This involves a description of the object interaction along with the context in which they're expected to act a certain way. Included in the description would be the object fields within a class since these dictate (to an extent) what the object can do in a given context. For example, an object cannot perform computations on fields it doesn't have, this would therefore require accessing such fields by utilising a function that accepts a parameter of the desired type, alternatively, one could opt for introducing a design pattern that can facilitate the interaction ideally. This is also a form of reflection that helps in reasoning about the program in order to examine the details of the implementation before introducing any code to it.

Pass Task 10, 12, 13 & Credit Task 1 – Involved defining UML diagrams and thereafter explanations of the implementation within the UML definition.

ILO 5: Program Quality

A good object-oriented program utilises (as needed) the four principles of object-oriented programming. Abstraction helps prevent too many class constructions with repetitive implementations. Encapsulations allows a minimalistic approach to the problem where the object can just contain only 'what it needs to know about' (a loose way of referring to object fields). Heeding these principles prevents making a huge monolith of an application that is hard to scale or even maintain.

Use of algorithms (whether they be in-built or custom implementations) should facilitate the program efficiency and improve response time.

A program should also be fault tolerant, it should accommodate the fact that the users are going to be humans and will make errors (intentionally or otherwise) while utilising it. It should, therefore, have functionalities that monitor areas of interactions (for example while expecting user input) in order to perform checks and provide hints as needed. The program should also be able to correct internal faults (for example running out of memory) it should identify such faults and issue a much more graceful exit of the program without befuddling the user. In testing for such possible faults, it's best to use unit tests to check for potential behaviour of the application.

Perhaps this goes without saying, the program should work and serve the intended purpose..

Pass Task 10, 12, 13 & Credit Task 2 – Involved unit tests which checked the program performance during development.

Reflection

The most important things I learned:

UML Diagrams – Pass Task 10 - 13
Unit Tests – Pass Task 7 - 13
Dotnet Console Applications – Pass Task 7 - 13
Dotnet NUnit package – Pass Task 7 - 13
OOP Principles – Pass Task 8 - 13
Design Patterns
CMake (for C++ projects) – Credit Task 2
Unit Testing using CMake (GoogleTest) – Credit Task 2
Software XML documentation – Pass Task 8 - 13
DOxygen Software Documentation – Credit Task 2

I enjoyed this unit. I learnt much more than what the unit covered because it pushed my curiosity and creativity and I wanted to explore further.

I'm greatly satisfied by this unit.

The things that helped me most were:

Lecturer's approachability and supportive attitude

Lecturer's method of teaching

The Lecturer was flexible and allowed me to try explore my creativity in the applications even regarding which tools to use.

I found the following topics particularly challenging:

Design Patterns – We only touched on the topic and didn't explore it in great depth

I found the following topics particularly interesting:

UML Diagrams – Pass Task 10 - 13
Unit Tests – Pass Task 7 - 13
Dotnet Console Applications – Pass Task 7 - 13
Dotnet NUnit package – Pass Task 7 - 13
OOP Principles – Pass Task 8 - 13
Design Patterns
CMake (for C++ projects) – Credit Task 2
Unit Testing using CMake (GoogleTest) – Credit Task 2
Software XML documentation – Pass Task 8 - 13
DOxygen Software Documentation – Credit Task 2

I feel I learned these topics, concepts, and/or tools really well:

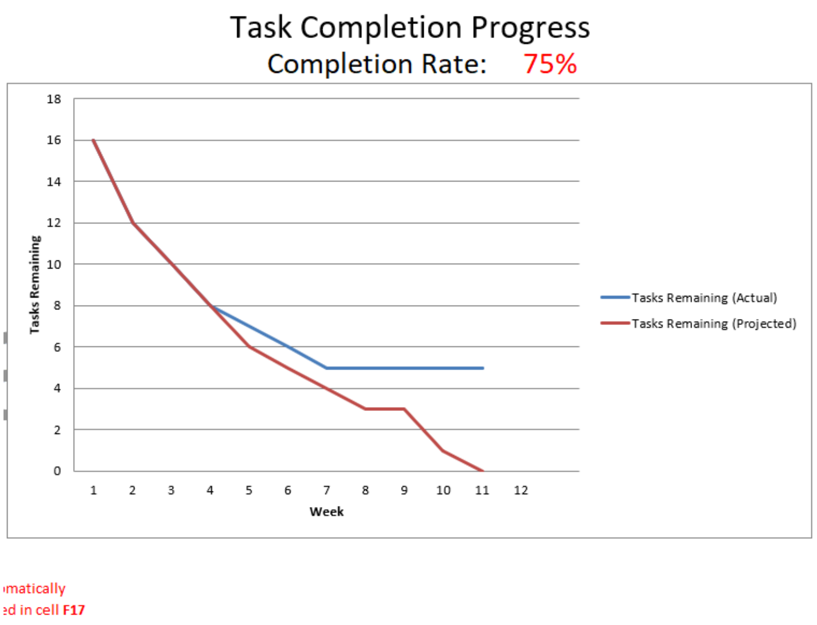
UML Diagrams – Pass Task 10 - 13
Unit Tests – Pass Task 7 - 13
Dotnet Console Applications – Pass Task 7 - 13
Dotnet NUnit package – Pass Task 7 - 13
OOP Principles – Pass Task 8 - 13
CMake (for C++ projects) – Credit Task 2
Unit Testing using CMake (GoogleTest) – Credit Task 2
Software XML documentation – Pass Task 8 - 13
DOxygen Software Documentation – Credit Task 2

I feel more than confident to apply the above concepts in projects

I still need to work on the following areas:

Design Patterns – This area was introduced towards the end, and I was hoping to apply it in my distinction task, however, since I didn't continue with the distinction task I didn't get to apply it as I'd intended. I'm hoping to implement it in my personal projects on GitHub.

My progress in this unit was ...:



This unit will help me in the future:

I'm currently pursuing Artificial Intelligence, this is a field that uses a lot of data which has to be collected from users using applications. Having the know-how on application construction will help me make applications that have a great user flow allowing seamless usability and cleaning out the data to ascertain that the expected types are accepted, and the opposite are rejected. Also knowing how to construct applications that manage failure successfully will aid in preventing loss of data during the collection process.

If I did this unit again I would do the following things differently:

Time Management – This was a great unit with completely relevant material for today's software development careers. I know this because I saw the tools we were covering, in Job descriptions on LinkedIn. However, my time management skills could have been better and would have allowed me to complete both the Distinction and High Distinction tasks

Concluding Statement

As shown above, both by the tools learned and the breath of their application, I think I have done ample work for a distinction grade in this unit. My projects demonstrate a confident understanding of development tools and the implement design principles ideals for scaling and maintenance, two very key factors in development.