ASSESSMENT 2 BRIEF	
Subject Code and Title	ISE102 Introduction to Software Engineering
Assessment	Prototype and Software Solution
Individual/Group	Group
Length	source code and comments
Learning Outcomes	The Subject Learning Outcomes demonstrated by successful completion of the task below include: a) Identify user requirements in computer-driven software applications. c) Apply software programming methodologies including object-oriented programming (OOP) paradigms to build software solutions. d) Test and validate software solutions that satisfy project requirements.
Submission	Due by 11:55pm AEST/AEDT Sunday end of Module 8.
Weighting	35%
Total Marks	100 marks

Assessment Task

In this assessment, your group will be required to demonstrate an understanding of object-oriented programming principles and knowledge using programming language features you have learnt in Modules 5 to 8.

Please refer to the Task Instructions for details on how to complete this task.

Context

A computer application solves any real-world problem, like booking a ride online or making a purchase using a touchscreen kiosk at a fast-food restaurant. A computer program or application instance has a lifetime—it starts, it progresses, and it will eventually end. Along the way, we will find ourselves in different *stages* of that task. This concrete approach can be used to write programming applications to embed in machines. As part of this assessment, assume that you are working as part of a large software company which writes software for business organisations. A business case study problem has been received by your company. The Learning Facilitator will provide and describe the business case project, and your task will be to understand it then complete it.

In Assessment 1, you applied and explained the variables, expressions, conditionals, loops and data structures like arrays. Assessment 2 will require your group to apply object-oriented programming

knowledge and concepts of modular design to begin building larger, more capable programs. This marks the setting up of a launching pad towards building confidence in your own ability to break down problems and design solutions.

Task Instructions

Group Work

1. **Group Formation**

Form a group of a maximum of 3 members.
Your group must be formed by the end of Module 5 (Week 5) and registered.

To register your group, you are required to send your Learning Facilitator an email with "ISE102 Group Registration" in the subject line. In the body of the email, please list the names and student ID numbers of all the members of your group.

You are required to send the registration email to your Learning Facilitator before the registration deadline. After the registration deadline, the students who are not in a group will be allocated to a group by your Learning Facilitator.

The deadline for the group registration is 11:45pm AEST on Friday at the end of Module 5 (Week 5). Please note that you will work with your group members for Assessments 2 and 3.

2. **Group Contract**

Please read the attached *ISE102_Assessments 2 & 3_Group Contract*. This document outlines the rules and conditions each group has to follow for both assessments as well as the roles and responsibilities of each group member. Each group is required to complete their group contract and attach it as an appendix to the Word document on software development cycle (Section B). The group contract accounts for 5% of the assessment grade, as indicated in the Assessment Rubric. For assessments where students are expected to work in groups, the workload must be shared equitably among all group members. Please refer to sections 6.1 and 6.2 of the TUA PL_AC_014: Student Conduct Policy.

When submitting the group contract, you are reminded not to 'recycle' (self-plagiarise) contracts from other assessments. Sections on deliverables, timeline and expectations should be unique to each assessment or project. Self-plagiarism constitutes a breach of Academic Integrity and can lead to penalties to the assessment or subject.
During Assessments 2 and 3, you should keep records of communication and drafts. Any serious concerns about an individual group member's contribution should be brought to the attention of your Learning Facilitator as soon as they occur or at least two weeks before the due date, whichever is earlier.

If a student has been accused of not contributing equally or fairly to a group assessment, the student will be contacted by the Learning Facilitator and given three working days to respond to the allegation and provide supporting evidence. If there is no response within three working days of contact, the Learning Facilitator will determine an appropriate mark based on the evidence available. This may differ from the mark awarded to other group members and would reflect the individual student's contribution in terms of the quantity and quality of work.

Your Learning Facilitator will provide your group with the business case project. Your group will explore the code beyond the class samples. Your group will be applying the principles and skills you learn throughout the modules to leverage and demonstrate your programming skills.

1. Read the business case study and the problem definition to understand the user, software and resource requirements.
2. Develop a program using C# considering all the users' requirements.
3. Apply object-oriented programming concepts where appropriate.
4. Debug and test your code.
5. Validate your code work to your client satisfaction.

Tasks your group will encounter include:

- Compiling the user, technical and resource requirements and representing them using a simple diagram.
- Creating classes with subjective names and appropriate attributes and functions.
- Defining constructors and access modifiers within class definitions and stating in comments why it has been used and where it can be used.
- Keeping track of the state of the program.
- Formatting output clearly and handling bad input gracefully and with useful errors.
- Providing necessary test samples to validate the code.
- As primary features of object-oriented design are reusability and maintainability, you should therefore make the code definitions modular by reducing the use of global variables and only expose functions to as much information as they need to do their job.

Please refer to the Assessment Rubric for the assessment criteria.

Crediting Sources of Acquired Code

As many online programming resources provide solutions to programming tasks along with documentation, if any such source code is acquired (reference works, documentation, help and tutorial sites etc), it must be preceded by a code comment that lists the original site/creator and followed by a comment that declares the end of the acquired code. Refer to the Academic Integrity Code Handbook for more information. Acquisitions should be kept to a few lines or less and solve single problems (i.e., changing the range of a randomly generated number, handling unexpected types of input data and so forth).

Submission Instructions

Final Submission by: **11:55pm AEST/AEDT on the last day (Sunday) of Module 8**.

a) Archive your completed project as a .zip file. Full instructions for how to do so will be provided and demonstrated by your Learning Facilitator. Label the file using the following naming convention: ISE102_LastnameFirstname_A2.zip
b) Create a share link for the project. Instructions will be provided and demonstrated.
c) Submit the link and the zip archive of your project on Blackboard in the **Assessment** section of *ISE102 Introduction to Software Engineering*. Only one group member should submit the final project to the Learning Facilitator.

Note: Your Learning Facilitator will provide the case problem and base code and work through part of it in class. It is critical that you attend or watch class streams and work along. Keep track of Blackboard announcements/emails and reach out to your Learning Facilitator with questions or issues well before the final due date.

Feedback will be provided via the Grade Centre in the LMS portal and can be viewed in My Grades. Before you submit your assessment, please ensure you have read and understand the conditions outlined in the Academic Integrity Code Handbook. If you are unsure about anything in the Handbook, please reach out to your Learning Facilitator.

Please note that ad-hoc university tech support for external apps and platforms is not available.

Academic Integrity

All students are responsible for ensuring that all work submitted is their own and is appropriately referenced and academically written according to the Academic Writing Guide. Students also need to have read and be aware of the Torrens University Australia Academic Integrity Policy and Procedure and subsequent penalties for academic misconduct. These are viewable online.

Students also must keep a copy of all submitted material and any assessment drafts.

Special Consideration

To apply for special consideration for a modification to an assessment or exam due to unexpected or extenuating circumstances, please consult the Assessment Policy for Higher Education Coursework and ELICOS and, if applicable to your circumstance, submit a completed Application for Assessment Special Consideration Form to your Learning Facilitator.

TORRENS UNIVERSITY AUSTRALIA

Assessment Rubric

Assessment Attributes	Fail (Yet to achieve minimum standard) 0-49%	Pass (Functional) 50-64%	Credit (Proficient) 65-74%	Distinction (Advanced) 75-84%	High Distinction (Exceptional) 85-100%
Interpretation and implementation of functional requirements **Total Percentage for this Assessment Attribute = 25%**	Demonstrates poor attention to detail and understanding of software engineering/programming terms and concepts. Reasons may include: • Does not build atop the code base supplied by the Learning Facilitator • Insufficient original work has been attempted atop the code base • Features and/or data values required by the brief are missing • Requirements supplied by the Learning Facilitator via Blackboard or streams were not followed.	Demonstrates some attention to detail and understanding of software engineering/programming terms and concepts by: • Building atop the code base supplied by the Learning Facilitator • Attempting features and using data values that reflect a correct interpretation of the brief's requirements • Following any requirements provided by the Learning Facilitator via Blackboard or streams.	Demonstrates good attention to detail and understanding of software engineering/programming terms and concepts by: • Building atop the code base supplied by the Learning Facilitator • Attempting features and using data values that reflect a correct interpretation of the brief's requirements • Following any requirements provided by the Learning Facilitator via Blackboard or streams.	Demonstrates excellent attention to detail and understanding of software engineering/programming terms and concepts by: • Building atop the code base supplied by the Learning Facilitator • Attempting features and using data values that reflect a correct interpretation of the brief's requirements • Following any requirements provided by the Learning Facilitator via Blackboard or streams.	Demonstrates outstanding attention to detail and understanding of software engineering/programming terms and concepts by: • Building atop the code base supplied by the Learning Facilitator • Attempting features and using data values that reflect a correct interpretation of the brief's requirements • Following any requirements provided by the Learning Facilitator via Blackboard or streams.

Understanding and application of concepts: Object-oriented paradigm **Total Percentage for this Assessment Attribute = 25%**	Uses variables and functions in main program, instead of defining classes with data and functions. Uses few or no classes. Does not use constructor functions or uses but does not achieve enough functionality in the code.	The project displays some object-oriented design, readability and maintainability through: • Use of classes and constructors • Use of principles and application of practices taught and demonstrated in class modules, base code and learning activities.	The project displays good object-oriented design, readability and maintainability through: • Use of classes and constructors • Use of principles and application of practices taught and demonstrated in class modules, base code and learning activities.	The project displays excellent object-oriented design, readability and maintainability through: • Use of classes and constructors • Use of principles and application of practices taught and demonstrated in class modules, base code and learning activities • Addition of one or more extra features.	The project displays outstanding object-oriented design, readability and maintainability through: • Use of classes and constructors • Use of principles and application of practices taught and demonstrated in class modules, base code and learning activities • Use of principles and application of practices reflecting further research • Addition of multiple extra features.
Testing of program features, polish of user experience, clarity of communication	The program has showstopping and/or severe bugs. Few or none of the following are somewhat true of the programs: • The user interface clearly communicates the state of the program • Progression is intuitive and smooth	The program has some bugs evident under expected usage. For several stages of the programs, some of the following are somewhat true: • The user interface clearly communicates the state of the program • Progression is intuitive and smooth	The program has minor bugs evident under expected usage. For several stages of the programs, the following are somewhat true: • The user interface clearly communicates the state of the program • Progression is intuitive and smooth	The program has minor or no bugs evident under expected usage. Throughout much of the programs, the following are true: • The user interface clearly communicates the state of the program • Progression is intuitive and smooth	The program has no bugs evident under expected usage. Throughout the programs, the following are true: • The user interface clearly communicates the state of the program • Progression is intuitive and smooth

Total Percentage for this Assessment Attribute = 25%	• Erroneous inputs are anticipated and met with clear, instructive error messages • The interface is easy to read due to spacing, use of colour, glyphs and screen clearing.	• Erroneous inputs are anticipated and met with clear, instructive error messages • The interface is easy to read due to spacing, use of colour, glyphs and screen clearing.	• Erroneous inputs are anticipated and met with clear, instructive error messages • The interface is easy to read due to spacing, use of colour, glyphs and screen clearing.	• Erroneous inputs are anticipated and met with clear, instructive error messages • The interface is easy to read due to spacing, use of colour, glyphs and screen clearing • Any extra features are well integrated.	• Erroneous inputs are anticipated and met with clear, instructive error messages • The interface is easy to read due to spacing, use of colour, glyphs and screen clearing • Extra features are seamlessly integrated.
Technical and delivery requirements **Total Percentage for this Assessment Attribute = 15%**	Uses a different development environment, programming language and/or version of each to those specified by the Learning Facilitator Fails to adhere to submission guidelines when packaging and delivering the correct working files/project Projects do not compile at all or only after significant modification. The group contract is incomplete or has been poorly completed.	Uses the development environment, programming language and version of each specified by the Learning Facilitator. Adheres somewhat to submission guidelines when packaging and delivering the correct working files/project. Projects compile and runs with little or no modification. The group contract has been partially completed. Insufficient consideration has been given to the details. The roles and responsibilities for each group member are stipulated but these are not always clearly defined.	Uses the development environment, programming language and version of each specified by the Learning Facilitator. Adheres mostly to submission guidelines when packaging and delivering the correct working files/project. Projects compile and run without modifications. The group contract has been completed but there is room for improvement. There is evidence that the group have applied effort. The roles and responsibilities for each	Uses the development environment, programming language and version of each specified by the Learning Facilitator. Adheres almost entirely to submission guidelines when packaging and delivering the correct working files/project. Projects compile and run without modifications. The group contract has been completed to a good standard. The document shows that the group have a good grasp of what they want to achieve and expectations are clearly detailed.	Uses the development environment, programming language and version of each specified by the Learning Facilitator. Adheres entirely to submission guidelines when packaging and delivering the correct working files/project. Projects compile and run without modifications. The group contract has been completed to an excellent standard. The document reflects all areas clearly. There are clear and well-structured explanations in

	The roles and responsibilities for each group member are not stipulated and/or are not clearly defined. The group contract has not been submitted.		group member are clearly defined.	There are clear explanations in relation to the roles and responsibilities for each group member.	relation to the roles and responsibilities for each group member.
Code commenting and documentation **Total Percentage for this Assessment Attribute = 10%**	Comments are: • Not present and/or minimal beyond those provided in the code base or adds little clarity to the intent of code • Excessive and/or drowns out the more helpful comments. • Does not credit sources of acquired code.	Comments somewhat improve clarity and maintainability of code by: • Describing intent of passages of code that are not quickly self-evident • Being concise and deployed in moderation and does not appear next to simple/self-describing code. • Credits some sources of acquired code.	Comments improve clarity and maintainability of code by: • Describing intent of passages of code that are not quickly self-evident • Being concise and deployed in moderation and does not appear next to simple/self-describing code. • Credits some sources of acquired code.	Comments substantially improve clarity and maintainability of code by: • Describing intent of passages of code that are not quickly self-evident • Being concise and deployed in moderation and does not appear next to simple/self-describing code. • Credits all sources of acquired code.	Comments greatly improve clarity and maintainability of code by: • Describing intent of passages of code that are not quickly self-evident • Being concise and deployed in moderation and does not appear next to simple/self-describing code. • Credits all sources of acquired code.

The following Subject Learning Outcomes are addressed in this assessment	
SLO a)	Identify user requirements in computer-driven software applications.
SLO c)	Apply software programming methodologies including object-oriented programming (OOP) paradigms to build software solutions.
SLO d)	Test and validate software solutions that satisfy project requirements.