Assignment Specification -V1

Learning objectives

1. Design and develop software adhering to basic modularity concepts, verify whether code is following basic modularity concepts, and refactor if not.
2. Design test cases using black-box and white-box testing approaches.
3. Implementing test codes based on test cases, execute them, and test a production code.
4. Use version controlling to keep track of a small software engineering project.

1. Introduction

Basic concepts of version controlling, modularity, and software testing are the key topics you have learned in the second half of this unit. This assessment evaluates your competency in above areas of software engineering. It expects the knowledge you have gained via lectures, practical classes, and continuous assessments to design and implement a simple piece of software following modularity principles, verifying your code against a review checklist and refactor it, if needed, and design test cases and test the code you have developed. You will document your work in each stage so that another person can understand the work you have done and reflect on your own work for improvements. You will use a version control system to keep track of all the activities of the whole assessment task.

Your work will be assessed based on the code, documentation you produced, and a demonstration of your work.

Mark of the assignment is given out of 100 and the assessment is worth 50% of your final mark (overall mark of the unit) as specified in the unit outline.

2. Scenario

A software company develops tools useful in numerology analysis and related areas. Two of the simple scenarios the project wishes to consider are as follows:

A. When a birthday is provided, find:
   - the Life Path Number is to be calculated.
   - the Lucky Colour is to be identified.
   - if the life path number is a master number or not.

   When two birthdays are given, indicate if the Life Path Numbers of the persons of the birthdays are same or not.
   There are different ways to calculate the Life Path Number and know the lucky colour based on the Life Path number. For simplicity, the calculation method, Master Numbers and the Lucky colours shown in the Figure 1 are agreed to be used. (Figure 1 is available in the Assignment page of Blackboard).

B. When the birthday of a person is provided, find the generation the person belongs to.
   For simplicity, generations shown in Figure 2 is agreed to be used. 1 (Figure 2 is available in the Assignment page of Blackboard)

   NOTE: Only the birthdays between year 1901 and 2024 are to be considered in both scenarios. The month can be either provided as a number(e.g., 10) or as month name (e.g., October)

3. Detailed description

Considering the scenarios given above (Figure 1 and Figure 2), you will (a) design and implement suitable production code following modularity guidelines, and (b) design test cases and test your production code using proper testing processes. You will use basic version control methods to keep track of your work.

You are expected to cover all functionalities expected in the scenarios, with any supporting functionality needs to fulfill the given tasks, such as a checking the validity of inputs (as per the scenarios). When designing your production code, you much think about testing also.

Choosing how to take input, and make available output:
You are free to decide how to handle imports/exports of your software modules. Imports can be parameters, keyboard inputs, or text files. Exports can be available via return values, displaying on screen, or writing to a text file. Look at the section 2 -4 of the detailed description to know why these considerations are important.

You are expected to think about how easily the overall software you design can accommodate more functionalities later and reusability (such as providing more information when a Life Path Number is known).

Your work will be organized in seven parts as described below.

1. Version Control: *Apply version control process to keep track of a simple software project* [16 marks]

   First, read all assessment tasks and get a clear idea of what you are supposed to do in this assignment. You are supposed to use version controlling and keep track of all your work (documents and code both) relate to this assignment in a single Git repository. All your code must be in a directory called "code" and documents should be in a directory called "documents".

   Derive a short plan, identifying what branches you will need, and why you need them, and when the branches will be merged.

   Now, create a **Git local repository** **(do not use GitHub)** for this assessment using your surname and ID as part of the repository name following the format <surname>_<firstname><ID>_ISErepo). Commit all code and documents you create in the rest of the assessment as you create and modify them. **Make sure you commit your changes as and when you are doing them, but not all at once at the end**.

   Note: There is no hard rule about what each commit should contain, and you are expected to show your ability to use version control meaningfully.

2. Modularity: Design modules following good design principles [9 marks]

   Looking at the given scenario, identify **most suitable modules** you will need for a software to achieve the required functionality, considering the good modularity principles discussed in lectures/ worksheets.

   a. Write down module description/s for modules you decided to implement. You must apply good modularity principles you have learned in lectures/ worksheets when identifying and planning modules. Each module description should include short, meaningful name, clear and detailed explanation on what the intended task of the module, how it gets inputs to do the intended task and how the outputs make available, as shown in the module description examples of Lecture 8, slides 24-39, Lecture 9 slides20-33.

How imports, if any, of the modules are to be handled:
You are free to decide which input method/s are to be used in which of your modules (parameter passing, keyboard entry, or reading from text files.).

How exports(results), if any, of the modules are to be handled:
You are free to decide which input method/s are to be used in which of your modules. (Return values, display on the screen, or write to files).
However, in the Testing section, you must demonstrate your ability to test different input methods/output methods. Therefore, design different modules to use different import/export methods so that you can demonstrate testing skills.

You should decide on suitable, meaningful names for all modules and variables. You may make assumptions, if you think they are required, and state them clearly.

b. Explain your design decisions, and how good modularity principles are considered in your modules.

Note 1
In this unit, "modules" refers to any sub-part of a program. The terms "module" and "sub-module" are used interchangeably. The module/s you design in this stage affect the rest of your assessment. **Therefore, read the whole assignment properly before finalizing the answer for this section. You are strongly advised to use iterative approach especially for task 2 and 4**.

3. Modularity: Implementing the production code, reviewing, and refactoring [20 Marks]
   a. Implement the modules designed in the part 2 above considering the good modularity principles discussed in lectures/ worksheets.
   b. You can use either Python (Python3) or Java for your implementation. Execute and verify that you have got your code running without syntax errors.
      Note: This step will create a production code, not a test code.
   c. Create a short review checklist to check whether you have followed good modularity principles. You are expected to cover all basic guidelines covered in lecture 7.
   d. Review your code using the prepared checklist, identifying any issues. You must use the format suggested in the worksheet 7 to record your results. Each module must be reviewed for modularity issues.
   e. If you have identified any issues, refactor your code to address such issues. Explain how your code is improved now. If no refactoring is required, justify your decision.
   f. After refactoring, revise your preliminary descriptions of your modules and shown as revised module descriptions.

4. Test design [25 marks]
   a. Black box test design [16 Marks]

   Based on the module descriptions written by you in part 2 above (OR the revised version after refactoring in Part 3 above, if any revision is done), design suitable test cases to test your modules using following methods:

      I. equivalence partitioning.
      II. boundary value analysis.

You must design test cases using equivalence partitioning approach for all modules you have defined. You may use boundary value analysis for some of the modules only. Mention clearly which approach is used in each test design. Describe briefly how you decide your test cases.

b. White-box test design [9 marks]

Look at your code implemented in part 3 above and identify <u>at least two modules</u> where white-box testing approach will be beneficial. Design test cases to cover functionality of the selected modules using white box testing approach. Test <u>at least two different types of constructs</u> relate to paths you have learned. Indicate clearly which modules are tested with white-box approach. Describe briefly how you decide your test cases.

Note 2:

In this section (Test design), showcase your ability to test wide verity of situations:

- Testing test data of different data types: Integers, Strings, Boolean values.
- Testing various forms of inputs: parameter passing, keyboard input, text files.
- Testing various forms of outputs: return values, console outputs, text files. (Refer "General instructions" section for more information)

You are not required to consider all above situations in all your modules, and different modules can be designed in such a way that different situations can be tested.

Note 3:

Your test data must include following data items among other test data items (in any test cases of your choice, either in a) or b) above):

Last four digits of your student ID, and your last name as appear in your ID.

5. Test Implementation [20 Marks]

Implement your test designs for part 4 and part 5 above using either Python or Java. You may use test fixtures to organize your test code. Using a unit test framework is optional.

Run your test case and obtain results. Identify any test failures and then attempt to improve your code.

6. Summary table of your work (Traceability matrix) [5 marks]
Produce a table which shows the following information to help you and the marker to check the work you have done (E BB : black-box, WB: white-box P: equivalence partitioning, BVA: boundary value analysis).

	Design of test cases					*Test code implementation and execution*		
Module name	*BB (EP)*	*BB (BVA)*	*WB*	*Data type/s*	*Form of Input/output*	*EP*	*BVA*	*White-Box*
xxx	done	done	not done					
yyy	done		done					
….								

You may note the total marks as per the above detailed description is 90. The rest of the marks (10 marks) will be allocated to marks of documentation (for the Cover page, Introduction, Discussion, formatting etc.) as explained in section 7.

4. Documentation

You need to document what you have done in each stage of the assessment so that another person can get a clear idea about what you have done. You must produce a short report, collating all your work as part of your submission.

Your report name must be of the following format:

"< YourFirstName><YourLastName>_<your student ID>_ISEReport"

Your report must be written in markdown. Refer https://www.markdownguide.org/ if you are not familiar with the markdown (.md) format. This file format is helpful when tracking your documentation using a version control system.

Your report should include following sections and content:

a. Cover page [1 mark]
   Include the assessment name, your name as in Blackboard, Curtin student ID, practical class (date/ time). This may not be in a separate page, but as the first thing in your document in a clear format.
b. Introduction [1 mark]
   A brief overview of work you have done.
c. Module descriptions [7 marks, part 2 of the assessment task]
   Original module descriptions you have created for part 2 of this assessment.
   An explanation on your design decision, and why you have chosen modules as the way you have done and any assumptions you have made, if any.
d. Modularity [10 marks out of 20 marks of the part 3 of the assessment task]
   A description on how to run your production code with correct commands.
   Sample output of running your production code. You must use screen shots to support your answer in this section.
   A brief explanation on how different modularity concepts is applied in your code.
   Your review checklist, results of reviewing your production code using the review checklist, with explanation on your results, and refactoring decisions.
   Revised module descriptions resulted after refactoring, if any (after doing the part 3 of the detailed description)
e. Black-box test cases [16 marks of the part 4 (a) of the assessment task]
   All test cases you have designed for part 4 of this assessment, produced in the tubular test design format shown in lecture 8, assumptions you made if any, and brief explanation on your test design decisions.
f. White-box test cases [9 marks of the part 4 (b) of the assessment task]
   All test cases you have designed as the answer for part 5 of this assessment, produced in the tubular format shown in lecture 10, brief explanation on your test design, and any assumptions you made.
g. Test implementation and test execution [3 marks out of 20 marks of the part 5 of the assessment task]
   A brief description of how to run your test code with correct commands.
   Results of test execution with outputs of test successes and failures, with short discussion of results/improvements from part 5 of this assessment.
   You must use screen shots to support your answer in this section.
h. Summary of your work (traceability matrix)[5 marks]
   Table you have produced in the part 6 of this assessment.
i. Version control [3 marks out of 16 marks of the part 1 of the assessment task]
   Log of the use of your version control system (image of the log is sufficient), any explanation

/discussion on version control. (refer part 1 of the detailed description)

j. Discussion [5 marks]
Reflect on your own work including summary of what you have achieved, challenges you have faced, limitations and ways to improve your work with other features you have not considered, and any other information you wish to present.

Your report would be around 12-20 pages.

5. What you will be submitting

Your submission will be done in **three** steps.
Submission step 1:
You must submit a signed and dated assignment cover sheet in PDF format to the "**Assignment submission: cover sheet"** link provided in the assignment folder. Blank assessment coversheet is available under Assessments page of Blackboard. You can sign a hard copy and scan it, or you can fill in a soft copy and digitally sign it.

Submission step 2:
Your documentation, i.e., your report (refer section 4), should be submitted to the Turnitin link ("**Assignment submission: step 2 documents only"**) provided in the assignment folder. Your report submitted to this link should be in PDF format. (**You must convert your report in markdown format to a PDF file before submitting to this link**).
Your report name must follow the following format:
" < YourFirstName><YourLastName>_<your student ID>_ISEReport"

Submission step 3:

You must submit single zip file of all the work produced in this assessment to the "**Assignment submission: step 3**" link provided in the assignment folder. First, create a folder with name

< YourFirstName><YourLastName>_<your student ID>_ISEAssignment. Then place all your work inside this folder. Example : *JohnWhite_12134567_ISEAssignment*

Avoid using file formats other than *.zip* file to reduce delays in marking.

The folder should contain:

a. All you code files: You must submit all your code files (production and test codes), any sample data files (if created) and any other files used/ resulted in part 3 and 6 of this assessment. Name your files in appropriate manner.
b. A single Readme file: a short text file indicating the purpose of each file you have submitted.
c. .git sub directory of the Git repository you have used in the assessment.
d. Your report (refer section 4) in markdown format (.md file)
e. Your report (refer section 4) in PDF format which was already submitted to the Turnitin link.
f. Your signed and dated assignment cover sheet in PDF format which was already submitted to the "Assignment submission cover sheet" link.

Zip this folder and submit to the "**Assignment submission: step 3**" provided in the assignment page.

Make sure that your zip file contains what is required. Anything not included in your submission may not be marked, even if you attempt to provide it later. It is your responsibility to make sure that your submission is complete and correct.

6. Demonstration

A short demonstration (15-20 minutes) will be held after the submission and the schedule will be announced later.

7. Marking guide

Your work will be assessed based on (a) the code submission (b) the report, and (c) the demonstration of your work. Look at section 5 of this document for submission instruction.
Different evaluation components will have marks allocated as per the table 1 below. Detailed mark allocation for sub-sections can be seen in table 2.

Table 1: Mark allocation for different submission/evaluation components

Evaluation component	Total mark
Code submission Implementation of production code and test code (Part 3 and Part 6 of the detailed description), and the .git directory (version controlling), fulfilling the assessment requirements.	20 marks
Documentation, based on the work you have submitted to the "Assignment submission: step 3" links	60 marks
Demonstration of your work	20 marks

Table 2: Mark allocation for assessment tasks and how they are assessed.

Assessment task	Allocated Mark	Mark allocation for evaluation components		
		Demonstration	Code submission (Submission step 3)	Documentation (Submission step 2)
Applying version controlling to a small software project using Git.	16	5	8	3
Defining suitable modules to solve a simple problem using sound modularity principles.	7	-	-	7
Implementing production code, prepare a review checklist and review code, and refactor code.	20	5	5	10
Test design using black- box design approach	16	-	-	16
Test design using white-box design approach	9	-	-	9
Implementing test code, execute test code, and improve code as per the results.	20	10	7	3
Introducing your work, traceability matrix, and reflection(discussion)	12	-	-	12
Total	100	20	20	60

8. Requirement to pass the unit.

As specified in the unit outline, you should score at least 40% of the final assessment to pass the unit. This assignment is your final assessment; therefore, you need to get at least 40% of the marks of this assignment to pass the unit.

Exact mark breakdown in Sections 3 - 4 and Section 7 of this document represent maximums, achieved only if you completely satisfy the requirements of the relevant section.

Plagiarism is a serious offence. This assignment has many correct solutions so plagiarism will be easy for us to detect (and we will). Please read the *Coding and Academic Guidelines* on Blackboard and for information about plagiarism, please refer to http://academicintegrity.curtin.edu.au
In summary, this is an assessment task. If you use someone else's work or someone else's assistance to help you complete the part of the assessment, where it's intended that you complete it yourself, you will have compromised the assessment. **You are prohibited from obtaining the support of AI tools when attempting this assessment**. Refer unit outline for more information on this aspect. You will not receive any marks for any parts of your submission that are not your original work. In the case of doubt, you may be asked to explain your code and the reason for choices that you have made as part of coding to the unit coordinator. A failure to adequately display knowledge required to have produced the code will most likely result in being formally accused of cheating. Finally, be sure to secure your code. If someone else gets access to your assignment for any reason (including because you left it on a lab machine, lost a USB drive containing the code or put it on a public repository) you will be held partially responsible for any plagiarism that results.

9. Late submissions

You must submit the assignment on the due date. Acceptance of late submissions is not automatic and will require supporting documentation proving that the late submission was due to unexpected factors outside your control.
Note that external pre-scheduled commitments including, but not limited to, work, travel, scheduled medical, sporting, family or community engagements are not considered for unexpected factors outside your control. If you know you have, or are likely to have, such engagements and that they may affect your ability to complete the assignment, you will be expected to have planned your work accordingly. This may mean that you need to start and/or complete your assignment early to make sure that you are able to hand it in on time. Also note that IT related issues are almost never a valid excuse.
Refer the unit outline to know how penalty will be applied in case of late submissions. As per the university policy, any submission made after seven (7) calendar days of the due date will not be marked and you will automatically fail the unit.
Note that if you are granted an extension, you will be able to submit your work up to the extended time without penalty – this is different from submitting late.

10. Clarifications and Amendments

This assignment specification may be clarified and/or amended at any time. Such clarifications and amendments will be announced in the lecture and on the unit's Blackboard page (not necessarily at the same time and not necessarily in that order). These clarifications and amendments form part of the assignment specification and may include things that affect mark allocations or specific tasks. It is your responsibility to be aware of these, either by attending the lectures, watching the recordings and/or monitoring the Blackboard page.

11. General instructions

- You must develop your code / perform testing in Linux environment. Your code must run in a Linux command line environment in Curtin computing labs. If not, the code will not be marked.
- Your code can be in either Python or Java.
- Remember to start small, come back, and revise your work iteratively, especially in **part 2 and 3** of the detailed description. If you spend more time in thinking and designing what you would do, rather than quickly try to implement something, you would be able to do much better in this assignment. Think about the total marks allocation (given as out of 100), time you would spend on each section and mark allocation for each section very carefully.
- **Read the assignment <u>fully</u> before start answering the assignment.** The design decisions you will taking in initial sections of the assignment will affect your ability to showcase your competency in testing/ modularity. Therefore, spend reasonable time on thinking about what you are going to do and how you are going to do it, before starting coding/ test designs.
- You may need to revise your work few times until you are satisfied with it. Your version control history will show these revisions.
- Design your test cases to demonstrate wide range of abilities you have developed:
    - Testing test data of numeric types (integers), strings, and Boolean values.
    - Testing inputs receiving as parameter passing, keyboard input, or text files.
    - Testing outputs that made available as return values, console outputs, or text files.
    - Testing loops, if-else statements, exceptions.

- Start your work by creating a repository and make sure all your work from the beginning is stored there. If you do all your work in your local machine without committing to Git and commit only the final version there, you will not be able to show your ability to use basic functions of git effectively.
- Your submission to Blackboard and your Git repository will be evaluated separately to avoid use of Git affecting other parts of the assignment.

End of the Assignment Specification