Question	Points	Score
Question 1	9	
Question 2	6	
Question 3	10	
Question 4	15	
Question 5	10	
Total	50	

Answer all questions in the space provided here.

Part A. Unit Testing

Question 1 .... **9 marks**

Originally, *Fizz buzz* is a group word game for children to teach them about division. It has gained popularity in being used as a simple interview question for screening software developers.

For example, in a simplified variation of the Fizz buzz question, the program takes as input one positive i integer between 1 and 100, both inclusive. As output, the program prints the number i itself within the range ($[1, 100]$) when it is not divisible by neither three nor five. For multiples of three the program should print "`Fizz`" instead of the number and for the multiples of five the program should print "`Buzz`". Finally, for numbers which are multiples of both three and five the program should print "`FizzBuzz`" instead.

(Continued overleaf)

(a) (5 marks) Develop equivalence classes given the above specification for the input variable.

(b) (4 marks) Based on your equivalence classes, develop a decision table for the above Fizz buzz problem.

Question 2 .... **6 marks**

In Lecture 3 we introduced the Equivalence Class Testing (ECT) technique. In ECT, we first partition the values of input variable into a number of equivalence classes. Depending on robustness requirements (normal or robust) and fault occurrence assumptions (single or multiple), test cases are then drawn from the equivalence classes.

For a specification with n number of variables $x_1, x_2, \ldots, x_n$ over some numerical domains. The valid values of each variable x_i is partitioned into a number of equivalence classes, each of which is bounded from both sides (in other words, none of the equivalence classes goes down to $-\infty$ or up to ∞). The number of equivalence classes for x_i is denoted r_i. There are gaps between the ranges. In other words, for each variable, all its equivalence classes are separated by ranges of invalid values.

The minimum number of test cases for ECT is a function of the number of valid ranges. Using the above notation,

(a) write a formula for the minimum number of test cases for **weak robust ECT**, and

(b) briefly explain why.

Question 3 . . . . **10 marks**

The binary search tree (BST) is a data structure that is very efficient in sorting, search and in-order traversal. A BST has the following properties:

- all nodes are comparable,
- all nodes of a node's left subtree are *less than* the node itself,
- all nodes of a node's right subtree are *greater than* the node itself,
- Each subtree is a BST, and
- there are no duplicate nodes.

The insertion of a node into a BST can be specified using the algorithm.

```
Input: node                                    ▷ Node to be inserted.
Input: root                                    ▷ The root node of the BST.
if root = null then
    root ← node
    return
end
while root ≠ null do
    if node = root then                        ▷ Node already in the BST
        return
    else if node < root then                   ▷ Insert left
        if root.left = null then
            root.left ← node
            return
        else
            root ← root.left
        end
    else                                       ▷ Insert right
        if root.right = null then
            root.right ← node
            return
        else
            root ← root.right
        end
    end
end
```

Algorithm 1: The insertion operation of the binary search tree.

(Continued overleaf)

(a) (5 marks) Draw the program graph for the above function.

(b) (5 marks) Recall that McCabe's essential complexity measures how *unstructured* the logic of a program is by calculating the Cyclomatic complexity of the condensed program graph. In this part,

i. draw the final condensed graph for the program graph you came up with in part (a) above, and

ii. calculate the Cyclomatic complexity of the condensed graph you draw.

Part B. Integration Testing, System Testing & Object-oriented Testing

Question 4 .... **15 marks**

During the semester we've been working on the *Fly me to Mars* project. In this project, the goal is to create a simple Web application for the management of Mars mission personnel registration, dealing with `Persons`, `Missions` and `Invitations`. In the code base of the last part of the project, individual components have been integrated together to make the Web application functional.

With some simplification, the steps and rules for creating of the `Invitation` can be described below.

1. The `Router` intercepts and dispatches all user requests. Once the router identifies a request to create an invitation for a given mission, it dispatches the request to the `CreateInvitation-Resource` object to handle. The resource then determines whether it requires authentication and authorisation.
2. Only an authenticated user can create an invitation. The authentication of a user is performed by `LoginManager`, together with `EntityRetriever` that retrieves all entities from the underlying database. The `LoginManager` then decides whether a valid user is presented. If not an error message is generated.
3. The authorisation is performed by the `AccessDecisionManager`, which is supplied with parameters a user, an entity type to be created (`Invitation`) in this case), and the mission for which the invitation to be created, all of which are provided by `CreateInvitationResouce`. Again, `EntityRetriever` is used to retrieve the mission by its ID. The `AccessDecisionManager` then grants/denies access based on whether the user is the captain of the mission.
4. If access is granted, the `CreateInvitationResouce` object then proceeds to create the invitation. It firstly retrieves, through `EntityRetriever`, the recipient `Person` object. It then checks whether the invitation's creator is the same as the recipient. If this is the case the creation process is aborted and an error message is generated.
5. Otherwise, the resource object invokes `EntityFactory` to create the invitation object, and then invokes the `InvitationHandler` object to send the invitation to the recipient. It also invokes the `EntitySaver` to save the invitation object. Finally, the `EntityUpdater` object is invoked to update the mission and the recipient objects.
6. Eventually, the resource completes processing and returns control to the router.

(Continued overleaf)

(a) (9 marks) Draw a sequence diagram depicting the interactions between the above objects. Note that for simplicity reasons, do not include DAO objects as `PersonDAO` and model objects such as `Person`, `Mission` and `Invitation`.

(b) (6 marks) The above components can be hierarchically organised into 4 layers. Based on the sequence diagram you developed in part (a), do the following.

(i). Draw the hierarchical call graph of the system.

(ii). Calculate the maximum number of *drivers* that need to be developed if the *bottom-up* integration testing approach is adopted.

(iii). Calculate the minimum of test cases if the *pairwise* integration testing approach is adopted.

Part C. Software Metrics

Question 5 .... **10 marks**

In lecture 8 we introduced Weyuker's 9 properties to evaluate software metrics. Some of the properties (for example, properties 1, 3, 4 and 8) are quite simple and intuitive. However, some other properties are a bit more complex and need further analysis.

Weyuker's property 9 states that the complexity of the composition of two programs may be greater than the sum of the complexities of the two taken separately. More formally,

$$\exists A, B \colon Program \bullet M(A) + M(B) < M(A + B)$$

where M represents a given metric and $A + B$ represents the composition of A and B.

The object-oriented *LCOM1* measures the lack of cohesion within a given class. It is defined as follows.

Input: C ▷ `The class that is being measured.`
Output: The LCOM1 value.

$mthds \leftarrow methods(C)$ ▷ `All methods of the class`
$P, Q \leftarrow 0$ ▷ `Initialise temporary variables`
foreach *pair of distinct methods* $m_i, m_j \in mthds$ **do**
  if $attr\_acc(m_i) \cap attr\_acc(m_j) = \emptyset$ **then** ▷ m_i `and` m_j `access disjoint sets of attributes`
    $P \leftarrow P + 1$
  else
    $Q \leftarrow Q + 1$
  end
end
if $P > Q$ **then return** $P - Q$ **else return** 0

Algorithm 2: The calculation of LCOM1.

Note that $methods(C)$ returns the set of methods of a given class C and $attr\_acc(m)$ returns the set of class attributes the method m accesses.

For clarity reasons, we will make the following assumptions.

- Programs A and B are both classes in the object-oriented sense.
- Composition is defined by inheritance of one class to another class: $A + B$ means adding B as a new super class of A.

For Weyuker's property 9 and the metric LCOM1, do the following:

(a) State whether the property holds or not.

(b) Prove your claim (informally).

End of the paper