Assignment 3—Solution

Handout: **Tuesday, November 1st, 2022**
Due: **23:59, Friday, November 11th, 2022**

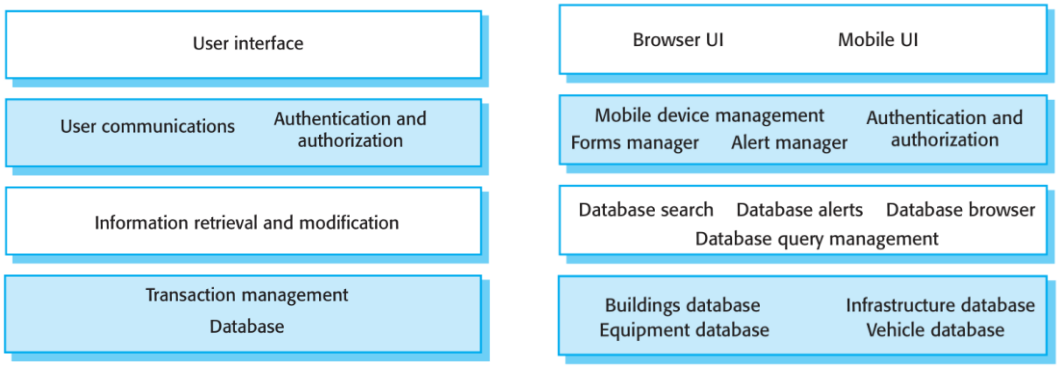
1. Architectural Design (10 marks)

When describing a system, explain why you may have to design the system architecture before the requirements specification is complete. (≤100 words)

The architecture may have to be designed before specifications are written to provide a means of structuring the specification and developing different sub-system specifications concurrently, to allow the manufacture of hardware by sub-contractors, and to provide a model for system costing.

2. Layered Architecture (18 marks)

An information system is to be developed to maintain information about assets like buildings, vehicles, and equipment owned by a utility company. It is intended that the information will be updatable by staff working in the field using mobile devices as new asset information becomes available. The company has several existing asset databases that should be integrated through this system. Design a layered architecture for this asset management system based on the generic information system architecture shown below. In other words, you need to decide and explain, e.g., using a similar diagram, which specific software components and/or functionalities of the information system should be assigned to which layers.



A possible design for the architecture is given above.

2. Defect Testing (12 marks)

Explain why testing can only detect the presence of errors, but not their absence. (≤100 words)

This is because complete testing is impossible for almost all non-trivial software systems. Here, complete testing means "There are no undisclosed faults at the end of test phase."

First, testing can never show certain requirements (e.g., a system should always terminate) are not satisfied. Second, it is often impractical to execute a program in all possible ways due to its large input domain and/or infinite execution environment.

4. Software Testing (12 marks)

Suppose a developer has written a method named `canFormTriangle`, as shown on the next page, that a) takes three `double` parameters named `x`, `y`, and `z` as the input, b) treats the input values as denoting the lengths of three line segments, and 3) returns a `boolean` value indicating whether the three line segments can legitimately form a triangle. Note that three line segments of lengths `x`, `y`, and `z` can legitimately form a triangle if and only if they satisfy the following two conditions: a) `x`, `y`, and `z` are all greater than 0; b) the sum of any two of them is greater than the third.

```
1  boolean canFormTriangle(double x, double y, double z){
2      if(x <= 0 || y <= 0 || z <= 0)
3          return false;
4
5      if(x + y <= z)
6          return false;
7      else if(x + z <= y)
8          return false;
9      else if(y + z <= x)
10         return false;
11     else
12         return true;
13 }
```

Please apply partition testing to devise a set `T` of unit tests for method `canFormTriangle` such that, for every possible unit test `t` of the method, there is a unit test `t'` in `T` satisfying that `t` and `t'` have the same branch coverage on the method. Here, two tests `t1` and `t2` are said to have the same branch coverage on a method if and only if both conditions 1) and 2) are satisfied: 1) `t1` and `t2` exercise the same set of `if` statements in the method; 2) for each `if` statement exercised by both `t1` and `t2`, the corresponding `if` condition expression evaluates to the same value during `t1` and `t2`'s executions. For example, if both tests `t1` and `t2` exercise only the first `if` statement in method `canFormTriangle` and the condition expression on line 2 evaluates to `true` in both tests, `t1` and `t2` are said to have the same branch coverage on method `canFormTriangle`. Write down all your tests, with one test on each line.

For each test, you need to specify the values of parameters `x`, `y`, and `z`, the expected return result from the method, and the branches covered by the test. In the example below, `c2==true` indicates that only the if condition on line 2 is evaluated during the test execution and the condition evaluates to `true`. You may separate the branches using commas.

x	y	z	ExpectedResult	BranchesCovered
1	1	-1	false	c2==true
1	*2*	*4*	*false*	*c2==false, c5==true*
1	*4*	*2*	*false*	*c2==false, c5==false, c7==true*
4	*1*	*2*	*false*	*c2==false, c5==false, c7==false, c9==true*
2	*3*	*4*	*true*	*c2==false, c5==false, c7==false, c9==false*

How to hand in:

Submit your typed, instead of handwritten, answers in a single PDF file on Blackboard.