Office Use Only

MONASH University

Semester One 2016
Examination Period

Faculty of Information Technology

EXAM CODES:	**FIT5171**
TITLE OF PAPER:	**SYSTEM VALIDATION AND VERIFICATION, QUALITY AND STANDARDS - PAPER 1**
EXAM DURATION:	2 hours writing time
READING TIME:	10 minutes

THIS PAPER IS FOR STUDENTS STUDYING AT: (tick where applicable)

☐ Berwick	☐ Clayton	☐ Malaysia	☐ Off Campus Learning	☐ Open Learning
☒ Caulfield	☐ Gippsland	☐ Peninsula	☐ Monash Extension	☐ Sth Africa
☐ Parkville	☐ Other (specify)			

During an exam, you must not have in your possession any item/material that has not been authorised for your exam. This includes books, notes, paper, electronic device/s, mobile phone, smart watch/device, calculator, pencil case, or writing on any part of your body. Any authorised items are listed below. Items/materials on your desk, chair, in your clothing or otherwise on your person will be deemed to be in your possession.

No examination materials are to be removed from the room. This includes retaining, copying, memorising or noting down content of exam material for personal use or to share with any other person by any means following your exam.
Failure to comply with the above instructions, or attempting to cheat or cheating in an exam is a discipline offence under Part 7 of the Monash University (Council) Regulations.

AUTHORISED MATERIALS

OPEN BOOK	☒ **YES**	☐ **NO**
CALCULATORS	☒ **YES**	☐ **NO**
SPECIFICALLY PERMITTED ITEMS **if yes, items permitted are:**	☐ **YES**	☒ **NO**

Candidates must complete this section if required to write answers within this paper

STUDENT ID: __ __ __ __ __ __ __ __ __   DESK NUMBER: __ __ __ __ __

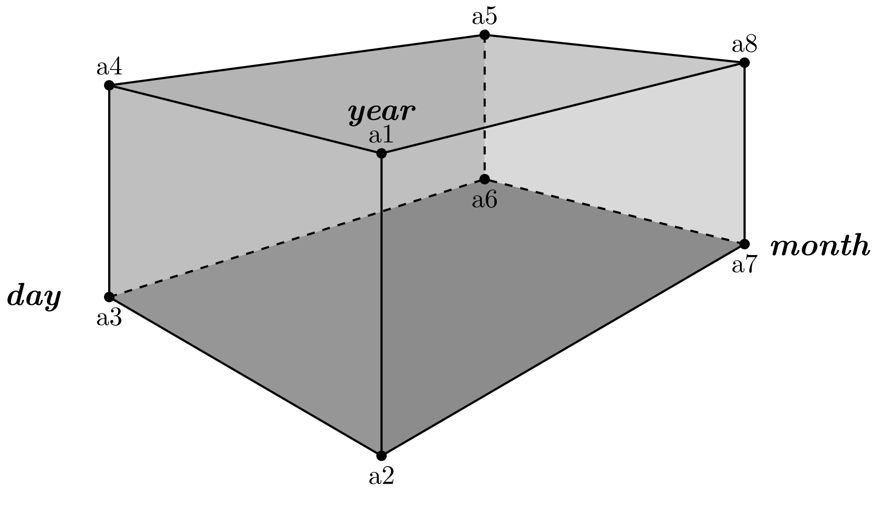
Answer all questions in the space provided here.

Question	Points	Score
Question 1	10	
Question 2	11	
Question 3	10	
Question 4	8	
Question 5	11	
Total	50	

Question 1 .... **10 marks**

In the lecture we discussed the testing of a **NextDate** method, which, given a day, a month, and a year, returns the date of the following day. Assume the year variable ranges over [1812, 2016].

Since there are three variables (day, month, and year), we can visualise the boundaries using a 3D plot, as follows.



Each of the 3 axes represents day, month, and year, respectively, and the internal of the 3D shape represents the valid values. Each of the end points in the rectangular prism, labelled a1, a2 to a8, represents a specific tuple of boundary values for the three variables day, month, and year. Point a2 represents the allowed min values for day, month, as well as year, i.e., (1, 1, 1812). Point a7 represents the allowed min values for day and year, but the allowed max value for month, i.e., (1, 12, 1812).

Given the above visualisation, we can reason about test cases for **strong**, **normal** boundary value testing (BVT) on points, lines, planes, and the prism itself. An example, for points, is given below.

Points. For each point (denoted a1 to a8), there are 8 test cases for the 3 variables. For example, for points a2 and a7, we have the following test cases at and around the min values of the 3 variables:

(a) Test cases for point a2.

day	month	year
1	1	1812
1	1	1813
1	2	1812
1	2	1813
2	1	1812
2	1	1813
2	2	1812
2	2	1813

(b) Test casse for point a7.

day	month	year
1	11	1812
1	11	1813
1	12	1812
1	12	1813
2	11	1812
2	11	1813
2	12	1812
2	12	1813

Hence, for all 8 points, we need 8*8=64 test cases.

Following the above example, for (1) lines, (2) planes, and (3) the prism, do the following

1. Give details of strong, normal BVT test cases.
2. Calculate the total number of test cases.

— blank page for answers if required. Will be marked. —
— Indicate clearly question number. —

Question 2 .... **11 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.

Algorithm 0: The insertion operation of the binary search tree.

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

(Continued overleaf)

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

(b) (2 marks) Calculate the cyclomatic complexity of the program graph in the previous part.

(c) (4 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,

- draw the final condensed graph for the program graph you came up with in part (a) above, and
- calculate the Cyclomatic complexity of the condensed graph you draw.

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

"Fizz Buzz" has been used as a simple interview question for software developers. In its simplest form, the program takes as input an integer value between 1 and 100 (both inclusive), and prints the number itself when it is not divisible by neither three nor five. For numbers which are multiples of both three and five the program should print "`FizzBuzz`" instead. Otherwise, for multiples of three the program should print "`Fizz`" instead of the number, for the multiples of five the program should print "`Buzz`". Code listing 1 below shows a simple implementation in Java.

Listing 1: An implementation of the "Fizz Buzz" puzzle.

```java
    public static String fizzBuzz(String input) {
        int x = Integer.parseInt(input);

        String result;

        if (x < 1) {
            throw new IllegalArgumentException("input should be >= 1.");
        } else if (x > 100) {
            throw new IllegalArgumentException("input should be <= 100.");
        }

        if (x % 15 == 0) {
            result = "fizzbuzz";
        } else if (x % 3 == 0) {
            result = "fizz";
        } else if (x % 5 == 0) {
            result = "buzz";
        } else {
            result = input;
        }

        return result;
    }
```

The 5 test cases in Code listing 2 below have been developed for the above method.

Listing 2: Test cases for the `fizzBuzz()` method.

```java
    @Test
    public void testFizzBuzz() {
        for (int i= 1; i <= 100; i++) {
            System.out.print(fizzBuzz(Integer.toString(i)) + " ");
        }
    }

    @Test(expected = Exception.class)
    public void wrongNumbersAreNotProcessed() {
        fizzBuzz("0");
    }

    @Test(expected = NumberFormatException.class)
    public void illealInputThrowsException() {
        fizzBuzz(" ");
        fizzBuzz("  ");
        fizzBuzz("   ");
    }

    @Test
    public void alphabeticTest() {
```

```java
22          try {
23              fizzBuzz("a");
24          } catch (NumberFormatException e) {
25              ;
26          }
27      }
28
29      @Test
30      public void threeGetsFizz() {
31          try {
32              String fizz = fizzBuzz("3 ");
33              assertEquals("fizz", fizz);
34          } catch (Exception e) {
35              e.printStackTrace();
36          }
37      }
```

(a) (2 marks) What is the statement coverage of this test suite?

Note that the lines you need to consider for calculation of coverage include line 2, lines 6–9, lines 12–19, and line 22. In other words, the total number of lines to cover is 14.

(b) (8 marks) There are some problems with some of these test cases. (1) List these errors, and (2) discuss how they can be fixed.

Question 4 .... **8 marks**

Weyuker's 9 properties were proposed 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,

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

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

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

Algorithm 1: The calculation of LCOM1.

```
   Input: $C$                                              ▷ The class that is being measured.
   Output: The LCOM1 value.
1  $mthds \leftarrow methods(C)$                            ▷ All methods of the class
2  $P, Q \leftarrow 0$                                      ▷ Initialise temporary variables
3  foreach pair of distinct methods $m_i, m_j \in mthds$ do
4      if $attr\_acc(m_i) \cap attr\_acc(m_j) = \emptyset$ then   ▷ $m_i$ and $m_j$ access disjoint sets of
        attributes
5          $P \leftarrow P + 1$
6      else
7          $Q \leftarrow Q + 1$
8      end
9  end
10 if $P > Q$ then return $P - Q$
11 else return 0
```

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 merging the two classes to form a new class. Attribute/methods with the same names are to be consistently renamed to avoid compilation errors.

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).

— blank page for answers if required. Will be marked. —
— Indicate clearly question number. —

— blank page for answers if required. Will be marked. —
— Indicate clearly question number. —

Question 5..**11 marks**

Mutation testing is a technique to assess the efficacy and quality of a test suite. It works by making *mutants*, syntactic variations of the program under test, and measuring how many of the mutants are *killed* by the test suite. The presence of non-equivalent *live* mutants represents inadequacy of the test suite.

The following Java method, `max`, returns the largest of the three integer parameters.

```java
    public int max(int a, int b, int c) {
        int temp = a;
        if (b > a) {
            temp = b;
        }
        if (c > b) {
            temp = c;
        }
        return temp;
    }
```

(a) (3 marks) Devise a test suite with the smallest possible number of test cases that achieves 100% *branch coverage*, or *DD-path coverage*.

(b) (5 marks) Come up with three *non-equivalent* first-order mutants of the original program, each making use one of the following mutation operators. Determine the *kill rate* of your test suite on each of the three mutants.

The mutation operators you can use are:

`ror:` Relational operator replacement.

`sdl:` Statement deletion.

`uoi:` Unary operator insertion.

`svr:` Scalar variable replacement.

`vie:` Scalar variable initialisation elimination.

(c) (3 marks) Is there a defect in the program? If so, develop the smallest set of test cases that achieves 100% statement coverage but *does not* reveal the defect. If not, develop the smallest set of test cases that achieves 100% statement coverage.

— Additional page for answers if required. Will be marked. —
— Indicate clearly question number. —

— Additional page for answers if required. Will be marked. —
— Indicate clearly question number. —

—— End of the paper ——