UNIVERSITY OF ESSEX

Undergraduate Examinations 2020

LARGE SCALE SOFTWARE SYSTEMS AND EXTREME PROGRAMMING

Time allowed: **TWO** hours (exam time) + **ONE** hour to allow for submission time (total **THREE** hours)
(Please see your exam timetable or check on FASER for the deadline to upload your answers)

The times shown on your timetable are in British Summer Time (BST) (GMT+1). Please check online for a conversion to your local time if you will be undertaking your assessment outside the United Kingdom

Candidates are permitted to use:
Calculator – Casio FX-83GT Plus/X or Casio FX-85 GT Plus/X ONLY

Candidates must answer **ALL** questions

The paper consists of **FOUR** questions

The questions are of equal weight

The percentages shown in brackets provide an indication of the proportion of the total marks for the **PAPER** which will be allocated.

If you have a query with the content of this exam paper please use the revision FAQ Forum on the module's Moodle page. Your academic will be available to answer any queries in real-time.

If you have a technical problem with FASER, or any other query, please go to Exams Website to find contact details of the teams that can help you.

Please note that the time allocated for this assessment includes time for you to download this question paper and answer paper and to upload your answers to FASER.

Please allow at least 30 minutes at the end of your exam time to upload your work. Once you have completed the assessment do not leave it to the last minute to upload.

Please save your work throughout the examination to avoid losing your work.

Please do not communicate with any other candidate in any way during this assessment. Your response must be your own work. Procedures are in place to detect plagiarism and collusion.

<u>Question 1</u>

Software development has always been hard. In a famous essay entitled “No Silver Bullet – Essence and Accidents in Software Engineering”, more than 30 years ago Fred Brooks wrote:

> *“There is no single development, in either technology or management technique, which by itself promises even one order-of-magnitude improvement within a decade in productivity, in reliability, in simplicity”*

There is ample agreement that the situation is largely unchanged if one uses traditional forms of software development.

(a) Explain what the reasons/causes are for this situation. [12%]

(b) Explain in what ways Extreme Programming acts on those reasons/causes, resulting in better software-development outcomes. [13%]

<u>Question 2</u>

(a) Extreme programming insists on the software being in a *deployable state at all times* (that is: at the end of every iteration the software can be demoed to the clients). What makes this possible? [12%]

(b) What role do customers/clients play in extreme programming? [13%]

Question 3

Your company, PneuFood, aims to reach every household and provide near-instant food deliveries from supermarkets via a network of pneumatic tubes. Pneumatic tubes are systems that propel cylindrical containers through networks of pipes by compressed air or by partial vacuum. They are used for transporting solid objects, as opposed to conventional pipelines, which transport fluids.

The company has built an initial network of pneumatic tubes with all of the necessary hardware to deliver food cylinders from any supermarket to any household, and now needs a software system that will allow users to operate the system. End users (households) pay fees to PneuFood to use the service. So, they are the company's clients and the software will need to satisfy their needs. [NOTE: Of course, also supermarkets are users of the system. However, you are asked to ignore this fact in your answers to this question.]

You are part of the extreme programming team that has been asked to develop the software system. End users will interact with the system through a web interface.

You are asked to sketch a release plan for the software. Given the specific nature of the project you can act both as a programmer and as a client.

In your release plan please specify:

(a) A list of stories identified for the release which capture key needs for the customers. [15%]

(b) A priority for each story in terms of value for the clients using a point system where 3 = very valuable, 2 = valuable, 1 = optional. [5%]

(c) A list of the stories that should be implemented in the first iteration. [5%]

<u>Question 4</u>

You have been asked to continue to develop a simple **Calculator** class, taking over from another programmer. Overleaf you will find two implementations of the class, one in Java and one in Python, together with unit tests for them.

The **Calculator** class can currently do two operations – addition and multiplication – on any two integer numbers. The result of such operations is stored in an accumulator register, **R1**, which is initially 0 (when an instance of calculator is created), overwriting any value previously stored in **R1**.

Users of the class call the method **evaluate** with a string argument representing the operation they want to execute. For instance, **evaluate("10 + 5")** will result in the value 15 being stored in **R1**, while **evaluate("5 * 5")** will result in 25 being stored in **R1**. Users can then retrieve the last result of an operation by calling the method **get_R1**.

(a) While the production code (the class **Calculator**) written by the programmer so far is reasonably good, *it violates some principles of clean code*, which is essential for Extreme Programming. Choose one implementation of **Calculator**, study it, and indicate which refactorings would be needed to make the code cleaner (use line numbers to indicate which elements of the code should be affected).
<u>Please, ignore the test class for this answer.</u> [10%]

(b) Write one or two tests which will force you to extend **Calculator** to add the following functionality: (1) the calculator will have a second register, **R2**, and (2) it will accept a new assignment instruction of the form **Register = Value**, which can set the value of either **R1** or **R2**. Thus, for example, **evaluate("R1 = 10")** will result in the value 10 being stored in **R1**, while **evaluate("R2 = 5")** will result in 5 being stored in **R2**. [8%]

(c) Suggest which changes to the code would be required for the implementation of **Calculator** to acquire the functionality and pass the test(s) mentioned above (use line numbers to indicate the elements of the code that would be affected). [7%]

calculator.py

```python
class Calculator:
    OPERATORS = ['+', '*']

    def __init__(self):
        self.R1 = 0

    def get_R1(self):
        return self.R1

    def add(self, a, b):
        self.R1 = a + b

    def multiply(self, a, b):
        self.R1 = a * b

    def evaluate(self, expression):
        elements = self.parse(expression)
        a = int(elements[0])
        b = int(elements[2])
        if elements[1] == "+":
            self.add(a, b)
        else:
            self.multiply(a, b)

    def parse(self, expression):
        position = self.find_operator(expression)
        return [
            # This is the first operand
            # (strip removes leading and trailing spaces)
            expression[:position].strip(),
            # This is the operator (e.g., *, /, -, ...)
            expression[position],
            # This is the second operand
            expression[position + 1:].strip()]

    def find_operator(self, expression):
        for operator in self.OPERATORS:
            operator_position = expression.find(operator)
            if operator_position != -1:
                return operator_position
        return -1
```

calculator_test.py

```python
import unittest
from calculator import Calculator


class CalculatorTest(unittest.TestCase):

    def setUp(self):
        self.calc = Calculator()

    def test_calculator_initial_value_of_register_should_be_zero(self):
        self.assertEqual(0, self.calc.get_R1())

    def test_calculator_can_add(self):
        self.calc.add(10, 20)
        self.assertEqual(30, self.calc.get_R1())

    def test_calculator_can_multiply(self):
        self.calc.multiply(10, 20)
        self.assertEqual(200, self.calc.get_R1())

    def test_calculator_can_parse_expression(self):
        parts = self.calc.parse("10 + 20")
        self.assertEqual("10", parts[0])
        self.assertEqual("+", parts[1])
        self.assertEqual("20", parts[2])

    def test_calculator_can_evaluate_expression(self):
        self.calc.evaluate("10 * 20")
        self.assertEqual(200, self.calc.get_R1())
```

Calculator.java

```java
public class Calculator {
   private int R1 = 0;
   private final char[] OPERATORS = {'+','*'};

   int getR1() {
      return R1;
   }

   void add(int a, int b) {
      R1 = a + b;
   }

   void multiply(int a, int b) {
      R1 = a * b;
   }

   public void evaluate(String expression) {
      String[] elements = parse(expression);
      int a = Integer.parseInt(elements[0]);
      int b = Integer.parseInt(elements[2]);
      if (elements[1].equals("+"))
         add(a, b);
      else
         multiply(a, b);
   }

   String[] parse(String expression) {
      int position = findOperator(expression);
      return new String[]{
              // This is the first operand
              // (trim removes leading and trailing spaces)
              expression.substring(0,position).trim(),
              // This is the operator (e.g., *, /, -, ...)
              expression.substring(position,position+1),
              // This is the second operand
              expression.substring(position+1).trim()};
   }

   private int findOperator(String expression) {
      for(char operator: OPERATORS) {
         int operatorPosition = expression.indexOf(operator);
         if ( operatorPosition != -1)
            return operatorPosition;
      }
      return -1;
   }
}
```

CalculatorTest.java

```java
import org.junit.Before;
import org.junit.Test;

import static org.junit.Assert.*;

public class CalculatorTest {

   private Calculator calculator;

   @Before
   public void setUp() {
      calculator = new Calculator();
   }

   @Test
   public void calculatorRegisterShouldInitiallyBeZero() {
      assertEquals(0,calculator.getR1());
   }

   @Test
   public void calculatorCanAdd() {
      calculator.add(10,20);
      assertEquals(30,calculator.getR1());
   }

   @Test
   public void calculatorCanMultiply() {
      calculator.multiply(10,20);
      assertEquals(200,calculator.getR1());
   }

   @Test
   public void calculatorCanParseExpression() {
      String [] parts = calculator.parse("10 + 20");
      assertEquals("10",parts[0]);
      assertEquals("+",parts[1]);
      assertEquals("20",parts[2]);
   }

   @Test
   public void calculatorCanEvaluateExpression() {
      calculator.evaluate("10 * 20");
      assertEquals(200, calculator.getR1());
   }
}
```

END OF EXAM PAPER CE320-6-AU/AT

Once you have completed your answers, please upload them to FASER http://faser.essex.ac.uk.

Remember to add your REGISTRATION NUMBER onto ALL documents that you upload.