CE320-6-AU/AT

UNIVERSITY OF ESSEX

Undergraduate Examinations 2021

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 and answer paper and to upload your answers to FASER.

You should complete the Answer Sheet supplied using a word-processed file format or PDF. If you need to handwrite/draw any or all of your answers, you should take a photo or scan these to be included **WITHIN** one answer sheet. (i.e. ALL YOUR ANSWERS SHOULD BE UPLOADED TO FASER IN ONE DOCUMENT – AN EXTRA HOUR HAS BEEN ADDED TO THE EXAM LENGTH TO ALLOW YOU TIME TO DO THIS AND DEAL WITH ANY TECHNICAL ISSUES).

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.

Question 1

Extreme Programming (XP) addresses particularly well some issues that have traditionally affected software development. Describe how XP addresses each of the following issues:

(a) Staff turnover. [9%]

(b) Business changes. [8%]

(c) Problems accumulate in a system making it regress. [8%]

Question 2

Clean code is an essential element of XP.

(a) Explain why clean code is so important. [9%]

(b) Code that does more than one thing and duplications are signs of code that is not clean. Explain why these two characteristics are obstacles in XP. [16%]

Question 3

You are part of an extreme programming team that has been asked to build the control software for a new type of kitchen appliance which is halfway between a *washing machine* and a *dishwasher*, named a *WishWasher*.

A WishWasher cannot be purchased. It can only be rented on a contract. All washing products (detergent, rinse-aid, softener, salt, etc.) are included in the contract. They are stored inside the machine and their levels are remotely managed by the company who makes the machine. Users need to only provide power and internet connection as well as to connect the machine to mains water and wastepipe.

From the point of view of the end user, the machine is loaded through its conveyor belt. Users simply place items (clothes or utensils, plates, etc.) on the conveyor belt and the machine automatically stores them. When there is a full load of clothes or utensils, plates, etc., WishWasher does not accept further items. To be washed, items are transferred into the machine's washing chamber. After the wash, the machine returns the washed items through the conveyor belt. The machine has an associated app through which customers can operate it or get in touch with the company making it.

You are asked to sketch a release plan for the software that will control the WishWasher. Given the specific nature of the project you can act both as a programmer and as a client for the purpose of planning the release. The hardware for the fundamental functions of the machine will be provided to you.

In your release plan please specify:

(a) A list of the stories identified for the release including at least 10 but no more than 15 stories. If you require further special hardware to implement a story, indicate so in the plan. [15%]

(b) A priority for each story in terms of value for the client (the company making the WishWasher) 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%]

Question 4

The ancient Egyptians wrote numbers in a manner that is quite different from our modern one. In our system, the position of a digit determines its value (for instance in the number 123, the first digit from the right, 3, is the number of units, the second digit, 2, is the number of tens, the third digit, 1, is the number of hundreds). In Egyptian numbers each digit was represented by a particular a hieroglyph (𓏺, 𓎆, etc.) which had a value (𓏺 = 1, 𓎆 =10, etc.) that did not depend on its position. So, the value of number 𓏺𓏺𓎆 was 12, as so was the value of the number 𓎆𓏺𓏺.

You have been asked to continue to develop an Egyptian-number Calculator class, taking over from another programmer. Below you will find two implementations of the class, one in Java and one in Python, together with unit tests for them. For simplicity, in that class the digit hieroglyphs have been replaced by characters: `a = 1, b = 10, c = 100, d = 1000`, etc. So, in this representation the number 12 is `aab` (or any permutation of the digits).

The Calculator class can currently do one main operation, `add`, on two Egyptian numbers. Adding two such numbers is easy as they can simply be concatenated to produce the result. So, adding the Egyptian number `aa` and `ab` (2 and 11, respectively) produces the number `aaab` (13). The result of additions is stored in an accumulator register, `result`, overwriting any value previously stored in `result`.

The Calculator class also has a method, `numberToDecimal`, that can convert Egyptian numbers to decimal. So, for example, `numberToDecimal("abbccc")` returns 321.

Choose one implementation of Calculator and its test class, study them and then answer the following questions.

(a) Write one or two tests which will force you to extend Calculator by introducing a `multiply` method, which returns the product of two Egyptian numbers (represented as an Egyptian number itself). [9%]

(b) Give a sketch of an implementation of the `multiply` method that would pass your tests. [8%]

(c) As we have seen above, in the Egyptian representation, there are multiple ways of representing the same number. Some of the alternatives can be quite confusing as it is difficult to quickly get an idea of how large a number is. For instance, it is difficult to see straight away that the number `aabdbdcdbaaacaa` is just over 3000 in decimal, while this would be much easier if the number was written as `aaaaaaabbbccddd`. How would you go about solving this problem? Also, which tests would you would modify (and how) and/or which new tests would you introduce to ensure that all operations (add and multiply) produce more easily interpretable results? [8%]

Implementation of Calculator in Java

```java
public class Calculator {
    private String result = "";

    String getResult() {
        return result;
    }

    void add(String a, String b) {
        result = a + b;
    }

    protected int numberToDecimal(String number) {
        int decimalValue = 0;
        for (int i = 0; i < number.length(); i++) {
            decimalValue += digitToDecimal(number.charAt(i));
        }
        return decimalValue;
    }

    private int digitToDecimal(char digit)  {
        return (int) Math.pow(10,digit-'a');
    }
}
```

Implementation of CalculatorTest in 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( expected: "",calculator.getResult());
    }

    @Test
    public void calculatorCanConvertEgyptianNumbersToDecimal()  {
        assertEquals( expected: 11,calculator.numberToDecimal("ab"));
        assertEquals( expected: 111,calculator.numberToDecimal("bac"));
    }

    @Test
    public void calculatorCanAddSingleDigitNumbers() {
        calculator.add( a: "a", b: "a"); // 1 + 1 = 2
        assertEquals( expected: "aa",calculator.getResult());
    }

    @Test
    public void calculatorCanAddMultiDigitNumbers() {
        calculator.add( a: "aa", b: "ab"); // 2 + 11 = 13
        assertEquals( expected: "aaab",calculator.getResult());
    }
}
```

Implementation of Calculator in Python

```python
class Calculator:

    def __init__(self):
        self.result = ""

    def get_result(self):
        return self.result

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

    def number_to_decimal(self, number):
        decimal_value = 0
        for i in range(len(number)):
            decimal_value += self.digit_to_decimal(number[i])
        return decimal_value

    @staticmethod
    def digit_to_decimal(digit):
        return 10 ** (ord(digit) - ord('a'))
```

Implementation of CalculatorTest in Python

```python
import unittest
from calculator import Calculator

class CalculatorTest(unittest.TestCase):

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

    def test_calculator_initial_value_of_register_should_be_zero(self):
        self.assertEqual("", self.calculator.get_result())

    def test_calculator_can_convert_Egyptian_numbers_to_decimal(self):
        self.assertEqual(11,self.calculator.number_to_decimal("ab"))
        self.assertEqual(111,self.calculator.number_to_decimal("bac"))

    def test_calculator_can_add_single_digit_numbers(self):
        self.calculator.add("a", "a") # 1 + 1 = 2
        self.assertEqual("aa", self.calculator.get_result())

    def test_calculator_can_add_multi_digit_numbers(self):
        self.calculator.add("aa", "ab") # 2 + 11 = 13
        self.assertEqual("aaab", self.calculator.get_result())
```

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.