**CE320-6-AU**

**UNIVERSITY OF ESSEX**

Undergraduate Examinations 2016

---

**LARGE SCALE SOFTWARE SYSTEMS AND EXTREME PROGRAMMING**

---

Time allowed: **TWO** hours

Candidates must answer **ALL** questions**.**

The paper consists of **FOUR** questions.

The questions are **NOT** 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.

**Please do not leave your seat unless you are given permission by an invigilator.**
**Do not communicate in any way with any other candidate in the examination room.**
**Do not open the question paper until told to do so.**
**All answers must be written in the answer book(s) provided.**
**All rough work must be written in the answer book(s) provided. A line should be drawn through any rough work to indicate to the examiner that it is not part of the work to be marked.**
**At the end of the examination, remain seated until your answer book(s) have been collected and you have been told you may leave.**

## Question 1

With traditional software development strategies (e.g., waterfall) adapting to change (e.g., in user requirements) during the development of a product is typically problematic.

(a) Please indicate the reasons for this. [10%]

(b) Indicate why, for extreme programming (XP), change is not a problem (in fact, it is welcome) and list a few key practices that provide XP teams with the ability to rapidly adapt to change. [10%]

**Question 2**

Beyond being functionally correct, extreme programming (XP) requires code to have additional properties that help with communicating the code's intent and design.

(a) Given this, explain why XP insists that including comments in source code to explain what the code does is bad practice? [10%]

(b) What properties of the code confer to it the ability to communicate its behaviour, design and intents? [15%]

## Question 3

Windows Explorer (which was renamed "File Explorer" in Windows 8) is an application included in Microsoft Windows that provides a graphical user interface for accessing the file systems.

You are part of an extreme programming team that has been asked to build a tool which complements the functionality of Windows Explorer so as to make it easier for users to remember the tasks they were working on in previous sessions and to resume work on any such tasks.

The main features the tool are best explained through a simple example. Imagine that in order to perform a particular task, such as writing a report on the workload of staff in a company, a manager needs to open and work with two Excel files and one Word document while at the same time consulting a particular Web site. The new tool to be developed should notice that those files/sites were opened within a few minutes of one another, that they were then kept open for some time, and that they were finally closed at approximately the same time; it should then take these events as an indication that such files/sites are all required to perform a particular task. When the user launches the tool's GUI, it should display graphically the tasks it has thus identified and should reopen all files and Web pages associated with the task at the click of a button.

You are asked to **sketch a release plan for the tool**. The plan should only include *one release*. The release will include 12 weekly iterations. Your team includes 8 programmers. Given the specific nature of the project you can act both as a programmer and as a customer. In your release plan please do the following:

(a) As a customer, identify and list a set of feature and stories to be implemented in the release. *Make sure you include, but also go beyond, the basic functionality described above.* [15%]

(b) As a customer, indicate the value of each story using a point system where 3 = very valuable, 2 = valuable, 1 = optional. [5%]

(c) As a programmer, estimate the programming effort required by each story based on the following scale: [5%]

E = Easy (takes 0 to 2 working days to the team)
M = Medium (takes 3 to 5 working days to the team)
D = Difficult (2 weeks)
X = eXtremely difficult (3 or more weeks)

(d) Rank the stories in the order in which they should be implemented in the release based on the values and difficulty identified in your answers to parts (b) and (c) above. [5%]

## Question 4

Test driven development (TDD) is a key practice in extreme programming. Your XP team has started developing a text-encryption class, `TextEncryptor`, using TDD.

The class works by taking an input string expressed in a particular alphabet and converting each of its characters into a character from another alphabet, resulting in an incomprehensibly scrambled output text. This task is performed by the method `translate()`.

At the moment the input alphabet includes the 27 characters `"abcdefghijklmnopqrstuvwxyz "` (the last character being a space), while the output alphabet includes the 10 characters `"0123456789"`. So, multiple input-alphabet characters may map to the same output-alphabet character.

At the moment the method that does the mapping, `scramble()`, simply replaces the characters "a" with "0", "b" with "1" and so on until it gets to the end of the output alphabet, at which points is restarts from the beginning. So, it maps "k" (the 11th character in the input alphabet) into "0", etc. and so on. The method does so by using the modulo operator (%) which returns the reminder of the division of its first argument by its second.

At the moment the team has defined two complete tests and produced corresponding production code that passes them. In addition, the team has produced two further tests, which currently fail, in preparation for further developments. These developments include the handling of input strings containing invalid characters and the ability to personalise the `scramble()` method to make the encryption less predictable.

Below you will find **two implementations** of the `TextEncryptor` class just described and of the test class `TextEncryptorTest`: one in Java and one in Python. Choose one implementation, study it, then do the following:

(a) indicate how one could **replace the body of the `testTranslateWithInvalidChar` method** in such a way to test whether the main class handles input strings including characters not in the input alphabet (see above) without errors. Each such character would need to be translated to `"."` (a dot). Then **indicate how the existing production code in `TextEncryptor` would need to be modified** in order to pass this new test without failing any of the previous ones. [10%]

(b) indicate how one could **replace the body of the `testTranslateWithPersonalisedScrambling` method and modify the `TextEncryptor`** to ensure that: (1) users would be able to store an integer key into objects of class `TextEncryptor`, (2) the key would modify the behaviour of the `scramble()` method to make the encryption less predictable. ***Hint**: you are <u>not</u> required to ensure that different values for the key lead to different encryptions, but more simply that there is a reasonable chance of that happening. Also, it this helps, feel free to limit the range of acceptable keys, e.g., from 0 to 9.* [15%]

<u>NOTE: for both part (a) and part (b) exact JUnit or PyUnit syntax is not essential.</u>

```java
public class TextEncryptor {
    private String inputAlphabet;
    private String outputAlphabet;

    TextEncryptor() {
        inputAlphabet = "abcdefghijklmnopqrstuvwxyz ";
        outputAlphabet = "0123456789";
    }

    public String translate(String clearText ) {
        String encryptedText = "";
        for (char c : clearText.toCharArray())
            encryptedText += mapCharacter(c);
        return encryptedText;
    }

    private char mapCharacter( char c ) {
        int indexInInputAlphabet = inputAlphabet.indexOf(c);
        int indexInOutputAlphabet = scramble(indexInInputAlphabet);
        return outputAlphabet.charAt(indexInOutputAlphabet);
    }

    private int scramble(int indexInInputAlphabet) {
        return indexInInputAlphabet % outputAlphabet.length();
    }
}
```

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

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.fail;

public class TextEncryptorTest {

    private TextEncryptor textEncryptor;

    @Before
    public void setUp() throws Exception {
        textEncryptor = new TextEncryptor();
    }

    @Test
    public void testTranslateAlphabetFragment() {
        assertEquals("01234", textEncryptor.translate("abcde"));
    }

    @Test
    public void testTranslateText() {
        assertEquals("7822073465418", textEncryptor.translate("riccardo poli"));
    }

    @Test
    public void testTranslateWithInvalidChar() {
        fail();
    }

    @Test
    public void testTranslateWithPersonalisedScrambling(){
        fail();
    }
}
```

```
1   class TextEncryptor:
2
3       def __init__(self):
4           self.inputAlphabet =  "abcdefghijklmnopqrstuvwxyz "
5           self.outputAlphabet = "0123456789"
6
7       def translate(self, clearText):
8           encryptedText = ""
9           for c in clearText:
10              encryptedText += self.mapCharacter(c)
11          return encryptedText
12
13      def mapCharacter(self, c):
14          indexInInputAlphabet = self.inputAlphabet.find(c)
15          indexInOutputAlphabet = self.shuffle(indexInInputAlphabet)
16          return self.outputAlphabet[indexInOutputAlphabet]
17
18      def shuffle(self, positionInInputAlphabet):
19          return positionInInputAlphabet % len(self.outputAlphabet)
```

```
1  from unittest import TestCase
2  from TextEncryptor import TextEncryptor
3
4  class TestTextEncryptor(TestCase):
5
6      def setUp(self):
7          self.textEncryptor = TextEncryptor()
8
9      def testTranslateAlphabetFragment(self):
10         self.assertEquals("01234", self.textEncryptor.translate("abcde"))
11
12
13     def testTranslateText(self):
14         self.assertEquals("7822073465418", self.textEncryptor.translate("riccardo poli"))
15
16     def testTranslateWithInvalidChar(self):
17         self.fail()
18
19     def testTranslateWithPersonalisedScrambling(self):
20         self.fail()
```

**END OF EXAM PAPER CE320-6-AU**