CE320-6-AU

UNIVERSITY OF ESSEX

Undergraduate Examinations 2018

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

Extreme programming (XP) designs software in a different way from more traditional forms of software development. In particular, the way in which the planning phase is done has a series of advantages (such as flexibility to change) when compared with the more rigid Waterfall methodology.

(a) Explain the meaning of **planning at the last responsible moment** and the role of **multiple planning horizons**. [10%]

(b) Briefly (using one or two sentences each) define the following concepts related to planning: *planning game*, *release planning*, *velocity*, *story*. [10%]

Question 2

Agile techniques, such as extreme programming, have radically changed the way software is developed.

(a) When a project is at risk of being late, there are two traditional actions that are used to try to get it back on track: hiring more programmers and asking existing programmers to work for longer hours. List the drawbacks of each of these practices and explain how the problem of schedule slippage is addressed in extreme programming. [10%]

(b) Compare and contrast the traditional notion of success for a software project with the notion of success that extreme programming embraces. [10%]

Question 3

You are part of an extreme programming team that has to develop a new blogging software.

You are asked to sketch a release plan for the blogging software. The project will include two releases, each composed of 2 one-week iterations.

The list of features identified for the project (together with their estimates, measured in story points) is as follows:

[Feature ID]	Feature	[Estimate]
[F1]	As a user I should be able to post	[20 story points]
[F2]	A user should be able to comment on someone else's posts	[10 story points]
[F3]	Only registered users should be allowed to post and/or comment	[20 story points]
[F4]	Editing a post should be through an interface that provides a WYSIWYG experience	[45 story points]
[F5]	As a user I should be able to include pictures and movies in posts	[25 story points]
[F6]	It should be possible to tag posts	[10 story points]
[F7]	It should be possible to search by content and tags	[30 story points]
[F8]	Posts should be dated and shown in reverse chronological order	[10 story points]
[F9]	A person should be able to get notifications when someone comments on his/her post	[20 story points]
[F10]	It should be possible to generate/receive RSS feeds	[45 story points]

(a) For each of the features identified above, specify a priority in terms of value for the customers using the following point system: 3 = Very valuable; 2 = Valuable; 1 = Optional. [5%]

(b) Based on the values assigned in (a) and the given estimates in story points, specify the order in which the features should be implemented. [5%]

Question 3 continues…

Question 3 (continued)

(c) The table below lists a set of features that were developed by your team in a one-week iteration of your last project. Calculate the velocity of your team using this information. [6%]

Feature ID	Status at end of iteration	Estimate (# story points)	Time invested (# story points)
G1	Done done	28	42
G2	Coded	27	30
G3	Coded, tested, refactored and integrated	13	13
G4	Done done	55	50
G5	Coded, tested and refactored	30	33
G6	Done done	10	10
G7	Done done	2	3
G8	Done done	7	9

(d) Using a **rigorous** approach and assuming your project entails no specific risks, calculate the number of story points you commit to achieve in the next release. Note: for the rigorous approach, the risk factors are: 1 for 10%; 1.4 for 50%; 1.8 for 90%. [7%]

(e) Taking into account the calculations from (d) and the prioritised list of features from (b), list the features that you would commit to for the next release using the **rigorous** approach. [7%]

Question 4

You have joined GreatSoftware Ltd as a lead developer. Your boss has asked you to have a look at a piece of code written by a previous employee. The code gets a string in input, splits it into substrings at the spaces, and returns an array with ten integers representing the number of occurrences of single-digit numbers (0-9) among such substrings. For example, the string "I want 3 slices of pizza with 4 toppings for 3 pounds" is first split into the sequence of substrings "I", "want", "3", "slices", "of", "pizza", "with", "4", "toppings", "for", "3", "pounds". Then, since this contains twice the string "3", once the string "4", and zero instances of all other digits, the code returns [0 0 0 2 1 0 0 0 0 0].

Below you will find an implementation in Java and one in Python. Study an implementation and do the following:

(a) Write the following JUnit-style unit tests:

(i) Test that when passing an empty string, the method `run` returns the array [0 0 0 0 0 0 0 0 0 0], representing 0 occurrences of each digit from 0 to 9. The current implementation should pass this test. [5%]

(ii) Test that when invoking `run` twice, with two different strings, the second time it returns the cumulative counts of single-digit numbers across the two strings, eg, after executing `run("4 me")`, calling `run("1 to 1")` would return [0 2 0 0 1 0 0 0 0 0] (ie, two instances of "1", one instance of "4" and zero instances of all other digits). The current implementation should fail this test. [5%]

(b) Explain how the current implementation should be modified to pass test (ii) above. [5%]

(c) Indicate in which ways the current implementation does not satisfy extreme programming requirements in relation to clean code. Focus on names, functions, comments, formatting, and classes. Use line numbers to point out violations. [15%]

Code for Question 4

Java code:

```java
import java.util.Arrays;

public class StringSplitter {
    private Integer[] occurrences = new Integer[10];
    private String[] digits_list = new String[]{"0","1","2","3",
            "4","5","6","7","8",
            "9"};
    // One of the most important methods in this class
    public Integer[] run(String s) {
        for (int d_index=0; d_index < digits_list.length; d_index++)

            occurrences[d_index]=0;
        String[] words = s.split(" "); // Split string into words
        for (String word: words) {
            int i = -1;
            for (int j = 0; j < digits_list.length; j++)
                if (word.equals(digits_list[j])) {
                    i = j;
                    break;
                }
            if (i >= 0 )
                occurrences[i]     +=      1;
        }
        return occurrences;
    }
    public static void main(String[] args) {...}
}
```

Python code:

```
class StringSplitter:
    occurrences = [0] * 10
    digits = ["0","1","2","3",
              "4","5","6","7","8",
              "9"]
    # One of the most important methods in this class
    def run(self,s):
        for d_index in range(len(self.digits)):

            self.occurrences[d_index] = 0
        words = s.split() # Split string into words
        for word in words:
            i = -1
            for j in range(len(self.digits)):
                if word == self.digits[j]:
                    i = j
                    break
            if (i >= 0 ):
                self.occurrences[i]       +=       1
        return self.occurrences

```

END OF PAPER CE320-6-AU