CE320-6-AU

UNIVERSITY OF ESSEX

Undergraduate Examinations 2015

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.

Candidates must answer ALL questions

Question 1

Two practices are traditionally used to try to get a late software project back on track: hiring more programmers and asking existing programmers to work for longer hours.

a) List the drawbacks of these two practices. [10%]

b) Indicate how the problem of schedule slips is addressed in extreme programming. [10%]

Question 2

The **ten-minute build** is one of the key practices in XP.

a) Explain why automating the build of a project and keeping the build under 10 minutes is important to XP. [10%]

b) List the key functions of the build-automation tools discussed in the module (e.g., make, ant, maven, etc.) [10%]

Question 3

It is more and more the case that programmers, particularly novices (but not only), make use of the search facilities of Google to identify suitable examples of API and code snippets that implement functionalities they need, instead of implementing them from first principles. Once one has found code that demonstrates or implements the functionality required, development is greatly accelerated as the programmer only needs to adapt the code to his/her own project.

You are part of an extreme programming team that has been asked to build a plug-in for an existing IDE which will automatise as much as possible the "programming-by-Google" process described above.

You are asked to **sketch a release plan for the plug-in**. 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 specify:

(a) A list the features identified for the release. [15%]

(b) The value of each feature for the customers using on a point system where 3 = very valuable, 2 = valuable, 1 = optional. [5%]

(c) An estimate of the programming effort required by each feature 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) The order in which the features should be implemented in the release based on the values and efforts identified in (b) and (c). [5%]

Question 4

Test-driven development (TDD) is a key element of extreme programming. In TDD unit tests must be written before the corresponding code is written.

You are asked to use TDD to develop a `StringJustifier` class that can wrap around and justify the text of a string into a series of lines (strings) which are all of exactly a given length.

The class will be used by simply creating an object of type `StringJustifier` which remembers the length of the lines it will produce and then invoking its `justify` method to perform the justification. More specifically, the `justity` method gets a string as an input and produces an `ArrayList` of strings as an output, where each element of the array is a line of text. For instance,

```
StringJustifier sj = new StringJustifier(20); // 20=line length
sj.wrap("As I said before, the best part of the course...");
```

would output an `ArrayList` containing the following strings:

```
"As  I  said  before,"
"the best part of the"
"course...           "
```

Note that words from the original string cannot be split across the lines and that, where required, the method adds extra spaces between words so as to fill up the available characters (20 in this example) in each line.

Imagine you are using TDD. Write 6 JUnit-style unit tests which would ensure the production code for `StringJustifier` would work correctly based on the above description. **Order or** [30%]

number the tests in the order in which you would create them when doing TDD. These must test a variety of limit conditions in addition to the normal behaviour of the class.

NOTES: (a) Exact JUnit syntax is not essential, (b) you are not asked to write any production code to satisfy the tests: only the tests.