SEOC2 Coursework 1

There are two equally weighted parts to this coursework.

Part 1's aims are:

- to give you a chance to get to grips, albeit in a toy example, with a genuine issue in object oriented design; for example to understand how patterns contribute to design discussions;
- to give you hands-on experience of designing a sequence of refactorings;
- to help you learn one important principle of OO design, the Dependency Inversion Principle, including its applicability to the Layers architectural pattern.

Part 2's aims are:

- to deepen your understanding of two important development methodologies that can be used with UML;
- to let you start to explore the issues that arise in fitting such a methodology into an existing organisational context;
- to give you practice in expressing your findings in this area succinctly and clearly.

Deadline and late submission policy

The deadline for this coursework is **9am Monday 14 February**. Submit Part 1 on paper to the mail tray outside my office; submit Part 2 by email to `Perdita.Stevens@dcs.ed.ac.uk`.

Late submissions will get 0 unless I have agreed with you before the deadline that you may have an extension. Even then, there will be an automatic penalty of 10% per day or part of day late, unless I have agreed that you have an *exceptionally* good reason for needing to submit late.

Part 1

Read the distributed paper "The Dependency Inversion Principle" by Robert Martin. (It and many other papers by the same author are available from www.objectmentor.com.)

Suppose that you have a system which uses the "naive approach" described in Figure 5 and Listing 5, and that you wish to transform it into the form described in Figure 7 (so that, for example, (a) it would be possible to replace the Lamp class by another from a library; (b) it would be possible to reuse the Button class in another application).

Consider what sequence of (at least 3) refactorings would enable you to make this transformation whilst finding any coding errors at as early a stage as possible. (You should view this toy example as standing for a similar situation with more complicated classes. You may assume that a good automatic test framework is available so that new tests can be written with minimal effort and the whole test suite can be run and checked with a single command, so that the overhead of testing is low, but that the system needs to compile in order for the tests to be runnable.) Each refactoring should, of course, end with a fully functional system.

Your answer should include modern UML diagrams of the system at the beginning and after each of your proposed refactorings. Your diagrams may, but need not, be produced using Argo/UML.

You should make clear:

- what code has been altered, added or deleted at each stage (but you need not submit any code)
- at what stage(s) all the tests should be rerun;
- what tests should be added at each stage;
- why you have chosen the refactorings you have.

One-third of the marks for Part 1 are available for each of: correct use of UML; sensible choice of refactorings; tests and justification.

Part 2

You are a software developer in an organisation which has recently adopted object orientation and the use of UML as a design language. In the past the company's main development language has been C, and the "official" development process has been a version of the waterfall model. In practice, it is well understood that in order to get products out you have had to iterate the stages of the waterfall. There has been some use of prototyping.

Your manager (who is clever and has a background in software development, but is not knowledgeable about any modern software development process) wants a briefing which will help her to decide whether the company should adopt the Unified Process or Extreme Programming, or neither, and about how any change should be made. In her request she explained "The two most important risks we run in any project are (a) that we fail to understand what the customer needs, or be distracted from building it; (b) that we find ourselves part-way through the project technically unable to build it." She does not need to be convinced about the benefits of having some iteration in the process, because she has observed iteration being needed in practice; but she has no pre-defined ideas about what kind of iteration might be best or what other factors might matter.

From the point of view of these two risks and any others you consider relevant, compare and contrast the Unified Process with Extreme Programming, making clear how each process will help to manage each of the risks. Briefly recommend a course of action for how the company should proceed: concretely, what should your manager do?

Your answer should take the form of a mail message to your manager; because your manager is busy, this should not be longer than 500 words, and if you can say what you need to in 250, so much the better (you are unlikely to be able to say enough to obtain full marks in less than 250). You may optionally precede this by a short description of any further assumptions you have made about the company, its business or anything else relevant; alternatively you may be able and prefer to make clear any assumptions in the email itself ("Because our business is mostly...").

Your answer will be marked on the basis of:

- your demonstrated understanding of the facts about UP and XP;
- your comparisons and contrasts;
- your analysis of how these relate to the two kinds of risk;

- your application of what you understand to the fictional business situation, including any further assumptions you introduce, and your suggestions about how to proceed
- the coherence and readability of your explanation.

each factor being roughly equal in weight and giving a total mark out of 50.

A pass mark could be obtained simply on the basis of what has been said in lectures plus the lecture notes, but for higher marks you will need to learn more about UP and/or XP. There are some links on the course home page; you may also find newsgroups useful.

Collaboration

For Part 1, I encourage you to discuss anything that is not clear about the Martin paper. You should not discuss your choice of refactorings, however.

For Part 2 I encourage you to discuss the task as much as you like. What you *write* must be your own unaided work and you should not show your submission to anyone else.