PRIFYSGOL ABERYSTWYTH - ABERYSTWYTH UNIVERSITY

DEGREE EXAMINATIONS 2011-2012 SEMESTER 1

FACULTY OF SCIENCE

Computer Science, CS31310: Agile Methodologies

Time allowed: 2 hours

Calculators are not allowed in this examination.

Answer **THREE** *from* **FIVE** *questions.*

All questions carry equal marks.

1. This question is about the Agile Manifesto and the Agile Practices.

   a) The Agile Manifesto, published in 2001, is stated below.
   We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

   - **Individuals and interactions** over processes and tools.
   - **Working software** over comprehensive documentation.
   - **Customer collaboration** over contract negotiation.
   - **Responding to change** over following a plan.

   That is, while there is value in the items on the right, we value the items on the left more. (Source: agilemanifesto.org)
   For each of these values, discuss why the words in **bold print** are valued more than the words in light print.

   [16 marks]

   b) A colleague states, *"If you can write down all of the top-level requirements at the start of the project, you should use a plan-based methodology for the project."* Discuss any strengths or weaknesses in this statement and justify your answer.

   [12]

   c) One of the principles from the *Principles behind the Agile Manifesto* states that *Agile processes promote sustainable development.* Briefly explain the purpose of this principle.

   [5]

2. In the book, *Extreme Programming Refactored: The Case Against XP, Stephens and Rosenberg, Apress, 2003*, the authors liken the XP practices to a *"ring of poisonous snakes daisy chained together."* They claim that if one 'snake' (practice) is detached then they all unravel and the methodology fails.
   Discuss the strengths and weaknesses of the relationships between different XP practices. In your answer, explain whether or not you support this view.

   [33]

3. This question is about antipatterns.

   a) Briefly define the terms: software design pattern, antipattern.

   [6]

   b) Suppose you are a new employee in an established software company. You are asked to add a new feature to a large Java system. When you come to inspect the code, you observe that one large class handles most of the processing. Most of the other classes just contain data, but some consist of a single method named after the class.

      (i) Identify and briefly describe the antipatterns that lead to these symptoms. Give reasons for your diagnosis.

      [11]

      (ii) Outline a strategy for improving the code.

      [11]

      (iii) What steps should you take before embarking on full scale code improvement?

      [5]

4. This question concerns Feature-Driven Development and Extreme Programming.

   Ace Software Ltd. is a reputable company that has had reasonable success in several large software projects. However, although all contracts have been signed off, clients have not been totally satisfied with the finished products. Moreover some of the larger projects had to be renegotiated, as they took longer to complete and cost more than anticipated.

   Hearing that agile development provides a way to address these problems, the company is considering adopting either extreme programming (XP) or feature-driven development (FDD). However, they are concerned that neither of these methods will scale to large projects involving 50 or more developers.

   Acting in the role of consultant, you are asked to:

   a) present a strategy for using extreme programming in large projects;

   [11]

   b) present a strategy for using feature driven development in large projects;

   [11]

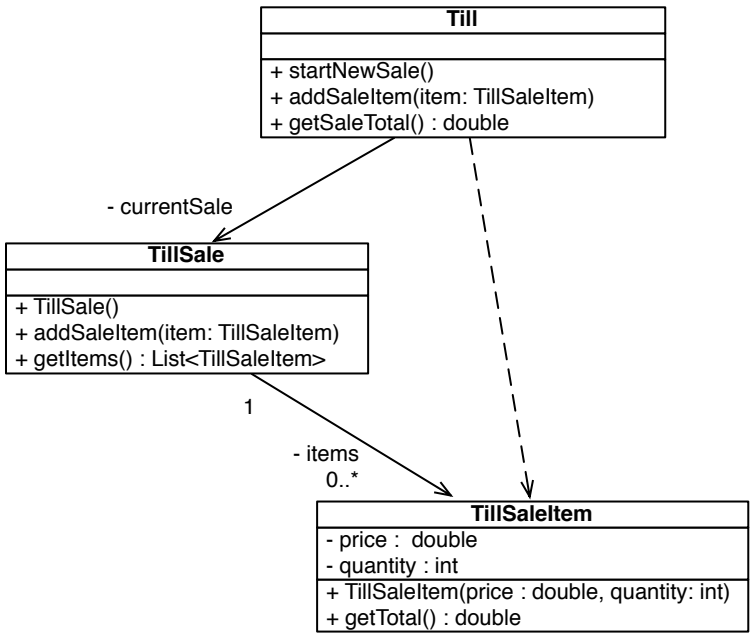
   c) discuss the challenges and merits of adapting to each approach and make recommendations to the company as to how to proceed.

   [11]

5. This question concerns Test Driven Development.

   You are working for a company that is developing a new electronic till system, using the Extreme Programming (XP) methodology. You have finished the daily stand up meeting and paired up with a colleague.

   The following UML diagram summarises the current state of the design. The diagram shows the main classes. In addition to these classes, there are three unit test classes called TestTill, TestTillSale and TestTillSaleItem, which test the functionality of the classes in the diagram.



   You have been told some background information about the design:

   - The Till manages the main actions of the till, including starting a new sale, adding items into the sale and getting the total for the current sale.
   - The TillSale holds details about the current sale for a single customer. It manages a list of TillSaleItem objects.
   - The TillSaleItem holds details about an item, including the price and quantity. The total for the item is defined as *price * quantity*.

   The code for the Till class is provided at the end of the question.

   Your pair has been asked to develop the following tasks:

   - Task 1: The system currently stores the *price* of a product only. The system should store the *price* and *description* for each *Product*.
   - Task 2: The Till should be able to store a list of each sale (TillSale) and report total sales for the day.

Describe *each step* that you would complete and the associated changes that you would make as you work on this task using *Test Driven Development*. In your answer, remember to consider issues such as tests and refactoring. You do not need to write out any code, but in your steps you should clearly indicate the purpose of any code that you would add/change and why you would make the change. State any assumptions that you make.

[33]

```
public class Till
{
   private TillSale currentSale;

   public void startNewSale() {
      currentSale = new TillSale();
   }

   public void addSaleItem(TillSaleItem item) {
      if(currentSale == null) {
         startNewSale();
      }

      currentSale.addSaleItem(item);
   }

   public double getSaleTotal() {
      double total = 0.0;

      for(TillSaleItem item : currentSale.getItems()) {
         total += item.getTotal();
      }

      return total;
   }

}
```