UNIVERSITY OF LONDON

IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 1999

BEng Honours Degree in Computing Part III
BEng Honours Degree in Information Systems Engineering Part III
MEng Honours Degree in Information Systems Engineering Part III
BSc Honours Degree in Mathematics and Computer Science Part III
MSci Honours Degree in Mathematics and Computer Science Part III
for Internal Students of the Imperial College of Science, Technology and Medicine

This paper is also taken for the relevant examinations for the Associateship of the City and Guilds of London Institute
Associateship of the Royal College of Science

PAPER 3.02 / I 3.8

SOFTWARE ENGINEERING – METHODS

Friday, April 30th 1999, 10.00 – 12.00

Answer THREE questions

For admin. only:
paper contains 4 questions

1a Define what is meant by the *state pattern* and explain what its benefits are.

b Explain the purpose of the *abstract factory pattern*, and the roles of classes $AbstractFactory$ and $AbstractProduct$ in this pattern.

c Give the class diagram of a system which meets the following requirements, using a suitable pattern:

> A banking system deals with two types of accounts: normal accounts and overdrawn accounts. Each type of account has a balance *bal* which is an integer, and operations $withdraw(x : nat)$ and $deposit(x : nat)$ where *nat* is the type of natural numbers.
> Normal accounts have $bal \geq 0$. For normal accounts depositing adds x to *bal* and withdrawing subtracts x. Overdrawn accounts have $bal < 0$, and withdrawing does not change the balance (i.e., it does not succeed).
> Depositing x increases *bal* by $x - 5$.
> The account of a given client starts with balance 0.

d Write pseudocode for the main classes. Assume that object creation of accounts can be performed by a call $new!C(val)$ where *val* is the initial value of the *bal* attribute of the new object, and C is the account class.

The four parts of this question carry 20%, 20%, 30% and 30% of the marks, respectively.

2a What is a *lifecycle* or *process model* for software development?

b Describe the characteristics of three (3) lifecycle models and, by comparing and contrasting them, place them in a historical context.

c Define the various forms of maintenence encountered in a typical software lifecycle.

d Describe the objectives and characteristics of the *Capability Maturity Model.*

(The three parts carry, respectively, 10%, 35%, 15%, and 40% of the marks.)

Turn over ...

3 Consider a simple language with: assignments (A), binary conditionals (C), loops (L), and sequencing (S). For this language, a measure, M, for "Number of predicate nodes" can be defined hierarchically as follows:

$$\begin{aligned} M(A) &= 0 \\ M(C(p1,p2)) &= M(p1)+M(p2)+1 \\ M(L(p1)) &= M(p1)+1 \\ M(S(p1,p2)) &= M(p1)+M(p2) \end{aligned}$$

where $p1$ and $p2$ are sections of the program.

a Give the structure as a tree of the following program fragment and draw the control flowgraph for it. Use the above definition to calculate (showing your working) the number of predicate nodes in the code.

```
while(x > 1) do
  while(y > 1) do
    if (x+y mod 2 == 0)
      then
        x := x-1
      else
        y := y-1
    endif
  enddo
enddo
```

b The "McCabe Structural testing" strategy requires the execution of a set of linearly independant paths through a piece of code. Define a measure for the number of tests required by this strategy and calculate its value (again showing your working) for the above program fragment.

c Describe the relationship between the number of predicate nodes measure and the number of linearly independant paths. Explain why this relationship arises.

d For the program in part a, give a test suite which gives 100% coverage for the McCabe strategy. (Consider pairs (x, y) as input and also (x, y) as output.)

e Give an infeasible path for this code and show how it can be viewed as a linear combination of the paths taken in executing the tests in this suite.

(The five parts of the question each carry 20% of the marks).

4a What distinguishes *adaptive* techniques for software testing from non-adaptive techniques? On what basis is it claimed that adaptive techniques are more readily automatable than non-adaptive techniques?

Explain the term *relative infeasibility* as it is applied to paths in adaptive testing techniques. What can be done to continue finding test cases when such paths arise?

b Use the path prefix approach to find a test suite for 100% branch coverage of the following program fragment. (Consider the tuple (x,y,z) as the input and (a,b) as the output. Start with the input (0,0,0).)

```
if (x = 0)
    then
        if (y = 0)
            then a := 1
            else a := 2
    else
        if (y = 0)
            then a := 3
            else a := 4
;
if (z = 0)
    then b := 1
    else b := 2
;
return
```

In general, what is the largest number of tests which can be required to achieve 100% branch coverage with this strategy? Why is it often the case that coverage can be achieved with fewer than this many tests?

c Explain the terms *on point*, *off point* and *extreme point* as used in boundary value analysis. Give an example of each for the region ($x \geq 0$ *and* $y > 0$ *and* $x + y < 1$) for real valued x and y.

(The three parts carry, respectively, 30%, 50%, and 20% of the marks).

End of Paper