UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 1997

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 / I3.8

SOFTWARE ENGINEERING - METHODS

Friday, April 25th 1997, 10.00 - 12.00

Answer THREE questions

For admin. only: paper contains 5 questions

Department of Computing: 1996-7 Session

Paper 3.02=I3.8

QUESTIONS

Question 1

1 a Describe in a few sentences the role of a *process model* in software development.

b With the aid of diagrams, outline the main characteristics of the following three process models:

i) V-model

ii) incremental model

iii) evolutionary model

c Briefly describe the kind of project for which each of the three models in part b would be most appropriate.

d Give one example of how a basic process model might be *supplemented* to meet the needs of a specific project, and one example of how two basic models might be *combined* to meet the needs of a specific project.

The four parts carry, respectively, 20%, 45%, 20% and 15% of the marks.

Turn over...

Question 2

2a i) Consider the passage below, extracted from one of the standard texts on software engineering:

A quality metric is a number that represents one facet of the quality factors identified by researchers such as McCall. In order to explain what this means, an example would be in order. Consider the task of programming and testing a module in a third-generation programming language. There are a number of countable features of such a module which can contribute to quality factors. It must be stressed that these are examples and would need thorough experimentation and analysis before they are used on a software project.

- First, there is the expression of the functionality of the software. This is normally contained within a module as a natural language comment. The longer the comment, in terms of words, the greater the chance that the module is being asked to do too many functions, thus going against the coupling and cohesion principles. The total number of words can be regarded as an indication of one aspect of the maintainability of a module.

- The number of branches in a module can be regarded as a metric that reflects one aspect of testability: that concerned with structural testing. The higher the number of branches, the more test data and testing are required.

- Another aspect of a module that can be used as a metric is the degree to which the module calls other modules. When a module is debugged, the programmer who carries out this process must remember a large amount of information, for example what the pattern of access is from called modules to a global variable that is read to, and written from, by the module being debugged. The number of calls to subordinate modules might, hence, be regarded as a metric that reflects some aspect of maintainability.

- Another numeric aspect of a module is the extent to which a module makes access to global data - either global variables declared in program code or external files. The larger the access to such entities, the lower the maintainability of a system. For example, if a global item of data is read from, or written to, by a large number of modules, then a change to this data occasioned, say, by a maintenance change would result in a maintenance programmer having to examine and possibly change all the modules that accessed the global data. A count of the number of modules that access common data is a good reflection of one facet of maintainability.

Turn over...

- Another numerical property of a module is the pattern of variable access in a module. A module might declare a number of local variables. When the module is debugged, the programmer carrying out this task will need to carry a large amount of information about the current values of variables in his or her head when debugging. The number of variables declared in a module would be reflection of this aspect of testability.

- Another aspect of the quality of a module is the number of non-standard features it contains. This is a reflection of one facet of the portability quality factor.

Comment critically on the above from the point of view of measurement theory. (Note that 'quality facets/factors' are to be understood in the context of quality factors, criteria and metrics as conventionally used in standards.)

Turn over ...

Question 3

3 a Define and describe the *definition-use* (DU) dataflow testing strategy.

b Consider the following program:

```
B1;
do while C1
      if C2
            then
                  if C4
                        then B4;
                        else B5;
                  endif;
            else
                  if C3
                        then B2;
                        else B3;
                  endif;
      endif;
enddo;
B6:
```

Give the corresponding control flowgraph representation of this program, indicating the correspondence of the nodes to statements.

c Assume that variable X is defined in the last statement of block B1, B2, B3, B4, and B5, and is used in the first statement of blocks B2, B3, B4, B5, and B6. The DU testing strategy requires an execution of the shortest path from each Bi, $0 < i \leq 5$, to each of Bj, $1 < j \leq 6$. (Such testing also covers any use of variable X in conditions C1, C2, C3, and C4.)

How many DU paths are there for X and what are they?

d How many execution paths are actually required to cover all DU paths and what are they?

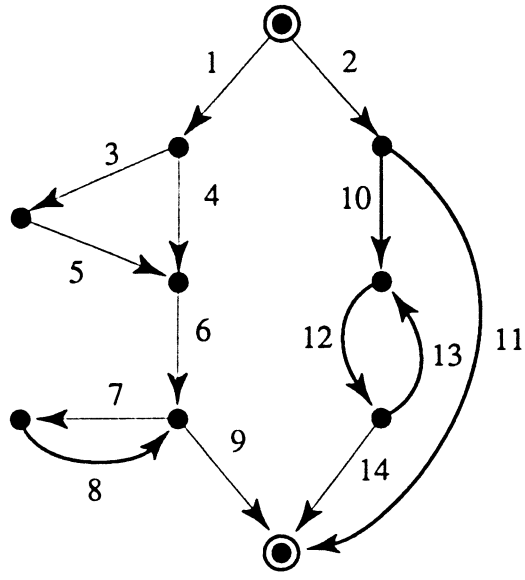
The four parts carry, respectively, 20%, 20%, 35%, 25% of the marks.

Turn over...

Question 4

4a Define each of the following test strategies: all paths, visit-each-loop, simple paths, branch coverage, statement coverage.

b Consider the control flowgraph below! For each of the strategies above, give the minimum number of paths required for that strategy and exhibit them. (For simplicity, in presenting your answers, the edges are labelled rather than the nodes.)



Turn over...

Question 5

5a Give and explain the *representation condition* as used in Measurement Theory.

b Suppose we are trying to solve a problem involving a fragment of the world (referent) which includes 3 people: Frankie (a centre forward for a basketball team), Wonderman (the Head of the Department of Computing) and Peter (a five year old). The problem requires us to use the concept of height and we choose a numeric measurement scale, the non-negative integers, say. Let M be the map from the referent to the measurement scale. Which of the following assignments satisfy the representation condition? Give reasons for your answer.

i. M(Wonderman) = 100; M(Frankie) = 90; M(Peter) = 60
ii. M(Wonderman) = 100; M(Frankie) = 120; M(Peter) = 60
iii.. M(Wonderman) = 100; M(Frankie) = 120; M(Peter) = 50
iv. M(Wonderman) = 68; M(Frankie) = 75; M(Peter) = 40

c Suppose that we could classify every software failure as either a) syntactic, b) semantic, or c) system crash. Suppose additionally that we agree that every system crash failure is more critical than every semantic failure, which in turn is more critical than every syntactic failure. Use this information to define two different measures of the attribute of criticality of software failures. How are these measures related? What is the scale of each?

d Formally, what do we mean when we say that a statement about measurement is meaningful? Discuss the meaningfulness of the following statements:

- "The average size of a Windows application program is about four times that of a similar DOS program."

- "Of the two Ada program analysis tools recommended in the Ada coding standard, tool A achieved a higher average usability rating than tool B." For this example, program usability was rated on a four-point scale:

  4: can be used by a non-programmer
  3: requires some knowledge of Ada
  2: useable only by someone with at least five years' Ada programming experience
  1: totally unusable

The four parts carry, respectively, 25% , 20%, 25%, 30% of the marks.

End of paper