THE BCS PROFESSIONAL EXAMINATION
Diploma

April 2002

EXAMINERS' REPORT

Software Development Environments

General Comments:

- It is necessary for the examiner to be able to read your answer. Try to write legibly. In many cases writing less, but more clearly, would gain more marks.

- The poorer answers were those which contained stock ('text-book') phrases. The better answers were those which provided a thoughtful answer from experience.

- Advice to candidates: Answer the question! (see detailed comments on individual questions).

Question 1

This question examines section 1 of the syllabus, "The Software Development Lifecycle"

(a) Name a software development method of your choice and identify its stages.

(b) Pick THREE stages from your method and, for each stage, name an appropriate language or notation and explain what contribution the stage makes to the overall software development.

(c) Suppose you were a programmer given the task of making a significant change to a piece of software. You have not yet seen the documentation for the software. List FOUR kinds of information that you would hope to find in the documentation and say why you would need them and how you would use them.

Answer Pointers

(a) Name:
e.g (from syllabus) Waterfall:

Stages:
Requirements, Prototyping, Specification, Design, Coding, Implementation & Testing, Documentation, Maintenance

The question asked "Name ... and identify its stages", so missing out the name lost some candidates a mark

(b) (there will be much variation here)
Three stages: name + contribution
- Specification: Z; link required output to specific input
- Design: Pseudo-code; formulate algorithm
- Coding: C++; capture design in code

The question said "pick 3 stages..." so writing about a 4th or 5th is a waste of time.

Again for each of the 3 stages, the questions says "name an appropriate language and ...". The name of a language was omitted in nearly half the answers leaving a maximum of 6 marks out of 9 for this part.

(c) Look for 4 kinds of information, why, how use:

- Original specification (to see what exactly has changed)
- Design (to see how much of program will be affected
- Subroutine hierarchy, who calls what (to see which subroutines need changing)
- Description of each procedure/function (to see how to change individual routines)

Full marks could not be obtained by simply writing a description of types of documentation - the question asks why & how it is USED.
The question asks for FOUR kinds of information so there should not have been any answers with FIVE kinds.

Mark Breakdown

(a) 1 mark for name, 1 marks for each of 3 stages, making 4 marks in total
(b) 3 marks per stage (1 mark for language/notation & 2 for explanation) making 9 marks in total
(c) 3 marks per kind of information (1 mark each for kind, why needed, how used) making 9 marks in total

Question 2

This question examines section 2 of the syllabus,, " The Programming Environment"

(a) Describe FIVE principle features common to modern operating systems.

(b) Name TWO operating systems that you have met and describe TWO differences between them that are not concerned with the user interface.

(c) The two extremes of operating system interfaces are command-line interface and WIMP (Window, Icon, Menu, Pointer). Briefly describe how each interface is used and give ONE advantage for each type.

Answer Pointers

(a) 5 features:

- Handle hierarchical file store
- Handle peripherals
- Run user programs
- Schedule processes
- Memory allocation (virtual?)

The question said "describe FIVE features" – not SIX or more.

The question explains that not all operating systems have a graphical user interface, so it was not acceptable to quote a GUI as a feature that operating systems have in common.

(b) Windows 2000, Linux
Differences: (accept anything that is accurate, but not re interface)
e..g. Store end of line differently in text files
e.g. Can chain several programs together using pipes in Linux

The question asked for differences that were NOT based on the user interface. Many gave answers which ARE based on the user interface.

(c) Command Line: textual version of a command with arguments is typed at a prompt
WIMP: Data file(s) dragged over icon of program using mouse pointer

Note that "Describe how it is used" is a different question to "Describe what it is".

Command Line: good for batch processing, system work, ability to write scripts
WIMP: good for novice user (does not have to learn command names, flags, argument patterns, etc

The question asked for one ADVANTAGE - not a DISadvantage.

(a) 2 marks per feature (1 for name, 1 for description) making 10 marks in total
(b) 1 mark for naming two operating systems and 2 marks for each difference, making 5 marks in total
(c) 3 marks for how each type is used and 2 marks for an advantage of each, making 10 marks in total

Question 3

This question examines section 3 of the syllabus “Program Design”

(a) When a program is executing, data can be read from a file and results can be written to a file. Describe and compare the following terms relating to files:

(i) Text file / binary file
(ii) Sequential / direct access
(iii) Local / remote
(iv) Open for reading / writing / appending

(b) In most programming languages there is a file open and a file close command. Describe the purpose of these two commands and include information on the parameters which are used.

Answer Pointers

(a) (i) **Text** file is a sequence of bytes representing ASCII characters

Binary file is also a sequence of bytes but the interpretation that is put upon them is not fixed. Groups of bytes might represent integer or real numbers or ASCII strings depending on the application

Some candidates wrote as though they were answering a question about how source code is converted into corresponding executable code by a compiler

(ii) **Sequential access** is where the reading position moves forward only through the file allowing the bytes to be read in order, once only.
Direct access allows the reading pointer to be placed at any point in the file at any time

Some candidates wrote as though the only files that exist are databases.

An explanation was required. It is not enough to say that a direct file uses direct access.

(iii) A **local file** is in the file system of the computer running the program.
A **remote file** is accessed across a network of some kind

(iv) **Open for reading** means the file cannot be changed.
Open for writing means arbitrary change.
Open for appending means only allowed to add bytes at the end.

Candidates often muddled writing and appending as though they were the same.

(b) **File open** usually has 3 parameters

- The name of the file to be opened
- The name of a program variable which will be used to reference the file
- The style of opening (r,w,a)

The open routine has to find the file, create a buffer, signal an error if file missing, mark the file as 'in use', etc.

Some candidates thought that opening a file meant loading all its contents into memory.

File close usually has one parameter. The reference to the file to be closed
For a file being written or appended the close has to empty any remaining data from the buffer to the file. The file should now be marked free to be used by other programs.

Some candidates described the open/close dialog typical of their favourite operating system instead but the question is centered around the file open/close command as written in a programming language.

(a) (i) 2 marks for each term, totalling 4
(ii) 2 marks for each term, totalling 4
(iii) 2 marks for each term, totalling 4
(iv) 2 marks for each term, totalling 6
totalling 18 marks for the part

(b) file open: 3 marks for parameters, 2 marks for purpose
file close: 1 mark for parameter, 1 mark for purpose
making 7 marks in total for the part

Question 4

This question examines Section 4 of the syllabus, "Program Development Environments."

(a) Program editors within programming development environments often provide features that aim to minimise the introduction of errors into a program. Describe three such features and explain how each reduces the possibility of errors.

(9 Marks)

(b) Describe five features of a debugging tool that enable it to assist the process of identifying the source of errors in a program.

(10 Marks)

(c) In order for a debugging tool to operate successfully, additional code has to be inserted by the compiler. Whether or not this additional code is included is determined by setting the appropriate compiler option. Identify and explain the purpose of three other compiler options that a program development environment may offer.

(6 Marks)

Answer Pointers

(a) Possible examples include:

Syntax directed editing

- Automatic completion of language structures, e.g. the insertion of an END when a BEGIN is entered.
- Reduces "spelling" mistakes in keywords and ensures the correct syntax is used

Colour coding

- The reserved words of a language are displayed in a different colour to, for example, comments.
- This enables errors such as a missing comment token or a mis-spelt keyword to be easily identified.

Automated indentation

- Language constructs are indented according to predefined rules.
- This ensures the logical structure of the code is clearly indicated.

Note that the examiners were expecting features specifically connected with programming environments, not features commonly found in many general purpose text editors, such as cut and paste facilities.

(b) Examples include:

Break points

User defined locations at which the debugger will pause execution of the code until further instructions are given.

Watches/Inspectors

Display the contents of a variable or a more complex expression when the program is paused at a break point.

Changing variable contents
Allows the content of a variable to be modified during the execution of the code.

Stepping into/through code
Allows the code to be executed statement by statement. Statements that represent calls to other functions and/or procedures can either be considered as a single statement (step through) or as a group of separate statements that get executed individually (step into).

Call stack
Presents information as to which functions/procedures have invoked other procedures.

Several candidates described how to use a variety of testing tools to find errors, for example, simulators and file comparators. However, the question explicitly indicated that the tool being examined was a debugger and thus it was features of this type of tool that were sought.

(c) Example compiler options include:
Select target platform
Used for multi-platform installations

Optimise code for speed or size
Depending on implementation environment

Read options from a file
Used when several options are to be selected

The question provided the example of a compiler option that allowed debugging code to be included or excluded and asked for OTHER options. Hence marks were not awarded to candidates who simply repeated this option.

(a) Three features required, 2 marks for describing each feature plus 1 for explaining how it reduces the possibility of errors, totalling 9 marks.
(b) Five features required, 1 mark for each feature identified plus 1 for its description, totalling 10 marks.
(c) Three options required, 1 mark for each option identified and 1 for an explanation of its purpose, totalling 6 marks.

Question 5

This question examines Section 5 of the syllabus, "Program Testing".

(a) Compare and contrast two possible approaches to the identification of test cases that could be used to adequately test computer software. You should clearly identify the advantages and disadvantages of each approach.

(12 Marks)

(b) Consider the pseudo code below and clearly identify the test cases that should be used to ensure the correct operation of the program.

```
Procedure leapyear(year)
BEGIN
  If (year > 1582)
    If (year is divisible by 400) Or
      (year is divisible by 4 but not 100) Then
      Display "Is a leapyear"
    Else
      Display "Not a leap year"
  Else
    Display "Unable to decide"
END leapyear
```

(5 Marks)

(c) Describe in detail a software tool that can be used to assist the testing process. Your answer should highlight the benefits of using the tool, give an example of where it would be appropriate to use it and identify any issues with which a user of the tool should be familiar.

(8 Marks)

Answer Pointers

(a) Two possible approaches are top-down and bottom-up testing:

Top-down

Starts testing at the system level with subsystems represented by stubs - simple components with the same interface as the subsystem. Each subsystem is then tested in a similar way.

Advantages

- Modules can be tested as soon as they are completed
- Structural design errors may be detected at an early stage
- Limited, working system is available at an early stage

Disadvantages

- Program stubs are required for lower level modules

Bottom-up

The opposite of top-down in that it starts with testing the lowest level modules and working its way up the hierarchy.

Advantages

- Does not require the development of module stubs to mimic lower level components.

Disadvantages
- To test a module, a test driver must be written.
- Structural design errors may not be detected until late in the implementation phase.
- System cannot be demonstrated until complete.

Other approaches were also acceptable, for example, white box and black box testing, equivalence partitioning and path testing.

(b) Test cases identified to include:

A year before 1582
The year 1582
Years after 1582 that are
- divisible by 400
- divisible by 4, but not a century
- divisible by 4 and which is also a century, but not divisible by 400
- Not divisible by either 4, 100 or 400.

(c) Any type of testing tool is acceptable, but an example (to indicate the level of detail required) is a test data generator:

Test Data Generator
Description
- Generates a large number of test cases using a specification of the syntax of the input.

Benefits
- Automates the generation of test cases, thus saving time (and therefore money)

Typical Uses
- Testing performance in a practical environment, e.g. a database system dealing with a large volume of transactions.

Issues
- Do not relieve the burden of checking test outputs.
- Need to ensure syntax of test inputs is accurate.

Many candidates briefly described several tools rather than concentrating on “a” tool as indicated by the question, thus not providing the level of detail required on any of the tools they described.

(a) Two techniques were required, 2 marks for a description of each technique, 4 for the advantages and disadvantages of each, totalling 12 marks.
(b) 1 mark awarded for each test case identified up to a maximum total of 5 marks.
(c) 2 marks for a description of the tool, 2 for its benefits, 2 for typical uses and 2 for any issues, totaling 8 marks.

Question 6

This question examines Section 6 of the syllabus, "Quality Assurance and Documentation".

(a) Software quality is often expressed in terms of quality factors such as re-usability, portability, maintainability, reliability and clarity. In the context of software engineering, explain the terms in italics and indicate how each may be achieved.

(10 Marks)

(b) Having observed several different commenting styles and standards in the code produced by different software engineers in a large software house, the quality division has decided that programming style should be standardised. Write a document that defines a suitable standard. You may assume the company uses a programming language with which you are familiar.

(15 Marks)

Answer Pointers

(a) **Re-usability**

- Refers to the easy with which the code could be reused within similar applications.
- Can be achieved through the use of modularisation in which modules are loosely coupled but highly cohesive.

Portability

- Refers to the ease with which the code can be transferred to other hardware platforms.
- Can be achieved by isolating hardware dependant code from the main application logic.

Maintainability

- Refers to the ease with which programs can be updated to extend their useful life.
- Can be achieved through good programming practice, e.g. commenting, use of meaningful variable names, modularisation etc. Can also choose the simplest algorithm where alternatives exist.

Reliability

- Refers to the probability that the program will operate as intended.
- Can be achieved through adequate testing, including appropriate load testing.

Clarity

- Refers to the ease with which the code can be read and understood.
- Can be achieved by using the same techniques as maintainability.

Many candidates failed to describe at least one of the quality factors requested and a many others failed to indicate how the desirable quality could be achieved. Further, several candidates simply indicated, for example, that reusability was the ability to reuse the code. The examiners were expecting a more detailed explanation.

(b) Possible answers included (actual description depends on language selected):

- Identifier naming conventions, e.g. for variables, functions, procedures, classes etc.

- Indentation standards, e.g. tab size
- Layout of language syntactic structures
- Format and content of comment block
- General commenting style
- Guidelines for the number of lines of code in any one procedure/function
- Guidelines for the number of procedures/functions in a single file

By using the phrase “define a suitable standard” the examiners were expecting candidates to detail a variable naming convention etc, not just simply state that one should be defined. Also, some candidates wrote several paragraphs detailing one aspect, e.g. commenting standards, but failed to cover the wider range of issues that need to be addressed in such a style document. In general, answers were typically shorter than expected, as indicated by the allocation of 15 marks to this part of the question.

(a) 1 mark for the description of each factor plus 1 for indicating how it may be achieved, totaling 10 marks.

(b) 2 marks each issue addressed up to a maximum of 12, plus 3 for the style of document, e.g. does it clearly define a standard or simply talk about what could be done, is it written in the style of a company report?