THE BCS PROFESSIONAL EXAMINATION
Diploma

April 2004

EXAMINERS' REPORT

Software Development Environments

General

- It is necessary for the examiner to be able to read your answer. Try to write legibly.

- Candidates should answer the question posed. There was a strong tendency to simply identify key words in the questions and then regurgitate large quantities of textbook material on these topics rather than applying this knowledge to the question asked. Much if what was written by many candidates was simply irrelevant to the question, even if accurate.

Question 1

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

1. *a)* Briefly describe the requirements, specification and design stages of the software development lifecycle. For each of the three stages give an example of a common error that can be made. **(9 marks)**

b) At what stage of development would a prototype be constructed and what would it be expected to achieve? In particular, why might a non-functional prototype interface be built rather than a more complete prototype? **(5 marks)**

c) Various diagramming notations can be used at the specification stage. Describe **ONE** such system giving a simple example of its use. **(6 marks)**

d) Of the various ways of creating a program or system specification, one is known as a "Formal Specification". Describe what is usually meant by a formal specification and give an example of a situation where a formal specification would be used, or preferred, over a less formal specification. **(5 marks)**

Answer Pointers

Part a)
Poor answers contained sentences like:

The requirements stage is where the requirements are obtained.
A common error in requirements is getting the requirements wrong

Some answers had sections missing (was this carelessness or lack of knowledge?)

Requirements

[**Description**] A list of features that should be present in the finished software.
[**Common Error**] A missing requirement could lead to a system being built to specification but not what the user wanted.

Specification

[**Description**] A list of (abstract) data structures & operations with implicit relationships between data & results.
[**Common Error**] An inconsistent specification with a contradiction in it will lead to problems at a later stage.

Design
[**Description**] A list of data structures and operations with explicit relationships between data & results (algorithms).
[**Common Error**] Lack of attention to the detail of interface issues can lead to individual modules being designed correctly but used wrongly (e.g. called with transposed parameters).

Part b)
Stage & Role of Prototype: An analyst creating requirements or specification documents may baffle a customer with notation in these documents; showing a prototype application is very direct but requires a huge amount of effort.
[Note: prototypes can be used in other phases of development. Full marks could be obtained by giving a suitable description of the role of a prototype in any phase]

Why non-functional Prototype: If there is a significant interface to the application then showing the customer the 'look and feel' of the interface (with no functionality behind it) can achieve much of the same communication with considerably less effort.

Part c)
Description of a diagramming structure required

e.g. dataflow diagrams
with definitions of symbols for

- dataflow (directed)
- user interaction
- transformation process
- data source

together with a (simple) example.

Part d)
This section was least well answered.

What is meant by a Formal Specification: A formal specification is usually presented as a set of mathematically precise, hierarchical relations regarding acceptable data (pre-conditions) and results obtained from the data (post-conditions). If a pre-condition is satisfied by some data, then the program is required to produce corresponding output that satisfies a post-condition.

Situation for use: This kind of specification is necessary if a formal proof of correctness of the eventual program is required. This might occur in a safety-critical application.

Mark Breakdown
a) 2 marks for each description and 1 mark for each common error (total 9 marks)
b) 3 marks for role of prototype in specific stage, 2 marks for reason to use non-functional prototype (total 5 marks)
c) 6 marks
d) 3 marks for description of a formal specification & 2 marks for example situation of use (total 5 marks)

Question 2

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

2. *a)* Describe **FIVE** functions of modern operating systems. **(10 marks)**

b) Windows and Unix are usually quoted as examples of operating systems with contrasting kinds of user interface. For these two operating systems (*or any other two with contrasting user interfaces with which you are familiar*)

i) Outline their differences by briefly describing their user interfaces.

ii) Describe **ONE** difference between the two operating systems that is not concerned with the user interface. **(7 marks)**

c) The central activity initiated by a user of an operating system is to cause it to execute a particular program with a particular file as data. Describe how this is achieved under the two types of operating system, using the two types of user interface in *b)* above, and give **ONE** advantage for each approach. **(8 marks)**

Answer Pointers

Mostly well answered. Poorer answers were typically those with missing sections or inappropriate answers.

Windows+Unix or Windows+DOS made good pairs of operating systems for this question.

Part a)

5 (general) functions of a modern operating system:

e.g.

- handle hierarchical file store
- handle peripherals
- run user programs
- schedule processes
- memory allocation (virtual?)

other possible functions include

- access to web/internet

provide user interface

Part b)

(i) **Outline interface differences:**

Windows=WIMP (Windows, Icons, Menus, Pointer) graphical i/f
Unix=Command line i/f - keyboard-based

(ii) **Difference (not concerned with interface):**

(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 Unix

Part c)

How is a program executed with a specific file as data:

WIMP: data file is dragged over icon of program using mouse pointer (keyboard not used)
Command Line: exact filename program to be run is typed at prompt along with exact name/path of data file

1 advantage of each approach:

WIMP: good for novices (do not have to learn/remember command names)
Command Line: good for batch processing, system work, ability to write scripts

Mark Breakdown
a) 2 marks per function (total 10 marks)
b) (i) 2 marks for each interface (ii) 3 marks for difference, not concerned with interface (total 7 marks)
c) 2 marks for how to achieve under each operating system and 2 marks per advantage (total 8 marks)

Question 3

This question examines section 3 of the syllabus "Program Design"

3. *a)* Novice programmers often need guidance to enable them to recognise when to use a procedure (*or function or subroutine*) in a program. How would you advise a novice in that situation? **(6 marks)**

b) Explain how it is possible to have a programmer write a procedure for inclusion in a larger program without being aware of the details of the larger program. **(5 marks)**

c) A procedural design for a computer program has split the problem into a selection between tasks A1 and A2 on the basis of test Q. Task A1 has been further broken down into a sequence of task B1 followed by B2. Task B1 is an iteration of task C1 with a pre-condition (*test at the beginning*) Q1. Task A2 is an iteration of task C2 with a post-condition (*test at the end*) Q2.

i) Present this design as a diagram (flowchart).
ii) Present the same design in procedural code (*in a language of your choice*). **(14 marks)**

Answer Pointers
There were a few answers along the lines of 'everything I know about procedures' rather than phrasing an answer to the actual question.

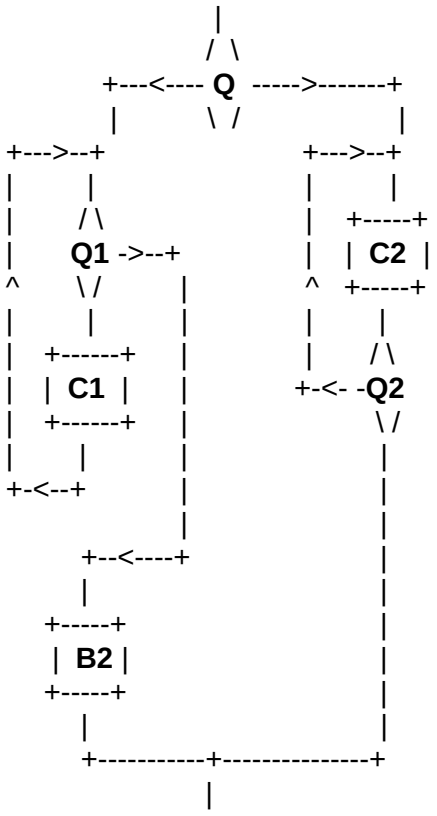
In c(i) the names A1, A2, B1 should not appear inside boxes on the finished flowchart. Similarly, unless used as procedure names, A1, A2, B1 will not appear in c(ii).

Part a)
Advice for novice on recognising when to use a procedure:
A procedure can be recognised by size (the program would be too big if it were not split up into (logical) pieces.
A procedure can be recognised by logical function.
A procedure can be recognised by repeated section in program.

Part b)
How can a procedure be written without being aware of the main program:
Apart from the specification of what the procedure is to achieve, the parameter pack of the procedure (and any intended result) must be specified precisely in order and type. Only then can one programmer safely call a procedure that has been prepared in isolation by another programmer.

Part c)

(i) Flowchart



(ii) **Procedural code**:

```
if(Q){
    while(Q1)
    do C1;
    B2
} else
    do C2
    while(Q2);
```

Mark Breakdown

a) 3 marks per point made (max 6 marks)
b) 5 marks for explanation
c) (i) 7 marks for flowchart (ii) 7 marks for procedural code

Question 4

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

4. *a)* Discuss how each of the following features, often found in program development environments, are of benefit to the programmer:

i) Colour coding of keywords
ii) Automated indentation
iii) Automated completion of code constructs **(9 marks)**

b) Describe how the following features of a debugger can be used to assist in the identification of errors within a computer program:

i) Break points
ii) Watches (also known as variable inspectors)
iii) "Step over" and "Step into" operations **(10 marks)**

c) Outline the purpose of **THREE** compiler options with which you are familiar and indicate circumstances under which each of these would be used. **(6 marks)**

Answer Pointers

In general, this question was well answered.

a) **Colour coding of keywords**
Key words of the language highlighted using colour, allowing typographical errors to be identified more easily. Comments are also often displayed in a different colour, ensuring the actual code stands out from comments and also highlighting missing end comment tokens.

Automated indentation
Indentation is used to aid understanding of the program's execution. For example, in an iterative structure, the body is often indented to clearly show which statements are to be repeatedly executed.

Automated completion of code constructs
Reduces the number of syntax errors in a program (e.g. misspelt keywords and/or missing "end" tokens). Can also be used to select methods within objects, with hints to indicate the number of parameters required and hence can help to ensure methods are correctly invoked, again reducing the number of syntax errors.

b) **Break points**
Allow the program to be run at normal execution speed until the textual area of the program where an error is thought to be located is reached. Thus breakpoints speed up the debugging process.

Watches (also known as variable inspectors)
Allow the user to view the value of an individual variable and/or an expression at any desired stage in the execution of the program. Watches can be used to view the values of internal variables at predetermined points (at which expected values of the variable can be pre-determined) and hence narrow down the location of an error.

"Step over" and "Step into" operations
These commands allow the code to be executed statement by statement. In particular, when executing function calls, the "step over" command treats the function call as a single statement, ignoring the internal operation of the function. In contrast, the "step into" command causes each individual statement within the function to be executed when determined by the user (using this "stepping" functionality).

c) Any three compiler options are acceptable. The following are examples:

Define target platform
Used when the same compiler can generate code for several different hardware platforms and the software is to be deployed in a multi-platform environment.

Optimise code
Typically code can be optimised for either speed or size. If the software is for an embedded system, the size of the executable may be more important than execution speed. However, if the code involves millions of complex calculations speed optimisation may be preferable.

Include/Exclude debugging code
Typically developers would include debug code when building for test purposes and exclude his code for a final shipping version, increasing the efficiency of the final software product.

Mark Breakdown
a) 3 marks for each feature
b) 3 marks for the description of breakpoints and watches, 4 marks for step-over and step-into
c) 2 marks for each option

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

5. *a)* Discuss the following statement: "The purpose of testing software is to prove it works as intended." **(6 marks)**

b) Identify **TWO** software tools that assist the testing process and for **EACH** indicate:

i) How each tool assists the testing process
ii) An example where it would be appropriate to use each tool
iii) Any disadvantages associated with using each tool **(12 marks)**

c) Describe **ONE** strategy that assists in the identification of actual test cases. **(7 marks)**

Answer Pointers
For part a), many candidates discussed why it was necessary to test a program rather than discussing the actual statement in the question. Most candidates failed to recognize the key point of the question, namely that software testing cannot normally ***prove*** a program's correctness, but rather it is simply an attempt to find as many errors as possible.

In part b) some candidates discussed approaches to testing (e.g. black box and white box) rather than specific ***software tools*** that support the testing process.

Answers to part c) in general lacked detail as to how ***actual test cases*** were identified. Some candidates simply discussed high level approaches, for example top down or bottom up, indicating which modules were tested at what point, but not how to identify ***actual test cases***.

a) Key points:
- Testing cannot normally prove a program is correct, it can only identify errors.
- Exhaustive testing is usually not possible, even for "simple" examples.
- The more tests carried out the more likely it is that errors will be identified but the more cost involved.
- Therefore there is a need for compromise between the number of tests and the costs involved.

b) Any software testing tool was acceptable. The following outline answer assumes the candidate has selected a simulator and a test data generator.

Simulator

How the selected tool assists the testing process

- Allows testing to be carried out without the need for potentially expensive hardware and/or without the risks associated with safety critical systems.

An example where it would be appropriate to use the selected tool

- Software for the control of a nuclear reactor

Any disadvantages associated with using the tool

- Simulation environment may contain errors
- Simulator may not accurately reflect the real environment
- Difficult to test functions that are dependant on timing

Test Data Generator

How the selected tool assists the testing process

- Automatically produces vast quantities of test data using a specification of the syntax of the input, thus saving time (and money).

An example where it would be appropriate to use the selected tool

- Testing the performance of a large database application

Any disadvantages associated with using the tool

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

Mark Breakdown

a) 2 marks for each of the first two bullet points above, 1 mark each for the last two
b) 6 marks for each tool discussed
c) 7 marks, split depending on strategy selected

Question 6

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

6. *a)* Discuss how maintainable software can be achieved. **(7 marks)**

b) A particular software development company has a small development team and concentrates on the production of a single, evolving product. Each developer works largely in isolation on different aspects of the software, writing code in their own individual style. All files are stored on a central server so that everyone has access to all code. When a developer has an idea for new functionality, they take a copy of the code from the central server, update it, test it themselves and then copy the modified files back to the server. If a customer requests a change to the software, the developer with responsibility for the appropriate aspect of the software makes the change. Again the resulting software is tested by the same developer and when complete, the new version is stored on the central server. Each developer schedules their own work, balancing new development tasks with bug fixes and customer change requests.

When customer installation disks are required, the code stored on the central server is copied onto a blank disk (along with an appropriate install routine) and dispatched. The only documentation is the code itself as the developers are told to use comments throughout their code and this is deemed sufficient within the company.

Discuss **SIX** problems associated with the situation described above and indicate how they may be resolved. **(18 marks)**

Answer Pointers
When answering part (a), many candidates focused on “well presented” code, failing to comment on structuring issues. Several candidates simply discussed why maintenance was necessary rather than how maintainability can be achieved. In general, part (b) was well answered, although some candidates failed to indicate how the identified problems could be ***resolved***.

a) For software to be maintainable, it should be (for example):

Well structured
The software should be highly cohesive and loosely coupled. This implies that related functionality should be encapsulated within parts of the program (e.g. within a class or file) and that the links between these units should be minimised.

Well presented
Code readability (and thus maintainability) will be enhanced through good program layout (e.g. using indentation, using meaningful variable names and the inclusion of comments that explain the operation of the code.

b) Examples of the types of problems to be discussed include (others are acceptable):

Problem: The company is highly dependant on individual developers as they are the only member of the team with knowledge of a particular aspect of the software.

Solution: Each team member could be assigned primary responsibility for individual aspects of the software and secondary responsibility for others, thus spreading the knowledge of how the software operates across a number of individuals

Problem: There is a lack of change management procedures both in terms of new functionality and change requests.

Solution: New ideas for additional functionality should be noted and a case for development time argued at say, a development team meeting. This would ensure that the “best” ideas were implemented first. Change requests should also be prioritised to ensure the most urgent changes are carried out first. However, care must be taken to ensure “less important” changes are still scheduled within an acceptable timescale.

Problem: There is a lack of overall software structure/architecture

Solution: The current system architecture should be documented and all subsequent changes to this documented and formally approved.

Problem: Lack of documentation, implying that should one developer leave, it will be difficult for someone else to take over the responsibility for their code.

Solution: Each developer should be asked to document the operation of the software for their own particular aspect. In addition, links between the functional aspects should be documented. New change management procedures should ensure that the documentation is then updated as changes to the code are implemented.

Problem: Lack of coding standards

Solution: Develop a “house style” including guidelines for variable naming conventions, indentation structures, commenting standards etc.

Problem: If two developers happen to make changes to the one program file simultaneously, one set of changes will be lost.

Solution: Invest in software such as SourceSafe to ensure only one developer can work on the code at any one time.

Mark Breakdown

a) 4 marks for discussing code structuring, 3 for code presentation

b) 1 mark for each problem identified and 2 for each solution up to a maximum of 18 marks