**THE BCS PROFESSIONAL EXAMINATION**
**Professional Graduate Diploma**

**April 2004**

**EXAMINERS' REPORT**

**Programming Paradigms**

## General

This module had a 94.5% pass rate. The highest mark was 84% and the mean was 56%. The general level of candidates' performance was therefore very pleasing.

Questions 1 and 2 were by far the most popular and question 3 by far the least. Question 4 was the best answered and question 3 the worst but the differences were not large.

## Question 1 (Syllabus section 1)

The merger of two international computer service companies in the UK and USA has led to the management of the new company consolidating their programming provision. For efficiency reasons, they have decided to standardise on using one programming paradigm for all future projects. The work they carry out is usually for information systems and database projects.

The employees from both companies have a mixed background of programming experience; at the moment there is no favoured programming paradigm.

Write a report for the management recommending a short-list of two programming paradigms to consider adopting. Give the strengths and weaknesses of both approaches, and say why the chosen paradigms would be appropriate in this environment and what type of applications they would be most suitable for. (25 marks)

A description of the characteristics of the chosen paradigms should be first of all given, with an evaluation of their strengths and weaknesses.

In their discussion candidates needed to consider what types of applications the language is most suitable for and a good answer would have related these types of applications to particular features. For example, imperative languages are more general-purpose languages and could be used for a variety of applications, whereas non-imperative languages, including functional and logical languages, are targeted at more specialised programming applications. OO languages may suit a company who use OO methodologies for designing the system.

A basic answer would have described two paradigms, with some evaluation. To get a good mark, some discussion of the type of system the paradigm is aimed at was required.

On the whole candidates made a good attempt at describing two approaches. A weaker answer just described the two paradigms, some only giving the advantages, with few discussing the disadvantages of one or both paradigms. A higher mark was given to candidates who tailored the answer to the background to the company, saying why for instance a particular paradigm was good for database or information systems projects.

A few candidates lost marks by concentrating just on one paradigm, with a brief attempt at the second one. On the whole most candidates did, however, describe two paradigms with one candidate even describing three. The object-oriented paradigm was the most popular paradigm, followed closely by the imperative approach, with a few looking at the functional and logic paradigms.

## Question 2 (Syllabus section 2)

The manager of an IT department has decided to buy the company's team of programmers an Interactive Development Environment (IDE).

*a)* Describe what features an IDE should have to support a team of ten programmers. (10 marks)

*b)* The manager needs to justify the expense to the Finance Director. What benefits will the IDE bring to offset the costs? (15 marks)

For the first part, the candidates were expected to describe the programme development tools found in a typical IDE. Candidates might have discussed debuggers, testing tools, linkers, configuration tools, loaders, screen painters, code generators, etc.

The second part required candidates to reflect on the real usefulness of these tools. Within their discussion they should have indicated why they think the tools are important or not. For example, they could have argued that they improve the speed and performance of the programmer, help with debugging, testing, etc. Disadvantages could include that the programmer may fall into bad programming practices by relying on the tool to correct mistakes, so encouraging hacking, rather than getting the programmers to check and test their code thoroughly themselves.

For both parts, examples are expected of appropriate tools, for example, Microsoft Visual Studio, Borland C++, Borland JBuilder, etc. They need to identify what elements of the tool are useful to productivity and say why they think they are beneficial or not.

Most candidates made a good attempt at describing the features of an IDE. A better answer tailored the question to say what features in particular were useful for a team of programmers, for example, the ability to check-in/out code, version control, shared code repositories and resources. In part (b), some candidates repeated most of part a again, without any further elaboration. The question asked focussed on the advantages of using an IDE, though a few students also highlighted what the problems could be, which made for a more thoughtful answer.

## Question 3 (Syllabus sections 1, 2 and 4)

*a)* Most modern programming languages, imperative and applicative (functional), support exception handling. Why is exception handling necessary, and what problems are associated with providing this facility? (12 marks)

*b)* A functional programming language can contain declarations in addition to defining functions and evaluating expressions. Discuss the nature of declarations within a functional programming language.

Choose a specific functional language and a specific imperative language and compare and contrast the use of declarations in them. (13 marks)

In part (a) candidates were expected to describe, for example, the need for exception handling in embedded systems such as aircraft software, where error recovery must be performed without human intervention. They were expected to comment upon the need for exception handling facilities to have low overheads if no exception occurs, and the need for exception handling to be easy to use and safe. Herein lie the problems of providing such a facility. Credit was given for any arguments correctly put.

In general the answers to part (a) were poor and few candidates mentioned the requirements for low overheads and for exception handling to be easy to use and safe. Very few candidates mentioned an embedded systems example.

In part (b) candidates were expected to demonstrate their functional programming specific knowledge, and to understand how environments are relevant to imperative programming languages also. Candidates were asked to comment upon similarities, or otherwise, and should have discuss issues such as bindings, structures, signatures and functors.

Answers to this part were considerably better than those for part (a). There were some very good comparisons between functional and imperative languages. However, few candidates discussed the issues of bindings, structures, signatures and functors.

### Qu*estion 4 (Syllabus section 5)*

*a)* Pure logic programming is built upon a small number of basic concepts. Describe these concepts and discuss the compromises that have to be made when the pure logic programming paradigm is implemented in a real logic programming language. (12 marks)

*b)* "Logic programming should be integrated with other programming paradigms such as functional programming". Comment upon this statement, presenting arguments for and against such integration. (13 marks)

In part (a) candidates were expected to demonstrate their knowledge of the basic concepts of pure logic programming. Based upon their familiarity with a real logic programming language, candidates were expected to understand that 'extra-logical' features are required, which are not in keeping with a pure logical approach. It was expected that the compromise discussion might refer to logical or non-logical aspects, such as interface design or time dependency versus simplicity and consistency. No reasonable approach was rejected.

In general the answers to part (a) were very good with a number of candidates being awarded full marks. There was clear evidence that many candidates understood that 'extra-logical' features are required, and many had an excellent familiarity with a real logic programming language.
In part (b) candidates were expected to present 'for' arguments that might have included the very close relationship between the mathematics of functions (lambda calculus) and logic. The 'against' arguments were expected to be greater in number, and might have included differences in unification, use of inference engine for logic programming, etc. Credit was given for any arguments correctly put.

Overall part (b) was not answered as well as part (a). A number of candidates did briefly discuss the very close relationship between the mathematics of functions (lambda calculus) and logic, and a few did mention in passing the differences in unification. Generally, insufficient detail was provided by the candidates to attract a high mark.

### Question 5 (Syllabus section 6)

*a)* Within the context of multi-programming operating systems, multi-tasking operating systems, and embedded systems, describe what is meant by the term concurrency. Illustrate your answer with examples for each system. (10 marks)

*b)* If a programming language claims to support concurrency, it must provide solutions to the problems of process synchronisation and inter-process communication. Explain the nature of these problems and discuss some of the proposed solutions. (15 marks)

In part (a), candidates were expected to demonstrate their wider knowledge of concurrent systems, and to illustrate the application of this type of computing using appropriate examples, drawn from their own experience, and/or background reading. Candidates were expected to define concurrency within the context of the three systems, and thereby introduce and describe additional terms such as time-sharing, task scheduling, real-time, and processes. No reasonable approach was rejected.

Whilst there were a number of excellent answers to part (a), in a few cases some candidates failed to provide sufficient detail. For some candidates there was clear evidence that they had a

good familiarity with a programming language that supported concurrency, and hence they were able to put this experience to good use within their answer.

In part (b) candidates were expected to present their understanding of the need for synchronisation and communication within concurrent software. This need presents many problems, and candidates were expected to describe the importance of synchronisation and communication in the context of the models that have been developed to overcome these problems. Answers could have included a discussion of the shared memory model, the mutual exclusion problem, monitors, and synchronous message passing. Credit was given for any discussion correctly put.

Once again there were a number of excellent answers to part (b). Many candidates included a discussion of the shared memory model, the mutual exclusion problem, monitors, and synchronous message passing. A number of candidates discussed Dekker's algorithm, and a few included a discussion of the problem of distributed termination and the Dijkstra-Scholten algorithm.