THE BCS PROFESSIONAL EXAMINATIONS
Professional Graduate Diploma

April 2006

EXAMINERS' REPORT

Programming Paradigms

General

This module had a pass rate of 92%. The highest mark was 76% and the mean was 53%. The general level of candidates' performance was therefore very pleasing, and very slightly better than last year.

As in 2005, questions 1 and 2 were by far the most popular and question 4 by far the least. The average mark for question 1 was significantly worse than for other questions and the average mark for question 4 was better, despite its unpopularity.

Question 1 *[Syllabus section 1]*

1. A national supermarket, *Mini-Mart*, wishes to allow its product database to be made available over the web. Customers initially will be allowed to browse what products it stocks and eventually be allowed to shop online. The company currently uses C++ for all its IT systems but is now considering moving to another language for the web system.

As the senior programmer, you have been tasked with writing a short report evaluating possible choices of language. Choose two different programming paradigms, such as object-oriented programming, scripting languages, logic programming, functional programming, or imperative programming, and evaluate their strengths and weaknesses for such a web-based system. **(25 marks)**

The candidate should have given a description of the characteristics of the two chosen paradigms, with an evaluation of their strengths and weaknesses.

Within their discussion they needed to discuss how the paradigm would be appropriate for the web system. A good answer would have related the features of the paradigm to this particular type of application.

For example, a scripting language might be the best language for such a system, since it has features useful for connecting a database to the web. Imperative languages are more general-purpose languages and could be used for a variety of applications, whereas non-imperative languages include functional and logical languages and are targeted at specialised programming applications so could be argued are not appropriate for such a system. An OO language, such as C++ or Java could be used, since they both can be used within the CGI framework for web applications.

A basic answer would have covered the description of two paradigms, with some evaluation. To get a good mark, candidates needed to justify why that paradigm was appropriate for such a system.

Most candidates made a good attempt at this question. The most popular paradigms chosen were object-orientated and scripting, which were the most appropriate paradigms for a web-based system.

A higher mark was given to candidates who gave thought as to why the paradigm was suitable to web-based systems, giving examples of their strengths and weaknesses, as opposed to candidates who just reviewed the features of the concept, without saying why they were important. A lower mark was given to candidates who chose a paradigm that was less relevant, such as logic

programming, and could not give a convincing argument as to why the approach would or would not be suitable.

Some candidates only discussed one approach, often the object-oriented paradigm.

Question 2 *[Syllabus section 2]*

2. *Cap-Saturn* is an international facilities management company. The Technical Director has decided to buy the company's programmers an interactive development environment (IDE). The programmers typically work in individual teams of around ten, but occasionally there is a need for teams from different countries to work together.

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

b) The Technical Director needs to justify the expense to the Company's Financial Director. What benefits will the IDE bring to offset these costs? **(15 marks)**

For the first part, the candidates were expected to describe the tools for program development typically found in an IDE. Candidates might have discussed debuggers, testing tools, linkers, configuration tools, loaders, screen painters, code generators, etc. A better answer would also consider the team aspects: some tools allow the program to be checked-in and out and support version management, which could be important if several programmers are working on the same area, particularly for teams not based physically close to each other.

Part b required the candidate 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. Therefore programmer productivity is increased. Most things have disadvantages too and these 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 were expected of appropriate tools, for example, Microsoft Visual Studio, Borland C++, Borland JBuilder, etc. Candidates were expected to identify what elements of the tool are useful to productivity and to say why they thought they are beneficial or not in a team environment.

Most candidates made a very good attempt at describing the features of an IDE. A higher mark was given to answers which tailored the question to say what features in particular were useful for a team of programmers, for example, version control, the ability to check-in/out code, shared code repositories and resources.

Often the candidates received a lower mark in part (b), because they repeated most of part a again, without any further elaboration. The question required the candidate to justify the cost of using an IDE, so they needed to focus on the advantages of using such as system for a team environment, though some students also highlighted what the problems could be, for example, using an IDE could lead to inefficient code, which made for a more thoughtful answer.

Question 3 *[Syllabus section 4]*

3. *a)* A run-time error is called an *exception*. In the context of an embedded system, why is exception handling necessary and what problems are associated with providing this facility? **(12 marks)**

b) What is meant by the term referential transparency? Illustrate your answer with examples from both an imperative and a functional programming language with which you are familiar. **(13 marks)**

In part a) of the question, 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 them to be easy to use and safe. Herein lays the problems of providing such a facility. Credit was given for any arguments correctly put.

In answer to part b), candidates were expected to put forward an explanation such as an expression E is referentially transparent if any sub-expression and its value (the result of evaluating it) can be interchanged without changing the value of E. With the aid of examples students should have shown, for example, that a routine written in an imperative language is (potentially) referentially opaque, whereas in a functional language referential opaqueness is forbidden and referential transparency is mandatory.

Question 4 *[Syllabus section 5]*

4. *a)* "To be truly practical, logic programs includes features that have nothing to do with logic programming." Explain why this statement is true. **(12 marks)**

b) A software inference engine conducts logical inferences from one formula to another until the problem is solved. What role does unification have to play in this process? Illustrate your answer with examples from a logic programming language. **(13 marks)**

In part a) of the question, 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. 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 answer to part b), candidates were expected to explain that an inference engine tries to match the goal and the conclusion of a formula; hence there is a requirement for generalised pattern matching. This process is called unification which enables links to be maintained between variables and data structures (or components of data structures). Candidates were expected to introduce the term instantiation, and to illustrate the unification process with the aid of a logic programming language with which they were familiar.

Question 5 *[Syllabus section 6]*

5. *a)* Explain the term concurrency using examples from multi-programming operating systems, multi-tasking operating systems and embedded systems. **(10 marks)**

b) In the context of concurrent programs, explain the mutual exclusion problem. Discuss the advantages and disadvantages of the possible solutions to this problem using examples from a programming language that supports concurrency. **(15 marks)**

In answer to part a), candidates were expected to demonstrate their knowledge of different types 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.

In answer to part b), candidates were expected to introduce and define terms such as interleaving and atomic instructions, and to use these to explain the mutual exclusion problem. Concepts such as critical and non-critical code sections and Dekker's algorithm were expected to be introduced, together with solutions such as semaphores and monitors. Credit was given for any clear illustration as to the advantages or otherwise of these solutions. Additional credit was given for using examples from real concurrency supporting programming languages, for example, the use of Ada protected variables .