BCS THE CHARTERED INSTITUTE FOR IT

BCS Higher Education Qualifications

BCS Level 6 Professional Graduate Diploma in IT

March 2015

EXAMINERS' REPORT

Programming Paradigms

General comments on candidates' performance

The questions covering the object-oriented paradigm and paradigms for web development were the most popular questions, with each being attempted by approximately 90% and 74% of the candidates; they were reasonably well answered. The remaining questions were less popular and generally not so well answered, although there were some good answers.

Question A1

Easy Rambles is a travel agency that specialises in walking holidays and wishes to allow its customers to book their holidays online. The company is currently using a mainframe based technology but wants to move towards a Linux platform that will support a web system. This will require a change of programming language from COBOL to something more suitable.

As the IT Manager, you have been asked to produce a short report assessing possible choices of language. Choose two different programming paradigms, such as object-oriented programming, scripting languages, logic programming, functional programming, or imperative programming, and assess their strengths and weaknesses for use in such a web-based system.

(25 marks)

Answer pointers

Syllabus Section: Nature of Programming Languages

Candidates were expected to describe 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 scenario. Good answers related the features of the paradigm to these particular types of applications.

For example, 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 would be less suitable. Object-oriented languages may be considered too, particularly if they consider using an object-oriented methodology for designing the system, such as

UML. For a web-based application the most likely paradigms to be chosen are object-oriented or scripting.

A basic answer would cover the description of two paradigms, with some evaluation. To get a good mark, some discussion must cover what type of system the paradigm is aimed at.

Examiners' comments

This question was attempted by 74% of the candidates, of whom 67% achieved a pass mark.

Most candidates could discuss one paradigm in detail, however, to achieve a high mark, the candidate did need to discuss two paradigms.

The most popular paradigms were scripting and object-oriented. There is evidence that some candidates assumed that all scripting languages run on the client, such as JavaScript, and could be viewed by the user. There was no appreciation that other scripting languages run on the server, such as PHP, and implied therefore that no server requirements were needed by the company.

Weaker candidates only discussed what features the chosen paradigms offered, with no discussion of the strengths and weaknesses. In order to gain full marks, the answer had to be tailored to the question asked, discussing the languages with respect to the scenario given.

Question A2

a) Software testing is an important part of producing a quality system. Discuss what tools can be used to test a large complex system, such as an air traffic control system, including the advantages and disadvantages of using such tools.

(15 marks)

b) Discuss the key reasons why it is important to have an international standard for any programming language.

(10 marks)

Answer pointers

Syllabus Section: Nature of Programming Languages and Programming Environments

In part (a) the candidates were expected to discuss tools such as debuggers, testing tools, linkers, configuration tools, loaders, screen painters and code generators.

The candidates should reflect on the real usefulness of these tools. Within the discussion, candidates should indicate why they think the tools are important or not. For example, how effective are debugging tools in the installation stage.

Examples of appropriate tools should be included, for example, Microsoft Visual Studio, Borland C++, JBuilder, Rational Quality Manager, Oracle Application Testing Suite, IBM Oliver (propriety tools), or open source/free tools include: CubicTest, Fastest or Java PathFinder.

In part (b) the candidates were expected to reflect on the need for standardisation as the process of producing a technical standard that can be used amongst both the competing entities producing the products and the users of the product, such as C++ or Java. For example, if there was no C standard, it could mean you have to learn one variation of C against one product and another version against a different product.

Example points for discussion could include: (i) inconsistencies can be introduced, (ii) need multiple users/international agreement of what is in a language, (iii) maintenance affected, (iv) communication difficult with different stakeholders, (v) establish world-wide recognition, and (vi) interoperability.

Examiners' comments

This question was attempted by 37% of the candidates, of whom 57% achieved a pass mark.

This proved to be a less popular question. Some candidates were able to answer either part (a) or part (b) well, but often not both.

For part (a), there is evidence that some candidates only read the tools part of the question and proceeded to describe the main features of an IDE. Whilst some IDEs have testing features, it was not appropriate to discuss things like screen painters and menu builders; such answers gained few marks. For those candidates who did discuss testing tools, often the answer just described the tools and did not make any attempt to discuss their advantages and disadvantages. Good answers included this comparison and also tailored the answer to thinking about what is appropriate in a large complex system.

Part (b) was generally answered well, with good answers discussing why the standard is important and not just describing what might be in one.

Question B3

a) "A logic program written in the Prolog programming language specifies what a program should do and not how it does it." With reference to this statement, discuss the differences between a **logic** programming language and an **imperative** programming language.

(12 marks)

b) With the aid of examples from functional and imperative programming languages, explain what the term **referential transparency** means.

(13 marks)

Answer pointers

Syllabus Sections: The Nature of Programming Languages, Logic Programming and Functional Programming

In part (a), candidates were expected to demonstrate their knowledge of how a logic programming language differs from an imperative language. In terms of a logic programming language, the code will specify rules and goals. The language will try to satisfy these rules by using search – the search mechanism is built into the language. Imperative programs need to define all of the steps to solve a problem – therefore defining how a problem is tackled. A strong answer might talk about side-effects, which are not part of a pure logic language. Prolog does have extra-logical features that allow for I/O, for example. Credit would be given for relevant issues and discussion.

In part (b), candidates were to discuss that an expression is referentially transparent if it can be replaced by the result of the expression without affecting the result of the program. The answer should explain that this is possible because of the lack of side effects. With the aid of examples students should have shown a function that is referentially opaque and a function that is referentially transparent and discussed the fact that imperative languages often encourage side-effects whereas functional languages often restrict side-effects.

Examiners’ comments

This question was attempted by 50% of the candidates, 45% of whom achieved a pass mark.

Part (a) of this question gave a statement and asked candidates to discuss with reference to this statement. Many candidates simply listed/described aspects of the two language styles. There is evidence that whilst most candidates were able to explain one paradigm and then independently explain the other, they did not link these points by comparing and contrasting them or refer to the statement in question. The answers would also often be illustrated by two entirely unrelated snippets of code that did not help the comparison. For example, they frequently gave a parent/grandparent code example to illustrate the logic language, and then a summation of numbers code example to illustrate the imperative language. This does not help them to compare the two. Showing the same problem solved by both language styles would be a much better way to discuss the differences.

In part (b), most students demonstrated a good understanding of side effects. There is evidence that almost all candidates confused "referential transparency" with "purity". A pure function will provide the same result when given the same input. This is a useful property for debugging code. An impure function may have side effects. An expression, which consists of constants, operators, and function applications, can be referentially transparent if the functions that it contains are pure. It can be evaluated and replaced by its value, because it will always result in the same value. Candidates did not distinguish between functions (which are reusable and usually take arguments) and expressions (which may be evaluated and then replaced by the value).

Question B4

a) Explain what is meant by the term concurrency. Use examples to illustrate your answer.

(12 marks)

b) Within the context of concurrent programming, explain what is meant by **mutual exclusion** and describe possible ways of achieving it.

(13 marks)

Answer pointers

Syllabus section: Related issues

In part (a), candidates were expected to demonstrate that they understand how concurrency is part of modern computing. It was likely that the answer would introduce the ideas of time-sharing, task scheduling, processes and real-time issues. Examples might include multi-threaded systems, embedded systems and multi-tasking operating systems. Stronger answers may draw out the differences between concurrent software on one machine and a software system running as a distributed system.

In part (b), candidates were expected to demonstrate their understanding of the concept of mutual exclusion and why it is important. Solutions to the problem should talk about ways to protect access to code in critical regions. Different solutions exist, including semaphores and monitors.

Examiners’ comments

This question was attempted by 49% of the candidates, 54% of whom achieved a pass mark.

The answers were generally good for part (a), and there is evidence that candidates understood the basic principles of concurrency and some of the scenarios in which this would happen. Scheduling was neglected, as was distributed systems. Some of the definitions of concurrency were vague or recursive.

The concept of mutual exclusion was generally understood. The answers to part (b) would have been improved by including a wider range of solutions, and ideally to explain something of the advantages and disadvantages of these solutions. Many candidates only gave the Java synchronized examples.

Question B5

a) Briefly identify the key concepts of the object-oriented paradigm which aid application development.

(5 marks)

b) For TWO of the features that you identified in part (a), discuss how they help a programmer to design and implement an application. Illustrate your answer with appropriate examples.

(2x4 marks)

b) With the aid of example code, explain the purpose of an exception, and how it is defined and used in an object-oriented language. In your answer, discuss the advantages and disadvantages of using this language feature.

(12 marks)

Answer pointers

Syllabus section: Object Orientation

In part (a), candidates should identify issues such as Classes & Objects, Single and Multiple Inheritance, Encapsulation, Abstraction and Polymorphism. Other issues will be given consideration.

In part (b), candidates were expected to show an understanding of how the chosen concepts help when designing and implementing a program. For example, encapsulation is used to create types that contain the data and the operations that act on the data. This can help to deliver cohesive code and help developers to manage the data and operations in the system. A more advanced answer might touch on the issue of information hiding, which is a related idea but sometimes confused as encapsulation. Information hiding focuses on the access to such data and operations in a type.

In part (c), candidates were expected to describe the role of exceptions to handle errors and exceptional circumstances in OO systems. A suitable example is required to illustrate how an exception is defined, created, thrown and caught in a language.

Advantages include (i) clearer code, (ii) a convenient and predictable mechanism to create and handle the exceptions, (iii) defined types that represent different classes of errors. Disadvantages include (i) overheads in generating and handling the exception,
(ii) mistakes in the ordering of handlers might prevent the correct error handling to be run and (iii) overuse can lead to code that is harder to read.

Examiners' comments

This question was attempted by 92% of the candidates, 91% of whom achieved a pass mark.

Part (a) was often too brief, with just 5 words written down. A sentence for each would suffice to show the candidate understands the words. While some of the concepts given do indeed aid OO application development, they also aid any program development (for example "abstraction"), so the candidates need to relate their answers to the question that was asked, in this case, to OO. What abstraction techniques are specific to OO? Polymorphism was often described as "having many forms" without explaining what has many forms.

Candidates seemed fairly comfortable with part (b). Weak answers would describe features but not discuss how they help the programmer. Encapsulation is not just about data hiding. Classes and objects were sometimes confused, especially in code examples. Polymorphism was poorly described. Some answers described overriding and overloading, but the examples did not show why they would be useful.

In part (c), most candidates could describe what an exception was and give an example of its use in a try-catch block. Very few candidates defined an exception or talk about their relevance to an OO language (such as creating classes of exceptions). Few candidates discussed throwing exceptions but did not demonstrate an understanding how these could be used to handle errors. Often, examples focused on presenting messages to the user or printing information to the console. The discussion of the advantages and disadvantages would be improved by clearly identifying what these are being compared to. For example, what advantages are there to using exceptions compared with an alternative such as an if-then-else construct? Some answers listed generic disadvantages that one might give as an answer for many computing questions (too slow, too much memory, too difficult) without clearly relating these to the question.