BCS THE CHARTERED INSTITUTE FOR IT

BCS Higher Education Qualifications

BCS Level 6 Professional Graduate Diploma in IT

March 2016

EXAMINERS' REPORT

Programming Paradigms

General comments on candidates' performance

There is evidence that the questions covering the use of developer tools and the use of object-oriented technology were the most popular questions and they were reasonably well answered. Once again, the remaining questions, which address topics including functional programming, declarative and logic programming and concurrency, seem less familiar to candidates. These questions were less popular and generally not so well answered, although there were some good answers.

It is important that candidates revise the material stated in the syllabus and learn how to apply it to different situations. Answers that do not address the question will, at best, be very general and not provide an answer with sufficient understanding. At worst, such general answers will fail to address what has been asked.

To be successful on this paper, candidates should ensure that their answers are relevant for the question on the paper. The better answers to this year's papers did demonstrate a deep understanding of the material and clearly showed that understanding by relating it to the question.

The following pages contain pointers about answers for the different questions, together with comments from the examiners.

Question A1

A1. a) Describe the range of tools available to support the programming development process for a team of programmers who work on collaborative projects.

(10 marks)

b) Evaluate the success of these tools in improving the productivity of programmers and the quality of the code they produce. Include examples to back up any points made. **(15 marks)**

Answer pointers

Syllabus Section: Programming Environment

In part (a), a good answer would discuss the sort of tools available for developing a program, from writing the code to testing it. The discussion would include Interactive Development Environments (IDEs) or Interactive Development toolkits that provide a set of tools for writing the code, debugging, and testing it, plus version control systems, automated build systems, continuous integration systems and configuration management utilities.

By mentioning a team of programmers, a better answer would have discussed what aspects of these could be used to support collaborative development.

In part (b), candidates were required to reflect on the real usefulness of these tools. Within the discussion, that candidate should have indicated why they think the tools are important or not. For example, they could argue 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, rather than checking and testing it thoroughly themselves.

For both parts, examples of appropriate tools were expected, for example, Microsoft Visual Studio, Eclipse, Netbeans, IntelliJ, Xcode etc. Candidates needed to identify what elements of the tool are useful to productivity and say why they think these are beneficial or not.

Examiners' comments

This question was attempted by most candidates and a lot of candidates achieved a pass mark.

Most candidates produced good answers for Part (a), with some obtaining full marks. Discussion of Interactive Development Environments (IDEs) was the most popular option, with some opting for configuration management or version control systems. Candidates lost marks by not discussing the tools with respect to teams of programmers working on collaborative projects, which was needed for full marks.

Part (b) produced weaker answers by not addressing the question asked. Some candidates mostly repeated Part (a) again by describing the features of a tool, such as an IDE, rather than evaluating how successful they were in helping a programmer to improve productivity. When evaluating the tool, the candidate should also think of any drawbacks too, as well as the advantages.

Better answers addressed the quality the code produced. A lot of candidates assumed the code generated by a tool would always be 100% correct, which is not always the case.

Question A2

You are the Development Team Manager of a company that produces Timetable software. The development team is currently using procedural technologies and a file based data storage system. It has been decided to move to an object-oriented programming language with a database for data storage.

Write a report to discuss the effect of using an object-oriented programming language in place of a procedural programming language. In your report, identify the concepts to be found in object-oriented languages and discuss the advantages, issues and disadvantages of taking such an approach. Illustrate your answer with appropriate examples. **(25 marks)**

Answer pointers

Syllabus Section: Nature of Programming Languages and Object Orientation

Candidates were required to consider the advantages and disadvantages of using an object-oriented (OO) language such as Java, an OO .Net language or C++. The answer should have included a discussion of the concepts found in OO languages that help the development of a system, for example: inheritance, polymorphism, encapsulation and information hiding.

The candidates needed to reflect on what benefits or problems these concepts bring to program development compared to traditional development. e.g. reuse. Examples from an OO programming language should have been included to illustrate the points made.

The example application developed by the company is database oriented. Therefore, a better answer would have focused on aspects of OO that would help develop such systems, e.g., JDBC classes in Java, or Object-Relational Mapping tools such as JPA.

Examiners' comments

This question was attempted by most candidates and a lot of candidates achieved a pass mark.

There is evidence that most candidates could describe the concepts found in object-oriented programming languages, however, the question also asked for the advantages, issues and disadvantages of using such an approach too, which weaker answers did not include, or they only looked at the advantages.

The background for the report was a company which currently uses procedural technologies and a file based data storage system. To achieve a high mark some consideration was needed of how to use a database for data storage from an object-oriented language.

Examples were also needed to illustrate the answers. Credit was given to candidates who tailored their examples to a timetabling system.

Question B3

a) Explain the terms **pure function** and **referentially transparent** expression. Give an example of each. **(8 marks)**

b) Explain the terms **domain** and **range**, as they pertain to functions in functional programming. Choose a programming language that has a square root function. What is the domain and range of the square root function? Is this a partial function or not? Explain your answer. **(10 marks)**

c) Using a functional language of your choice, write a recursive function **duplicate** which duplicates every item in a list. For example, applying duplicate to the list [3,5,6,8] should give the result [3,3,5,5,6,6,8,8]. **(7 marks)**

Answer pointers

Syllabus Sections: Functional Programming

In part (a), a Pure Function is a function that has no side effects; candidates would also need to explain the term side effects. The outputs of a pure function are determined only by its inputs. Given the same input again it will produce the same output. A referentially transparent expression can be replaced by its value without changing the behaviour of the program. The value of a referentially transparent expression remains the same regardless of context. If an expression contains only pure functions, then it will be referentially transparent. Marks were awarded for any suitable examples.

In part (b), the Domain is the set of values that the function permits as its argument. The Range is the set of values the function produces as a result. The candidates could have chosen any relevant programming language to illustrate their answers. For example, if the candidate had chosen to talk about Haskell, the answer could have discussed the sqrt function. The Haskell sqrt function has the set of positive or zero floating point numbers as its domain, and the same set as its range. Negative input values will give an error rather than a result. Negative output values are not produced (although both 3 and -3 will give 9 when squared, the sqrt function will only return the positive option). sqrt is partial because there are no answers produced for negative inputs. Marks were awarded for any suitable examples.

In part (c), any suitable examples were given credit. For example, Haskell could produce a duplicate function using the following definition.

```
duplicate :: [a] -> [a]
duplicate []     = []
duplicate (x:xs) = x:x:duplicate xs
```

Examiners' comments

Very few candidates answered this question, and those who did, mostly lost marks by leaving large parts of it blank. Very few candidates attempted to write a function in any functional programming language.

Functional programming is an area that candidates should be familiar with from their studies. With more functional languages available and as more programming languages include aspects of functional programming, it is even more relevant to be aware of this programming paradigm. The examiners encourage future candidates to study this area.

Question B4

a) Logic programming languages such as Prolog are said to be declarative. Describe what is meant by the word **declarative** in this context. Describe the advantages and potential disadvantages of declarative programming. **(10 marks)**

b) Show how Prolog determines the truth or falsehood of the goal `celebrity(victoria_beckham).` from the facts and rules in the following declarative program. Briefly describe what is meant by unification and backtracking, and give examples of the unification and backtracking that will occur when solving this goal.

```
celebrity(X) :- famous(X), alive(X).

famous(X) :- footballer(X).
famous(X) :- author(X).
famous(X) :- married_to(X,Y), famous(Y).

married_to(wayne_rooney, coleen_rooney).
married_to(coleen_rooney, wayne_rooney).
married_to(david_beckham, victoria_beckham).
married_to(victoria_beckham, david_beckham).

sister(charlotte_bronte, emily_bronte).
sister(emily_bronte, charlotte_bronte).

author(jk_rowling).
author(charlotte_bronte).
author(emily_bronte).
singer(victoria_beckham).
singer(madonna).
footballer(david_beckham).
footballer(wayne_rooney).
alive(jk_rowling).
alive(david_beckham).
alive(victoria_beckham).
alive(madonna).
alive(wayne_rooney).
alive(coleen_rooney).
```

(15 marks)

Answer pointers

Syllabus section: Logic Programming

In part (a), a declarative language is a language that encourages the programmer to state what they want to produce rather than how they want to produce it. A discussion of advantages such as the following: A statement of what is wanted rather than how to achieve it will enable programmers to think at a higher level about the problem they are solving, rather than become lost in details. It can make code easier to read and debug. It can allow a good optimising compiler to generate potentially more efficient code than the programmer would have created. Potential disadvantages: the programmer has less control over efficiency of the solution. Sometimes, coding a more efficient solution may require the use of non-declarative aspects, such as the cut operator, or may require a more complex statement of the problem. All relevant answers were considered.

In part (b), to solve the goal `celebrity(victoria_beckham)`, Prolog will attempt to find this fact in its database, either as a ground fact or as a result of application of the rules. The facts and rules will be tried in the order that they are presented in the program.

The first rule will fire, since the head of the rule, `celebrity(X)`, unifies with the goal `celebrity(victoria_beckham)`. Unification is the process where variables within a term are substituted by variables or atoms in order to make two terms match each other. In this case, `victoria_beckham` unifies with the variable `X`. The body of the rule under this substitution then produces the two subgoals `famous(victoria_beckham)`, `alive(victoria_beckham)`. Both of these subgoals need to be found to be true for the original goal to be declared true. We first explore the first subgoal, `famous(victoria_beckham)`.

Three rules describing `famous(X)` occur next in the program. The first of these will fire, because the `X` in the head of the rule can unify with `victoria_beckham`, but the body of the rule will produce the subgoal `footballer(victoria_beckham)`. This cannot be satisfied by the definitions of footballer in this database (which are all simple ground facts). So backtracking will occur, that is, the variable unification will be undone, and the next matching rule will be applied instead.

The next case for `famous(X)` will fail similarly. After backtracking again, the final rule will produce the two subgoals `married(victoria_beckham,Y)`, `famous(Y)`. The subgoal `married(victoria_beckham,Y)` will succeed with `Y` unifying with `david_beckham`. This causes the second subgoal to become `famous(david_beckham)`.

Now we need to apply the rules for `famous(X)` again, where `X` unifies with `david_beckham`. The first rule's body produces the subgoal `footballer(david_beckham)`, which finally succeeds because it is present as a ground fact in the database. So this means that the subgoal `famous(victoria_beckham)` is now satisfied.

The remaining subgoal `alive(victoria_beckham)` is then trivially satisfied because it is present as a ground fact in the database. So, as all subgoals are satisfied, the overall goal `celebrity(victoria_beckham)` is satisfied.

Examiners' comments

This question was attempted by a reasonable number of candidates, although only a low number achieved a pass mark.

For part (a), the evidence shows that many candidates gave a description of Prolog or of logic programming instead of giving a description of declarative programming. Some did give wider answers that also referred to other languages for declarative programming such as functional languages or the environments used in spreadsheets such as Excel, which was good to see. However, some candidates wrote very long descriptions without using the word 'declarative'. Candidates should ensure that they read the question carefully.

Higher marks were awarded to candidates that said that it was a style that allowed the programmer to state what they wanted rather than how they wanted it done, but those answers still needed more explanation or some examples to illustrate that difference.

Some candidates contrasted declarative programming with imperative programming, which was good to see. We note that Lisp is not a great choice as an example of a declarative language, as it allows imperative programming - there are other better choices of language to illustrate this. Advantages and disadvantages were generally poorly written. Most candidates fell back on saying that one paradigm or the other was 'difficult to learn' or 'more efficient' without presenting any examples or evidence.

For part (b), candidates who drew diagrams of the order in which goals and subgoals were to be tested were mostly correct, though many omitted steps. Some candidates mistakenly thought that the correct facts would just be selected from the database without testing the others first. The process should be driven from the goal. Many candidates were unclear on the meaning of 'unification' and 'backtracking'.

Question B5

The Dining Philosophers problem describes an issue found in concurrent processes. The problem is as follows.

Five philosophers sit around a round table for dinner. They are each served a bowl of spaghetti. The spaghetti is slippery and a hungry philosopher will need 2 forks to be able to eat it, one in each hand. If a philosopher has just one fork then that philosopher cannot eat. Philosophers must think and eat, alternately. When they are ready to eat they will try to pick up two of the forks. After eating, a philosopher will put down his or her forks so that other philosophers can use them. There are just five forks on the table.

a) What are the problems with this scenario and how do they correspond to problems raised by concurrent processes? **(13 marks)**

b) Explain three of the methods that are available to help solve the problems that you identified in part a)? **(12 marks)**

Answer pointers

Syllabus section: Related Issues

In part (a), candidates needed to explain that the philosophers could deadlock, with each taking one fork, or they could have a situation where some philosophers do all the eating and others are too slow to get the forks and therefore starve. Concurrent processes that need to access limited shared resources have the same problems. Some candidates might have given an example of this, such as files, databases, shared memory, etc.

In part (b), candidates needed to explain that the philosophers could deadlock, with each taking one fork, or they could have a situation where some philosophers do all the eating and others are too slow to get the forks and therefore starve. There are various solutions or partial solutions, including having a waiter who decides whose turn it is to have the forks, or a queue to get to the table, or take a numbered ticket when they want to eat (such as at the delicatessen counter), or by giving an ordering to the forks, so that philosophers have to pick up a higher numbered fork before a lower numbered fork, thereby ensuring that diners who could only pick up one fork are not allowed to do so. They might discuss timeouts on resources (which could lead to livelock). They might discuss why various approaches still do not work for reasons of fairness, deadlock or livelock.

Examiners' comments

This was attempted by more than half of the candidates, although the overall pass rate was low.

In part (a) candidates were asked to state the problems that the diners would have and also how the problems correspond to problems in concurrent processes. Both of these areas need answering, whereas there is evidence that many candidates only discussed the diners. Some candidates explored all options (all philosophers picking up one fork simultaneously, some picking up 2 before others had one, etc.), whereas some candidates limited themselves to only one scenario. Candidates could discuss deadlock, livelock, fairness and starvation. Some thought it would be fine if a philosopher could eat until full and the others would have to wait

their turn, without saying how turns would be managed, how long this would take and whether this would be a problem for the waiting diners. Many candidates did not recognise the problems in having to acquire 2 resources (both forks, or a database writer connection and a file writer handle) and the problems arising from acquiring and holding one but not the other.

In general, fuller answers, and more examples of concurrent systems that could show these problems were required.

In part (b), candidates mostly limited themselves to describing mutexes and semaphores as low level concurrency tools without explaining whether or how they could be used to solve this problem.