# BCS THE CHARTERED INSTITUTE FOR IT

## BCS Higher Education Qualifications
## BCS Level 6 Professional Graduate Diploma in IT

**March 2017**

# EXAMINERS' REPORT

### Programming Paradigms

## General comments on candidates' performance

There has been an increase in the pass rate this year, in line with strong answers in the topics of developer tools and object-oriented technology.

The evidence suggests that questions that look at other topics are often not answered as well, although there were some good answers. This illustrates that students would benefit from expanding the range of material studied in preparation for the course, looking to cover the full range of the syllabus.

There are also some examples where students were not able to provide examples in certain languages. For the Programming Paradigms paper, it is a requirement for students to be proficient in at least one language and have had the opportunity to learn and write some example programs in relevant languages for the other paradigms.

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

### Section A

A1. a) Explain the meaning of the terms **compiled** languages and **interpreted** languages.
**(7 marks)**

b) Discuss how the following tools can support the different stages of the software development lifecycle. For each stage, supply at least ONE real-world example.

i) Interactive Development Environments (IDEs)

ii) Testing and Debugging Tools

iii) Configuration and Revision Control Tools

**(3 x 6 marks)**

**Answer Pointers**

**Part a)**

Basic: Illustrates a core description of the process to translate a set of source files into a machine-readable version that can be executed. Compilation does this once and can be run multiple times. Interpretation is where this is done each time the code is executed.

Issues include: The concept of transforming source code written in a high-level programming language (the source language) into another computer language (the target language) usually having a binary form known as an executable program. Compilers

versus Interpreters – where the term compiler means a program that produces a separate executable from the compiler (which may require a run time library). By contrast, an interpreter executes the program directly, translating each statement into a sequence of one or more subroutines already compiled into machine code – it does not generate a permanent separate executable file. Compilation does this once and can be run multiple times. Interpretation is where this is done each time the code is executed. Comments regarding the various stages of translation: lexical analysis, parsing, semantic analysis, code generation and code optimization should be credited. Better answers might talk about languages where there is a mixture of compilation and interpretation, e.g. Java.

**Part b)**

**Interactive Development Environments**

Issues include: an IDE provides a set of tools that can work together to make it easier to develop software solutions. There will be editors, which would typically provide syntax highlighting for the supported languages and 'intellisense' operations to make it easier to find the relevant API elements. There will be support to create project templates to aid starting with certain types of projects, e.g. desktop or server projects. There will be functions to execute the projects, which could involve a compilation step. There will be functions for interactive debugging sessions and the ability to run test suites. The tools will help simply the task of editing, executing, debugging, testing and deploying code. Other points that emphasise the ability for different tools to work together in this environment will be given appropriate credit.

Examples include VisualStudio, Eclipse, IntelliJ, Xcode.

**Software Testing and Debugging Tools**

Issues include: Testing tools are used to help developers create tests and run tests to reveal deficiencies in the system. There can be tools to support static analysis of the code as well as tools to support dynamic analysis by running test cases. Tools such as unit test tools provide a mechanism to define a set of tests and then run these tests. The results are presented, identifying tests that passed and tests that failed. Tools can script other aspects of system development, e.g. testing the user interface or the information generated by web applications. Testing tools will help to simplify the process of developing a repeatable set of tests.

Debugging tools provide a way to interactively step through the execution of a system. The software developer will be able to execute multiple statements or step through one statement at a time. It will be possible to set breakpoints so that the debugger will pause and allow inspection of the code and data at the breakpoint. These tools will help the developers to explore the system as it runs, investigate problems and identify the parts of the system that are the source of the problems.

Other relevant discussion will be given appropriate credit.

Examples include JUnit, Selenium, GDB, LLDB, Microsoft Visual Studio Debugger.

**Software Configuration and Revision Control Tools**

Issues include: Source code management, version control, documentation management, issuing patches and updates etc. Examples include Git - a free and open source distributed version control system, CVS (Concurrent Versions System) and IBM's Rational ClearCase. There are many more. Tools are available to help developers manage the complex configuration issues and code version issues. The aim is to provide ways to track versions that have been created, access different versions of the system and documentation and all resources necessary to rebuild the versions. The tools allow teams to work on the system in parallel and offer ways to merge the code and resolve conflicts that arise from merging.

More advanced answers might comment on some features in examples or comment the ability of teams to work on different branches.

**Syllabus Coverage**: Programming Environments, 2.1, 2.2, 2.4,2.5, 2.6

**Examiners' Comments**

The question was answered well overall, showing a good appreciation of this part of the syllabus. Most candidates answered all parts of the question, addressing the several issues. There are no significant issues to report. Where the answers failed to achieve a pass mark, it was clear that the candidate was not well prepared for this section of the syllabus.

A2. Imagine that you are an IT manager looking to recruit a new software developer and a new database administrator (DBA). As part of the recruitment process, applicants are asked a series of technical questions. Two sample questions are listed below – the first aimed at the developer role, the second at the DBA position. Imagine you have applied for both jobs and answer both questions.

**a)** What is a **scripting language** and where, when and why would it be used? Give specific examples of scripting languages. Support your answer with specific code examples. **(10 marks)**

**b)** Discuss the key differences (in terms of fundamental concepts and programming constructs) between a purely **data-oriented language** like SQL and **procedural languages** that are used with databases – such as Oracle's PL/SQL or Microsoft's T-SQL. Support your answer with specific code examples. **(15 marks)**

**Answer Pointers**

**Part a)**

Answers should show an understanding that a script is the automation of a set of tasks that would normally be manually entered. The essence is that a batch or set of commands can be packaged-up into a 'script' and executed automatically just as if entered via the keyboard by a human. At least one example scripting language should be discussed and ideally a small amount of code provided. A scripting language controls the operation of a normally interactive (user-driven) program, giving it a sequence of work to do all in one batch. For instance, we could put a series of editing commands in a file and tell an editor to run that "script" as if those commands had been typed interactively. Similarly, with 'SQL scripts' where the DBMS executes the statements just as if a human had typed them in. Another aspect of scripts is acting as the 'glue' that connects software components. Examples could include UNIX pipelines and in web development where scripting can link the back-end database to the web server (server-side scripting). Scripts can also be embedded in a larger host software component – like a webpage – to add functionality and data validation features (client-side scripting). Scripts are normally interpreted rather than compiled. Examples include JavaScript, PHP, Perl and Python. There are many others such as Adobe ActionScript and Microsoft VBScript. Any suitable code example, from any language will suffice. More advanced answers might briefly comment on a comparison of scripting languages and 'full' programming languages such as VBScript and VB.

**Syllabus Coverage:** Nature of Programming Languages, 1.2 Scripting Languages.

**Part b)**

Answers should show an understanding that SQL is not a 'full' programming language and does not have all normal programming constructs and is declarative. In comparison, languages such as PL/SQL and T-SQL do have programming constructs and are

imperative. SQL is a fourth-generation language (4GL) and declarative in nature ('this is what I want') rather than the traditional third-generation (3GL) and imperative paradigm ('this is how to do it') employed by PL/SQL and T-SQL. SQL is based on set-theory and relational algebra (SELECTION, PROJECTION, JOIN etc.) and its sole job is to manipulate relational tables in a database. It has none of the mainstream programming constructs such as IF THEN ELSE or LOOPS etc. For that reason, it tends to be used within a larger 'host' language or environment – like a webpage to query a back-end database. By contrast, PL/SQL and T-SQL are full programming languages with all the normal programming constructs and modularization features of functions, procedures and packages plus they support database-specific objects like triggers and cursors. Any suitable code examples will suffice.

**Syllabus Coverage:** Nature of Programming Languages, 1.3 Data-Oriented Languages.

**Examiners' Comments**

This question was attempted by fewer students and overall the answers offered limited detail.

The evidence suggests that answers for part a) demonstrated an understanding of scripting languages although some answers would be improved by more discussion on when they are used.

Part b) was sometimes missed out. Where part b) was attempted, the answer focused more on the SQL aspect. This suggests an opportunity to look further at data-oriented languages such as SQL and procedural languages used by databases such as T-SQL and P-SQL, understanding their differences from a programming paradigms perspective. Some candidates did talk about procedural code, such as PHP, that links to the database. Whilst not the intended area for the question, some credit was given where appropriate.

A3. 'Legacy Systems Incorporated' have been known as industry leaders for software development. This reputation was based on the use of procedural languages and relational databases. The company's clients are now demanding that future projects be developed within the 'object orientation' paradigm. Convince the CEO of the company that making the switch to a new paradigm is worthwhile by answering the following questions.

a) Briefly discuss the key features and differences of **procedural** languages and **object-oriented** languages. **(10 marks)**

b) Discuss each of the following features typically found in object-oriented languages. For each feature, using your own suitable examples, outline how it could help the company to build software systems.

i) Inheritance

ii) Encapsulation and Information Hiding

iii) Polymorphism

**(3 x 5 marks)**

**Answer Pointers**

**Part a)**

Answers should demonstrate knowledge of SSI concepts – sequence, selection and iteration – as the key procedural building blocks, while OO languages are based around the class/object concept utilizing encapsulation, inheritance, polymorphism etc.

The answer should be able to compare these different approaches – where procedural languages pass data items to be processed by different functions and procedures. In comparison, OO combines the idea of data and the valid operations on that data. There would be discussion of how these respective techniques work at a general level.

Examples might be included and will be judged on relevance to covering key features and differences.

More advanced answers might draw a distinction between the general type (class) and specific instantiations of that class (objects). Consideration might be given on whether such type organisation is used in procedural approaches.

**Syllabus Coverage:** Object Orientation, 3.1 Basic concepts.

**Part b)**
Marked holistically but the key issues/concepts include:

i) Inheritance: The concepts of parent (super) classes and child (sub) classes, the implicit idea of hierarchies of classes, the differences between single, multiple and multi-level inheritance, the concept of extending the sub-class to produce a range of unique child classes, all stemming from a common ancestor – just as in real families. More advanced answers might briefly discuss issues such as access modifiers, e.g. private, and their use with inheritance. Benefits company through code reuse.

ii) Encapsulation and Information Hiding: The core idea of 'binding' or 'merging' data (attributes) and logic (methods) that act upon those attributes together into a single, well-defined and named entity called a class – which acts a template or type for specific examples (instantiations) of that class. Provides the concept of 'information hiding' – using the visibility ideas of public, protected and private. The key role of abstraction in the sense that what happens inside the class is hidden from the external world. A class is defined by what it does, not how it does it. The inner workings are hidden. More advanced answers might expand on some issues, such as the detail for visibility ideas. Benefits company by having code that manages part of the system. It logically links the data and operations, which can help with maintaining the code. Information hiding enables future change of inner workings without affecting other code in the system.

iii) Polymorphism: The essence of 'having many forms' - the ability to choose the exact called-function, depending on the current context. The different types of polymorphism should be described briefly – parametric, sub-typed or ad-hoc. The ability to specify an action at an abstract level – such as 'go to the dealer and buy a new car'. Each object (class instantiation) then provides its own concrete way of *how* to implement that request (implementation) plus allowing for variation in exactly *what* it's going to do (behaviour) – such as using a different car dealership, buying different models of cars or paying by different mechanisms. For this scenario, you might have a PERSON base class that defines methods like 'Go To Dealership', 'Select Best Car' and 'Buy Car'. You could then have implementations of PERSON called TEENAGER, FAMILY and PENSIONER - each of which will define its own method of going to the dealership (walk, cycle, drive, bus etc.), selecting the best car (Convertible, People Carrier or Compact) and paying for the car (Loan, Cheque or Cash). Those methods would be polymorphic, because each implementation would have its own way of carrying out the higher-level (abstract) command. More advanced answers might address issues such as static versus dynamic

polymorphism – the precise time when a specific implementation object is selected - at compile time (static) or at run-time (dynamically). Benefits company by having a flexible way to pass data to methods, to have different versions of methods (overloading) and to process similar items that are liked by a common higher-level definition. Can lead to less code duplication.

**Syllabus Coverage**: Object Orientation, 3.1 Basic concepts.

**Examiners' Comments**

A popular question that was attempted by most students. Overall, the answers showed good knowledge of the different issues.

The evidence suggests that some answers for part b) were general explanations that answered some of the question but missed out on the wider discussion. Such answers would have benefited from discussion on how the features were relevant for the company.

## Section B

B4. This question is about **functional programming**.

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

**b)** Explain the term **recursive function**. Explain the main similarities and differences between recursion and iteration. **(7 Marks)**

**c)** Using a functional language of your choice, write a recursive function **insert** which should take two arguments, an item and a sorted list of items, and inserts the item into the correct location in the sorted list. For example, `insert 7 [3,5,6,8]` should give the result `[3,5,6,7,8]`. **(10 marks)**

**Answer Pointers**

**Part a)**

A pure function is a function that has no side effects (candidates will need to explain the term side effects). Its outputs 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 will be awarded for any suitable examples.

A more advanced answer would select examples that demonstrate side effects, purity and the value of an expression. Further, advanced answers will be clear in the use of terminology (expression, function, side effects, etc.).

**Syllabus Coverage:** Functional Programming, 4.3 Referential Transparency

**Part b)**

A recursive function is a function that calls itself. Recursion and iteration are equally expressive and can both solve the same problems. A recursive solution has an equivalent iterative solution and vice versa. In some languages (e.g. C and Java) a recursive solution can be slower because of the overhead of calling functions, and a recursive solution can

also use more stack space. In functional languages, this isn't the case because function calls are cheap and good compilers will optimize the recursion to be tail-recursive to avoid stack issues. A recursive function will often avoid the need for variable initialization and reassignment, which is common in iterative solutions. Other relevant points of distinction will be awarded marks also.

More advanced answers will be clear about what both iteration and recursion have in common, as well as their differences. They may go into detail about efficiency, or give examples of languages and scenarios that favour the different approaches, and they may indicate that they understand the choices can be based on language, speed, memory, readability, maintainability and more.

**Syllabus Coverage:** Functional Programming, 4.2 Recursion

**Part b)**
An example in Haskell is:

```
insert :: a -> [a] -> [a]

insert x []     = [x]

insert x (y:ys) = if x < y then x:y:ys else y:insert x ys
```

**Examiners' Comments**
Of the candidates that attempted this question, there were some good answers, but many answers demonstrated limited understanding of this programming paradigm. Few candidates attempted to write a function in any functional language, and some tried to use a procedural language such as C to illustrate the issue.

The evidence suggests that some candidates were unclear about programming language, for example some stated they were providing code in Python but then provided code that was more like C, C# or Java.

Candidates and centres are encouraged to look at functional programming in more detail and where possible try out some of the basic concepts as part of studying for this module. here are more functional languages available, e.g. Haskell and F#, and other programming languages are including aspects of functional programming.

B5. This question is about **distributed systems**.

**a)** What is meant by a **distributed system**? In your answer, describe communication, memory, coordination and failure. **(15 marks)**

**b)** A massively multiplayer online game is one example of a distributed system. Two of the algorithmic challenges inherent in distributed systems such as this are clock synchronization and mutual exclusion. Describe what these problems are and how they may be solved in a distributed system such as a massively multiplayer online game. **(10 marks)**

**Answer Pointers**

**Part a)**

A distributed system is a collection of computers that acts as a single system. Examples include the Internet, a company intranet, load balancing web servers, music file sharing services, massive multi-player online gaming, telephone networks and High Performance Compute clusters in science. Computers/nodes in a distributed system do not in general share memory and cannot access the memory of other nodes. Instead they communicate via message passing.

There are various ways of coordinating distributed systems, including having a central coordinator, multiple coordinators, or no coordinators. A central coordinator distributes the jobs among the nodes and sends and receives messages from each of the nodes (in this scenario, the individual compute nodes may message each other, or may just message only the central coordinator). If the distributed system has a central coordinator then this is a single point of failure, which can bring the whole system down. Decentralised or P2P systems don't have a single point of failure, but may take a long time and many messages to determine which node has the information that is needed by another node, if indeed it is every determined at all.

**Syllabus Coverage:** Related Issues, 6.3 Distribution

**Part b)**

Clock synchronization is the issue of ensuring all physical clocks involved in the system (on users gaming machines and on any servers involved in the game) are within a reasonable tolerance of each other. Physical clocks diverge over time because of hardware issues. The time at which a player fires a shot, arrives at a goal, opens a door etc. needs to be understood by the other players in the game in order that the consequences of the actions can be determined. Common solutions include frequently updating the time after requesting it from a central time server, using NTP, or having one server fetch the time from all distributed processors and averaging those times to report back the adjustment that each processor needs to make. Mutual exclusion is the issue of coordinating access to critical resources so that only one process or player can alter or use the resource at any one time, for example if two players try to pick up the same object at the same time. Not managing mutual exclusion can lead to race conditions. Solutions include the use of semaphores, wait-free algorithms, etc.

A more advanced answer would clearly describe the problems, describe multiple solutions and give examples to demonstrate how the problems relate to the scenario presented in the question.

**Syllabus Coverage**: Related Issues, 6.3 Distribution

**Examiners' Comments**

Of the candidates that attempted this question, there were some good answers but many answers demonstrated limited understanding of distributed systems. Some answers were too general and tried to focus on general concepts of concurrent programming and not the specific topic of distribution across machines. Whilst some ideas from concurrency are relevant, some answers talked about shared memory solutions between processes on the same machine, but that would not be viable for systems distributed across different machines. Generally, answers could more specifically address the scenario and show how the issues apply rather than relying on general answers.