THE BCS PROFESSIONAL EXAMINATIONS
Diploma

April 2006

EXAMINERS' REPORT

Software Engineering 1

General Comments

Most of the scripts produced by candidates this year were well structured and readable, showing advancing knowledge beyond the certificate level and a good grasp of higher essentials necessary for the Graduate Diploma examination. However, in some centres the general standard of the written work, and the level of comprehension (of the question set as well as the subject matter), needs particular attention.

Above all, this is a technical paper about Software Engineering. Questions seek to test candidates' knowledge and ability to apply that knowledge. It is a poor answer that simply describes recalled information. It is a good answer that can then show application of the recalled knowledge.

Time management is an issue that still challenges many candidates. Questions are framed with an allocation of marks; an answer that provides 6 pages for 9 marks and two paragraphs for 16 marks makes poor use of time. Equally, answering more questions that required signals either lack of preparation or inability to count; both are issues of concern. In a few instances, the issue still persists in which some candidates attempt to recognise one or two keywords without regard for the context and scenario in which the question was presented. Candidates should make themselves familiar with the format of previous question papers, the better to recognise and be successful with the test paper.

Finally, similar to the previous year, some candidates had not completed the front cover with the number of the question and part-questions attempted. This made it very difficult to find all the parts of an answer, then determine whether it was a continuation of a previous answer, and assess and record the mark for that answer as a whole.

Question 1

1. *a)* Discuss <u>at least</u> FOUR key roles that must be undertaken by staff within a software engineering project team. **(12 marks)**

b) Outline FOUR factors that can influence communication within a software engineering project team. **(8 marks)**

c) In his book, *The Mythical Man-Month*, Fredrick P. Brooks, Jr. stated Brooks's Law: "Adding manpower to a late project makes it later". Discuss the evidence for and against Brooks's Law. **(5 marks)**

Answer Pointers

a)
Expecting any four of the following project leader/project manager; software librarian, chief/lead software engineer, software tester, software architect, software toolsmith, and related discussion of role with the team. (4x3=12 Marks)

b)
Expecting any 4 of the following with suitable outline:

- size of group and subsequent communication overhead;
- number of modules under development and technical communication to support interfacing these;
- group structure - problems with rigid, hierarchical groups for technical working;

- the physical work environment - office layout and adequate meeting rooms;
- group composition - personality problems and balance of roles within the group;
- communication channels available especially if the group is distributed geographically.

(4x2=8 Marks)

c)
Expecting the answers to highlight the problems of bringing new people up to speed before they can usefully contribute to the work in hand; the additional communication overhead that new people will bring; that adding new people to a poorly managed project if they are too low level will not address the problem of poor management, adding manpower and retaining a poor design, poor architect and module breakdown will not address the core problem. Also based on Brooks OS/360 experience and subsequently validated by other studies and also reflected in Boehm's COCOMO model where project duration is a non-linear function of effort.

Against where the effort allocated is such that the project will never be finished, e.g. inadequately qualified staff, then adding effort with an appropriate skillset may be the only way to ensure progress towards completion. (5 Marks)

Examiner's Guidance Notes
This question probes students' understanding of the work and working of a software engineering team by getting them to discuss what people actually do within a team.

Communication is obviously key to ensuring that a group of people can work as team; and in software development communication amongst the various developers and the project leader/manager is critical to the success of the project.

Brook's insights into the nature of teams and problems introduced by adding members to the team of a late project are often quoted; here students were given an opportunity to critically appraise Brooks's Law.

Question 2

2. The outline of a project you have been asked to develop is given below:

> **Enquiries and Orders Information Management System**
> A manufacturing company, ABC Composites, is the client.
> The client wants development of a new management information system (MIS) to enable multi-user data entry and querying. It is expected that the system will encompass the following areas:
>
> - User-friendly interface
> - Data entry validation
> - Defined reports, especially job control sheets, which the client already uses on paper
> - Querying of data to produce reports
>
> The client has a SQLserver system that should be the database and will support intranet access from several desks.

The development models suggested for this project are

- **Incremental Development** using a series of prototypes and
- **Systematic, sequential Development** with progression through analysis, design, coding, testing and maintenance.

Compare the advantages and disadvantages of these two life cycle models for this project. Give <u>at least</u> THREE examples to support your analysis of each life cycle model. **(25 marks)**

Answer Pointers

A good answer will be balanced, with arguments on both sides and a conclusion.

Incremental development offers a way to determine unclear requirements, tackle complex developments and anticipate difficult interfaces. Incremental development suggests build as a series of prototyping developments, where each cycle might involve considerable reworking of previous developments at the cost of developer and client time, and client cost. This project is mildly undefined – a 'user friendly interface' and not really difficult, and 'pre-defined reports' is, again, a matter of analysis not exploration. The extra time spent developing prototypes seems not valuable. (10 Marks)

Sequential development, also called Waterfall development, reckons that the project can be achieved in a single pass. It requires firm and stable requirements, known tools and unexciting interfaces. This project has most of these characteristics. (10 Marks)

On balance, with sample arguments as above, a sequential development would achieve this project in short time with good results. (5 Marks)

Examiner's Guidance Notes
This question is set to test candidates' powers of analysis and reasoning. The aim is to analyse the scenario and reason about its implications for life cycle management. Some answers gave very guarded conclusions, saying that both types of suggested lifecycles would be used. Some answers gave extensive descriptions of lifecycles but did not relate their features to any characteristic of the scenario given in the question. Good answers related aspects of the scenario to claimed benefits of each life cycle model, and came to a reasoned conclusion and a recommendation.

Question 3

3. *a)* Give THREE reasons for the use of CASE tools to support the work in a large software project. **(9 marks)**

b) Your company has agreed to develop a large software system using Object-Oriented methods. Choose and describe THREE capabilities or functionalities you would want in a CASE tool to support this O-O development. Give your reasons for each choice. **(16 marks)**

Answer Pointers
A good answer covers the following points.

a) The main reason for using CASE tools is Productivity, and this can be expressed in a variety of software-specific ways
- better communication (including database repository of work, requirements, tests, test results, configuration status, etc.)
- better work rate (including lower costs and fewer people),
- higher quality (including fewer errors, easier user validation, and easier iteration to converge on requirements).9 MARKS

b) Three capabilities/functionalities for O-O development: from -
Diagramming capability – Use cases, Object class/messaging diagrams.
Repository functionality – storing diagrams, code, test cases etc. in a recoverable and reusable way.
Test case generation from object descriptions.
Requirements tracing reporting from use-case descriptions
Configuration control of build states.
Metric analysis for project management Etc.

Any three with supporting rationale at 5 marks each. Extra point for holistic answers. (16 Marks)

Examiner's Guidance Notes
This question is designed to test candidates' knowledge of CASE tools and their use. A good answer to Part (a) described why CASE tools are used, and a good answer to Part (b) related the O-O methods of the scenario to functions and features of a supportive tool. In each case, marks are given to answers that support their arguments with reasons.

Some answers described what CASE tools were without identifying why they should be used. Some answers identified features of a CASE tool without relating these features to the O-O scenario in the question.

Question 4

4. Some definitions of quality are given below:

- Quality means "to make without any errors'';
- Quality means "to make fit for the client's purpose";
- Quality means "to make each product statistically the same as the previous product".

Which view, or views, of quality do you think best address the development of software? Give your reasons for accepting, rejecting or modifying each view. **(25 marks)**

Answer Pointers

A good answer will give a discussion of each definition, along these lines.

Absence of errors is an attractive goal for software engineers, but has never been achieved because of software complexity. (6 Marks)

Fitness for a client's purpose is also attractive but it's not simply user requirements that determine purpose; other purposes introduced by software engineers are e.g. reliability and maintainability. (6 Marks)

Statistical identity with 'other products' is not useful for software projects, because the making of 'products' involves digital cloning that ensures no statistical variation at all, or if the candidate interprets 'products' to be separate projects where each is different from each other but yet addresses a similar requirement, this only makes commercial sense if for some sort of experiment (or the Space Shuttle or Airbus, where reliability is a primary requirement and dissimilar builds avoid common-mode failure). (6 Marks)

Hence conclusion that most useful definition is Fitness for Purpose, with rider that this must include the several stakeholders with purposeful intentions; users, development engineers and maintainers. (7 Marks)

A similar answer, balanced with a reasoned conclusion, is acceptable.

Examiner's Guidance Notes
This question is set to test the candidates' understandings of software quality. In order to limit a possibly wide-ranging answer, three specific definitions are given. A good answer considers the relevance of each definition to software quality. Candidates are expected to factor the 25 points over the implied 7 parts of an answer; three sets of for-and-against; and one conclusion.

Some answers decided the preferred definition and concentrated only on that one without giving reasons for discounting the others.

Question 5

5. *a)* Define what is meant by *black box testing* and *white box testing*. **(4 marks)**

b) Discuss TWO examples of a black box testing technique and TWO examples of a white box testing technique. **(6 marks)**

c) How is software documentation used in the testing processes of unit testing, integration testing, system testing, and user acceptance testing? In each case, explain which software documents are most relevant. **(10 marks)**

d) Explain the importance of the practice of regression testing during software maintenance. **(5 marks)**

Answer Pointers

Black box testing is a test design method which focuses on the functional requirements of the software without regard to its internal logical structure. It treats the software under test as a black box whose contents are unknown.
White box testing is a test case design method which uses the control structures of the software source code to derive test cases. It examines the logical paths through a program.
(2x2 = 4 Marks)

Discuss of any of the following examples:
Black box techniques: equivalence partitioning, boundary value analysis, cause-effect graphing; comparison testing (2x3 = 6 Marks)

Unit testing involves the testing individual units of source code, e.g. functions, procedures, classes, and it is primarily supported by the source code and detailed design documents.
Integration testing involves the testing of two or more units which have a common linkage usually through a well defined interface; and it is primarily supported by design documentation.
System testing involves testing of the software in combination with other system elements such as hardware, databases, people, etc and is supported by the system documentation including the initial customer requirements; it is often coupled with user acceptance testing which involves the final testing of software in the customer's installation before it is finally accepted and put into use. The most relevant documentation here is the customer requirements and any contracts made regarding the software to be delivered. (5x2 = 10 Marks)

Regression testing is important during maintenance for the following reasons:
Regression testing is the systematic re-testing of software after changes have been made. Its primary purpose is to ensure that after changes have been made to the software, no new errors have been introduced.
Regression testing requires the systematic recording of previous testing and its practice ensures that structured maintenance is being carried.
As the previous tests are re-used in regression testing, it saves the maintenance team the effort of recreating any tests. (5 Marks)

Question 6

6. An often quoted software design principle is "aim for low coupling and high cohesion".

a) Define what is meant by software coupling and by software cohesion. **(4 marks)**

b) Give examples of both of these with respect to the design of component-based software system. **(6 marks)**

c) Explain the rationale for this principle and describe at least THREE benefits that follow from its practice. **(10 marks)**

d) Explain how refactoring of software can be used to improve its cohesion. **(5 marks)**

Answer Pointers
Examples of cohesion: functional (high), sequential, communicational, procedural, temporal, logical and coincidental
Examples of coupling: data coupling, control coupling, common coupling via global data, content coupling
(suitable examples of any of the above in the context of a component based software system).
(2 x 3 = 6 Marks)

Expecting a short essay type answer which covers the following:
The rationale is that functional independence is a key to good design and design is a key to software quality.

Benefits are that independent modules are easier to maintain, reusable, can be developed independently by individual engineers within a software team, result in improved program comprehension with supporting discussion. (10 Marks)

Refactoring is a technique practised in Object Oriented software development, particularly amongst those following the extreme programming and agile methods of development. It involves examination of previously developed classes and the identification of any common aspects of these which can be factored out and implemented independently thus simplifying the original classes and making them more cohesive. (5 Marks)

Examiner's Guidance Notes
This question aims to determine students' understanding of a key design principle by getting them to define the key concepts and provide illustrative examples.

Students' deeper understanding is probed by part c) of the question; and finally students are asked to relate a practice that is common in extreme and agile programming, refactoring, to the improvement of cohesion.