THE BCS PROFESSIONAL EXAMINATION
BCS Level 6 Professional Graduate Diploma in IT
September 2017

EXAMINERS' REPORT

Software Engineering 2

General Comments

The pass rate was 40% representing the lowest mark in the last 5 sittings. Future candidates would benefit from carrying out the following:

1. Coverage of the syllabus. The evidence from candidate responses to questions around topics such as Software Architecture, Open Source Software Engineering, and Requirements Engineering need to be revised. It is important that candidates have access to a broad range of resources to add breadth and depth to their knowledge at the point of delivery.

2. Software Engineering 1 is not a pre-requisite for this unit. Centres should, however, encourage candidates to adopt a staged approached to their learning, by completing the content of the Software Engineering 1 syllabus, before attempting the Software Engineering 2 paper. This is of particular concern, as much foundational knowledge would seem to be missing, resulting in candidates using common sense interpretation as a substitute for depth in basic principles and concepts of software engineering.

3. Examination techniques. Candidates need to bear in mind when answering parts of questions to show regard for their indicative weightings, resulting appropriate details being provided for relevant parts of the questions.

4. Presentation. It is important that candidate responses to questions are legible, well-structured and formatted. Further, if the question asks for a report, the response should be structured as a report.

a) Explain what is meant by software product complexity, and demonstrate how measures of module coupling, cohesion, and size can help the engineer monitor the build quality of software.

(10 marks)

b) Functionality is one of the most important software quality characteristics.
It has been suggested that the functionality is influenced by the following software quality sub-characteristics: suitability, accuracy, interoperability and security. Justify this claim.

(10 marks)

c) Which of the following software quality characteristics are 'easier' to measure:

- Efficiency or functionality,
- Efficiency or reliability?

Justify your answers.

(5 marks)

Answer Pointers

a) A good answer should recognize and explain complexity at different levels in terms of structure (e.g. graphs, trees, lists), cognition (e.g. abstraction, induction, recursive processes), and magnitude (scale and size beyond what is accepted as manageable).

Coupling is concerned with the degree of connectivity between the modules of a system. Typically, with good engineering practice most modules are loosely coupled via their interfaces such that there are no side effects resulting from changes in implementation. Nevertheless, many modules are coupled via their parameter list, and the greater the number of parameters exchanged the more likely that errors will arise from missing or incorrect type parameters;

Cohesion is concerned with the internal strength where each module exhibits the single responsibility principle. For example, using a given value parameter, each statement will act upon the intermediate result of the previous statement until the result is produced. Thus, too many parameters would indicate that the module is in danger of violating the single responsibility principle, resulting in the likelihood of performance inefficiencies, and defects when change is affected.

Module size is concerned with the lines of code, internal data structures, and components. For the engineer the optimal module size can get the best performance from the target platform. However, that same value for the maintenance team may be indicating poor comprehension or understandability.

(10 marks)

b) Functionality – the capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions.

(2 marks)

A good answer should provide definitions of suitability, accuracy, interoperability and security. This should be followed by a brief justification for each of these sub-characteristics. For example – security can be defined as the capability of the software product to protect information and data from unauthorised persons. So, if the product is not sufficiently 'secure' its functionality is questionable.

(8 marks)

c) Efficiency is influenced by time behaviour and resource utilisation, so it can be measured directly. So, it is 'easier' to measure than functionality.

Reliability is influenced by maturity, fault tolerance, recoverability. Only fault tolerance can be measured directly. So, efficiency is 'easier' to measure than reliability.

(5 marks)

Examiner's Guidance Notes:

This question was attempted by nearly 80% of candidates. The pass rate was close to 50%.

Part (a): The evidence shows that many candidates did not provide a proper explanation of software product complexity. Most candidates explained coupling, cohesion and size metrics well, but only a small number properly discussed how these metrics can help the engineer to monitor the quality of software.

Part (b): There is evidence that only a few proper definitions of functionality were provided. Most candidates gave sufficient explanations of the quality sub-characteristics, but only a small number properly justified their answers.

Part (c): There is evidence that only a small number of candidates clearly justified that efficiency is 'easier' to measure.

A2

a) As a software engineer, you have been given the task of reverse engineering and re-engineering a large legacy system written in languages which are no longer widely used in modern development with out-of-date and incomplete documentation.

Give an assessment of the problems likely to be encountered in this task. Explain how you would go about the reverse engineering of this system. Outline the techniques and tools that you could use, and the results that you would expect to produce.

(15 marks)

b) (i) The four categories of software maintenance are: Perfective, Adaptive, Corrective and Preventive.
Explain the meaning of each category.

(6 marks)

(ii) How would you classify the following maintenance activities:
(i) hardware and software platform change,
(ii) correcting errors found by users,
(iii) producing a design document (as the original document has been lost), and
(iv) modifying some parts of software due to changing user requirements?

(4 marks)

Answer Pointers

a) Problems likely to be encountered are as follows:

1. As the system is written in languages no longer widely used it may be necessary to convert it through automated and manual transformations to a more modern programming language so that in the future it will be easier to find knowledgeable software engineers to maintain the system; however, it be difficult to software engineering with adequate knowledge of the old programming languages to undertake the necessary manual transformation work..

2. To aid the re-engineering process and assist future maintainers, redocumentation may be desirable; this is likely to prove difficult in view of the inaccurate and missing documentation and as this is an old system, the staff involved in its development may no longer be available.

3. Depending on the versions of the old programming languages in which the system was originally written, it may prove difficult to find any translators that will convert the old source code into more modern languages.

4. Data re-structuring may be needed if data structures are altered during the re-engineering of software. 5. If the re-engineering involves moving from a procedural programming language to an object oriented programming language, significant work may be involved and there may be little potential for automation.

(7 marks)

The process of re-engineering involves the following steps:
1. Source code analysis and conversion

2. Documentation Recovery and Re-documentation
3. Program re-structuring
4. Program modularization
5. Data re-structuring

The process of reverse engineering could also be discussed. In this case the emphasis should be on various 'recovery' activities such as e.g. design recovery etc.

(3 marks)

Techniques and tools and results produced:

1. Source code analysis and conversion: The old source code is analysed and converted to the new language using source code analysers and old-code to new-code converters. This process is likely to be semiautomatic and results in program in a more modern version of the old language or a program in a completely new modern language.

2. Documentation recovery and re-documentation: code analysis tools can assist with this as well as automated source code documentation tools; results in improved and more comprehensible documentation to assist reengineering and future maintenance.

3. Program re-structuring: automated re-structuring tools can be applied once the code has been converted to the new language. The result is more maintainable software with an improved structure and a more modern architecture may be introduced.

4. Program Modularisation: Re-factoring techniques and clone detection tools may be used. The individual modules/subsystems are grouped together, and any redundancy is removed.

5. Data re-structuring: To transform data, it is likely that specific data transformation tools will have to be developed using lexical analysis techniques.

(5 marks)

(15 marks)

b) Explanations of the four categories of software maintenance should be provided.

(6 marks)

(i) adaptive maintenance
(ii) corrective maintenance
(iii) preventive maintenance
(iv) perfective maintenance.

(4 marks)

Examiner's Guidance Notes:

This question was attempted by nearly 90% of candidates. The pass rate was close to 60%.

Part (a): There is evidence that many candidates identified some problems encountered in the task of reverse and re- engineering. However, only a few managed to identify most of them.

Only a small number of candidates managed to answer the rest of Part (a) reasonably well. Many answers focused on traditional steps of forward engineering and on traditional techniques/tools used in forward engineering e.g. UML techniques.

Part (b): Most candidates provided adequate explanations of different categories of maintenance and many candidates properly classified the given maintenance activities.

B3

A small specialist language training company would like to improve the services offered to existing clients and increase its client base by replacing existing call centre and paper-based mailshots, with online web technology deployment.

a) As a consultant requirements engineer, discuss the tools and techniques that you would deploy to elicit, analyse, document, and check services requested by the company and any actual or implied constraints.

(16 marks)

b) Discuss whether it is advantageous for the company to continue requirements engineering beyond the first phase of the development process.

(9 marks)

Answer Pointers

A good answer should cover

a)

Definition: Requirements elicitation is the first step in requirements engineering. It involves software engineers working with customers and system end-users to find out about the application domain, what services/functionalities the system should provide, as well as the desired performance, software and hardware constraints, and other non-functional requirements with respect to usability, security, reliability and so on; and

Methods: (i) Interviewing Customers: Actual and Potential Interviews are a way of directly finding out what customers want, their present and future needs, and to gather information about the system they require. The requirements engineers can meet with customers individually or in groups; if the potential user community is large then identifying representative users is an important first step. At the early stages, focus groups can also be used and interviewed. Interviews can either be unstructured (informal) or structured (formal). The interviewing method seeks to establish a dialogue with users, so that one interview is likely to be followed up by further interviews to clarify requirements and present identified use cases; this dialogue continues until agreement on the requirements has been reached.

Methods: (ii) Prototyping: building a demonstration system to illustrate the user interface and simulate the possible system behaviour to users, a working model or prototype of the system can be built. This approach is particularly useful when there are unclear or conflicting requirements, as it allows users to evaluate alternative solutions. It is also relevant to demonstrate new functionality to users with which they might not be familiar. Prototyping may be either “throw-away” prototyping at the early stages of development to clarify the poorly understood requirements prior to system design and implementation; or it may be part of an evolutionary approach to development which involves progressive refinement of the prototype working with the customers until a fully working system is developed.

b)

the fact that whilst traditional methods would endeavour to complete requirements engineering in the early life cycle phases, modern approaches engage such practices throughout the lifecycle, with the client as an important stakeholder and project team member, and a point of reference for checking timeliness and accuracy of requirements capture.

Examiner's Guidance Notes:

More than 73% of candidates attempted this question but it had one of the lowest pass rates (36%).

Part (a) was answered reasonably well, but there is evidence that a substantial number of candidates appear side-tracked by attempting to deliver a solution to the problem scenario rather than discussing aspects of applicability of the range of requirements engineering tools and techniques available.

In answering part (b) many candidates showed a lack of awareness of modern approaches such as the "agile" philosophy in this regard.

B4

a) A small software tool construction business is considering whether adopting Open Source Software Engineering (OSSE) would improve software quality, and increase overall programmer productivity. As a Freelance IT Consultant, write a report that presents an outline of OSSE, compares it to similar process methods, and demonstrates how it might be deployed in the business.

(16 marks)

b) Briefly discuss the significance of Collaborative Development Environments (CDE) to the short-term and long-term future of Open Source Software Engineering.

(9 marks)

Answer Pointers

A good answer should

a)

- Provide an overview of OSSE
- Identify areas for potential quality improvements. These are testing, walkthrough; reuse; refactoring. Comparison can be made with an agile method such as XP.

Thus,

(i) Standards in the open source development community can reduce increase developer productivity, esp. standard practices in the use of languages and version control toolsets.

(ii) Reuse is key to developer productivity by providing access to large libraries, using standard and well documented interfaces to proven and fully tested components.

(iii) The early release of software attempts to address the tendency of a proposed development becoming unaligned through changes in the business environment. Early release means the client and some stakeholders can enjoy the benefits of the software much sooner.

(iv) Product Quality. Highly skilled developers are in short supply, are expensive to employ, and difficult to support. A large community of volunteers, often intrinsically motivated, and providing a readily available repository of project and development

skills, self-supporting, and self-correcting groups to improve the quality of the final product. Effective communities such as these reduce costs and increases quality. Further, issues of correctness and reliability are addressed by the extensive community of users providing peer review and feedback.

b)

- note that in the short-term, development communities can reduce the administrative and training cost of using powerful tools and make secure access to diverse development artefacts practical. It can reinforce and compound the effects of individual tools, leading to long-term benefits including accumulation of development knowledge in a durable and accessible form, increased quality and reuse, and more consistent adoption of the organization's chosen methodology.

Examiner's Guidance Notes:

41% of candidates attempted this question and returned the second lowest pass rate of near 33%. There is evidence that whilst "open source" software is well understood; the software engineering of open source was not. This lack of knowledge extended to part b) where no real future perspective on its growth or decline developed or argued. Answers were often shallow, of questionable relevance, failed to go further than "free" source code, and were of little academic merit in many instances.

B5

A local transport service requires proprietary software that can manage passenger access of services by introducing automated entry and exit barriers.

a) Briefly discuss the applicability of the following architectural models, and illustrate how each might be applied to the scenario described:
   (i) The repository model;
   (ii) The client-server model;
   (iii) The abstract machine model.

(15 marks)

b) Briefly discuss the potential benefits and challenges of code, architectural and design refactoring

(10 marks)

Answer Pointers
A good answer should cover

a)

(i) Repository model – either shared data in central database; or local databases shared via message-passing; the former is more commonplace.
The local transport system can have a common customer database, where each entry-exit barrier can pick up passenger status information and act accordingly;

(ii) Client-server model – data and processing distributed across a range of processors. Much more prevalent in contrast to the Repository. Many of the sub-systems such as the entry-exit barriers can be considered as clients. Decision

whether processing is performed centrally by servers or locally at the barriers, can determine the level of network traffic between server and client, and fault-tolerance capability of the central server.

(iii) In this model, a layered approached is considered where the nucleus of the system is represented by firmware/operating system, surrounded by another layer of services pertaining to the entry-exit barriers and their location; layered by passenger data-information services.

b)

Code refactoring is a technique for restructuring an existing body of code. The challenge is to alter its internal structure and keeps it fully working without changing its external behaviour. A series of small behaviour preserving transformations means it's less likely to go wrong and reduces the likelihood of system failure.

Architecture Refactoring is about changing the software architecture, into new logical layers, resulting in an increase in the overall software quality in terms of scalability, testability and robustness of the actual solution system. The major challenge of this kind of refactoring is trying to reuse as much as possible existing code, without writing new software.

Design refactoring seeks to improve the design of existing code. Over a period of time code maintenance activities to adapt to new requirements, operating environments and platforms can result in a 'labyrinth' of links and arcs of control. This reduces code simplicity and increases the complexity of the software and the resulting design representation. The challenge is producing a design through refactoring that is both easy to comprehend whilst reflecting the current and expected behaviour and practice of existing operational software.

Examiner's Guidance Notes:

This question was the most unpopular of all questions attempted by only 22% of candidates. It also had the lowest pass rate of 21%. The evidence shows a general lack of knowledge of architectural models, and many answers were often shallow. Some candidates appear to be more concerned with delivering an actual solution to the problem scenario rather than discussing the models listed and aspects of their applicability to the scenario presented.

There was some good understanding shown re code refactoring in Part b). However, many candidates showed little knowledge or awareness of the various refactoring levels.