THE BCS PROFESSIONAL EXAMINATION
BCS Level 6 Professional Graduate Diploma in IT
March 2018

EXAMINERS' REPORT

Software Engineering 2

General Comments

The pass rate was a disappointing at 41% but is not out of line with previous years. The following issues (some of which have been identified in previous examiner reports) appear to be significant:

1. Coverage of the syllabus. This issue is borne out in candidate responses to questions around topics such as Requirements Engineering, and modern software process models. These subjects are extensively covered in the recommended text. It is important that candidates have breadth and depth of knowledge in these areas at the point of delivery.

2. Software Engineering 1 is not a pre-requisite for this unit. However, candidates are encouraged 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. There is evidence that some 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. There continues to be instances of candidates answering parts of questions without regard for their indicative weightings, resulting in too detailed an answer for one component part that cannot compensate for the marks lost by incomplete or "shallow" coverage of the second and subsequent components of questions. Further, candidates should make sure they understand and answer the question being asked, instead of giving a detailed account of a keyword or abbreviation identified, and loosely related to the subject.

4. Presentation. It is important that candidate responses to questions are legible, well-structured and formatted.

Section A

A1

a) Software evolution processes vary considerably depending on the type of software being maintained, the development processes used in an organisation and the people involved in the process. However, they include some fundamental activities. Write a brief report that outlines a typical software evolution process.

(12 marks)

b) Software maintenance costs are influenced by a number of technical and non-technical factors. Some of the factors are: module independence, programming style, documentation, staff stability, hardware stability, and software age.

Which of the above factors is a software engineer able to control when **developing** new software? For each of these 'controllable' factors, explain how a software engineer would attempt to minimize future maintenance costs.

(6 marks)

c) After a major release of a software system, there often follows a period of corrective maintenance. Discuss why this can occur, how it could be avoided and whether it is inevitable.

(7 marks)

Answer Pointers
[1.5, 2.1 Software maintenance and evolution]

a) A good answer should include a brief explanation of the following fundamental activities: Impact analysis/Change analysis, Release planning, Change implementation and validation, System release.

8 marks

A good answer should also emphasise the iterative nature of a typical software evolution process (a simple diagram can be used) and it should mention three different types of maintenance: Corrective, Adaptive, Perfective.

4 marks

b) Definitely the first three factors can be fully controlled by software developers while the remaining factors not.

For example – if you and your team design a system with highly independent modules (i.e. with a low coupling) then changes in one module are unlikely to affect other modules i.e. they are localised. Therefore, the actual modifications in the system are easier to implement and the testing of these modifications is also easier (as these modifications are unlikely to affect other modules).

Proper programming style (e.g. clear layout of the source code, good comments, etc.) make the corrections and modifications easier. The same applies to good documentation.

So, you and your team should make sure that when you develop a system these factors are properly addressed.

On the other hand, you have no influence (as a developer) on software age, hardware stability and staff stability after your system was delivered.

6 marks

c) A good answer should discuss the evidence that Lehman and his co-workers found when studying the evolution of large system. It seems that the many enhancements and new features added made in a major release will involves substantive changes and therefore there is a risk of errors being introduced. Pressure to bring out a new release may also add to the risk of errors being introduced. Insufficient regression testing may mean that any errors introduced with the additional of new features and functionality are not detected and only come to light after the new system is released to users. Truly innovative features and functionalities may not be sufficiently well understood by the software engineers and hence programming errors may arise in their implementation. Corrective maintenance can be minimized by the application of sound software engineering practices in the form of change control, version control and configuration management, extensive regression testing as well as comprehensive testing all new features and functionalities, and better management of releases. In some cases, it is better to limit the features and functionalities than to risk a release with bugs. Of course, it is never possible to eliminate all causes of bugs, so some degree of corrective maintenance is always needed.

7 marks

Examiner's Guidance Notes:

Nearly 90% of candidates attempted this question. The pass rate is above 40%.

There is evidence that part a) proved a challenge to some candidates. Only a small number properly discussed fundamental activities of a typical software evolution/maintenance process. Many candidates discussed traditional software development stages (Analysis, Design, etc.) instead.
Part b). Generally, candidates provided reasonable answers.
Part c). Only a small number of candidates properly explained why corrective maintenance is inevitable. However, many provided reasonable explanations of why this can occur and how it could be avoided/minimised.

A2

a) Outline three reasons why assertions are useful in software design.

(9 marks)

b) A system is required to keep track of students who belong to a World Cinema club. The maximum number of club members is 50. Some of the typical functions provided by the system are as follows:

- join (sNumber) – to add a new student (whose number is sNumber) to the club provided that the upper limit (i.e. 50) is not exceeded,
- leave (sNumber) – to delete a student (whose number is sNumber) from the club,
- retrieve (sNumber) – to retrieve details of the student whose number is sNumber,
- is_a_member (sNumber) – to return true if a student (whose number is sNumber) is a member of the club and false otherwise,
- no_in_club – to return the current number of club members.

Define and develop formal (e.g. using OCL) and natural language specifications for this system. The specifications should include pre and post conditions for the following operations: join, leave, retrieve as well as the invariant for the club. State all assumptions made.

(16 marks)

Answer Pointers
[3.4 Advanced use of UML including OCL and use of assertions, pre- and post-conditions]

a) Three reasons why assertions are useful in software design are as follows:

1) Assertions make explicit in formal language the logical properties of software and constraints on the software. Using Boolean Logic or Predicate Calculus, these assertions can be subjected to formal proof.

(3 marks)

2) Assertions assist developers to write correct software, since they specify the expected behaviour of the software to be implemented. The developer can include code within their implementation to check that the assertions are true.

(3 marks)

3) Assertions provide documentation of interfaces and can be used by developers when debugging. Assertions provide guidance to developers using an interface and to debuggers about what values to expect when debugging an interface error.

(3 marks)

b) A good answer should present semi- or formal descriptions of the operations, clearly identifying some typical OCL constraints for these. For example:

context join

pre: Club.no_in_club() < 50 Club.is_a_member(sNumber) = false

post: Club.no_in_club() = Club.no_in_club()@pre + 1
Club.is_a_member(sNumber) = true

--The precondition states that the number of club members is less than 50 and the student to be added is not already in the club.

--The postcondition states that the number of club members is one greater than its precondition value. Further, the student to be added is now present in the club.

(4 marks)

context leave

pre: Club.is_a_member (sNumber) =true

post: Club.no_in_club() = no_in_club()@pre - 1
Club.is_a_member (sNumber) = false

--The precondition states that the student to be removed is already present in the club.

--The postcondition states that the number of club members is one less than its precondition value. Further, the student to be deleted is no longer present in the club (indicated by the return of a "false' value).

(4 marks)

context retrieve

pre: Club.no_in_club() >0 Club.is_a_member (sNumber) =true

post: no_in_club() = no_in_club()@pre Club.retrieve(sNumber) = s

--The precondition states that at least one student is already present in the club and this includes the student to be retrieved. -- The postcondition states that the number of club members remains the same as its precondition value, and the member is retrieved.

(4 marks)

context Club

inv 0 <=no_in_club() <=50

-The invariant states that the number of club members is always within the above range.

(4 marks)

Examiner's Guidance Notes:

This was the least popular question. Less than 10% of candidates attempted it. The pass rate however is good – 60%.

The evidence shows that for part a) most candidates provided reasonable answers.

For part b). there were some very good answers. The candidates who provided these answers were able to properly specify pre- and post- conditions for all three operations and to specify the invariant.

Section B

B3

a) Discuss **TWO** aspects of the requirements engineering process and assess their usefulness in capturing, specifying, and communicating requirements throughout the software lifecycle.

(16 Marks)

b) Discuss and suggest how advances in natural language processing can help engineers detect requirement ambiguities and defects.

(9 Marks)

Answer Pointers

[1.3 Software Requirements Engineering including requirements management]

a) A typical answer should present a brief definition of the topic and present a brief outline of the requirements engineering process. For example:

Requirements engineering involves tasks that lead to an understanding of what the business impact of the software will be, what the customer wants and how end-users will interact with the software. The process includes Feasibility Study, elicitation and analysis, specification, and validation.

It is the practice of obtaining the requirements of a system from users, customers and other stakeholders. The practice is also sometimes referred to as Requirement gathering.

The practice of elicitation includes such things as: interviews, questionnaires, user observation, and prototyping. The challenges are to do with scoping (ill-defined system boundary), comprehension (users having partial understanding of the problem domain), and environmental volatility. Modern approaches use multiple elicitation methods, solicit participation from different point of views, and create usage scenarios or use cases to help users identify key requirements.

(16 Marks)

b) Given the prevalence of natural language for documenting requirements, a great need exists to automate or semi-automate elicitation and analysis documents that may containing hundreds or thousands of requirements.

However, whilst some progress has been made, natural language's inherent complexities, including grammar and semantics, makes completely automated analysis difficult.

The semi-automated approach allows the requirements analyst to execute ambiguity and defect detection tool as they document requirements. These tools show promise in helping engineers to identify defects and remove ambiguities before they lead to problems later in the cycle.

(9 Marks)

Examiner's Guidance Notes:

More than 66% of candidates attempted this question but it had one of the lowest pass rates (33%).

There is evidence that many responses given in part a) showed a lack of knowledge of requirements engineering and its processes. Many candidates appeared to have resorted to using general knowledge as a substitute. This appeared to be the case also in part b) where many candidates demonstrated some general knowledge of the meaning of "natural language" but a lack of awareness of natural language processing in its academic and technical context.

B4

a) Give a brief outline of a traditional software-process-improvement (SPI) model used in large organisations and discuss the challenges very small software companies might face when implementing these models.

(16 Marks)

b) Discuss how small software companies can adopt aspects of SPI in practice, to realise external and internal opportunities and benefits.

(9 Marks)

Answer Pointers
[1.1 Software Process Improvement]

a) A typical answer may outline one of the more common traditional software process improvement models such as CMMI, and ISO/IEC 15504 (also known as SPICE).

For example, a model answer where CMMI was chosen would:

Provide a brief introduction and description of the model. For example, the Capability Maturity Model consists of 18 main Key Process Areas (KPA), 52 goals, and 316 key practices, in 500 pages of documentation. There are five levels of process capability from "Initial" (lowest) to "optimizing" (highest). Initial is where the software process is characterized as ad hoc, and occasionally even chaotic; and optimizing is where continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies.

(6 Marks)

Highlight issues such as:

- software quality is a key to competitive advantage;
- organizations are using only a few of the most popular ISO/IEC systems and software engineering standards;
- experience difficulty relating ISO/IEC standards to their business needs and justifying their application to their business practices;
- lacking expertise or is unable to afford the necessary employees, cost, and time;

(10 Marks)

b) Various initiatives have been undertaken since 2011, devoted to small companies such as ISO/IEC 29110 standards and guides:

- focusing on project management and software implementation;

- comprising of profile groups, each containing profiles related by process composition (such as activities or tasks), capability level, or both;
- targeting companies with no experience or expertise in selecting a project's appropriate software processes; and
- addressing specific software process needs.

Surveys indicate that the areas posing the most problems are software implementation, verification and validation, test cases, test procedure, software components, and software architecture and detailed design.

(9 Marks)

Examiner's Guidance Notes:

50% of candidates attempted this question and returned the lowest pass rate of under 30%. A significant number of candidates don't appear to be aware of SPI and, consequently resorted to a speculative description of how to "improve software". As part b) seeks to apply this knowledge, many candidates were not able to suggest a possible solution.

Had the candidates made the link between SPI and models such as CMM, is it possible that more may have been able to demonstrate knowledgeable responses to part a) and b).

B5

a) Compare and contrast **TWO** software engineering approaches that aim to satisfy the need to develop software quickly, in an environment of rapidly changing requirements.

(16 Marks)

b) Discuss the view that agile methods lack maturity and problems remain, as well as opportunities, for improving the process of software construction.

(9 Marks)

Answer Pointers

[1.2 Various Software Life Cycle Models (Waterfall, V-model, Prototyping, Spiral Model, Incremental Development, Evolutionary Development, Agile models including Extreme Programming)]

a) The expected answer should include two of the following: Prototyping, Incremental Development, Evolutionary Development, and Agile models such as DSDM or XP.

For example, a model answer where the choices made were Prototyping and DSDM would:

Provide a brief introduction and definition for each model

(6 Marks)

Highlight issues such as:

- The use of iterative development and frequent releases to customers (incremental prototyping);
- Close collaboration with customers encouraged (throwaway prototyping); however, this may be contrasted with DSDM (preferably onsite) and accepting requirements change more so in the latter as a welcome, part of the process;
- The emphasis on small self-organizing teams, the idea of continuous design improvement, test-driven development and continuous integration can be compared but contrasted in terms of time frame and organization or managerial structures;
- Knowledge management compared and contrasted in terms of level of documentation, and face-to-face communication.

(10 Marks)

b) The answer should recognize that in historical terms, Agile Methods is relatively new, and there is no universal agreement on what agile software development is. However, many approaches have attempted to demonstrate their “agileness” according to the underpinning concepts and most of the practices associated with agile software development.

The outstanding issues might be considered to be around ad hoc, unprincipled development, lack of documentation and possible audit trail, and concerns about future maintenance, especially for mission critical systems. However, the increasing use of automated tools for reverse engineering, design and testing present opportunities for these issues to be addressed.

(9 Marks)

Examiner's Guidance Notes:

This question was one of the two most popular questions attempted by 87% of candidates. It also had the second highest pass rate of over 46%. The evidence shows that many candidates demonstrated good knowledge of appropriate methods for the modern environment, characterised by agility, rapid change and fast delivery. However, there were a number of candidates who counted the Waterfall Model as such without any supporting arguments, and their awareness of modern process methods seemed limited.