BCS THE CHARTERED INSTITUTE FOR IT

BCS HIGHER EDUCATION QUALIFICATIONS
BCS Level 6 Professional Graduate Diploma in IT

SOFTWARE ENGINEERING 2

March 2016 – Morning
Answer **any** THREE questions out of FIVE. All questions carry equal marks.
Time: THREE hours

Answer any Section A questions you attempt in Answer Book A
Answer any Section B questions you attempt in Answer Book B

The marks given in brackets are **indicative** of the weight given to each part of the question.

Calculators are **NOT** allowed in this examination.

Section A
Answer Section A questions in Answer Book A

General Comments

The following issues have been discovered in the marking of this paper:

1. Coverage of the syllabus. Of particular concern is the lack of foundational knowledge demonstrated in the responses, resulting in candidates using common sense interpretation as a substitute for depth in basic principles and concepts of software engineering. In terms of breadth of coverage, there is evidence that many candidates appeared to have good knowledge of traditional methods, but limited knowledge on modern methods, tools, and techniques as identified in the new syllabus.

2. Subject awareness. To achieve high marks, candidates should provide more breadth in responses given to all parts of the question by reading more widely publications within the profession as well as recommended text.

3. Examination techniques. There are instances of candidates answering too many questions resulting in very “shallow” answers”; or too few, which resulted in too detailed an answer but insufficient to compensate for the marks lost by not completing another question, or the second and subsequent components of questions.

4. Presentation. It is important that candidate responses to questions are legible, well-structured, formatted, and appropriate to the questions and rubric of the paper itself.

A1. [Software metrics and models and associated measures of software quality]

a) Distinguish between software process and software product metrics.

(5 marks)

b) At which stages of software development projects would you use the following software metrics and software quality 'measures':

(i) Function Points
(ii) Lines of Code Count (LOC)
(iii) Cyclomatic Complexity
(iv) Coupling and Cohesion?

Explain why.

(8 marks)

c) Comment on applicability of Function points, LOC, Cyclomatic complexity and classical Coupling and Cohesion to 'object-oriented' projects.

(12 marks)

Answer Pointers

A good answer should cover the following points:

a) Software product metrics measure properties and aspects of the software product itself, such as its size in lines of code, its complexity, no of functions, execution speed, etc. They can involve direct measurements or indirect measurements of product properties. Process metrics measure properties and aspects of the software processes during development and maintenance, either directly or indirectly, such as time taken by a team to perform an activity, team effort per activity, staff turnover on a project, etc.

Total (5 marks)

b) The Function Point metric is a 'high level' metric which can be used effectively for 'measuring' the functionality provided by a system. This functionality is 'measured' in terms of external inputs, external outputs, external inquiries, internal and external files i.e. in terms of data which are normally known at the analysis stage of a software project.

LOC can only be calculated when the actual code is available, so during or after the implementation stage.

Cyclomatic complexity is based on a control structure of a program e.g. represented by a pseudocode. So, it can be calculated at the detailed procedural design stage or later e.g. during or after the implementation stage.

Coupling and cohesion are 'metrics' related to modularity. Therefore, they are suitable to assess the quality of a 'high level'/component level design.

2 marks for each

Total (8 marks)

c) The Function Point metric is paradigm independent, so it can be used in 'object-oriented' projects.

LOC is a rough measurement of the size of traditional/structured programs. Theoretically it can also be used as a rough size measurement of object-oriented programs as most of constructs provided by structured and object-oriented programming languages are similar.

A similar argument could also be used for cyclomatic complexity. However traditional/structured and object-oriented programs differ in the way they are modularized. In traditional/structured programs modules are procedural and some may be relatively complex. In object-oriented programs the cyclomatic complexity of a class is usually low, because many classes typically include a large number of small, straightforward operations/methods. Cyclomatic complexity also ignores data complexity. Because data and operations are equal 'partners' in the object-oriented paradigm, cyclomatic complexity ignores a major element that could contribute to the complexity of a class. Therefore, traditional cyclomatic complexity is not very suitable to measure class complexity.

Classical coupling is a 'measure' of the strength of the intercomponent/intermodule connection. A high degree of coupling indicates a strong dependence between components/modules. Classical cohesion can be viewed as the 'glue' that keeps the component/module together. It is a 'measure' if the mutual affinity of the elements of a component/module. Both concepts can be easily extended to object-oriented designs. Coupling can be viewed as a 'measure' of the degree to which classes are connected to one another. Cohesion implies that a class encapsulates only attributes and operations/methods that are closely related to one another.

3 marks for each

Total (12 marks)

Examiner's Guidance Notes:

This question assesses a candidate's knowledge and awareness of software measurement concepts and processes.
More than 50% of candidates attempted this question, but only some produced reasonable answers. Most candidates provided proper explanations of software product and process metrics in parts (a), but many did not give proper examples of the corresponding metrics. Part (b) was reasonably answered by a small number of candidates only. Many answers provided irrelevant information.
Only a small number of candidates answered part (c) reasonably well. Many answers were irrelevant or/and incorrect.

A2. [Software maintenance and evolution]

a) Explain what is meant by a legacy system and why such systems may be critical to the operation of an organization. Discuss ways in which organizations can lessen their reliance on legacy systems. (10 marks)

b) Explain the differences between reverse engineering , re-engineering and re-structuring in software projects. (6 marks)

c) Outline the process of preventative (or preventive) maintenance and discuss why such maintenance is needed. (9 marks)

Answer Pointers
A good answer should cover the following points:

a) A legacy system is large and usually complex computer system that has been developed over a long time and typically is based on obsolete technology or at least technology that is also old. Legacy systems are example of socio-technical systems in that they embody organizational processes and ways of doing things that may themselves be old. A legacy system may include old hardware, old applications, historical data stores, as well embedding old business policies and rules. Nevertheless, these legacy systems may be critical to the operation of an organization which may find it hard to change its long established processes and ways of doing things. The organization may be worried that replacement or radical change to the legacy system may result in a system which doesn't work properly and puts the running of the organization at risk. (4 marks)

There are various ways in which an organization can lessen its reliance on a legacy system. They are as follows:

It may be possible to replace old hardware with newer hardware without affecting the support software and applications that are needed for the organization's business processes.

It may be possible to reverse engineer and re-engineer the system, or to isolate components and reverse and re-engineer these using more modern technology. Often the latter approach reduces the risk of the system failing catastrophically.

Addressing the hardware layer, separately from the software and business process layers, in a bottom up approach starting with newer hardware, may allow gradual (and 'safer') change to take advantage of the new facilities. For example, introducing a database server might allow the introduction of web access via a browser to important data and business processes could be modified to incorporate this data. (6 marks)

Total (10 marks)

b) Reverse engineering – analyzing the source code to recreate design and/or system specification information from the code e.g. design recovery. Note that the code is not modified.

Restructuring. The source code is changed by transforming ill-structured code into well-structured code. Note that the system's functionality is not changed.

Reengineering usually involves both reverse engineering and restructuring. In addition, it also involves 'forward engineering' to make changes to the system's specification (e.g. to update functional requirements) and to implement them.

2 marks for each

Total (6 marks)

c) A good answer should explain that preventative maintenance concerns activities aimed at increasing the system's maintainability such as improving and updating documentation, updating the code of a system to improve its understanding, improving the modular structure of the system, etc. All these activities aim at decreasing future maintenance costs.(3 marks)

The process of preventive maintenance starts with a systematic investigation of the system's designs, code, and associated documentation to identify problems. Code analysis tools can be used to study the code of the system. Past maintenance records may be examined to identify hot spots which have been subject to repeated changes as they may be in need of re-engineering. Poorly understood sections of the code should be identified and improved as well. This may involve restructuring of the system architecture, separation of large code segments into smaller independent modules, identification of code clones to replace them with shared code, introduction of design patterns and replacement of old obscure code, removal of 'dead' code, etc. (6 marks)

Total (9 marks)

Examiner's Guidance Notes:

Nearly 90% of candidates attempted this question.
For part (a) some candidates produced irrelevant answers e.g. they discussed 'legal' practices instead of legacy systems.
Part (b) was generally answered well, but some answers were irrelevant.
For part (c) there were many wrong answers as candidates discussed such issues as e.g. risk analysis, security, 'prevention' of errors, etc.

A3. [Software management]

a) Discuss whether software project management differs from the management of projects in other sectors of society. (10 marks)

b) As a software project manager what factors would you take into account when selecting staff to work on a software development project? (7 marks)

c) There is no simple way to make an accurate estimate of the cost and effort required to develop a software system. Two popular software cost estimation techniques are Expert Judgement and Algorithmic Cost Modelling. You have been given a task to briefly explain these techniques and discuss their potential disadvantages. (8 marks)

Answer Pointers

A good answer should cover the following points:

a) The answer should briefly outline the supposed unique characteristic of software, namely: constantly increasing levels of technology; change is inevitable and hard to deal with; projects are often unique so historical data does not help with future projects; and software is intangible, does not wear out and therefore, the maintenance task is not about replacing worn components. In the case of the latter, when is a project 90% finished? Often, its size has no direct relationship with effort, time, and resources; the fact that it is 90% complete provides no evidence (that all or some of the investment (capital) can be recovered or is any nearer completion.

The discussion should indicate that most projects in today's society, suffer from constantly increasing levels of technology; having to deal with change constantly, and being so unique that very little can be learnt from past projects. Of more concern is the intangible nature of software itself, and this is unlikely to change whilst there is a general lack of advances in practical commercial estimation and measurement techniques and tools.

Total (10 marks)

b) Ideally, skilled and experienced staff will be available to work on the project. However, in most cases, various constraints such as e.g. the project budget, insufficient number of experienced software engineers, etc. influence the staff selection process. Assuming that there are no constraints the following factors should be taken into account:

(2 marks)

Application domain experience,

Platform experience,

Programming language experience,

Problem solving ability,

Educational background,

Communication ability,

Adaptability,

Attitude,

Personality.

5 factors with brief explanations will be enough to get 5 marks. (5 marks)

Total (7 marks)

c) Expert judgement – a number of experts on the proposed application and software development techniques are consulted. Each of them provides his/her estimated project cost. This estimation process iterates until an agreed estimate is reached.

Algorithmic cost modelling – a model is developed using historical cost information that relates some software metric (usually its size e.g. LOC, FP, etc.) to the project cost. There are various published algorithmic cost models e.g. COCOMO1 and COCOMO 2.. An estimate of the size is made and the model 'predicts' the effort and cost required.

In general, the basic formula is Effort = a * $Size^b$ * m (where a and b are constant factors and m is a multiplier which depends on a number of process, product and development attributes).

(4 marks)

Disadvantages:

Expert judgement:

It relies on the experience of experts, so it is subjective.

It depends on current data. The data on which this approach is based must reflect current practices, so they must be updated often.

Most expert judgement techniques are simplistic (i.e. they do not take into account many factors which affect the effort needed on a project).

Algorithmic cost modelling:

It is often difficult to estimate size e.g. LOC at an early stage in a project (when only a specification is available). This problem can be partly alleviated by using Function point and/or Object point estimates, but these are often still inaccurate.

The estimations of the factors contributing to b and m (see the basic formula above) are subjective. (4 marks)

Total (8 marks)

Examiner's Guidance Notes:

More than 80% of candidates attempted this question. Part (a) was answered reasonably well, but a substantial number of candidates discussed general 'managerial' activities instead (e.g. scheduling, costing, staff management, etc.).
Many candidates answering part (b) produced adequate answers.
Many candidates did not attempt Part (c) or and some produced incorrect answers (this in particular applies to algorithmic cost modelling).

Part B

B4. [Open Source Software Engineering]

a) Discuss the merit of each of the following Open Source Software Engineering (OSSE) project practices, giving clear explanations and appropriate examples:

i. Adopting standards and facilitating reuse

ii. Early product release and peer review

iii. Develop communities of volunteers both as users and developers

(15 Marks)

b) Discuss the view that the future advancement of open source practices amongst all software developers is wholly dependent on the widespread adoption of Open Source Software Engineering tools.

(10 Marks)

A good answer should cover:

a)

i. Adopting standards in the open source development community can reduce the effort needed to start a new project, and increase developer productivity. Whilst training is important in standardizing engineering practices, peer review remains a powerful vehicle in propagating many project practices. Standard practices in the use of languages and version control toolsets.

Reuse is key to developer productivity by providing access to large libraries, using standard and well documented interfaces to proven and fully tested components. However, it is still common for projects to use tools and create interfaces with little or no standardization across projects.

ii. The early release of software attempts to address the tendency of a proposed development becoming unaligned through changes in the business environment. It would address a long-standing problem of a large percentage of software being developed but either never used, or become obsolete very quickly. Early release means the client and some stakeholders can enjoy the benefits of the software much sooner. However, concerns about its correctness and reliability, are partially addressed by the extensive community of users providing peer review and welcome feedback.

iii. 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. However, not every project will attract the critical mass of quality personnel for such benefits to be realized.

And recognize in this proposition that:

b)

- Open source software engineering tools are widely used today, with examples such as: Version Control using CVS and Subversion; Issue Tracking and Technical Support using Bugzilla; Quality Assurance using ToolsJUnit, and PHPUnit, Collaborative Development Environments with SourceForge providing support for managing the frequent release.

- Growths of this nature will by nature attract other developers and so directly advance the use of open source tools. However, whilst advance in adopting tools may effect indirectly growth in open source practices, inability to quantify this growth and observable evidence would indicate that collaborative communities are selective in nature, with standards and product quality varying greatly. This in itself may well act as an upper limit on growth and subject the advancement of its practice to the law of diminishing returns.

Examiner's Guidance Notes:

This question assesses a candidate's knowledge of open source software engineering and general awareness of its adoption in both current and future practice. The question was the third most popular question attempted, with the second lowest pass mark (45%). There were some very good answers, but most candidates spent much time describing open source, rather than the "engineering" aspect of development processes and practices.

B5. [Software Process Improvement]

a) Present a brief outline of the hierarchical structure of the Capability Maturity Model (CMM) and give examples of key process areas and goals.

(8 Marks)

b) Discuss how the many process areas and goals of CMM might be tailored for a small organization wishing to move into e-commerce, by adopting processes that are rigorous, disciplined, and of industrial strength.

(9 Marks)

c) Discuss the view that agile methodologies can adequately address many Level 2 and 3 practices of software process improvement.

(8 Marks)

A good answer should cover:

a) The capability maturity model

- consists of five levels from “Initial” (lowest) to “optimizing” (highest) maturity in terms of process capability;
- The general progression is Initial ->Repeatable ->Defined ->Managed ->Optimizing

  Initial : The software process is characterized as ad hoc, and occasionally even chaotic;

  Repeatable: Basic process management processes are established to track cost, schedule, and functionality; KPA - Requirements Management; Software Project Planning

  Defined: The software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization; KPA - Software Product Engineering; Peer Reviews

  Managed: Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled; KPA - Quantitative Process Management; Software Quality Management

  Optimizing: Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies; KPA - Process Change Management and Defect Prevention.

- All the CMM goals primarily address process end states with a preference for quantitative measures and can be considered as a "sub process area".

b) Key Process Areas (KPA), maturity levels, and the small (eCommerce) business

- With 18 KPAs, 52 goals, 316 key practices, and 500 pages of documentation, the model is appropriate for large systems developed by large teams of engineers. However, many small organizations would find its adoption expensive and unwieldy, and cannot be applied without modification. Further, all the goals of KPA need to be achieved for the organization to satisfy that KPA, and done on a continuing basis across all projects. The amount of overhead and reporting for a small company makes it expensive to use.

- In the very competitive area of e-commerce, product quality and company reputation are key success factors. To achieve success, the small organization will need a CMM tailored to its particular resource envelope and business objectives. Firstly, research data indicate that many organizations do not progress beyond level 2, esp when a rating at level 2 in the CMM may be regarded as ISO 9000 compliant. Further, it is not clear if levels 4 and 5 add much value to an organization.

- A capability assessment would need to be performed. The outcome may indicate the company to be at level 1. To progress to the next level (level 2), the process areas to be considered include: Software configuration management, Software quality assurance, Software subcontract management, Software project tracking and oversight, Software project planning Requirements management

c) Agile methods equivalence to CMM practice

- In terms of Agile methods such practices as Peer reviews, Intergroup coordination, Organization process focus, and training programmes found at level 3 of CMM are underlying themes. Similarly, at level 2, Software configuration management and Software quality assurance are important elements of agile methods.

- However, with so many KPAs, goals, key practices, and 500 pages of documentation, CMM is more extensive and far-reaching than agile methods.

Examiner's Guidance Notes:

This question assesses a candidate's knowledge of the Capability Maturity Model, and general awareness of commercial quality frameworks. The question was the least popular of all questions and exhibits the lowest pass rate (35%). In most of the candidate answers, there is evidence that there was a lack of subject knowledge with no grasp of key concepts, principles and practices in CMM.