BCS HIGHER EDUCATION QUALIFICATIONS
BCS Level 5 Diploma in IT

April 2011

EXAMINERS' REPORT

Software Engineering 1

General Comments

This is a technical paper about Software Engineering. Questions seek to test candidates' knowledge, and ability to apply that knowledge. A poor answer is one that simply describes recalled information. A good answer will show application of the recalled knowledge.

This exam encourages reflection, meaning critical review of what the topic means in the wider context of software engineering. Candidates are encouraged to read more widely about 'high end' goals of software engineering, such as risk management and ethics, and reflect on how development processes hinder or help the higher aims.

Overall, the candidates' responses to the Software Engineering examination questions showed a good understanding of the subject. Most marks for individual questions are in the average range.

The rate of response was unevenly distributed for booklet A: Q1, Q2 and Q3 received roughly the same number of attempts (some 50-60% of candidates). Most candidates attempted Q4 and Q6 in booklet B. This is understandable as Q6 required the candidate to provide justification for the Graphical Interfaces within CASE tools; while Q4 required the candidate to identify and define different testing techniques and terminology. Unsurprisingly Q3, regarding UML diagrams, was very popular among candidates.

Question A1 – Software Processes

A1. A company supplies a Tax Returns Automation process to its clients, visiting their sites and inspecting their revenues for a given year, giving advice and completing the necessary forms for Tax Returns purposes. Once the forms have been completed, they are saved as paper copies; one is kept by the client, the other is filed locally within the company's paper-based archives. The corresponding electronic copies are also saved in a word-processor format and saved on a local computer.
The company is seeking to develop a more fully automated process: the tax consultant visiting the client's premises periodically would log-in to an on-line application (within the company website) and input the data to an on-line form. The data collected would be used to keep the clients informed of the results of the consultant's visits and the date of the next visit.

Considering the above scenario:

a) List and describe at least **THREE** possible risks that the company faces in its current, paper-based business process.

(9 marks)

b) List and describe at least **THREE** possible risks that the company will be likely to face when the new more fully automated system has been implemented.

(9 marks)

c) Analytically select and describe **ONE** technique for estimating the effort and the costs associated with the project of building the more fully automated on-line system.

(6 marks)

Answer Pointers

a) A good answer addresses these points:

- using the wrong forms when visiting the clients
- lose/destroy forms in the travel from the client's site to the company
- risk of wrongly copying the paper-based data onto the desktop machine for archiving purposed

b) A good answer addresses these points:

- security issues, hackers access the website
- clients access the wrong personal details
- synchronisation of database against various branches of the company

c) Since the system does not contain technology unknowns, advanced techniques for effort estimation like Delphi or COCOMO are not needed. Techniques based on the analogy of similar projects developed by the same company, with attributes such as size, technology, staff experience, effort, duration, etc could be useful to estimate cost and duration of such project.

Examiner's Guidance Notes

This question requests students to provide an answer based on the concepts of software processes and risk management based on a real case scenario. Part (a) and (b) were answered well in many cases, especially part (b) that deals with the creation on a website, and students pointed out correctly that security of data should be guarded properly.

Part (c) was more evenly distributed, and in general students did not have a clear understanding of when to use specific estimation models and why. Most often the answers are drawn from known terms ("COCOMO") while no real reasoning was performed before the selection of the optimal solution.

Question A2 – OO Design

A2. a) Define the following three types of UML relationships:

- g*eneralisation*
- a*ssociation*
- *dependency*

In each case, give an illustrative example of the relationship in the form of an UML diagram.

(12 marks)

b) Define the concepts of "abstract class" and "interface" within the UML notation. Give examples of both concepts within a single UML diagram.

(8 marks)

c) Explain how polymorphism is implemented via the use of an interface.

(5 marks)

Answer Pointers

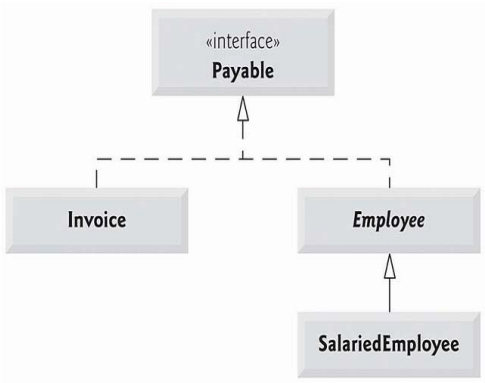
a) In UML modeling, a generalization relationship is a relationship in which one model element (the child) is based on another model element (the parent). Generalization relationships are used in class, component, deployment, and use-case diagrams to indicate that the child receives all of the attributes, operations, and relationships that are defined in the parent.

A dependency relationship is a relationship in which one element, the client, uses or depends on another element, the supplier. You can use dependency relationships in class diagrams, component diagrams, deployment diagrams, and use-case diagrams to indicate that a change to the supplier might require a change to the client.

An association is a relationship between two classifiers, such as classes or use cases, that describes the reasons for the relationship and the rules that govern the relationship.

An association represents a structural relationship that connects two classifiers. Like attributes, associations record the properties of classifiers.

b) An abstract class is a class that can't be directly instantiated. Abstract class exists only for other classes to inherit from and to support reuse of the features declared by it. An interface is a class that has nothing but pure virtual functions.



c) Each interface method must be declared in all the classes that implement the interface. Once a class implements an interface, all objects of that class have an is-a relationship with the interface type, and all objects of the class are guaranteed to provide the functionality described by the interface. This is true of all subclasses of that class as well. Interfaces are particularly useful for assigning common functionality to possibly unrelated classes. This allows objects of unrelated classes to be processed polymorphically

Examiner’s Guidance Notes

Although designed to be a question on basic UML characteristics, less than one in three students attempted this question. Apparently, interfaces and polymorphism are not yet well understood by the candidates for Software Engineering I, since parts b) and c) of this question scored in general very low marks.

Question – A3. (Software Reuse)

A3. a) In the context of *software reuse*, describe the differences between *design for reuse* and *design with reuse. O*utline the main steps for achieving each.

(10 marks)

b) Provide and comment on at least **TWO** scenarios where *software reuse* would not be appropriate and could not be recommended.

(10 marks)

c) In the context of *software reuse*, explain why access to the source code may be desirable and in some cases necessary for the validation of the reusability of a component.

(5 marks)

Answer Pointers

A good answer addresses these points

a) Design patterns are reusable solutions to a known problem in a well-defined context, they are problem-centred, not solution-centred, and complement existing techniques by capturing and communicating “best practice” and expertise. A framework is a library that provides a foundation layer to use to build application without writing everything from scratch framework word can be also address defined schemes of software design and build, a collection of procedures that the developer can follow to achieve better results. A framework is not a pattern. A pattern tells you how to solve a particular design problem. A framework, like Unified Process, SCRUM, Extreme Programming, etc, are methods to productively develop software that cover every aspect in the development process: how to understand the domain of interests, how to design robustly the interfaces, how to exchange ideas among team, best way to achieve a common vision, best way to deal with clients, best way to organize the work, etc...

b) SCENARIO A: The source of software is not credible and might bring potential errors in the components. This is dangerous as reused component might bring numerous bugs to the system and high cost- ineffective.

SCENARIO B: Critical-system is not a good subject of software reusing since we have to have a good understanding of the program flow. Since software reusing is susceptible to compatibility issue, it is not worth to implement it and will bring further time-consuming fix to the system.

c) The candidate should stress the importance of the black box accessibility to source code as a way to understand the inner working of the component to be reused. The validation of the component reusability will ensure that its functionality is the intended behavior of the reused component; such validation will involve the actual testing with other components and as stand-alone.

Examiner's Guidance Notes:

Reuse is always a very popular question attempted by Software Engineering I candidates. Part a) and c) had reasonable solutions in the majority of attempts, while as in question A1, the presence of scenarios produces very different answers by candidates. When not requesting out-of-the-book answers, candidates seem not to particularly like the challenge of producing a convincing rationale for a set of assumptions within a scenario.

Question B4. (V&V Testing)

B4. a) In the context of Software Engineering, define the two terms: *validation* and v*erification* giving examples of each; and discuss the importance of these two activities in the Software Life Cycle.

(8 marks)

b) Discuss the roles that the customer(s) and the software development team play in software validation.

(12 marks)

c) Discuss the feasibility of certifying that a software product is error-free.

(5 marks)

Answer Pointers

a) Validation is answering the question “Are we building the right product?” Example: functionality against promised requirements
Verification is answering the question “Are we building the product right?” Example: reverse-evaluate the design documents in the light of the implemented source code
The V&V phase has two principal objectives

1. The assessment of whether or not the system is useful and usable in an operational situation.
2. The discovery of defects in a system.

b) It should be stressed the importance of continuous feedback by customers as long as prototypes are made available (validation). Verification is carried forward by the software teams. New approaches like the Agile one put an emphasis on customers as part of the team.

c) Certification bodies would need to assume a responsibility in affirming that a component is error-free. The very credibility of such bodies would be under threat since virtually no software, as long as it is evolved, will be error-free, since the set of assumptions with which it was built will need to periodically change

Examiner’s Guidance Notes

The vast majority of candidates attempted this question. Generally they were able to give good answers to parts a) and b) of this question. On the other hand, several candidates proposed that “enough testing” or “careful attention” could be enough for certifying that software was error free.

Question B5. (Software Processes)

B5. a) Explain what CASE tools are and give an illustrative examples of ONE of such tools, classifying it by the *function*, *activity*, and *breadth of support* parameters.

(10 marks)

b) Discuss the use of a common *repository* within a software company that uses different CASE tools, describing the advantages and disadvantages of such approach, and by depicting a diagram exemplifying the interaction of such CASE tools and the named repository.

(15 marks)

Answer Pointers

A good answer addresses these points:

a) CASE stands for Computer Aided (or Assisted) Software Engineering and CASE tools are software systems used by software developers to aid/assist them in their work of developing software and managing software development projects. Upper CASE tools are used in the strategic planning and management of the project; they focus on the upstream activities at the start of the project, e.g. project planning software or requirements management software. Lower CASE tools are used in the down stream activities and support the technical work of engineers designing, developing, and testing the software, e.g. a UML editor, a compiler.

CASE systems can be classified according to their

1. Functionality - what functions do they provide
2. Process support - what software process activities do they support
3. The breadth of support which they provide

As an example, students could provide:

Name	Functionality	Process Support	Breadth
PERT	Planning tool	Specification, Design, Implementation, V&V	Moderate/Good
CVS, SVN, Mercurial, GiT	Version management systems	Design, Implementation	Excellent

b) CASE tools must exchange data throughout the phases of the software development. This may be done in two ways:

Shared data is held in a central database or repository and may be accessed by all CASE tools. Each CASE tool maintains its own database and passes data explicitly to other tools. When large amounts of data are to be shared, the repository model of sharing is most commonly used

Advantages

– Efficient way to share large amounts of data

– Sub-systems need not be concerned with how data is produced

– Centralized management e.g. backup, security, etc.

– Sharing model is published as the repository schema

Disadvantages

– Sub-systems must agree on a repository data model so inevitably a compromise

– Data evolution is difficult and expensive

– No scope for specific management policies

– Difficult to distribute efficiently

Examiner's Guidance Notes

Q5. Although half of the candidates attempted this question, and most were able to spell out what a CASE tool is, very few could propose a classification of such tools based on defined attributes. The importance of a repository was not perceived correctly: part b) requested candidates to illustrate the relationship among the various tools in the same shared workplace, but many candidates failed to produce a convincing answer.

Question B6. (Testing)

B6. a) Distinguish between the following: *unit testing*, *system testing* and *user acceptance testing*, and give illustrative examples of each.

(15 marks)

b) Describe the use of "test drivers" and "test stubs" in the phase of Integration Testing. Explain how they are used in the top-down and bottom-up approaches to integration testing, and provide an example for each approach.

(10 marks)

Answer Pointers

A good answer addresses these points:

a) Unit testing is a software development process in which the smallest testable parts of an application, called units, are individually and independently scrutinized for proper operation, and affects

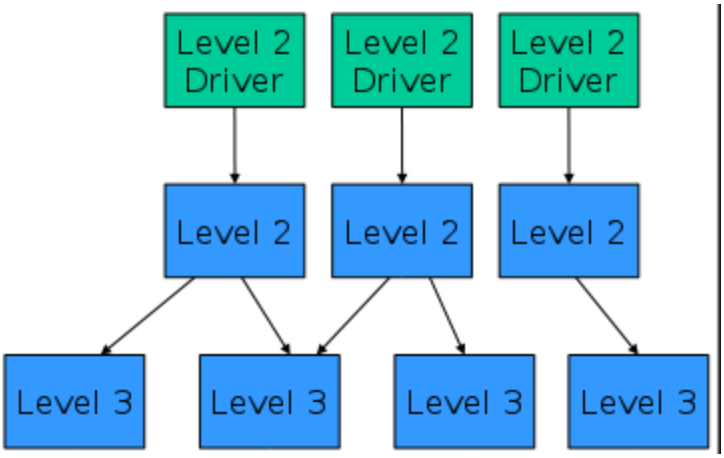
- Testing all operations associated with an object;
- Setting and interrogating all object attributes;
- Exercising the object in all possible states.

Component testing is a software development process where groups of related classes are tested together. This can be achieved by a white-box component testing, or a black box one.

Integration testing is a software development process in which program units are combined and tested as groups in multiple ways. Integration testing can expose problems with the interfaces among program components before trouble occurs in real-world program execution.

b) the bottom-up method and the top-down method are two major ways of carrying out an integration test.
Bottom-up integration testing begins with unit testing, followed by tests of progressively higher-level combinations of units called modules or builds.

In top-down integration testing, the highest-level modules are tested first and progressively lower-level modules are tested after that. In a comprehensive software development environment, bottom-up testing is usually done first, followed by top-down testing. The process concludes with multiple tests of the complete application, preferably in scenarios designed to mimic those it will encounter in customers' computers, systems and networks.



Examiner's Guidance Notes

Q6. This question was by far the most popular and the one that achieved the best results overall. Most students were able to list the differences between the various types of testing. Also, most students could produce a visualization of the top-down and bottom-up approach to system integration testing.