BCS HIGHER EDUCATION QUALIFICATIONS
BCS Level 5 Diploma in IT

March 2012

EXAMINERS' REPORT

Software Engineering 1

General Comments

This is a technical paper about Software Engineering. Questions seek to test candidates' knowledge and their 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 relatively limited understanding of the subject. Most marks for individual questions are in the low range.

Most candidates attempted Q1 and Q3 in booklet A, and Q4 and Q5 in booklet B.

Question A1 – Project and Product Risk Management.

A1.

A small taxi firm has been operating for many years based: the details of the existing customers and the number of completed services (to and from airports) have been regularly saved on disc. The drivers manage all the bookings, and they also keep the details of the transfers to and from airports; the courier services are allocated among the drivers based on their timetable. Once completed, these services are regularly backed up.

The taxi service owner has lately decided to introduce a Content Management System into the business, in order to manage the bookings more efficiently: customers would be able to pay online or by phone; the courier service will have to be booked online; and customers will be able to leave comments on a blog.

Considering the above scenario:

a) List and describe at least THREE possible risks that the service is exposed to in its current business process.

(9 marks)

b) List and describe at least THREE possible risks that the service will be likely to face when the Content Management 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 Content Management System.

(7 marks)

Answer Pointers

a) A good answer indicates these (or similar) risks:

1 – misplacing/loosing/not syncing the bookings by drivers
2 – CD could fail due to hardware problems
3 – timetables not updated regularly by paper-based system, and allocating drivers to multiple and concurrent tasks

b) A good answer indicates these (or similar) risks:

1 – security issues, payment details syphoned out during transactions
2 – failures on the booking systems, wrong confirmation given
3 – blogs used for denigration purposes

c) A good answer focuses on this:

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 (past) size, technology, staff experience, effort, duration, etc could be useful to estimate (future) cost and duration of such project.

Examiner's Guidance Notes

This question tests the candidate's understanding of concepts and practices related to project management and risk analysis. Together with A3, the majority of the candidates attended also this question, with various results. If the type and description of risks in the two scenarios were recognised and analysed by most students correctly, the concept of estimation models, and the core idea that different software engineering techniques can be applied to different scenarios, are still not clear to the candidates, who most of the times refer to a generic COCOMO model, without reflecting on its relevance or applicability with the problem at hand.

Question A2 – Software Engineering Tools and Environments

A2.

a) Consider a program implementing an *instant messaging system* and outline THREE possible types of error that can occur. The errors should be related to different ways in which the software implementing the messaging system could fail, by reproducing and completing the following table in your answer book:

Type of error	Example of this error type in an instant messaging system

(9 marks)

b) For two of the examples given in part a) provide a suitable error message that would be appropriate to inform users in the event of a failure occurring and explain the reasons why this can be considered informative to the users.

(10 marks)

c) Discuss the difference between a software *error*, a program *fault* and a system *failure* illustrating your answers with relevant examples.

(6 marks)

Answer Pointers

a) A good answer indicates these (or similar) types of error:

Type of error	Example of this error type in an instant messaging system
SYSTEM ERROR	the system cannot connect to the requested protocol (yahoo, MSN, IRC, etc) due to the wrong port used
USER ERROR	the user tries to use the wrong codes for emoticons [:=*] that are not recognised and not converted by the application
EXTERNAL ERROR	the user cannot save a file sent through the messenger channel due to insufficient rights on the file system

b) A good answer indicates these (or similar) types of messages to the user:

Case 2: *The combination of codes [:=*] does not convert to any emoticon!. Do you want to use it anyway? Yes/No*

Reasoning: This lets the user know that their choice is unavailable on their system. It indicates that this may in fact be a typing error. It gives them the option of continuing with their original choice which may be available on other systems.

Case 3: *The file cannot be saved in the requested location. Do you want to save it somewhere else?*

Reasoning: This explains what has happened due to insufficient user rights, and proposes an alternative location.

c) A good answer indicates these definitions:

Software error - A software system state that can lead to system behaviour that is unexpected by its users. For example, the user expects the word processor to save all the text they have entered but the software erroneously truncates long lines of text.

Program fault - A characteristic of a computer program that can lead to a software error, for example, a failure to initialise a variable which could lead to the wrong value being used.

System failure - This is an event that occurs at some point in time when the system does not deliver a service expected by its users. A web site crashes when its users place a heavy demand on its services and cannot respond in the time expected.

Examiner’s Guidance Notes

This question tests the candidate’s understanding of concepts and practices related to software engineering tools and environments. Albeit the concepts of error, failure and fault should be pivotal in any software engineering course, it is clear that the candidates do not appreciate the differences among the three concepts. The examples for the errors reported were often wrong, and in some cases not in relation with the situation described.

Question A3 – Software Reuse and Evolution

A3.

a) Discuss the practice of software reuse within the software life cycle and describe at least THREE of its benefits.

(9 marks)

b) In the context of software reuse, give an example of a reusable component.

(4 marks)

c) Identify three possible risks that can occur when a system is built using reusable components and explain how these risks could be reduced.

(6 marks)

d) 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.

(6 marks)

Answer Pointers

a) a good answer addresses this point:

Software reuse involves the reuse of existing software systems or components of a system in the development of a new system. It enables a new software system to be built using software components that have already be tried and tested in other systems.

Three benefits are: increased dependability (tried and tested already), increased productivity (and time to market) as not developing the system from scratch, and effective use of specialised knowledge in a domain, e.g. mathematicians can be employed to develop a standard maths library which can be reused again and again and by non-experts.

b) a good answer provides this (or similar) examples:

COTS = Component Off The Shelf or commercial-off-the-shelf; this refers to readily available software that is available in component form ready to reuse in the development of a new system. COTS may be commercial software, freeware or open source software. There are many “shopping cart” COTS available for reuse in e-commerce systems; they can be found by searching the web.

c) a good answer identifies the following risks:

1. No control over COTS future development – especially a risk if the source code of the component is not available – can be alleviated by obtaining the source code, only using Open Source components, also buying the company producing the COTS or making sure that a contract is made which ensure this is not the case.

2. Lack of support from COTS vendors – COTS vendors may be small companies and they may not be able to deliver the level of support needed by users, even if they have committed to supplying support, their business may change or they may even go out of business. Put in place an agreement if possible to cover these circumstances, e.g. the source is held by a third party and will be supplied if the company goes out of business. Also consider second sourcing important COTS.

3. A product using a number of COTS may suffer from COTS interoperability, i.e. the different COTS may not work together properly. This risk can be lessened by obtaining software documentation explaining the assumptions underlying the design and operation of the COTS software.

d) A good answer addresses this point:

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 behaviour of the reused component; such validation will involve the actual testing with other components and as stand-alone.

Examiner's Guidance Notes

This question tests the candidate's understanding of concepts and practices related to software reuse and evolution. In general, good answers were proposed by the candidates to this question, which was overall the most attempted of both part A and B. Sections A and B were solved correctly in most cases, whereas part D contained the right reflection in most cases. The advantages and disadvantages of reusable components are still not clear in most of the cases, and candidates tend to give conflicting and unstructured answers.

Question B4 – Validation, Verification and Testing

B4.

a) It is important that software is *fit for purpose*. Explain what “fit for purpose” means in this context. Describe various ways in which software developers can try to ensure that software is fit for purpose.

(8 marks)

b) Define the following types of testing and briefly outline scenarios in which each type of testing would be used.

i. Regression testing
ii. Acceptance testing
iii. Unit testing.

(12 marks)

b) Outline the role and nature of testing in *agile* software development. What practices can or should be used, and why?

(5 marks)

Answer Pointers

a) A good answer addresses these points:

(Definition of fitness for purpose) Adherence to stated requirements is one measure of fitness for purpose that has traditionally been used. Payment may depend on delivery of the functionality stated in requirements documents. Software must of course also be sufficiently error-free. However, this approach ignores the problem that stated requirements are often incomplete or incorrect and that requirements may evolve substantially as development proceeds. A better way of assessing fitness for purpose might therefore be to quantify realised benefits from implementing the software; this allows the developer leeway to give the customer what they need, rather than what they ask for. However, it can be difficult to measure realised benefits in an objective way and contracts may preclude this kind of developer-customer relationship.

(How developers can help ensure that software is fit for purpose) Verification and validation (quality control) processes are intended to ensure that software is fit for purpose.

Validation ensures that the developer builds ‘the right thing’:

- If requirements are specified in advance, they should be accurate, complete and workable, insofar as this is possible. Success will be measured according to conformance to stated requirements (rather than realised value, for example).
- Alternatively, developers can use an iterative (agile) development approach. Development is then guided by dynamically-

discovered requirements, which could change. This is likely to provide a result which is more in line with the customer's actual needs. However, the developer cannot guarantee that any particular requirements or benefits will be realised.

- Whichever approach is taken, it is often necessary to design software to fit particular business processes and to adjust business processes to fit the new software. This type of change can be problematic for organisations; many software projects fail because the organisation is unwilling or unable to adapt itself and its processes in the way required to make the software work.

Verification ensures that the developer 'built it right'.

- The software can only be fit for purpose if it is sufficiently free from defects. Therefore developers must do thorough unit, integration, system and regression testing.
- Inspections, walkthroughs and other forms of static analysis can also be used.
- The customer should perform acceptance testing of the software to assess its fitness for purpose (but it is generally expensive to rectify problems at the acceptance testing stage, particularly if they stem from inadequate or inappropriate design).

b) A good answer addresses these points:

i. Regression testing is testing that is redone once a change has been made to a system; this relies on a careful record being kept of tests that are performed so that they can be repeated. For example, in a product development scenario, the release of a new software version for beta testing would normally be preceded by extensive regression testing, often in an automated way. Regression testing is important because modification of software can easily cause it to malfunction, often in unforeseen ways.

ii. Acceptance testing is testing that is done (typically by the end user or customer) to determine the acceptability of a whole system or incremental delivery. Because the focus of acceptance testing is on business acceptability and utility, it is important that it be preceded by careful functional testing by the developer so that the customer does not waste time in encountering and working around bugs and other low-level defects. Because the cost of fixing unacceptable software is high, it is helpful to use iterative development so that acceptance testing can become an iterative or even continuous process, helping to identify issues early and avoid costly rework.

iii. Unit testing is the testing of small units of code, normally in isolation from one another. The definition of a unit in this context depends on the development platform, but an example might be the testing of a particular object class, or package of classes, or the testing of a particular web page. Unit testing can be automated so that regression testing is easier to perform.

c) A good answer addresses these points:

- In agile development, software is developed iteratively and testing practices must be adapted accordingly.
- Each developer must debug, unit test and integration test his/her own code during the process of development.
- In test-first development, the developer writes and may implement test cases before developing code. The code is then repeatedly subjected to the test cases during development. Writing the test cases in advance is said to improve quality because it forces the developer to understand the requirements very well and ask questions in cases where clarification is needed.
- At the end of each day (or period of days, depending on which methods are in use) each developer checks in his/her code. An automated system build may then be performed, including automated system and regression testing. This can help to ensure that defects and conflicts are identified quickly.
- The customer (or a customer representative) is typically involved throughout the development process in order to assist in design decisions and review new system functionality. This amounts to a form of continuous acceptance testing which helps steer development in the right direction, so that expensive surprises are avoided.
- Work proceeds in iterations (e.g. sprints of 3-4 weeks in Scrum) at the end of which version of the system is provided to the customer, who may then conduct more formal acceptance testing.
- In agile development, requirements may change as the project proceeds; this means that test cases may need to change also.

Examiner’s Guidance Notes

This question tests the candidate’s understanding of concepts and practices related to verification and validation and software quality. Part (a) was answered in a limited way by many students, referring merely to meeting requirements or testing. In part (b) most candidates were able to define unit and acceptance testing, although fewer could explain their context of use; many candidates were unable to define regression testing. In part (c) few candidates were able to give a good account of how the use of agile methods affect testing practices.

Question B5 – Software Development Models and Methods

B5.

A health clinic provides medical services to patients in a small town. Five doctors and three nurses work at the clinic; they consult with patients, prescribe medicines and carry out minor medical treatments. Patients with more serious conditions are referred to specialists at the local hospital. A medical information system is being designed for use in the clinic. The system will manage information about employees (doctors, nurses and administrator), patients and their contact details, appointments and consultations, medicines and prescriptions, treatments given, and referrals.

a) Produce a UML class diagram for use in constructing the system using an object-oriented programming language. Your diagram must include all applicable classes and relationships. There is no need to show the attributes and operations for each class.

(12 marks)

b) Explain and give an example of the *generalisation* relationship in UML class diagrams. What construct(s) in code does it correspond to?

(7 marks)

c) Explain the relationships between UML class diagrams and UML use case diagrams.

(6 marks)

Answer Pointers

a) A good answer would include the following:

- Classes representing clinic, employee (sub-classes administrator, doctor, nurse), patient, consultation, condition, condition type, treatment, treatment type, prescription, medicine type, referral, specialist.
- Correct identification of relationships (generalisation for employees and possibly also other persons, aggregation for clinic and classes related to clinic, optional use of aggregation for patient and classes related to patient, other associations).
- Other quality aspects (good class naming, correct cardinalities, absence of many-to-many associations, use of appropriate relationship types, implementation feasibility, UML notation)

b) A good answer addresses the following points:

(Definition of generalisation relationship) The generalisation relationship represents situations where one class inherits the properties of another class. This may reflect a real-world relationship between concepts, as in the example of students and postgraduate students: a postgraduate student is a student and therefore inherits the properties applicable to students (an illustration should be included).

(Corresponding code structures) In code, the generalisation relationship corresponds to object-oriented inheritance. This is typically implemented using a suitable declaration in the class header which refers to an abstract class. For example, in Visual Basic.NET the “inherits” keyword is used and in Java the “extends” keyword is used. The inheriting class may then refer to (and override if necessary) properties defined for the inherited class. Some languages allow a class to inherit from more than one class.

c) A good answer addresses the following points:

- Both types of diagram are used in designing software systems.
- UML class diagrams and UML use case diagrams represent different perspectives on a software system. The elements of a class diagram represent software structures, such as classes, while the elements of a use case diagram represent actors (e.g. end users) and their goals in using the system.
- The terms used in naming and describing actors and use cases may correspond (directly or loosely) to classes, properties or relationships in the class diagram.
- The system must be capable of supporting the use cases as shown in the use case diagram. This means that
    a. the system must be able to store relevant data in respect of each use case; hence the class diagram should contain appropriate classes, attributes and relationships, and
    b. the system must be able to provide the functionality implicit in the use case diagram; hence classes in the class diagram should contain appropriate methods (operations).
- Each class, property and relationship in the class diagram must be used in some way in at least one use case; otherwise it is redundant.

Examiner’s Guidance Notes

This question tests the candidate’s understanding of UML diagrams and the relationship between UML and software constructs. In part (a) a minority of candidates were able to construct a class diagram with suitable classes and relationships. In part(b) most candidates were able to explain generalisation but fewer could say how it is implemented in code constructs. Part(c) asked candidates to give ways in which class and use case diagrams are related; most candidates simply summarised the notations rather than identifying links between specific diagram elements.

Question B6 – Software Engineering Tools and Environments

B6.

a) List the types of software tool available for use in the *development and maintenance of software systems*. Explain how each type of tool is used and what benefits its use can bring. Express your answer using a table as shown overleaf.

Type of tool used	How tool is used	Benefits of tool use
1. …		
2. …		
3. …		
4. …		
5. …		
6. …		
7. …		
8. …		
etc.		

(8 marks)

b) List the types of software tool available for use in *planning, analysis and design of software systems*. Explain how each type of tool is used and what benefits its use can bring. Express your answer in a table similar to that used for part (a) above.

(8 marks)

c) The software tools used in system planning, analysis, design, development and maintenance offer various forms of integration. Explain how different tools may be integrated and what capabilities this integration makes possible. Express your answer in the form of a diagram showing the different software tools and types of integration available between them, annotated to explain the capabilities that result from integration.

(9 marks)

Answer Pointers

a) A good answer addresses some or all (e.g. 6-8) of the following tools used in system development and maintenance.

Tools available	**How used**	**Benefits**
IDE (integrated development environment) such as Eclipse or Visual Studio	Offers intelligent editing; allows code base to be browsed, navigated, searched, compiled, executed, etc.	Requires less expertise than command-line operation; offers various integrated tools and plug-ins
Compilers e.g. javac, Visual C#	Checks code for syntactic correctness and turns it into binary executable form	Allows code to be run; essential part of developing software

Source/revision control software e.g. CVS, Subversion	Maintains integrity of code base (developers check code out and in); complete change history maintained	Allows tracking of changes, automated system build, multiple branches, reverting to previous release, etc.
Development frameworks e.g. Spring, Struts, Ruby On Rails	Provides suitable code structure for many applications, and reusable code and code templates	Avoids the need to “reinvent the wheel”; provides a common approach for teams of developers
Project management and tracking software e.g. MS Project, GanttProject	Progress on development and maintenance tasks is tracked at a detailed level	Allows project manager to control development activities as they proceed
Code generators e.g. Apache Velocity, CodeSmith	Builds code automatically	Speeds up development process, reducing rote coding tasks; provides a helpful structure to code
Build automation tools e.g. make, Ant	Automates the process of compiling, assembling and deploying applications	Allows complex applications to be built and deployed quickly and automatically
Automated testing tools e.g. JUnit, WinRunner	Allows tests to be run automatically in various ways on individual code units, groups of units or whole systems	Permits frequent regression testing; facilitates test-first development; assists in development of comprehensive test cases
Issue/bug tracking systems e.g. Zoho, Bugzilla	Allows problems to be reported and info about their history and correction to be maintained	Helps organisation keep track of user problems and software defects and to manage the process of addressing them
Code analysis tools e.g. JDepend, NDepend, Doxygen	Produces code metrics, diagnoses, documentation and visualisations such as dependency analyses	Large legacy codebases can be difficult to understand and modify; code analysis makes it easier

Refactoring tools e.g. ReSharper (tools are often built into IDE)	Allows existing code structures to be automatically transformed in various ways	Helps make code simpler, and therefore easier to modify, or better prepared for addition of new features

b) A good answer addresses some or all (e.g. 6-8) of the following tools used in planning, analysis and design.

Tools available	**How used**	**Benefits**
Project management and tracking software e.g. Microsoft Project, GanttProject	Analysis, design, development and maintenance tasks are broken into small units that are assigned to individuals	Allows project manager to plan project activities in detail
Issue/bug tracking systems e.g. Zoho, Bugzilla	Allows problems to be reported and info about their history and correction to be maintained	Useful in planning implementation and maintenance activities
Code analysis tools e.g. JDepend, NDepend, Doxygen	Produces code metrics, diagnoses, documentation and visualisations such as dependency analyses	Useful in planning enhancement and maintenance activities (which constitute the majority of software development)
Process modelling tools e.g. Business Process Visual Architect	Allows business processes to be expressed and analysed	Helps analyst understand business context, requirements; helps organisation plan for IT-related business changes
Data modelling tools e.g. ERWin	Allows information requirements and structure to be expressed and analysed	Helps analyst understand business context, requirements; helps organisation state info needs; gives designer basis for data design

Behaviour (state) modelling tools	Allows relationships between business processes and data resources to be analysed in detail	Helps analyst understand business context, requirements; gives designer deeper understanding of functional and data requirements
Object modelling/ object-oriented design tools e.g. Rational Rose, Visual Paradigm for UML	Allows software architectures and structures to be designed in advance of coding	Allows designer to specify high-level architectures and detailed class structure of software
Prototyping tools e.g. Visual Basic, Microsoft Access	Allows prototype systems to be constructed quickly to test designs/elicit requirements	Helps analyst and customer understand requirements and agree on designs
Visual design/ wireframing tools e.g. WireframeSketcher, Creately	Allows user interfaces to be designed visually	Helps analyst and customer understand requirements and agree on designs
Office productivity tools e.g. Microsoft Word, Visio, Excel, Outlook	Allows specifications to be written and communicated	Essential since much development activity relies on documents of various types
Collaboration environments e.g. SharePoint	Allows team to interact and collaborate more easily even if not physically co-located	Improves teamwork; provides centralised project "memory"
CASE repository	Theoretically, stores all analysis, design artefacts in an integrated manner	Provides centralised support for analysis and design (and maybe development) activities

c) A good answer includes the following.

- A diagram, with a box or circle for each different tool type (e.g. process modelling tool, behaviour (state) modelling tool, data modelling tool, object modelling tool, code generation tool, refactoring tool, code analysis tool, integrated development environment (IDE), automated testing tool, compiler, build automation tool, source version control system, issue/bug tracking tool).

- Lines between boxes representing linkages between specific tools, with explanation of the nature of the linkage (how integration is achieved) and a summary of what this type of integration makes possible. Example: an *automated testing tool* (such as JUnit) may be integrated with an *IDE* (such as Eclipse) as a plug-in. The integration allows developers to run unit tests on specific pieces of code repeatedly, quickly and easily during development. It facilitates agile and test-first development methods.

Examiner's Guidance Notes

This question tests the candidate's knowledge of the software tools used in software engineering. In part (a) many candidates were able to list a good number of different tool types although fewer could explain well how they are used and their benefits. In part (b) few candidates listed more than two or three types of tool and many referred merely to CASE without explanation. Although part (b) asked for the answer in table form, many candidates ignored this. Part (c) was attempted by few candidates and only a minority drew a diagram as requested.