THE BCS PROFESSIONAL EXAMINATION
Diploma

April 2004

EXAMINERS' REPORT

Software Engineering 1

General
This was the first setting of this examination, with mostly satisfactory results.

Candidates mostly avoided the temptation to write a lot of descriptive material and focussed on the evidence-based reasoning that the examiners were seeking.

Question 1
1. The cost of maintenance for the development and deployment of a software product has often been linked with the cost of quality. Give your reasons for this link with the cost of quality. **(25 marks)**

Answer Pointers
The question expected a balanced answer, starting with description/explanation of cost of maintenance, then same approach for cost of quality, then some synthesis of views to test for linkage.

Pressman Ch 20 remarks on cost of maintenance rising to expected 80% of software budget during 1990s. Factors driving this are direct costs of the mini-lifecycle analyse, evaluate, design code, re-implement and the indirect costs of customer dissatisfaction, degradation of performance by 'pathological' errors (e.g. propensity of Win95 and Win98 to virus attack), and delay to other programmes as resource is diverted away to maintenance. Candidates are not expected to quote from sources texts. Up to 5 marks.

Both **Pressman** Ch 2 and **Sommerville** Ch 24 locate maintainability, portability and efficiency as critical quality attributes that are rarely listed in any specifications document. An aim of quality management is to develop functional software with one or more of these key product attributes embedded, as appropriate. **Sommerville** quotes **Crosby**'s definition that "a quality product is one that meets its specification". Candidates are not expected to quote from sources texts. Up to 5 marks.

This analysis should leads to a realisation that much hands-on quality is located during the development phases, while the lifetime cost of the product runs away in the maintenance phase because lifetime quality attributes have been neglected during development. Up to 7 marks for linkage of conclusion to previous discussions.

The suggestion is that if lifetime quality attributes were managed into the development phase, this could be achieved for less cost that retrospectively repairing during maintenance. Is this a realistic suggestion? What measures exist to test developing product for acquisition of lifetime quality attributes? Up to 8 marks for quality conclusion linked to previous discussion.

Examiners' Guidance Notes
Answers that dealt in a 'one hand this, other hand that, and this is synthesis' way with a traceable argument that is supported by textbooks or experiences earned high marks.

Few answers followed the model above. Many answers failed to link development costs with maintenance costs, and instead interpreted both costs as incurred during the development phase.

Not many answers followed the ‘debating’ style; first describe features of element-1 (in this case, Cost of Maintenance), the features of element-2 (cost of quality), then a comparison of these features , either points of similarity or points of difference. Most answers dealt summarily with a description of the two costs without drawing any conclusions about possible linkages. While a survivable exam technique, the examiner would like more confidence that candidates can perceive some wholeness to the life cycle model that must include total lifetime costs for a software product.

Question 2

2. *a)* Explain what is meant by the phrase *paradigms of software engineering*. **(9 marks)**

b) Explain the purpose of each of the following in the software engineering process. Give an explanation for a SUCCESSFUL outcome and also an UNSUCCESSFUL outcome. What are the consequences of a successful outcome? What are the consequences of an unsuccessful outcome?

i) Review **(8 marks)**

ii) Validation **(8 marks)**

Answer Pointers

a)

Pressman (Ch 1) uses the phrase ‘paradigms of software engineering’ in the sense of modelling an abstraction that represents the processes of software development and software product lifecycle. **Sommerville** (Ch3) divides software process models into ‘one-pass’, such as Waterfall model, evolutionary development, re-use-oriented development; and ‘iterative’, such as incremental development, and Spiral-model development. Pressman (Ch1) goes on to include 4GL ‘generator’ models.

So, paradigms of software engineering means the set of different ways in which a disciplined software development process can be described and still make sense with the tools used, the functionality developed and the expectation of the customer.
Candidates are not expected to quote from sources texts. Up to 9 marks for a reasoned description with examples.

b)

The purpose of a **software review** is to examine the outputs from a specific activity or development phase against the requirements laid on the activity/phase at the start to test for satisfactory achievement of outputs from inputs. 3 marks.

A **successful outcome** of a review means that the outputs are baselined as ‘achieved’ and become the input definitions for the next activity or development phase. 2 marks.

An **unsuccessful outcome** of a review means that the outputs do not meet their expected performance. This results in a rework of the activity or phase at extra cost in time and money to repair the outputs so that they meet the specifications laid on them at the start. 3 marks.

The purpose of **validation** is to test user requirements against built functionality and attributes of the software product. 3 marks.

A **successful outcome** of Validation is customer pays final bills. Consequence is income received by developer. 2 marks.

An **unsuccessful outcome** to validation is that developer embarks on rework at zero cost to client in order to repair product to meet customer specification. Consequence is that developer's income is delayed. 3 marks.

Examiners' Comments
Answers here show that students find bookwork easier than developing examples under exam conditions. The question required three definitions and one explanation, but also three examples associated with the definitions. A popular question with good results.

Most candidates associated 'paradigm' with 'software process development model' and earned good marks in section a).

However, very few candidates took a 'process' view of the Review event. Most could describe what it was, and many could describe the result of a failure to pass a Review in the development process. Strangely, few appreciated the positive side of a successful Review in the development process.

For a Validation review, very few candidates appreciated the effects, both positive and negative, on developer's income if a Validation events is successful or not. The inevitable linkage of a client with a Validation event ties income to a successful result.

Question 3

3. *a)* In the context of software design, define what is meant by the following design principles and illustrate each with an example piece of design:

i) transparency, also known as information hiding
ii) abstraction, and
iii) modularity **(21 marks)**

b) Explain what is meant by the concept of *requirements traceability* with respect to software design. **(4 marks)**

Answer Pointers
a)
i) Transparency, or information hiding, is an implementation of 'need to know' (see **Sommerville** Ch 18.1). Only those who need a specific piece of information for their duty are given that information. In software design, this most frequently refers to separation of function from implementation. The user can call the function name but cannot interfere with the implementation of that function.

For example, the Java definition of a queue makes available what it can do without revealing how it is done:

```
Interface Queue {
        public void put Object o);
        public void remove (Object o)
        public int size ();
                } //Queue
```

Up to 7 marks for definition and illustration.

ii) Abstraction is described in **Pressman** Ch 10.3 as a technique of layering a problem. At higher levels of abstraction, a solution is stated in broad, non-specific, terms using the language of the problem environment. At lower levels of abstraction, there is a more procedural expression of how the solution is implemented. In software design, subsystems are frequently specified at a high level of abstraction in order to define their function without compromising their implementation.

For example, (see **Sommerville** Ch 9.2) an abstract subsystem definition will tell
Data types or object classes available
Names for each
Operations on each
Syntax for using each
Axioms for semantic description of each.

A candidate should show either this abstract level of answer, or a detailed answer showing how such an interface might be represented in a programming language. Up to 7 marks for definition and illustration.

iii) Modularity (see **Pressman** Ch 10.3) is a technique of software architecture whereby software is divided into separately named and addressable components, called modules, that are integrated (after development) to satisfy problem requirements. Modularity is credited with being the single biggest contribution to intellectual management of the complexity of a software product.

To illustrate modularity, a candidate is expected to draw some form of module breakdown chart. Up to 7 marks for description and illustration.

b) Requirements traceability is the capability to track all design components and deliverables that result from a specific requirement specification (forward tracking). Also it can mean the ability to identify which requirements generate any given deliverable (backward tracking).
Up to 4 marks for a reasoned answer.

Examiners' Comments
Answers here show that students find bookwork easier than developing examples under exam conditions. The question required 3 definitions and one explanation, but also three examples associated with the definitions.

The requirement in the question to develop examples under exam conditions may have put some students off even attempting this question. Even with the bookwork, some students simply picked up on keywords and reproduced inappropriate notes about design in general.

As the associated examples required more thought and effort, more marks were awarded for them. Thus, an answer without examples could only achieve a maximum of 10 marks as each example was worth 5 marks.

Students need more practice studying and producing examples of design concepts. Chapter 10 of Pressman contained all the bookwork for part a) of this question along with examples. Going beyond Pressman's examples was expected for the student to receive top marks. While the material is covered by Sommerville, it is not all covered in a single chapter.

Question 4

4. *a)* Describe the features and capabilities of a CASE tool with which you are familiar. Give an example of a CASE tool with which you are familiar and describe how some of the features and capabilities are exhibited. **(10 marks)**

b) Give TWO examples where this CASE tool has shifted the engineering emphasis of the development paradigm in which it is used. **(15 marks)**

Answer Pointers

a)
CASE stands for Computer Aided Software Engineering; it can be used to mean any computer-based tool for software planning, development, and evolution.

A description of any particular tool that clearly shows the candidate's appreciation of method-supporting or lifecycle-supporting tool will be accepted (**Pressman** Ch1, p24). Alternatively, (**Sommerville** Ch 3.7) CASE can be analysed by function, by process and by integration.

Pressman's set of likely characteristics follows. **Sommerville**'s is longer.

Structured Analysis (SA), Structured Design (SD), Editors, Compilers, Debuggers, Edit-Compile-Debug environments, Code Generators, Documentation Generators, Configuration Management, Release Management, Project Management, Scheduling, Tracking, Requirements Tracing, Change Management (CM), Defect Tracking, Structured Discourse, Documentation editing, Collaboration tools, Access Control, Integrated Project Support Environments (IPSEs), Intertool message systems, Reverse Engineering, and Metric Analysers.

Up to 10 marks for a fully-featured answer that showed life-cycle appreciation.

b)
The core shift is away from coding towards other human-intensive areas of development. Examples of shifted paradigm (**Sommerville** Ch 3.7) include

- Support for individual tasks such as consistency checking, compiling, test results comparisons.
- Workbench support such as drawing tools for specification and design with integration capability to check or infer correctness.
- Environments that package a substantial part of the development into a disciplined and defined methodology.

Any two choices with illustration of the shift of emphasis gets full 15 marks. Less are graded down accordingly.

Examiners' Comments

Part a) was well answered. Many candidates drew from experience to describe CASE tools with which they were familiar. A few candidates confused 'tool' with 'methodology', for example asserting that SSADM was a development tool, or that structured programming was a development tool. However, on the whole, this part was well answered.

However, in part b), few candidates could formulate two distinct examples of any change in working practices caused by the ready availability of tools. Candidates must reflect more from their learning on to their practical experience of *how* they develop software.

Question 5

5. *a)* Because software is so easy to change, for example by a few keystrokes, software configuration management is often made difficult.

Explain why software is so easy to change, and give TWO examples of your interpretation with reference to software development and maintenance. **(10 marks)**

b) The following is an outline specification for a small project. Discuss the criteria you would use to determine the life cycle model that this project should follow, and make a recommendation about selecting a suitable life cycle model. You should assume you would have high-productivity development tools. **(15 marks)**

"This project is about developing visual and graphical web-presentations to show the client what graphical effects are possible with a web site.

"The client owns some houses for renting, preferably to contractors such as consultants and other 'long-stay' business managers for durations of 3-6 months at a time. More and better views of the houses and their interiors will increase the number of rentings.

"The client also wants to develop other lines of business to increase his income. He has an idea to start workshop events about teaching skills such as making glass models or knitting complex designs. He suggests the website should also have ways to illustrate and promote this with such things as downloads of patterns, diagrams, hints-and-tips or other knowledge-intensive 'free' giveaways as part of rewarding visitors to come to the site and be told about – frequently refreshed – dates for workshop events."

Answer Pointers

a)

'Easy to change' is a characteristic rooted simply in the speed of editing, compiling, linking etc. of software – especially assembler code and 3GL code - to form a piece of functionality. Another way of expressing this is that software engineers don't 'produce' software (as does a Production Engineer) but they clone their prototypes.

Root of the answer is flexibility/speed coupled with electronic filing systems coupled with on-screen editing.

Examples (**Pressman** Ch 20 p678) any two from

- Subprogram deleted or changed
- Statement label deleted or changed
- Identifier deleted or changed
- Changes made to improve execution performance
- File open or close sequence is modified
- Logic operations are modified
- Design changes are cascaded into major software adaptations
- Changes made to logic operations at boundaries of data.
- Redefined global or local variable value
- Redefined file or record format
- Increment or decrement to an array size
- Change of global data
- Re-initialisation of flags and pointers
- Re-arrangement of arguments in a subprogram parameter list

5 marks for any two.

One set of criteria might be stability of requirements and productivity of tools.

b)
In the scenario given, requirements are not stable (the job to do is vague). but the tools available are productive. The vague work consists of the look-and-feel of the website, and the range of services it offers. Such HCIs are known to be hard to get right, so the issue to management is getting user involvement in approval of the chosen web designs.

Most lifecycle ‘methods’ are variants of incremental development, so the question becomes which incremental development method suits ‘getting the users involved’? A series of exploratory design-build-test episodes, such as evolutionary prototyping, seem to fit the bill.

A balanced argument choosing suitable criteria and analysing the context as given against them, then identifying issue(s) of concern, then identifying a life cycle that has an ability to cope with these issues would earn marks as follows:
4 marks for the criteria (2 each)
4 marks for analysing the features of the scenario in contact of the criteria (2 marks each)
and finally, 4 marks for identifying a life cycle model and giving the rationale (2 marks each).

Examiners’ Comments
In part a), few students appreciated that a simple keystroke is enough to change a software component. Every item in Pressman’s list in the model answer is caused by a few keystrokes that immediately remove any confidence that has been built by testing because the integrity of the tested software has just been destroyed!

In part b), few answers followed the detail of the model answer, though many candidates produced alternative, valid answers. A valid answer was one that defended choice of lifecycle by a discussion of criteria in the context. Some answers gave a solution without any reasons drawn from the context. Some answers gave reasons for choosing a lifecycle model, but these reasons were nothing to do with the scenario provided in the question.

Some answers could not come to any conclusion, and instead discussed several lifecycle models that might suit and refused to decide on any one of them.

Question 6

6. *a)* Briefly describe TWO issues of product quality that should be considered during the planning process of a software development for EACH of the following, with your reasons.

i) stable requirements, low productivity development tools[1] **(9 marks)**
ii) unstable requirements, high productivity development tools[2] **(9 marks)**

Notes
[1] *Low productivity means tools that require intensive hand-crafting, like assembly language and 3GL programming systems without specific support for requirements engineering.*
[2] *High productivity means tools that provide specific support for requirements engineering, or software generator environments.*

b) For the situation in *ii)* above, identify the roles for two other people whose skills complement those of a software engineer. Give your reasons. **(7 marks)**

Answer Pointers
a)
Sommerville Ch 24.2 echoes the difficulty of a ‘raft’ of quality attributes and recommends selection of critical attributes.

i) stable requirements, low productivity development tools'. Clearly, this is a slow development against defined targets. 4 marks.

The attributes to manage tightly belong to the original requirements specification. Following this or similar reasoning, any two attributes of function or performance will be accepted. 5 marks.

ii) unstable requirements, high productivity development tools. Clearly, the speed of development becomes part of the requirements definition/convergence process. 4 marks.

Attributes to manage here are emergence of customer-agreed business-facing requirements and incremental build-up of functionality to deliver these requirements. Following this or similar reasoning, any two attributes of form or function will be accepted. 5 marks.

b)
Aside from the expert in the toolset, the team (**Sommerville** Ch 22.2) should contain a domain specialist, someone who knows what is wanted (perhaps someone from the customer group), and an integration/test specialist, someone who knows the failure modes of the products from the toolset and is probably equivalent in skill to the toolset expert but acts as a reference or test agent. Alternative expressions of these roles are communication specialist (client person) and test specialist (engineer person). Up to 7 marks for two roles and descriptions.

Examiners' Comments
Many candidates addressed this question well, understanding the need for requirements control when using low productivity, but few had insight into client-relations if using high-productivity tools.

Most answers realised the need for documentation specialist and a testing specialist, which were very appropriate answers.