THE BCS PROFESSIONAL EXAMINATION
The Professional Graduate Diploma

April 2000

EXAMINERS' REPORT

System Design Methods

Question 1
Answer Pointers

(i) ***Several design methods make use of data flow diagrams. Name the four main symbols that are used to construct a data flow diagram and provide an illustration and brief description of the meaning of each of the symbols.***

(6 marks)

Expect process, external entity, data store, data flow (or other common variants of these terms). A valid symbol (from one of the DFD variants) for each notational element is expected.
The assessment will include correct naming, and correct use of the symbols.

(ii) ***Consider the following simple warehouse system:***
A warehouse stocks batteries of various different types. A simple computer system is used to maintain a record of the current stock level of each type of battery. Batteries are delivered to the warehouse, and shipped from the warehouse in batches. The stockman is responsible for entering delivery and shipment information into the computer system as batches arrive or are shipped. From time to time the warehouse manager needs to find out the current stock level associated with a particular type of battery. She does this by querying the computer system.

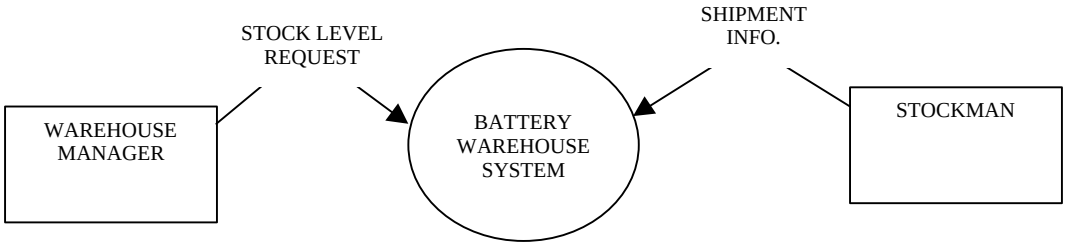
(a) Produce a context diagram (also known as a fundamental system model, context model or level 0 data flow diagram) to provide a high level overview of the battery warehouse system.

(5 marks)

(b) Produce the corresponding level 1 data flow diagram for this system. Make sure that your diagram is consistent with the context diagram given above.

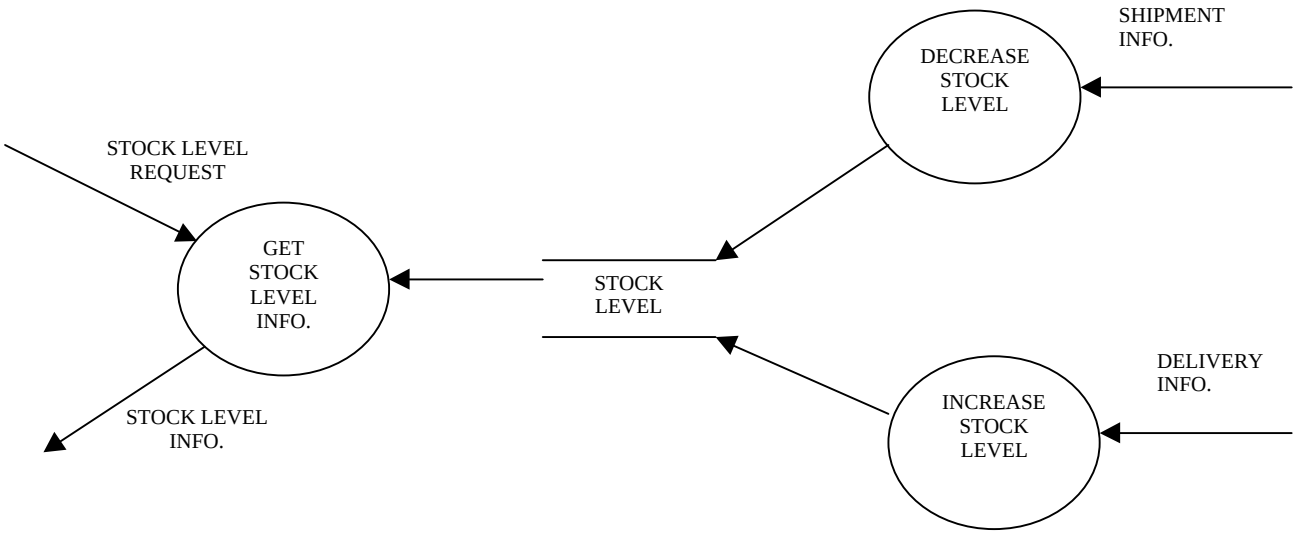
(8 marks)

(a) Something along the lines of the following is expected (any valid DFD notation is allowed, consistent with their answer to part (i)):



The candidate should pay attention to the correct usage of symbols and layout.

(b) Something along the lines of the following is expected, with consistent notation and balance with answer to (i) and (ii):



Consider alternative models on own merits. Must balance with context diagram. If context was wrong, but level 1 is sensible with respect to it, then award credit.

The candidate must be careful with the identification of the valid components and the layout.

(iii) Briefly discuss the scope and limitations of using data flow diagrams alone for system modelling, and indicate how other diagram notations can be used to provide graphical system models from different perspectives.

(6 marks)

DFDs are good for modelling information transformation and flow. Good for systems that fit this paradigm naturally (e.g. transaction processing), but not good for other types of system (e.g. real-time, database). Expect discussion of other notations that model systems from different perspectives, e.g. STD for dynamic behaviour, ERD for static relationships, real-time extensions to DFDs to model control, OO perspective, etc

Examiner's Guidance Notes:

- Most candidates seemed to have a reasonable grasp of the basic notation, although some confused the notation with that of flow-charts
- Some candidates got the symbols for external entities and processes confused
- The context diagram was done well by most, although some candidates provided more than one process

- Flow and process names were not always well chosen. Some candidates chose names for flows that had verbs in them, and so some of the flow names read more like processes (i.e. “doing things”)
- Several candidates provided more than one data store, which was not a problem in itself, however some of the DFDs became over complicated which resulted in a lack of model clarity
- Many of the answers to part (iii) indicated that candidates were not clear that different types of diagram are used to represent systems from different perspectives. Some stayed focused on DFDs and did not demonstrate clear knowledge of other diagram notations and their uses.

Question 2

Answer Pointers

Software design is an essential process of software engineering and can be performed with an appropriate design method. A typical design process covers four aspects: data, architecture, interface and procedural design.

(i) Describe the objectives of each aspect of the design and the activities conducted to achieve them.

(14 marks)

Data design: transforms the information domain model created during analysis into the data structures. The data objects and relationships defined in entity-relationship (or class) diagram and the data content depicted in the data dictionary provide the basis for the data design.

Architectural design: defines the relationship among major structural elements of the program. The design representation, the modular framework of a computer program, can be derived from the analysis model(s) and the interaction of subsystems.

Interface design: describes how the software communicates within itself, to systems that inter-operate with it, and with humans how use it.

Procedural design: transforms structural elements of the program architecture into a procedural description of software components.

(ii) State the techniques you would choose (e.g. structured or object-oriented techniques) for each aspect of design, and justify your choice.

(11 marks)

Data design: entity-relationship diagram, data dictionary. Architectural design: data flow diagram. Interface design: data flow diagram. Procedural design: state transition diagram. Appropriate selection of OO techniques is also acceptable.

Examiner's Guidance Notes:

This question examines the candidate's understanding of the design methods and techniques. Strong candidates demonstrated the rationale of choosing certain design techniques.

Question 3

Answer Pointers

(i) With the aid of illustrations describe the classic Waterfall model of a software process. Provide a brief but clear definition of the meaning of each process stage included in the model.

(7 marks)

Expect the following stages in linear sequence: analysis, design, code, test.
The candidate must identify the stages at correct point in sequence, and provide valid definitions. Extra credits will be given if maintenance is also included.

(ii) Give details of the support that is provided by a particular software development method for each of the distinct Waterfall lifecycle stages. Base your answer on one method with which you are familiar.

(6 marks)

Answer based on own experience. Give 1 mark for each valid point made. Must refer to all lifecycle stages identified (even if to indicate no support) to get full marks.

(iii) Briefly discuss the problems associated with the Waterfall model's idealistic view of a software process. Use examples from your own experience to support your answer.

(7 marks)

Expect problems along the lines of:

- Real projects are rarely linear
- Difficult to state requirements correctly and clearly at the start
- Working program not available until late in the lifecycle
- Possibility for blocking states where project team members are waiting for other members to complete dependent tasks
- Etc…

The answer must reflect on own experience for full credit.

(iv) Explain possible ways in which the traditional Waterfall model could be improved to make it reflect a more realistic and less idealistic software process.

(5 marks)

Ideas could include adding iteration (for a more evolutionary approach) and feedback loops to the model; de-emphasising the linearity; introducing prototyping; RAD approach for short projects; etc…

Examiner's Guidance Notes:

- Most candidates had a reasonable idea of what the waterfall stages are and how they can be represented as a diagram
- Candidates were least clear on the distinction between analysis and design
- Most candidates chose SSADM as an example software development method to illustrate the waterfall model. Some failed to indicate that they had considered each waterfall stage when relating to the method of their choice.

- Some candidates failed to fully appreciate the broad range of problems associated with the Waterfall model's idealistic view and several candidates did not relate the Waterfall problems to their own experience
- Several answers to part (iv) lacked vision on how to improve the waterfall model

Question 4

Answer Pointers

Many legacy systems are critical to the operation of the organisation that uses them. Software re-engineering must be performed on these systems to ensure that they continue to satisfy the organisation's requirements.

(i) Compare the notions of "forward software engineering" and "software re-engineering", and discuss the need for software re-engineering.

(14 marks)

A forward engineering is a conventional system development process. It starts from systems specification, to design and implementation. The outcome is a new system.

Software re-engineering is to produce an improved system. It starts from an existing software system, to seek for understanding and perform transformation (or re-engineering). The outcome is an improved (or re-engineered) system.

The legacy systems need to be re-engineered to enhance their functionality and performance in order to meet the business requirements. The re-engineered system is expected to meet the changing requirements, and should be more maintainable. Sometimes, the need for systems re-engineering is derived from the business process re-engineering, as the legacy system may incorporate implicit dependencies on the existing business processes.

(ii) One of the tasks in software re-engineering is to convert data from various sources in different data structures into a managed environment. This is sometimes called data conversion or data re-engineering. Identify and explain TWO problems that may be encountered in data conversation.

(11 marks)

- Data naming problems, typically synonyms (different names given to the same logical entity in different programs) and homonyms (the same name used in different programs to mean different things).
- Field length problems, the same item may be assigned different lengths in different programs or the field length may be too short to represent current data.
- Record organisation problem, typically in COBOL (but not in languages like C++ or Ada where the physical organisation of a record is the compiler's responsibility).
- Hard-coded variables: absolute values, such as tax rates, are included directly in the program rather than referenced using some symbolic names.
- No data dictionary.
- Inconsistencies in data values, e.g. inconsistent default values, inconsistent validation rules, inconsistent measuring units, inconsistent representations, inconsistent handling of negative values, etc.

The candidate is required to identify any two of these problems with an explanation of the problems.

Examiner's Guidance Notes:
For part (i), many candidates demonstrated an understanding for two types of software engineering, for which the basic marks were awarded. A few strong candidates discussed the need for software re-engineering with illustration of examples, which gained extra marks.
For part (ii), the difference in the answers by the candidates lied in the explanation of the problems. Those, who elaborated the problems with examples, showed their understanding and ability of analysis, gained good scores.

Question 5
Answer Pointers

(i) Explain what is meant by the term "metric" when used in the context of a software development process and indicate three outcomes from a software development process that can be used to derive process metrics.

(5 marks)

Metric – a quantitative measure of the degree to which a process possesses a given attribute. Sets of metrics are based on the outcomes that can be derived from the process. These include measures of errors uncovered before release, human effort expended, schedule conformance, etc…
The candidate must give explanations of metrics and valid outcomes.

(ii) Software process metrics can provide significant benefits in terms of software process maturity; however, if metrics are misused they can cause more problems than they solve. Indicate potential problems that metrics misuse could cause and suggest a list of guidelines that a manager should adhere to when instituting a process metrics programme.

(8 marks)

Potential problems: use of metrics for individual evaluation/appraisal, potential obsession with one or two metrics to exclusion of others, inadequate choice of metrics could give an inaccurate impression, …

Guidelines ("software metrics etiquette")

- Use common sense and organisational sensitivity when interpreting metrics data
- Provide regular feedback to those who have worked to collect measures and metrics
- Don't use metrics to appraise individuals
- Never use metrics to threaten individuals or teams
- Metrics that indicate a process problem should not be considered negative – quite the opposite – indicator for process improvement
- Don't focus on a single metric, to the exclusion of all others
- …

This is available in the texts, but a good candidate should be able to provide a number of common sense points by reflecting on their own past experience.

(iii) Software maintainability is one of the most important quality metrics. There are a number of technical and non-technical factors affecting maintainability. Identify THREE such factors and suggest on how maintainability can be improved with respect to these factors.

(12 marks)

- Module dependence. It should be possible to modify one component of a system with out affecting other system component.
- Programming language. Programs written in a high-level programming language are usually easier to understand (and hence maintain) than programs written in a low-level language.
- Programming style. The way in which a program is written contributes to its understandability and hence the ease with which it can be maintained.
- Program validation and testing. Generally, the more time and effort spent on design validation and program testing, the fewer errors in the programs. Consequently, corrective maintenance costs are minimised.
- The quality of program documentation. Program maintenance costs tend to be less for well-documented systems than for systems supplied with poor or incomplete documentation.
- The configuration management techniques used to keep track of all system documents and to ensure the consistencies. Effective configuration management can help control the cost of maintenance.
- The application domain, clearly defined and well understood domains are likely to lead to complete requirements, which leads to less perfective maintenance.
- Staff ability.
- The age of the programs.
- The dependence of the program on its external environment. A program dependent on its environment must be modified as the environment changes (e.g. changes of a taxation system might require payroll, accounting and stock control programs to change).
- Hardware stability.

The candidate is required to identify any THREE factors and suggest (as appropriate) how to improve the maintainability in these respects.

Examiner's Guidance Notes:

Part (i) and (ii) examine the candidate's ability of understanding and applying metrics in software engineering projects. Part (iii) offered the candidate an opportunity to demonstrate how quality metrics are used in software maintenance.

- Most candidates illustrated a reasonable understanding of what a software development metric is, although some were unsure about what outcomes from a software development process could be used to derive process metrics
- Most candidates indicated some valid problems that can be caused by metrics misuse, although several did not provide an adequate list of clear management guidelines to guard against metrics misuse
- Strong candidates were able to identify the relevant metrics and quality issues, and discuss the functions of appropriate application of metrics in depth