Examiner's report 2009

CIS226 Software engineering, algorithm design and analysis – Zone A

Introduction

The CIS226 examination paper has been designed to examine your knowledge of the many different topics introduced in the unit.

Of course, the topics that feature in the exam paper were chosen by the examiner to assess students' knowledge this year, and will not necessarily be the same topics featuring in next year's exam. Also, even if the same topics are featured, the questions could address different aspects or issues. So make sure you have good knowledge of the full spectrum of topics included in the curriculum for this unit.

A good understanding of the basic concepts, ideas, methods and techniques is expected from you, and the proper use of terminology and notation is particularly important.

Some of the questions are straightforward bookwork type question that can be answered basically by using material from the subject guide. However, you can also use material from other software engineering educational sources to enhance and supplement your answers. This can demonstrate greater knowledge and deeper understanding, and may lead to the award of higher marks.

When answering, take note of the mark weighting of the question (section or subsection), and size your answer accordingly. To save exam time, generally use the tactic of short direct answers when the question carries small mark weighting. However, elaborate more if there is a hint in the question that the Examiner is looking for discussion or justification.

Section A

General remarks

This report deals with only Section A (Question 1, Question 2 and Question 3) of the exam paper, and the following are not 100 per cent model answers; rather they are indicative statements showing how to answer the questions in a better way. Also, for each question there is a summary of the common pitfalls/mistakes that candidates experienced/made while undertaking the examination. However, it is worth mentioning here that, in general, most mistakes and incomplete answers normally occur as a result of poor preparation and revision.

Question 1

This question has many sections that are based on the description of a proposed time plan to develop a database and therefore are related. Read the description carefully before attempting to answer the sections. Your answers should be direct and to the point, and their length should be relevant to the mark weighting of each section.

Problems with this question seem to have resulted from candidates not reading the description carefully, which resulted in choosing the wrong model (d), which subsequently affected their answer for following sections. Other problems included using the wrong notations and wrong sequence of stages for the UML state diagram in (g), and confusion over how to represent the state diagram in (j).

a) Requirement capture/Requirement analysis.

b) Tests that system functions on target hardware with acceptable speed (and/or memory use).

c) Design and/or analysis, implementation, testing.

d) Waterfall.

e) Development would have to over-run or buggy system would have to be deployed.

f) Mainly no, because inflexible, poor at recovering from bugs, requirement drift or changing user requirements; user feedback only at the end.

   If you prefer yes, because clear and explicit timing for development stages and for overall delivery, limits time for clients to interact with engineers, so useful when client time is limited or expensive, potentially faster turnaround for short project.

g) States are rounded rectangles, pseudostates are present and correctly drawn, states are appropriate (including requirement capture/analysis, analysis and/or design, implementation, testing, deployment/maintenance), ne-way (or, optionally, two-way) transitions should be correctly shown and are only to neighbours.

h) Do activity in state box (underneath state name) deadline check. Exactly one unlabelled transition to next state. The same effect can be achieved with a looping transition and a date guard – no more than one mark given for these unless wholly correct.

i) Any of: Iterative, Incremental, Agile, Extreme Programming, Proto-typing or Hybrid.

j) For Iterative, this can acceptably be a loop from an appropriate late stage to an appropriate early stage with a guard against infinite looping.

k) For Iterative approaches: more engagement of client/user; encouragement of modular development; more responsive to bug fixes; more flexible against requirement drift.

Question 2

This question is based on a description of use case, and like Question 1 requires careful reading of the description and precise, direct answers to the sections. For answers that requires a diagram (e.g. (V) of section (a)) make sure you use the appropriate notation and clear drawing and labelling.

a) For this section the main mistakes came from representing the extensions required for (ii), like missing the numbering, wrong numbering, and wrong or unclear description. Also, picking out wrong candidate classes in (iv) or mislabelling them as suitable for ruling out. Common problems in answering (v) were producing an activity diagram instead of a sequence diagram, missing to represent the reader in the diagram, wrong sequence in message passing, and using wrong drawing notations.

  i. Reader, fetcher, issue desk, librarian.

  ii. 4a: Too many items on request

    1: System displays warning and returns user to request list.

   9a: Too many items at the desk

    .1 System warns librarian

    .2 Librarian gives reader the option of returning a book or cancelling.

  iii. Reference to (or inclusion of) another use case.

  iv. Reader; Request list; Book; Desk; Barcode; Catalogue; book storage area; system. For these if crossed out: pass number; password; fetcher; conveyer; shelf; (librarian at) issue desk.

  v. Sequence diagram should contain classes Catalogue and Request list and showing Reader. Messages: login(pass no, password), add items(items) and order books(seat no).

b) Most problems that affected answering all three parts of this section came from the confusion between the types of association and what they represent. Also, using wrong notation in producing the diagram for (iii).

  i. Generalisation/specialisation (all inheritance): Library item is generalised version of (super class of) the others.

  ii. Composition: Journal issue is a component part of Journal. Score ½ if aggregation instead of composition.

  iii. Article is to Journal Issue as Journal Issue is to journal (i.e. a filled diamond on the Issue side of the line connecting the two). Multiplicity should show a 1 on the Journal side and either just a * or a 1..* on the other.

Question 3

This question is a book work type question and has many sections and sub-sections. Take extra care in clearly labelling your section answers. Answer directly, briefly and precisely and draw on material found in the subject guide.

A general common problem in answering this question was over description/long definitions which affected (a) to (d). Also, there was widespread confusion between Verification and Validation in (a). In section (e) the main problem was including non-integers and not testing for sign in (i) and making the wrong calculations in (ii) and (iii). Common mistakes in section (f) were wrong names for diagram (i), mistakes in following the testing suite (first and second) in (ii). Most other mistakes were obviously related to poor revision.

a) Verification: Checks product against specifications / evaluates whether it is what it is supposed to be; check design against requirements; check system against design; checks that compilation runs.

   Validation: Tests that product works; includes system testing, performance, acceptance and usability testing. (Other testing types name can count, if not all in the same general category.)

b) A recurrence of an existing bug or a bug in a previously working piece of functionality. Avoided by regression testing – building and running tests for existing desired functionality and/or for previously fixed bugs.

c) Responds to knowledge about code used; tests implementation by looking at structure; often involves finding edge and corner cases. Tests functionality rather than implementation; can be defined in advance; tests against specs.

d) Rewriting code, while preserving functionality, making easier to read. No credit for performance optimisation. Refactored code is easier to debug.

e) i. No non-integers. Should include tests of sign, of large numbers and possibly zeros, and should be varied (not all testing the same thing for the same variables).

   ii. Combination for which $4ac>b^2$. Making b too high to square within normal integer spaces. Making all values 0.

   iii. A=0

f) i. Activity diagram.

   ii. The test suite is run the first time to check whether the functionality in the new tests is already present or the tests are badly cast. If the tests are passed, then new tests are needed. The second testing is to check whether the new code adds the new functionality and doesn't break old functionality. If the tests are failed, more programming is needed.

   iii. Yes.

   iv. Requirements churn occurs when requirements change as a project progresses. This model allows system engineers to add new functionality as the project progresses. This diagram does not allow for changing tests (only addind), which limits the ability to handle changes in requirements, but this is easily fixed.

   v. Adaptive.

Section B

General remarks

This year we have included some unseen questions in the examination. We found most candidates prefer answering routine questions even though the unseen questions are not necessarily more difficult than others.

A good answer would include clearly marked question numbers before each answer. The answer to a new question should start on a new page.

We discuss the questions one by one as follows:

Question 4

a) This is an unseen question and only a handful of candidates attempted it. It is understandable that the choice was made due to the desire to achieve better marks, but it may also be due to the lack of confidence of some candidates. It was a great pleasure to see the few candidates who attempted this question gave a very good answer to this question. A good answer to this question would consist of three parts, as the question suggests. You should adopt the section numbers in the question.

   i. The first part should include a definition of the problem. The key point is to know that two integers (x, y) can be used to define a point in a two dimensional space. The Euclidean distance of two points can be computed easily between the two points. A good answer would then include an instance of the problem and a special case, for example, there should be an error if $x<0$, or $y<0$.

   ii. A good answer would first suggest a data structure with an explanation on why you think the data structure is suitable. You also need to demonstrate the design process and the final algorithm.

   iii. To analyse the time complexity of your algorithm, you need to write all the steps in order to demonstrate.

b) A good answer to this part of the question would simply consist of a correct algorithm with a sufficiently clear structure and comments. Note that you are not expected to write an algorithm from the top line to the end instantly. Instead, you should include all the relevant intermediate stage of work, demonstrate all your rough work in algorithm design, then cross it out if it is not for marking.

   An easy approach to answer this question is to first work out the input and the output of the algorithm, perhaps with the aid of an example graph, or adjacency lists. You may then work on a flowchart. It would then be easy to derive the algorithm from the flowchart.

Question 5

a) This is a straightforward question asking to distinguish two concepts: **algorithms** and computer **program**. A good way to answer a question like this one is to first define or explain the two concepts and then highlight the differences between them. It would be wise to give only a concise answer to this part of the question because it only carries a maximum of 2 marks.

b) This is an unseen question. You should make sure that you understand the question and plan your answers before writing. There are many ways to answer this part of the question. However, a good answer would demonstrate sufficient understanding of the technical issues involved in the argument.

A good answer would consist of following sections: first, a yes or no answer to clarify whether you agree with Professor John or not. Next, state your views clearly. Then develop your argument by further discussion or explanation and, finally, provide a good example.

Note that a good answer needs to demonstrate a good understanding about the given argument by Professor John. Otherwise, your argument and the support evidence may run the risk of leaving the topics. Some candidates lost some marks this year due to misunderstanding the argument.

c) This part of the question requires a diagram that shows the original pixel block. Most candidates answered this question well this year.

d) A good answer to this part of the question would consist of three sections. First, a definition in your own words about the greedy approach. You then need to explain what a matrix multiplication problem is, and further describe briefly how the problem can be solved by a greedy approach.

e) An easy way to answer this part of the question is to demonstrate the relative positions of the patten to the text in each step, starting from the beginning position of the text and until a match of the pattern is found (in this case, or until the end of the searching).

A good answer also should highlight the characteristics of the required algorithm. For example, the comparison is conducted character by character and beginning from the most right character in the pattern instead of left. The shift can be from 1 character to the length of the entire pattern.

Question 6

a) This part of the question is about the computational complexity. The examiner would expect two sections in the answer to this part of the question: first, a sentence of definition; secondly a brief analysis of the time complexity in the worst case for the given algorithm. A good answer would follow precisely the other requirements such as definition in one sentence only, and specify the basic operations used.

b) An easy way to answer this part of the question is to first compute the hash value for each of the given numbers in the given order. Note that the order of your hashing matters because of the collisions. A good answer would also summarise the final hash table.

c) A good answer to this part of the question would include a correct algorithm that fulfils the given task. Since it is often not straightforward to write an algorithm, you are expected to include some design sketches in your answer. You may always cross out the rough work if you like, but intermediate design work is just as important as the final results, because the corrected ideas or approaches may be credited, especially if the algorithm is flawed.

d) The answer to this part of the question can be less straightforward. You need to understand certain concepts, such as a binary search tree, and know how a tree can be traversed before being able to derive the answer. If you understand that an **inorder** traversal on a binary search tree will return a sorted list in ascending order, you may simply write down the recursive inorder traversal algorithm as the answer. A good answer will also include necessary explanation or comments.