Examiners' commentaries 2016–17

CO2226 Software engineering, algorithm design and analysis

Zone A

General remarks

The examination was set generally as a mixture between questions that test your basic understanding of the subjects, and questions that require you to apply knowledge and demonstrate deeper understanding. You are reminded to read each question carefully and address all aspects of the question.

The paper is perhaps unusual in that it covers two separate (though loosely related areas): software engineering as well as algorithms.

This examination was therefore split into two parts. Part A assesses the software engineering and analysis material, Part B the algorithms material. Each part has three questions, of which two are to be answered (therefore four questions answered in total; two from A and two from B).

You should be sure to number your answers clearly and correctly. Though Examiners will do their best, if we cannot read or make sense of what is written it makes it very hard to award marks.

Furthermore, if you wish Examiners to disregard an attempt at a question, the answer should be crossed out. Examiners are directed to mark only the first two answers given for each section.

You should also put a single line through any rough work, so that it is not inadvertently marked.

All questions require a mixture of bookwork (e.g. explaining terms, giving definitions etc.) and application (e.g. calculations, building UML models), with the emphasis on application. You should therefore work on practising likely calculation/application tasks.

For Part A (software engineering) the key application skill is to produce a (usually UML) diagram from a scenario. To save reading, the same scenario may be shared between questions. You are advised to practise producing diagrams well and quickly. The detailed discussion below will highlight areas where marks might be lost.

For Part B, some brief explanatory comments on answers involving pseudocode would be helpful, as some answers were unclear.

You should also note that some questions may explicitly instruct to show your working and give reasons for your answers. You are advised that it is generally sound practice to show working on any questions that involve calculation or the application of a process. This applies particularly in Part B where a mistake midway can lead to a wrong answer in many questions. If you simply write down what you think the answer is and it is wrong, then you will gain no marks. If Examiners see working, they can see what you understand and may be able to award partial marks.

Question 1

This question involved bookwork; candidates were asked about the relationship between UML sequence diagrams and use case diagrams. This was followed by the usual task to generate a software engineering model from a scenario task in the domain of a membership discount website (in this case, UML class diagrams).

Part a) as noted above was bookwork. Candidates were generally able to explain what a sequence diagram was. Where candidates were asked to say how they correspond with use case diagrams some answers became vague.

A few short paragraphs would suffice for a good answer here. Some candidates either gave very terse answers or wrote long essays.

In part b) the quality of class diagrams was variable. There were a number of areas where candidates lost marks.

- In general, candidates were able to capture most of the scenario.
- Some candidates produced diagrams with few classes (5-6 is definitely too few), and/or omitted key classes.
- In general, sensible proposals were given for class attributes and methods.
- More commonly, many candidates did not fully illustrate associations, composition and aggregation relationships between the objects, despite these being explicitly requested in the question.

Hint: the scenario is designed to give you a chance to show you can use all of the above, so make sure you take it.

The construction of class diagrams seems to be a consistent weakness compared with other UML diagrams. You should prepare fully for these.

Question 2

This question started with a bookwork question on the meaning and differences between <<include>> and <<extend>> in the context of use case diagrams, followed by a scenario task (use case diagram, extending the scenario from 1b).

Part a) was not answered well. There was confusion about what the concepts meant and as a result the examples given were not effective. Again, a few paragraphs would suffice here.

Part b) was generally answered well, though there were some common errors:

- In general candidates were able to capture most of the scenario.
- A few candidates gave use cases (as text) rather than use case diagrams. It is hard for Examiners to give credit if the question is not answered as stated. Please read the question carefully.
- Some candidates produced diagrams with missing actors (hint: sometimes time can be modelled as an actor).
- Generally, candidates were able to put most of the use cases on the diagram.
- More commonly, candidates failed to fully illustrate include, extend, and generalisation relationships between the use cases, despite these being explicitly requested in the question.

Hint: the scenario is designed to give you a chance to show you can use all of the above, so make sure you take it.

Question 3

This question comprised a bookwork question about packages in UML diagrams and then the development of a state machine diagram based on the same scenario as the previous two questions.

Part a) was generally not answered well, with many vague answers. Examples were given in the stronger answers and sometimes lifted the quality of the written explanation. Again, a few paragraphs would suffice for a good answer.

Part b) was generally answered well. State machines seem to be an area of relative strength for candidates. Areas where candidates lost marks included the following (in descending order of seriousness):

- Sometimes flow diagrams were given instead of UML state diagrams. Note: Examiners can only give credit for the tasks specified in the question.
- Important states missing.
- Flow of control not matching the scenario.
- Missing operations on transitions.
- Errors in notation.

Question 4

This question aimed to test candidates' understanding of linked lists and their use in stacks and queues. The question is a combination of tasks, plus some bookwork.

Part a) was generally well answered – most candidates gave a valid and correct answer. A few sentences would have sufficed for a good answer.

Part b) in most cases was answered very well though some quadtrees were represented in rather interesting ways (how it is described in the Subject guide is good practice).

Part c) tended to be vague and confused. You are advised to understand how octrees work and how they relate to quadtrees.

Part d) asked candidates to execute the Boyer-Moore algorithm correctly, and most did this well. Marks were lost for not clearly indicating comparisons, and sometimes skipping stages. The example in the Subject guide is an excellent model for what an answer should look like.

Part e) asked candidates to describe a binary heap, its properties and application. Most candidates were able to provide a description, and give its properties (albeit with varying degrees of clarity). The examples given were often not clearly stated, hence losing easy marks.

Part f) asked candidates to draw a binary heap. The vast majority did this well, though some marks were lost where no short explanation was provided.

Question 5

This question aimed to examine candidates' higher level understanding of abstract data types and hashing.

Part a) answers were often disappointingly confused – the key is to distinguish between and abstraction and its concrete implementation. The examples sometimes managed to mitigate the confusion, but often did not. This is an important distinction for you to understand.

In part b), most candidates were able to give sensible reasons as to why ADTs were useful in software development.

Candidates gave good answers to part c) and often were able to explain why hashing is useful.

Part d) was tackled well by most candidates. A large minority did not understand modulo arithmetic – you must ensure you can do this. Marks were lost where working was not shown clearly, and some candidates tried to resolve collisions when this was not asked for (though usually this did not result in loss of marks unless we could not see the unresolved state).

Part e) was answered consistently well by candidates.

Part f) was answered well by most candidates, though a minority gave a different resolution method by mistake.

Part g) was answered surprisingly poorly by candidates, given that it was rather straightforward.

Question 6

This question aimed to test candidates' understanding of Huffman coding.

A majority of candidates were able to answer part a) satisfactorily, most importantly getting across the assumptions behind how it achieves text compression.

Some candidates were unable to provide the pseudocode asked for in part b); this is covered in the Subject guide.

The majority of candidates answered part c) poorly as they were unable to produce a tree that would achieve compression. Often the basic process was unclear, so Examiners were limited in awarding marks for evidence of understanding.

Some candidates were unable to provide the pseudocode asked for in part d); this is covered in the Subject guide.

In part e) candidates were not penalised twice for errors they made in part c). Candidates who could show how their tree in c) could be applied gained marks depending on the clarity of their answer.

Candidates were generally able to make sensible points in part f).

Examiners' commentaries 2016–17

CO2226 Software engineering, algorithm design and analysis

Zone B

General remarks

The examination was set generally as a mixture between questions that test your basic understanding of the subjects, and questions that require you to apply knowledge and demonstrate deeper understanding. You are reminded to read each question carefully and address all aspects of the question.

The paper is perhaps unusual in that it covers two separate (though loosely related areas): software engineering as well as algorithms.

This examination was therefore split into two parts. Part A assesses the software engineering and analysis material, Part B the algorithms material. Each part has three questions, of which two are to be answered (therefore four questions answered in total, two from A and two from B).

You should be sure to number your answers clearly and correctly. Though Examiners will do their best, if we cannot read or make sense of what is written it makes it very hard to award marks.

Furthermore, if you wish Examiners to disregard an attempt at a question, the answer should be crossed out. Examiners are directed to mark only the first two answers given for each section.

You should also put a single line through any rough work, so that it is not inadvertently marked.

All questions require a mixture of bookwork (e.g. explaining terms, giving definitions, etc.) and application (e.g. calculations, building UML models), with the emphasis on application. You should therefore work on practising likely calculation/application tasks.

For Part A (software engineering) the key application skill is to produce a (usually UML) diagram from a scenario. To save reading, the same scenario may be shared between questions. You are advised to practise producing diagrams well and quickly. The detailed discussion below will highlight areas where marks might be lost.

For Part B, some brief explanatory comments on answers involving pseudocode would be helpful, as some answers were unclear.

You should also note that some questions may explicitly instruct to show your working and give reasons for your answers. You are advised that it is generally sound practice to show working on any questions that involve calculation or the application of a process. This applies particularly in Part B where a mistake midway can lead to a wrong answer in many questions. If you simply write down what you think the answer is and it is wrong, then you will gain no marks. If Examiners see working, they can see what you understand and may be able to award partial marks.

Question 1

This question involved bookwork; candidates were asked about the relationship between UML sequence diagrams and state diagrams. This was followed by the usual task to generate a software engineering model from a scenario task in the domain of a membership house-move website (in this case UML class diagrams).

Part a) as noted above was bookwork. Candidates were generally able to explain what state and sequence diagrams were. Where candidates were asked to say how they correspond with each other some answers became vague.

A few short paragraphs would suffice for a good answer here. A significant minority of candidates gave very terse answers, while a few wrote long essays.

In part b) the quality of class diagrams was variable. There were a number of areas where candidates lost marks.

- In general, candidates were able to capture most of the scenario.
- Some candidates produced diagrams with few classes (5-6 is definitely too few), and/or omitted key classes.
- In general, sensible proposals were generally given for class attributes and methods.
- More commonly, many candidates did not fully illustrate associations, composition and aggregation relationships between the objects, despite these being explicitly requested in the question.

Hint: the scenario is designed to give you a chance to show you can use all of the above, so make sure you take it.

The construction of class diagrams seems to be a consistent weakness compared with other UML diagrams. Candidates should prepare fully for these.

Question 2

This question started with a bookwork question focusing on where use case diagrams are employed in the software development process, followed by a scenario task (activity diagram, extending the scenario from 1b).

Part a) was not answered well. This is a basic question, and candidates should be prepared to explain at what stage a given UML diagram is used and why. Again, a few paragraphs would suffice here.

Part b) was generally answered well, though there were some common errors:

- Generally, candidates were able to capture most of the scenario, though some diagrams were missing swim lanes.
- More commonly, candidates failed to fully capture concurrency and flow of control between actors in their answers.
- In general, candidates should pay particular attention to identifying activities, decision points and fork/join nodes.

Hint: the scenario is designed to give you a chance to show you can use all of the notation, so make sure you take it. In particular, pay attention to identifying activities, decision points and fork/join nodes.

Question 3

This question comprised a bookwork question about sub-states in UML diagrams and then the development of a state machine diagram based on the same scenario as the previous two questions.

Part a) was generally not answered well, with many answers being vague. Examples were given in the stronger answers and sometimes lifted the quality of the written explanation. Again, a few paragraphs would suffice for a good answer.

Part b) was generally answered well. State machines seem to be an area of relative strength for candidates. Areas where candidates lost marks included the following (in descending order of seriousness):

- Sometimes flow diagrams were given instead of UML state diagrams. Note: Examiners can only give credit for the tasks specified in the question.
- Important states missing or poorly named.
- Flow of control not matching the scenario.
- Missing operations on transitions.
- Errors in notation.

Question 4

This question aimed to test candidates' understanding of optimisation concepts applied to a specific optimisation problem. The question is a combination of tasks, plus some bookwork.

The optimisation concepts are covered in the Subject guide, but the question set the context of a new example/scenario. This was answered by relatively few students and generally poorly.

There is very limited point in learning the concepts/solution for one optimisation problem. To demonstrate understanding, you should be able apply these concepts across any properly described optimisation problem.

Part a) was generally answered in a vague and confused manner, suggesting that these candidates did not know what a greedy algorithm was or why it can be useful. A few short sentences would have sufficed for a good answer.

Part b) in many cases was answered poorly. Candidates often did not demonstrate understanding between the difference between decision/optimisation and NP-complete/hard problems, though some did have an idea of the link between NP and intractability.

Part c) only a few candidates got right, indicating a wide misunderstanding of how to set up depth-first search.

Part d) was answered well only by those candidates who answered part c) well.

Only in part e) were answers better, with candidates showing some understanding of what their previous answers entailed.

Question 5

This question aimed to examine candidates' higher level understanding of graphs and sorting lists.

Part a) answers were often good, though marks were lost by those candidates who misunderstood what an adjacency list is (the Examiners took a broad view here, but recommend that the topic should be included in revision).

In part b) most candidates were able to give sensible answers, though easy marks were lost where steps were not shown.

The answers to part c) were sometimes unclear. Examiners saw a lot of answers that either missed key stages or were a blur of swap operations, lacking clarity on how the algorithms work.

The problem with such displays of swap operations is that it can be hard to distinguish between the actual algorithm or one of the candidate's own devising. Examiners made every effort to do so, and kept in mind there may be alternative implementations of a sort algorithm.

Also, for sorting algorithms, showing a correct result at the end does not give Examiners evidence of understanding – the correct end point is obvious.

Candidates are therefore strongly advised to show the intent of their algorithms clearly, giving a few sentences to explain the key features of the algorithm's approach (e.g. pivots in quicksort). This will help demonstrate understanding for maximum marks.

Part d) was tackled well by most candidates. Some marks were lost for limited bookwork understanding, or not looking at average case or mostly sorted lists.

Question 6

This question aimed to test candidates' understanding of iteration/recursion and binary (search) trees.

A majority of candidates were able to answer part a) satisfactorily. The iterative version sometimes contained basic errors (suggesting that versions were memorised in at least some of these cases).

Candidates were mostly able to explain what tail recursion was (with some varying degrees of clarity) and correctly identify whether it applied to part a).

The majority of candidates answered part c) well, though the properties of the BST in some cases could have been described more clearly.

Candidates were able to provide the iterative pseudocode asked for in part d) in a minority of cases, though they were able to provide the recursive version. It appears that some candidates are unable to write code in both forms and may in some cases be memorising pseudocode without understanding. This is not a good strategy, and the VLE discussion boards are ideal for raising such issues.

In part e) candidates consistently gave good answers.

In general, candidates were again able to give good answers in part f).