Examiners' commentary 2017–2018

CO2226 Software engineering, algorithm design and analysis – Zone A

General remarks

The exam was set generally as a mixture between questions that test the candidate's basic understanding of the material, and questions that require the candidate to apply their knowledge and demonstrate deeper understanding. Candidates 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 exam 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).

Candidates should, however, be sure to number their answers clearly and correctly. There were some cases where it was unclear. Though examiners will do their best, if we cannot read or make sense of it, it makes it very hard to award marks.

Furthermore, if candidates wish an attempt at a question not to be marked, they should indicate this clearly (for example, by crossing it out). Examiners are directed to mark only the first two questions from each part. There were a few instances of this.

Again, examiners would appreciate it if candidates were to cross rough work through so 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, etc.) with the majority being application. Candidates should 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, the same scenario possibly being shared between questions. Candidates should ensure that they can do this well and quickly. We hope that the discussion below will help candidates to be mindful of where marks can be easily missed.

Some brief explanatory comments on answers involving (pseudo)code in Part B would be helpful, as some answers were unclear.

Also, candidates would do well to show working on questions that involve calculation or the application of a process. This applies doubly in part B where a mistake midway can lead to a wrong answer in many questions. Where this has happened often will be highlighted later.

If the candidate simply writes down what they think the answer is and it is wrong, then they will get no marks. If we see working, we can see what the candidates understand and they may gain partial marks.

Candidates should also note that mark schemes often explicitly allocate marks to showing working and giving reasons for their answers (and questions may ask for this).

Comments on specific questions

Question 1

This involved a question where candidates were asked about the verification of UML class against sequence diagrams. This was followed by the usual generate a software engineering model from a scenario task in the domain of helpdesk software (in this case, UML class diagrams).

Part (a), as noted above, was bookwork. Candidates were generally able to explain what a sequence diagram was (not specifically asked for). But when the question of verification came up, it was then that the answers sometimes became vague.

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

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 the majority of the scenario.
- Some candidates produced diagrams with few classes (5-6 is definitely too few), and/or omitted key classes.
- Sensible proposals were generally given for class attributes and methods.
- More commonly, candidates failed to fully illustrate associations, composition and aggregation relationships between the objects, despite these being explicitly requested in the question.

Please note, 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 lose marks needlessly by not preparing fully for it.

Question 2

Question 2 started with a bookwork question on the meaning and differences between an extension use case and an alternative flow in a use case diagram, 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. Candidates were able to capture much of the scenario, but common errors included:

- A few candidates gave use cases (as text) rather than use case diagrams. It is hard for examiners to award marks if the question is not answered as stated. Candidates are therefore advised to read the question carefully.
- Some candidates produced diagrams with missing actors (hint: sometimes time can be modelled as an actor).
- More commonly, candidates failed to fully illustrate include, extend, and generalisation relationships between the use cases, despite these being explicitly requested in the question.

Generally, candidates were able to put at least most of the use cases on the diagram.

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 part about the concept of polymorphism in Object-Oriented Design, and how and why it can be useful. This was the development of a state machine diagram based on the same scenario as the previous two questions.

Part a) was generally answered adequately, although some answers were 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 (in descending order of seriousness):

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

Question 4

This question aimed to test candidates' understanding of optimisation and computability/intractability concepts applied to an optimisation problem (the well-known travelling salesman problem). The question is a combination of tasks, plus some bookwork.

This question covers the optimisation concepts given in the study guide, but in the context of a new example/scenario. This was answered by relatively few candidates and often poorly.

There is no point knowing these concepts for one optimisation problem, as candidates will be unable to apply them in the real world, so candidates need to apply these concepts across any properly described optimisation problem.

Part a) was generally answered in a vague and confused manner indicating that candidates did not know why computability research makes use of abstract models.

In answers to part b), unfortunately only a minority of candidates managed to clearly distinguish and describe the concepts of decidability and intractability.

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

Only a minority of candidates did well at part d), indicating a misunderstanding of how to set up depth-first search. The greedy algorithm was usually executed well, however.

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

Question 5

This question aimed to test candidates' understanding and application of stack and queue data structures. As usual, the question is a combination of tasks, plus some bookwork.

Part a) was generally answered very well, with a good use of an example.

Part b) – most candidates managed to describe this implementation well; though some explanation alongside pseudocode would have strengthened the answer in some cases.

Part c) in most cases was answered very well. Candidates showed good understanding of this concept.

Part d) almost all candidates did well at, indicating a good understanding of how queues work

Question 6

This question aimed to test candidates' understanding and application of recursion, divide and conquer and dynamic programming. As usual, the question is a combination of tasks, plus some bookwork.

Part a) was generally answered well, through some answers were confused despite it being covered in the subject guide.

Part b) most candidates managed to describe this implementation well, though some explanation alongside pseudocode would have strengthened the answer in some cases.

Part c) in most cases was answered very well. Candidates showed good understanding of this concept.

Part d) almost all candidates did well at, indicating a good understanding of how merge sort works.

Parts e) and f) were often described well, though some answers were rather unclear with a fair amount of confusion about the nature and advantages of a dynamic programming approach (e.g. bottom-up, reuse of earlier calculations). That said, some candidates were able to give classic examples such as Fibonacci.