Examiners’ commentaries 2015–16

CO2226 Software engineering, analysis and algorithms – Zone A

General remarks

The examination was based on a mixture of questions that test basic understanding of the material, and of questions that require candidates to apply their 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 and analysis, as well as algorithms.

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

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

Furthermore, if you wish an attempt at a question to be ignored, you should indicate this clearly (e.g. striking through – a single line is sufficient). Examiners are directed to mark only the first two questions in each part.

Similarly, examiners appreciate it if you strike through rough work so it is not inadvertently marked. (Leave it legible so that examiners can refer to it if your points are not fully made in your answers.)

All questions require a mixture of bookwork (e.g. explaining terms, giving definitions) and application (e.g. calculations, building UML models), with the majority of marks awarded for 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. (The same scenario may be shared between questions.) You should ensure that you can do this well and quickly. We hope that the discussion below will help you be aware of where marks can be easily missed.

For Part B, some brief explanatory comments for questions involving (pseudo)code in Part B would help clarify answers.

You should show working on 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, you will not gain any marks. If we see the working, we can see what you understand, and you may gain partial marks.

Comments on specific questions

Part A

Question 1

This question involved bookwork; candidates were asked about the purpose of UML activity diagrams. This was followed by the usual task to generate a software engineering model from a scenario (in this case, UML class diagrams).

Part a), as noted above, was bookwork. Candidates were generally able to explain what an activity diagram was. However, where candidates were asked to say in which part of the software development process activity diagrams were used, the answers often became vague and confused (in a few cases listing a number of stages in scattergun fashion).

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

In part b), candidates overall did fairly well. However, there were two areas where candidates lost marks.

- Some candidates produced diagrams with few classes, or omitted key classes.
- More commonly, candidates failed to fully illustrate associations, aggregations, and generalisation relationships between the objects, despite these being explicitly requested in the question.

Question 2

This question started with a bookwork question on static operations and attributes, followed by a scenario task (sequence diagram, extending the scenario from Question 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. Common errors included:

- flow of control not matching scenario
- diagram at too high a level, lacking detail
- errors in notation
- poor selection of lifelines
- incomplete/omitted sets of operations.

Question 3

This question comprised bookwork about white and black box testing and development of a state machine based on the same scenario as the previous two questions.

Part a) was generally answered well in terms of definitions. But answers were weaker in terms of explaining the differences. Examples were given in most cases. Again, a few paragraphs would suffice for a good answer.

Part b) was generally answered well. Areas where candidates lost marks included (in descending order of seriousness):

- sometimes flow diagrams given instead of state diagrams
- important states missing

- flow of control not matching that in 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 paragraph and a diagram would have sufficed for a good answer.

Part b), though in many cases good answers were given, some answers were vague, confused or incomplete. Two paragraphs would have sufficed for a good answer: one each on the advantages and disadvantages of linked lists versus arrays.

Part c) contains a few subtasks. Candidates did a good job overall of explaining what a stack is, albeit confusing LIFO with FIFO. A paragraph with a diagram would suffice to get the idea across.

Candidates were generally also able to describe the array implementation of a stack. For this, a diagram with one or two paragraphs of explanation, and perhaps some pseudocode was a common feature of good answers.

Part d) most students knew what a queue was, though LIFO and FIFO were sometimes confused. There were a number of strong attempts at this question. A walkthrough of the process with a diagram was a feature of most good answers.

The most common error in an array implementation was to neglect pointers to the front/rear of the queue.

Question 5

This question aimed to examine candidates' higher-level understanding of hashing and especially collision detection and resolution.

Part a) answers started well – most candidates gave a short description of what linear probing is. Good answers gave a simple example.

In part b), a few candidates demonstrated a resolution algorithm that was different from the one that the question asked for. (Candidates should note that examiners will use the terminology given in the subject guide, not alternatives.)

Otherwise, common errors were in calculations or errors/lack of clarity in the steps of applying the algorithm. A number of candidates showed weakness in doing simple modular arithmetic.

Despite the above, most candidates answered this question well.

Candidates gave variable answers to part c), and some answers were rather unclear, with some confusion about the nature and criteria for a recursive algorithm (e.g. the need for a base case/stopping criterion). That said, many candidates were able to provide pseudocode of classic examples such as factorial or Fibonacci.

Part d) was challenging for many students, which suggested a lack of in-depth understanding of this concept – some left it blank or gave a few lines of code. Others gave non-recursive algorithms. Good answers also provided some explanation of the code.

Question 6

This question aimed to test the candidates' understanding of sorting algorithms (shell sort) and graphs.

A minority of candidates were able to answer part a) satisfactorily. Some candidates showed the operation of a different sorting algorithm; others lost marks by missing key stages. Good answers comprised of a full walkthrough of the sort, plus a paragraph of explanation.

Part b) was generally well answered. A minority of candidates unfortunately decided to explain what a Cartesian (X–Y) graph is, which indicates prior mathematical weakness.

The majority of candidates answered part c) well. Some candidates lost marks for:

- failing to produce both a graph and a digraph
- errors in translation of the matrix to a (di)graph, often missing that some nodes were self-connected
- failing to produce an adjacency list (or using the wrong format).

Good answers also included a description of the process of how the above were created; one or two paragraphs would suffice.

Examiners' commentaries 2015–16

CO2226 Software engineering, analysis and algorithms – Zone B

General remarks

The examination was based on a mixture of questions that test basic understanding of the material, and of questions that require candidates to apply their 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 and analysis, as well as algorithms.

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

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

Furthermore, if you wish an attempt at a question to be ignored, you should indicate this clearly (e.g. striking through – a single line is sufficient). Examiners are directed to mark only the first two questions in each part.

Similarly, examiners appreciate it if you strike through rough work so it is not inadvertently marked. (Leave it legible so that examiners can refer to it if your points are not fully made in your answers.)

All questions require a mixture of bookwork (e.g. explaining terms, giving definitions) and application (e.g. calculations, building UML models), with the majority of marks awarded for 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. (The same scenario may be shared between questions.) You should ensure that you can do this well and quickly. We hope that the discussion below will help you be aware of where marks can be easily missed.

For Part B, some brief explanatory comments for questions involving (pseudo)code in Part B would help clarify answers.

You should show working on 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, you will not gain any marks. If we see the working, we can see what you understand, and you may gain partial marks.

Comments on specific questions

Part A

Question 1

This question involved bookwork; candidates were asked about the concept of dynamic binding in object-oriented programming This was followed by the usual task to generate a software engineering model from a scenario (in this case, UML class diagrams).

Part a), as noted above, was bookwork. Candidates were generally able to explain the concept of dynamic binding and gave good, relevant examples (usually about how it is used to implement polymorphism).

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

In part b), candidates overall did fairly well. However, there were two areas where candidates lost marks.

- Some candidates produced diagrams with few classes, or omitted key classes.
- More commonly, candidates failed to fully illustrate associations, aggregations, and generalisation relationships between the objects; despite these being explicitly requested in the question.

Question 2

This question started with a bookwork question on agile software development, followed by a scenario task (activity diagram, extending the scenario from Question 1b).

Part a) was not answered well. Reflecting the widespread misunderstanding of agile development in the wider IT profession, there was confusion about agile concepts, and as a result the discussions around their role in handling requirements changes were not effective. Again, a few paragraphs would suffice here.

Part b) was generally answered well. Common errors included:

- flow of control not matching scenario, in particular that multiple attachments should be possible
- diagram at too high a level, lacking detail
- errors in notation
- poor or omitted use of forks/joins
- incomplete/omitted decision nodes.

Question 3

This question involved bookwork about UML state and sequence diagrams and development of a state machine diagram based on the same scenario as the previous two questions.

Part a) was generally answered well in terms of the kinds of things the two diagrams depict. But answers were confused in terms of explaining the differences. Again, a few paragraphs would suffice for a good answer.

Part b) was generally answered well. Areas where candidates lost marks included (in descending order of seriousness):

- sometimes other diagrams were given instead of state diagrams
- important states missing

- flow of control not matching that in the scenario (including multiple attachments and dealing with critical/overdue tasks)
- missing operations on transitions
- errors in notation.

Question 4

This question aimed to test candidates' understanding of queues and hashing. The question is a combination of tasks, plus some bookwork.

Part a) was generally well answered – most candidates gave a valid and correct answer and examples. A paragraph and a diagram would have sufficed for a good answer. Unfortunately, some candidates mixed up FIFO and LIFO.

Part b) requires pseudocode and a few paragraphs of explanation for a good answer. There was a wide range of quality of answers. Common areas where marks were lost included:

- missing or incorrect methods
- not keeping track where the front/rear of the queue is
- dealing with the situation where a queue may reach the end of the array, with empty cells in front.

Part c) required candidates to explain what hashing was, which most managed to do. Many candidates were also able to provide a walkthrough of a hashing example.

In part d), a few candidates demonstrated a resolution algorithm that was different from the one that the question asked for. (Candidates should note that examiners will use the terminology given in the subject guide, not alternatives.)

Otherwise, common errors were in calculations or errors/lack of clarity in the steps taken in applying the algorithm. A number of candidates showed marked weaknesses in doing simple modular arithmetic.

Candidates should clearly indicate both collusion detection and resolution. Therefore, a paragraph of explanation was a feature of the best answers.

Despite the above, a majority of candidates answered this question well.

Question 5

This question aimed to examine candidates' higher-level understanding of recursion in the context of sorting and dynamic programming.

Candidates gave variable answers to part a) – many candidates could give a high-level description of what recursion is, but became confused in explaining the details. Good answers gave a simple example.

Candidates also found part b) challenging. Some candidates showed the operation of a different sorting algorithm; others lost marks by missing key stages. A number of answers lost marks by unclear presentation suggesting that some did not understand the central ideas behind the algorithm. Good answers comprised of a full walkthrough of the sort, plus a paragraph of explanation.

Candidates gave variable answers to parts c) and d), which suggested a lack of in-depth understanding of this concept – some left it blank or gave a few lines of code. Some answers were rather unclear, with confusion over the nature and advantages of a dynamic programming approach (e.g. bottom-up, reuse of earlier calculations). That said, some candidates were able to provide pseudocode of classic examples such as Fibonacci.

Question 6

This question aimed to test the candidates' understanding of tree traversal and its application to binary search.

A majority of candidates were able to answer part a) satisfactorily as a bookwork question. Good answers comprised pseudocode for the three tree traversal methods, plus a paragraph of explanation.

Part b) was bookwork and generally well answered. However, a minority of candidates got full and complete trees the wrong way around. Diagrams were effectively used in many good answers.

The majority of candidates answered part c) well. Strong answers by candidates exhibited the following features:

- an explanation of the concept of binary search
- some pseudocode plus (often) a diagram to clarify the above
- a detailed and clear walkthrough of binary search on the example dataset given (where the majority of marks were given).