BCS Higher Education Qualification

Certificate in IT

October 2025

EXAMINERS' REPORT

Software Development

Questions Report:

A1	This question covered syllabus area 1.1, 1.2, 1.3 Overall, this was a popular question consisting of four parts. There was a large range of marks with weaknesses in coding in parts b) and c). Part a) Most candidates explained what distinguished an algorithm from a written specification, a fundamental part of software development. A number of candidates lost marks by failing to adequately address the different objectives of these techniques. Part b) Part c) These two parts were related and required candidates to translate a written specification into pseudocode (part b) and then into actual code (part c), again a key part of software development. A vast range of knowledge (of the process) and practical experience was evident. One of the key problems that was observed was that some candidates could not define an algorithm in pseudocode. Pseudocode should be programming language neutral and follow a step-by-step approach expressed in English. It was necessary to include conditional logic and iteration in the pseudocode solution. A number of answers produced pseudocode that was identical to the program code in part c) (popular example Python) resulting in marks being lost. There were a small number of candidates who showed good practice of coding in part c), for example using a function and the necessary calling code. This revealed how actual code is different from pseudocode. Also, it was good practice when writing actual code to avoid the use of built-in functions such as LEN (of the array) in this question. Minor syntax errors in part c) were not marked down. Part d) An algorithm's time complexity is explicitly stated on the syllabus, and most candidates demonstrated an understanding of this topic. However, marks were mainly lost if they omitted the impact that increasing the size of the input data has on time complexity of the algorithm. In this case the size of

	data array was irrelevant as a single iteration through the entire array was required to find the maximum element, so the time complexity was O(n).
A2	This question covered syllabus area 3.1, 3.2 on the syllabus and consisted of four parts. The first three parts were related to data typing and how these are handled either at run time or compile time.by either compilers or interpreters. Part a) Most candidates could identify three data types and give examples of their use in general, in terms of the operations that are possible. A small number of candidates thought that arrays and stacks are data types; these are abstract data types. The question related to primitive or base data types such as integers Boolean and characters The impact on memory allocation for a given data type was fairly well understood and some candidates who achieved high marks discussed the impact of different encoding (e.g. UTF ASCII for characters) and scale (e.g. BIGINT DOUBLE). Part b) Most candidates understood the basic concept of type checking code, but many candidates lacked knowledge of how it is actually carried out in their chosen programming language. Part c) Only a few candidates understood what was meant by type coercion; the conversion of one data type to another. Part d) This part was unrelated to the previous parts of this question. It was surprising to find some candidates struggled with producing good explanation, but specifically examples of code, for the three fundamental building blocks of coding. Very few candidates could give an example in code of a method, using for example Java.
A3	This question covered syllabus topics 7.1, 1.4. and consisted of three parts with part b) challenging candidates the most. This question was quite popular with half of candidates' attempts, but produced the lowest average across Section A with part b) causing the most concern. Part a) was generally well answered, it was important that candidates understood the concepts of planning and design and the different ways that pseudocode facilitates design and planning during software development. Part b) This part proved to be very challenging. It required candidates to produce an algorithm from a set of requirements allowing candidates to demonstrate good practice ideally based on candidates' practical experience of program design. Candidates produced a range of solutions, some that were too close to actual code at one extreme, and some that were a reproduction of the requirements. To gain high marks, the question required candidates to

	produce pseudocode that had sufficient detail to translate to actual code and covered all the requirements of the specification. What was important was to express the logic and flow of interaction between the users (borrower, librarian) and the processes and constraints imposed by the library system. Thus, simple statements that revealed the basic code constructs such as assignment; selectio; iteration for instance was required. Furthermore, answers should avoid the effects of device technology (for example mobile) and system software (for example database system) and be programming language neutral. A number of candidates scored high marks if they applied the suggestions above in their answers, but the majority of candidates scored below average marks on this part, revealing that candidates needed more practice in algorithm design as a pre-cursor to programming a solution. Despite stating the use of pseudocode in the question, some candidates used a flowchart to express an algorithm, which did not attract any marks. Again, some answers attracted low marks from candidates who expressed the requirements that were similar to the original requirements, hence unsuitable to translate to actual code. Part c) Most candidates understood the benefits of flow charts, mainly through its visual appeal as an aid for communication (of ideas) and collaboration. There were some good answers by candidates who had experience of using flow charts playing a useful role in a white box testing strategy to identify flaws in logic during debugging of faulty code. Examples of good practice usually attract high marks.
A4	This question covered syllabus topic 5.1. This was the most popular question on Section A. The average marks were the highest in Section A. Overall, a range of answers were provided and generally accepted particularly in parts a) and c). The examiner was mindful of the different views of user interface (UI) design given candidates' different backgrounds and experiences including recent developments in AI techniques such as Chatbots, for example. Part a) The most common principles covered by candidates covered variations of the keywords **consistency, simplicity and feedback**. Good answers were from candidates who could provide examples that illustrated best practices mostly from design experience and examples of poorly designed user interfaces they may have encountered. Generally, the examiner was pleased with the response from candidates. The only concern was that some candidates duplicated material from part of their answers to

	part a) into part c). and lost marks as a result. The context to these two parts is different, though choosing accessibility and responsive as two of the three key principles was valid. Part b) Although not stated in the question it was good practice to include a sketch of a login screen, and this was helpful for those candidates to articulate and highlight points raised in answering this part. Candidates generally lost marks if they didn't state a relevant UI component (for example a button; a drop down, etc). The best answers related to specific design issues of mobile devices (mainly apps) in particular. Detailed functionality of password recovery and multi-factor authentication (MFA) was NOT required, only what UI components were provided to support user interaction. Part c) Many of the previous comments apply and most candidates developed what had already been covered in more detail. Again, a variety of answers were produced with the best answers from candidates who could apply their knowledge of modern techniques of authentication. It was pleasing to see a number of candidates discussing biometrics techniques provision in the UI, such as fingerprints and face detection as part of the process of secure access through MFA.
B5	Part A focussed on octal conversion. Most candidates made a good job of this question. Where mistakes were made, these were typically related to incorrect remainders/quotients, forgetting to reverse the digits at the end, skipping intermediate steps, producing 401 (unreversed) as the final answer. Part B focussed on use of a stack. Where mistakes were made, these were typically concerned with not explaining LIFO behaviour, not relating stack behaviour to reversal of digits, saying stack is "faster" without justification, confusing stacks and queues. Part C was the least well-answered section in this question, with some candidates skipping it all together. Common mistakes included not linking condition to quotient > 0, incorrect loop termination and missing remainder assignment.
B6	Part (a) related to steps of bubble sort. This was a popular question with some good answers. Typical mistakes included missing passes, missing swap steps and producing an incorrect final sorted array. Part (b) – focussed on Bubble Sort code. Common mistakes included missing the swapped flag, forgetting the outer repeat loop, incorrect comparison operator (must use < for descending).

B7	This was a popular question and generally well-answered. The question focussed on process vs product documentation. Common mistakes included: not clearly separating the two categories, giving examples but no explanation of purpose or users, mixing up developer vs end-user documentation, giving only one example per category, etc.
B8	This question was concerned with following and modifying code. Part (a) Candidates were asked to improve layout/indentation. Common mistakes included: Keeping original unreadable formatting, forgetting braces, inconsistent indentation. Part (b) Boolean + comparison operators. Common mistakes included: Giving operators not used in code, misclassifying != as logical instead of comparison, giving only one example per category. Part (c) Which conditions print. Common mistakes included: Incorrect logical evaluation, not understanding NOT ! Operator, saying all conditions are printed, not showing working/explanation.
B9	This question was focussed on software comparisons. This was a generally well answered, popular question. 3 pairs of comparisons were required. Common mistakes included: Describing only one item in the pair, not explaining similarities, not providing examples, confusing bytecode with machine code.
B10	This question concerned White Box & Dry Run Testing. Part (a) – White box testing. Common mistakes included: Describing black-box testing, not discussing limitations. Part (b) – Dry run testing. Common mistakes included: Confusing with debugging or unit testing, not mentioning trace tables, saying it is done by automated tools (it is manual).

B11	This was a popular question focussed on Software Quality. Part (a) Quality definition + QA. Common mistakes included: Giving vague definition (“good software”), not linking answer to buyer-supplier scenario, listing tests but not explaining why they matter, ignoring documentation, support, and maintainability.
B12	This question focussed on documenting a Speeding Fines Algorithm. Part (a) Flowchart. Common mistakes included: Wrong fine amounts, putting conditions in the wrong order, missing loop for 25 speeds, wrong symbols (diamonds for decisions, rectangles for processes). Part (b) Selection & iteration. Common mistakes included: Giving definitions without examples from their flowchart, confusing iteration with sequence, not naming specific decision points.