Examiner Report	
Qualification Name	Higher Education Qualification
Qualification Level	Certificate in IT
Date/ Series	April 2026
Module	Software Development

Question no.	Comments
A1	This was the most popular question on Section A and produced a good range of marks between 0 and the maximum of 30. The average mark was around the pass mark of 12/30. Candidates gained most marks in Parts A and D. However, less than a quarter of candidates could produce a satisfactory answer to Parts B and C. Clearly, as in previous sittings, writing code from a specification revealed weakness in coding in what was a fairly simple solution. Some candidates could not express in code basic coding concepts in particular iteration. Some candidates could not differentiate pseudocode from actual code - in other words produced almost identical answers to Parts B and C. Python was the most common language used in Part C. Part B needed to be language neutral as many candidates had stated in answers to Part A ii).
Question no.	**Comments**
A2	This was the second most popular question and was generally well answered producing (not by very much) the highest average mark in Section A. Again, the marks ranged from 0 to 30. The main weakness was from those candidates who produced shallow answers that usually covered one topic. Practical experience of working through SDLC would have been beneficial to answer all parts of this question. Part A - This question expected a range of answers, which was the case mainly around the topic of testing. Apart from testing other activities were acceptable, such as quality assurance measures such as code

	reviews, static code analysis, alongside performance measures and auditing of activities to formally measure quality. Part B - This was another part that was generally well answered. Most candidates had good knowledge of the specific content that was designed to address the needs and skill levels of its intended audience. The style of documentation was often overlooked and how important it was to tailor the content and style to match the needs of different audiences. Effective documentation ensures that all stakeholders can use, maintain, develop, and support the software successfully. Part C - An open question that was fairly well covered by most candidates. Some of the objectives of early testing needed to be stated rather testing in general and addressed in terms of delivery to gain high marks. Early testing objectives had to be covered to gain high marks, such as defect prevention, cost efficiency, quality assurance, and continuous improvement, allowing more robust, reliable, and user-friendly software. Part D - There was a 50/50 split between black box vs white box testing as a matter of choice. The preferred answer was black box testing. Some candidates gave a good defence to choosing white box testing and were awarded marks if they were convincing enough and didn't confuse the two approaches to testing. The main reason for choosing black box testing was because we do not need to know the internal implementation details. We are primarily interested in verifying that the function produces the correct output (the GCD) for a wide range of input pairs. Finally, very few candidates described an actual testing approach that was required.
Question no.	**Comments**
A3	Part A - Generally well answered not unexpected as the term callable unit is clearly stated in the syllabus. Part B - Generally candidates were well informed particularly on the object-oriented paradigm. Subparts i) and ii) were often merged into one answer. Clearly, the main focus was to demonstrate the different needs of OOP versus FP, in that it depends on the nature of the problem, the need for data encapsulation versus data transformation, and whether the software requires complex interactions or stateless, predictable operations. Part C - Generally candidates had good knowledge of modular and defensive programming. Some candidates were unaware that defensive programming was a practice and not a paradigm that is generally employed on critical software.

	Most candidates could express the benefits of having both modular designed coded along with defensive approach to programming, in that it was very effective way of producing high quality easily maintained resilient code.
Question no.	**Comments**
A4	This was an unpopular question attempted by around 20% of candidates. Overall performance compared well with the other questions in Section A being only slightly down by a few marks on average. Part A – Generally, most candidates answered Part A very well with a range of depth to the answers to the second subpart. Part B - An application of a stack was provided so that candidates could apply their knowledge of how a stack works and processes data. Most candidates could express in code the standard operations of a stack to solve the problem of string reversal. Part C was very similar to Part B but this involved operations on a queue. More difficult in that candidates had to devise an application that simulates a queue that handled a queue of processes in the order that they arrive, adhering to the LIFO properties. Only a few candidates had sufficient knowledge of writing code that simulated process scheduling using enqueue (added) and dequeue (removed) operations. Part D - A varied set of answers as expected being fairly open-ended. Most candidates chose four, with the most familiar being compilers; file management; operating systems; antivirus. But there were many more to choose from. Some candidates had a problem distinguishing system software from application software - this was key and often overlapping. There was some flexibility in that some application software, such as IDE, embraces a lot of system software and therefore could be regarded as supporting crucial system software operations in one package.
Question no.	**Comments**
B5	This was a popular question, but most candidates made mistakes. Some candidates confused the behaviour of the SUM and CARRY outputs in a half adder, particularly when both inputs are 1, where the SUM becomes 0 and the CARRY becomes 1. In Part B, some wrote OR instead of AND for the carry expression.

	In Part C, common mistakes included incomplete nested IF statements, failing to account for all four input combinations, and producing code that works for some cases but not others because the decision structure is not properly nested.
Question no.	**Comments**
B6	A less popular question. A frequent error was confusing a linked list with an array, with candidates drawing adjacent memory locations instead of nodes connected by pointers. When describing insertion at the beginning of a list, many place the steps in the wrong order and lose the existing list by changing the head pointer too early. In the discussion of static and dynamic structures, candidates often described examples correctly but failed to explain why one has a fixed size and the other can grow or shrink during execution.
Question no.	**Comments**
B7	Another popular question. Candidates commonly mixed up the terms class, object, and attribute, often identifying node1 as a class rather than an instance of the node class. In the encapsulation question, many simply stated that data is "hidden" without explaining how access modifiers, such as private and public implement encapsulation, or why restricting access improves program reliability.
Question no.	**Comments**
B8	Many candidates struggled to distinguish between structured and procedural programming because the concepts overlap. Answers often described both as simply "step-by-step programming" without mentioning modules, procedures, or control structures. For parameter passing, candidates frequently know the definitions of call by value and call by reference but failed to explain the key consequence: whether changes made inside the procedure affect the original variable.
Question no.	**Comments**
B9	A common weakness was confusing logical errors with runtime errors. Some candidates assumed any error that occurs while the program is running is a runtime error, even when the program executes successfully but produces an incorrect result.

	In the debugging question, many provided general advice such as "test the code" without discussing specific IDE tools such as breakpoints, stepping through code, and inspecting variable values.
Question no.	**Comments**
B10	Some candidates confused sequential, serial, and random file organisation because the terms sound similar. A common mistake was describing sequential files simply as files that are read one record after another, without mentioning that records are stored in key order. Candidates also frequently overlooked the maintenance overhead involved in reorganising sequential files and struggled to provide realistic applications for each file organisation method.
Question no.	**Comments**
B11	Candidates frequently gave examples of data types without actually defining what a data type is. In the typing question, many confused strongly typed and statically typed languages, or weakly typed and dynamically typed languages. Answers often provide examples of languages but fail to discuss the advantages and disadvantages of each approach. The concept of type inference was also commonly misunderstood as automatic type conversion.
Question no.	**Comments**
B12	Many candidates focused entirely on the appearance of the vending machine interface and neglected broader usability principles such as consistency, simplicity, feedback, accessibility, and user control. Another common issue is producing a sketch without explaining how the design supports the user's interaction with the machine. Candidates also tended to overlook accessibility considerations, such as button size, readability, and support for users with differing needs.