BCS LEVEL 4 CERTIFICATE IN IT

SOFTWARE DEVELOPMENT

SYLLABUS

September 2021 v4.0

This is a United Kingdom government regulated qualification which is administered and approved by one or more of the following: Ofqual, Qualifications Wales, CCEA Regulation or SQA.

bcs The Chartered Institute for IT

CONTENTS

Introduction

Encompassing three core modules, the Level 4 Certificate in IT explores the fundamentals of computer and network technology, processor architecture, operating and information systems, software development, and networks.

Candidates will gain a solid foundation upon which they will be able to build a career pathway into information technology. Career opportunities include entry-level positions in the rapidly growing fields of computer science and software development.

Upon successful completion of this qualification, candidates will be equipped with the knowledge and understanding to enable them to progress on to a broad range of further development areas such as Big Data management, software engineering and web application development. Candidates will be prepared to progress onto the BCS Level 5 Diploma in IT, with the ability to customise their learning pathways based on their areas of special interest.

Software Development Core Module

The Software Development module is one of three core modules that forms part of the Level 4 Certificate in IT – the first stage within the BCS three-stage Higher Education Qualification programme. Candidates will develop an understanding of fundamental concepts of the programming process, consider issues related to the various phases of software development, and will be introduced to different types of programming concepts.

Qualification Suitability and Overview

There are no specific entrance requirements for the Certificate in IT. Candidates can study for this certificate by attending a training course provided by a BCS accredited Training Provider or through self-study, although it is strongly recommended that all candidates register with an approved centre. Studying with an approved centre will deliver significant benefits.

Candidates are required to become a member of BCS, The Chartered Institute for IT, to sit and be awarded the qualifications. Candidates may apply for a four-year student membership that will support them throughout their studies.

The Level 4 Certificate is suitable for candidates new to the profession who are keen to develop industry-relevant skills and knowledge, as well as professionals wishing to gain a formal IT qualification. Candidates taking this module may be interested in career opportunities such as games or mobile app development, database architecture or webmaster roles.

Total Qualification Time (Certificate)	Guided Learning Hours (Module)	Assessment Time (Exam)
734 hours	**200 hours**	**2 hours**

SFIA Levels

This award provides candidates with the level of knowledge highlighted within the table, enabling candidates to develop the skills to operate successfully at the levels of responsibility indicated.

Level	Levels of Knowledge	Levels of Skill and Responsibility (SFIA)
K7		Set strategy, inspire and mobilise
K6	Evaluate	Initiate and influence
K5	Synthesise	Ensure and advise
K4	Analyse	Enable
K3	**Apply**	**Apply**
K2	**Understand**	**Assist**
K1	**Remember**	**Follow**

SFIA Plus

This syllabus has been linked to the SFIA knowledge skills and behaviours required at Level 4.

ASUP2

Assists in the investigation and resolution of issues relating to applications. Assists with specified maintenance procedures.

ICPM2

Understands technical publication concepts, tools and methods and the way in which these are used. Uses agreed procedures to publish content. Obtains and analyses usage data and presents it effectively. Understands, and applies principles of usability and accessibility to published information.

PROG2

Designs, codes, verifies, tests, documents, amends and refactors simple programs/scripts. Applies agreed standards and tools, to achieve a well-engineered result. Reviews own work.

TEST2

Defines test conditions for given requirements. Designs test cases and creates test scripts and supporting data, working to the specifications provided. Interprets, executes and records test cases in accordance with project test plans. Analyses and reports test activities and results. Identifies and reports issues and risks.

HCEV3

Applies tools and methods to design and develop users' digital and off-line tasks, interactions and interfaces to meet agreed usability and accessibility requirements for selected system, product or service components. Creates workable prototypes. Assists, as part of a team, on overall user experience design. Assists in the evaluation of design options and trade-offs. Consistently applies visual design and branding guidelines.

Further detail around the SFIA Levels can be found at www.bcs.org/levels.

Learning Outcomes

Upon completion of this module, candidates will be able to:

- Distinguish between systems software and application software
- Understand the phases of software development
- Be able to develop and understand algorithms
- Be able to develop code from algorithms in a high-level programming language
- Be able to follow high level code and apply modifications to it
- Develop competence in the techniques of systematic problem analysis, program construction and documentation
- Gain an understanding of the basic concepts of good user-interface design
- Understand and develop test strategies
- Understand the need for quality assurance/security in software development and its operation
- Gain an understanding of the principles of multiple module program construction
- Understand the need for compilers, interpreters, code generators
- Develop a knowledge and understanding of a range of fundamental algorithms

Syllabus

1. Fundamental concepts of the Programming Process

Learners will be able to:

1.1 Explain the nature of information.

Indicative content	Guidance
a. Differentiate an algorithm from a written specification.	An algorithm describes the steps that must be taken to solve a given problem that has to be translated into computer code. Coders will generally employ patterns to determine the translation of an algorithm design into actual code.

1.2 Design algorithms.

Indicative content	Guidance
a. Decompose a problem into a set of steps which may be executed by a computer.	Candidates should be able to break down a given practical situation into smaller sections and continue to do this until the problem can be expressed in terms of the control structures covered in 1.3. Candidates should create an independent solution in a programming language.

1.3 Develop code from an algorithm.

Indicative content	Guidance
a. Produce code from an algorithm developed from the outcome of the process described in 1.2.	Candidates should be able to express sequence, selection, iteration in their chosen programming language. For example, they should be able to write if statements, case statements, while loops and for loops; the way this is done will vary from language to language.

1.4 Utilise pseudocode and flowcharts.

Indicative content

a. How these are used in the creation of code in the design phase
b. How these can be written in different ways
c. Specific symbols for flowcharts

Guidance

Pseudocode and flowcharts relate to the way in which algorithms can be described. Both techniques are ways of representing sequence, selection and iteration: pseudocode is textual, whereas flowcharts are graphical.

2. Phase Specific issues of Software Development

Learners will be able to:

2.1 Explain programming paradigms.

Indicative content

a. For example:
   - Modular/structured programming
   - Object oriented programming
   - Functional programming

Guidance

Candidates should be aware of various types of programming, including modular programming, OOP and functional programming, which are examples of current practices in software development. Modular programming requires a program to be broken down into individual modules rather than create a monolithic application (where the smallest component is the whole), several smaller modules are written separately so when they are composed together, they construct the executable application program. OOP associates data structures with the code that operates on it and seeks to prevent programmers interacting with data other than through pre-defined interfaces. Functional programming represents every action as a function call. Most functional languages use recursion instead of iteration.

2.2 Describe the objectives and principles of testing, derive test cases from a given specification.

Indicative content

a. Test-case specification
b. Testing and debugging strategies including:
   - Dry-running
   - White-box
   - Black-box

Guidance

Candidates will be expected to demonstrate an appreciation of the importance of testing in the software development process. They should be able to describe the terms dry-run, white-box testing and black-box test. Given an algorithm or a piece of code, they should be able to dry run the code or develop a set of inputs/outputs suitable for a complete white or black box test.

2.3 Discuss the need for software documentation and the nature of software documentation in a given context.

Indicative content	Guidance
a. Suitability of documentation for a given context b. Content of software documentation such as GUI descriptions and maintenance details	Candidates must appreciate the need to document code. Such documentation will range from user manuals to in-line comments in the code. They should be able to understand that the documentation should be written with reference to its potential user: an end-user will expect to see documentation which is relevant to their use of the software, whereas a maintenance programmer will require more technical details.

2.4 Describe mechanisms for assuring software quality and security within the software development process.

Indicative content	Guidance
a. ISO/IEC 25000 b. Quality models for software product evaluation	Software should be reliable, secure, efficient and maintainable. Candidates should be able to define each of these terms and describe basic mechanisms for achieving each characteristic, as covered by the ISO/IEC 25000 standard.

2.5 Discuss a range of new or emerging software technologies.

Indicative content	Guidance
a. For example: • Parallel computing • Quantum computing	Candidates should be aware of alternatives to traditional software development approaches covered elsewhere in this syllabus associated with programming languages and programming techniques. A general appreciation of current trends in software development might include techniques to improve computation speed, such as quantum computing or parallel programming.

3. Introduction to Programming concepts

Learners will be able to:

3.1 Discuss the use of data types and type checking in programming languages.

Indicative content	Guidance
a. Numeric and non-numeric b. Elementary and derived c. Subtypes d. Expressions such as: • Assignments • Input/output • String handling • Logical operators	Candidates should be able to list basic types such as integer, floating point, character, Boolean and string. They should also be able to distinguish between weak typing, strong typing and type inference. They should also have an elementary understanding of derived types. They should be able to describe the relationship between types and the operations permitted on those types.

3.2 Discuss the use of callable units in the development of code and write code examples which use callable units.

Indicative content	Guidance
a. Subroutines b. Procedures c. Functions	Callable units allow the programmer to refer to blocks of commonly-used code. Subroutines and procedures are essentially identical, with 'procedure' being the more modern name for a callable unit which does not return a value but may manipulate global variables. Functions will return a value. Subroutines, functions and procedures will accept zero or more parameters. Calls to functions, subroutines and procedures may be call by value or call by reference. Candidates are expected to be able to discuss the difference between these two types of calls.

3.3 Explain the concept of a data structure and illustrate the explanation with reference to commonly used data structures.

Indicative content	Guidance
a. Arrays, lists and tuples b. Implementation of queues, stacks and collections c. Concept of data abstraction	Arrays, lists and tuples are data structures that are used to store related sequences of data items that can be individually selected using iteration statements. An item in an array and a list can be changed or replaced. Arrays contain data of the same type. Stacks and queues are also linear data structures with special built in properties. Stack is a sequence of data that are inserted and removed according to the last-in first-out (LIFO) principle. Queue is a list of data items that are inserted and removed using a first-in first-out (FIFO) principle. Data abstraction is the reduction of a particular body of data to a simplified representation of the whole.

3.4 Explain the advances in technology and impact of emerging trends in IS.

Indicative content	Guidance
a. Comparative effectiveness of algorithms re. computation and storage, e.g. bubble sort, merge sort and quicksort	These are different techniques for sorting an unordered set of data contained in a list. The efficiency (also known as 'the big O') of a particular sorting technique can be measured and depends on various factors that candidates need to be aware of.

4. Files

Learners will be able to:

4.1 Describe techniques for storing data in secondary storage.

Indicative content	Guidance
a. Creating, storing, and/or retrieving the contents of a file located on a secondary storage device b. Sequential, index-sequential and random-access files c. Text files d. Semi-formatted files, e.g. Comma Separated Value (CSV) files	Data consumed or output by a program needs to be permanently stored and there are many ways this is achieved. In particular, candidates will need to know how to write code to access files of data held on a connected device (such as a memory card, external hard drive). It is also important to be aware of how data is structured to facilitate access to individual items of data or to sequentially read the file using iteration.

5. Discuss aspects of user interface design.

Learners will be able to:

5.1 Discuss aspects of user interface design.

Indicative content	Guidance
a. User requirements and characteristics of user interfaces b. Principles and techniques of dialogue control, navigation and selection	An important part of user interface design is to gather what the user requires from an application which the user wants developing. This is usually manifested in the User interface which is where the user is able to interact with the development of an application. Therefore the design of the user interface needs to accommodate the needs of users, but most importantly to provide access to the underlying functionality of the software. Candidates need to be aware of the basic principles such as Help and appropriate dialogue control particularly when relaying information. 'Keep it simple, stupid' (KISS) is a design principle stating that design of a UI should be as simple as possible to guarantee the greatest levels of user acceptance and interaction.

6. Role and need for system software

Learners will be able to:

6.1 Discuss the nature and the utility of system software.

Indicative content	Guidance
a. Fundamental utilities that form the bulk of built-in system software: • Editors • Debuggers • Compilers • Interpreters • Linkers • Loaders b. Programming languages requiring a virtual machine environment to run (e.g. Java)	System software is necessary to support the developer in many different ways. All software is built on top of an operating system and programs are built on top of compilers and interpreters linkers supporting different programming languages. An understanding of software utilities that support the development process such as editors debuggers and Integrated development Environments (IDEs) is important. Candidates are advised to gain this knowledge from practical experience of using the system software environment they use to write programs. More obvious system software – such as operating systems or database management systems, for example – are important, but candidates only need to be aware of the role they have in supporting software development.

7. Case studies in problem solving/algorithm analysis

Learners will be able to:

7.1 Develop a software solution to a real-world problem.

Indicative content	Guidance
a. Examine a software application b. Look at a case study or scenario that describes the problem and the functional requirements	This section is about giving candidates a chance to practise examining a case study or a scenario. The case study or scenario will set out a problem that candidates have not encountered before and ask them to give a solution possibly using code or pseudocode, i.e. a typical software development problem.

Examination Format

This module is assessed through completion of an invigilated written exam.

Type	Two questions from Section A and five questions from Section B
Duration	Two hours
Supervised	Yes
Open Book	No (no materials can be taken into the examination room)
Passmark	10/25 (40%)
Delivery	Paper format only

Adjustments and/or additional time can be requested in line with the BCS reasonable adjustments policy for candidates with a disability or other special considerations.

Question Weighting

Section A and Section B each carry equal marks. Candidates are advised to spend about one hour on Section A (30 minutes per question) and one hour on Section B (12 minutes per question).

Recommended Reading

Primary texts

Title: Grokking Algorithms
Author: A. Bhargava
Publiser: Manning Publications
Publisher Date: 2015
ISBN: 978-1617292231

Title: The Self-Taught Programmer
Author: C. Althoff
Publiser: Self-Taught Media
Publisher Date: 2017
ISBN: 978-0999685907

Additional texts

Java texts

Title: Data Structures & Algorithms in Java (6th edition)
Author: M. Goodrich, R. Tamassia and M. Goldwasser
Publiser: Wiley
Publisher Date: 2014
ISBN: 978-118771334

Title: Java How to Program (11th edition)
Author: H. Deitel and P. Deitel
Publiser: Pearson
Publisher Date: 2018
ISBN: 978-9353062033

C/C++ texts

Title: Problem solving with C++ (10th edition)
Author: W. Savitch
Publiser: Pearson
Publisher Date: 2018
ISBN: 978-1292222820

Title: C How to program (8th edition)
Author: H. Deitel and P. Deitel
Publiser: Pearson
Publisher Date: 2016
ISBN: 978-0133976892

C# texts

Title: Visual C# How to Program (6th edition)
Author: H. Deitel and P. Deitel
Publiser: Pearson
Publisher Date: 2017
ISBN: 978-1292153469

Title: C# in Depth (4th edition)
Author: J. Skeet
Publiser: Manning Publications
Publisher Date: 2019
ISBN: 978-1617294532

Python texts

Title: Introduction to Python for the Computer and Data Sciences
Author: H. Deitel and P. Deitel
Publiser: Pearson
Publisher Date: 2019
ISBN: 978-0135404676

Title: Data Structures and Algorithms in Python
Author: M. Goodrich, R. Tamassia and M. Goldwasser
Publiser: Wiley
Publisher Date: 2016
ISBN: 978-8126562176

Using BCS Books

Accredited Training Organisations may include excerpts from BCS books in the course materials. If you wish to use excerpts from the books you will need a license from BCS. To request a license, please contact the Head of Publishing at BCS outlining the material you wish to copy and its intended use.

Document Change History

Any changes made to the syllabus shall be clearly documented with a change history log. This shall include the latest version number, date of the amendment and changes made. The purpose is to identify quickly what changes have been made.

Version Number	Changes Made
Version 1.0 **July 2021**	Document Creation

CONTACT

For further information please contact:

BCS
The Chartered Institute for IT
3 Newbridge Square
Swindon
SN1 1BY

T +44 (0)1793 417 445

www.bcs.org

The Chartered Institute for IT