BCS LEVEL 5 DIPLOMA IN IT

SOFTWARE ENGINEERING

SYLLABUS

September 2021 v3.0

This qualification is regulated by one or more of the following: Ofqual, Qualifications Wales, CCEA Regulation or SQA.

bcs
The Chartered Institute for IT

CONTENTS

Introduction

Level 5 Diploma in IT

The second stage within the BCS three-stage Higher Education Qualification programme, the Level 5 Diploma enables candidates who have already achieved the Level 4 Certificate in IT to progress to higher levels of knowledge and competency.

This internationally-recognised qualification introduces you to the business-related aspects of the IT industry, developing your technological expertise while also considering the potential challenges of the day-to-day running of an organisation, such as legal obligations and intellectual property.

Our modules have been created in-line with the latest developments in the industry, giving you a competitive edge in the IT job market. You will have the opportunity to learn about object-oriented programming, user experience, systems analysis and design, as well as to build upon knowledge and skills developed during the Level 4 Certificate.

To successfully achieve the qualification, candidates need to complete:

- One core module
- Three optional modules
- One Professional Project in IT

Candidates who wish to progress onto the next stage will need to complete the Project at end of the Level 6 Professional Graduate Diploma in IT.

Software Engineering 1 Optional Module

The Software Engineering 1 module is an optional module that forms part of the Level 5 Diploma in IT – the second stage within the BCS three-stage Higher Education Qualification programme.
Candidates will be introduced to software engineering and its theoretical models, software design principles and will learn about the software development process, including project planning and product risk management.

Qualification Suitability and Overview

Candidates must have achieved the Certificate in IT or have an appropriate exemption to be entered for the Diploma in IT. Candidates can study for this diploma 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 5 Diploma is suitable for professionals wishing to gain a formal IT qualification, and this module may be particularly relevant for candidates interested in career opportunities such as intelligent systems, forensic computing, or computer security.

Total Qualification Time	Guided Learning Hours	Assessment Time
1086 hours	**225 hours**	**2 hours**

SFIA Levels

This module 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 5.

ASUP3

Identifies and resolves issues with applications, following agreed procedures. Uses application management software and tools to collect agreed performance statistics. Carries out agreed applications maintenance tasks.

DESN4

Designs components using appropriate modelling techniques following agreed architectures, design standards, patterns and methodology. Identifies and evaluates alternative design options and trade-offs. Creates multiple design views to address the concerns of the different stakeholders of the architecture and to handle both functional and non-functional requirements. Models, simulates or prototypes the behaviour of proposed systems components to enable approval by stakeholders. Produces detailed design specification to form the basis for construction of systems. Reviews, verifies and improves own designs against specifications.

PROG3

Designs, codes, verifies, tests, documents, amends and refactors moderately complex programs/scripts. Applies agreed standards and tools, to achieve a well-engineered result. Collaborates in reviews of work with others as appropriate.

DLMG5

Defines systems development projects which support the organisation's objectives and plans. Selects, adopts and adapts appropriate systems development methods, tools and techniques selecting appropriately from predictive (plan-driven) approaches or adaptive (iterative/agile) approaches. Ensures that senior management is both aware of and able to provide the required resources. Facilitates availability and optimum utilisation of resources. Monitors and reports on the progress of development projects, ensuring that projects are carried out in accordance with agreed architectures, standards, methods and procedures (including secure software development). Develops road maps to communicate future development activity.

TEST3

Reviews requirements and specifications, and defines test conditions. Designs test cases and test scripts under own direction, mapping back to pre-determined criteria, recording and reporting outcomes. Analyses and reports test activities and results. Identifies and reports issues and risks associated with own work.

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 regarding the SFIA Levels can be found at www.bcs.org/levels.

Learning Outcomes

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

- Explain the background of the software crisis and the need for an engineering approach.
- Appreciate the distinction between software programming and an engineering approach to the development of a software product.
- Create models of software data and processes using object oriented modelling approaches such as the UML.
- Describe and evaluate software tools and technology to enhance productivity and quality of software development.
- Demonstrate skills of software documentation, quality assurance and evaluation, and testing as part of software development.

Syllabus

1. The Nature of Software

Learners will be able to:

1.1 Discuss the nature of software.

Indicative content

a. Defining software
b. Software application domains
   i. Systems software
   ii. Application software
   iii. Engineering/scientific software
   iv. Embedded software
   v. Artificial intelligence software
c. Legacy software
d. Changing nature of software
   i. Web apps
   ii. Mobile apps
   iii. Cloud computing
   iv. Product line software

Guidance

Candidates should be able to understand that producing software is an engineering task and that engineering disciplines are often applicable to it. They should also appreciate that producing software is more than just programming.

1.2 Discuss theoretical models.

Indicative content

a. Prescriptive models:
   i. Waterfall models, e.g. classic lifecycle or V model
   ii. Incremental process models
   iii. Evolutionary process models, e.g. prototyping or spiral model
   iv. Concurrent models

Guidance

Candidates should understand how models have brought useful structure to the discipline of software engineering, as well as how modern software is characterised by constant change, tight schedules and the need to meet users' expectations. They should also understand and be able to discuss both strengths and weaknesses of evolutionary process models, which were originally conceived to address these issues.

1.3 Explain the motivation for development of software engineering.

Indicative content

a. Projects running over budget
b. Projects running over time
c. Inefficient software
d. Software not meeting requirements
e. Unmanageable projects
f. Code difficult to maintain
g. Failure to deliver

Guidance

Candidates need to be able to understand the motivations for the development of software engineering as a discipline, as well as to describe issues that have led to its development.

1.4 Describe the cost of maintenance.

Indicative content

a. Characterising maintenance, e.g. ISO/IEC 14764
b. Corrective maintenance
c. Adaptive maintenance
d. Perfective maintenance
e. Preventative maintenance

Guidance

Candidates need to be able to understand a variety of reasons for which maintenance is necessary, and appreciate that the maintenance costs may often be larger than initial development costs.

1.5 Explain software quality.

Indicative content

a. Software functional quality
b. Software structural quality
c. The cost of quality
d. Quality of design
e. Quality of conformance
f. Performance quality
g. Feature quality
h. Reliability
i. Durability
j. Serviceability
k. Aesthetics

Guidance

Candidates need to have an appreciation of the wide variety of factors which contribute to a measure of software quality. They should also be able to appreciate that given factors will be differently emphasised in a variety of contexts.

2. Software Engineering key practices

Learners will be able to:

2.1 Describe and analyse the multidisciplinary nature of software design.

Indicative content

a. Requirements analysis and specification
b. Software design
c. Software development
d. Software testing
e. Software maintenance
f. Software configuration management
g. Project management

Guidance

Candidates are expected to be able to identify distinct sub-disciplines of software engineering and to be able to identify their characteristics.

2.2 Explain team work in software engineering.

Indicative content

a. Characteristics of a software engineer
b. Psychology of software engineering
c. Cohesiveness of the software team
d. Team structures
e. Agile and global teams
f. Collaboration tools

Guidance

It is important for candidates to appreciate the overall concept that software is never developed in isolation. Paired programming would be one example. Candidates should also understand how having a sense of purpose, involvement, trust and improvement will all contribute to a software team's overall effectiveness.

2.3 Describe productivity in software engineering.

Indicative content

a. Lines of code
b. Function points
c. Constructive cost model (CoCoMo)
d. Cyclomatic complexity

Guidance

In general, candidates should be able to measure the effectiveness of software engineering processes for this section, possibly in lines of code, but other measures may be applicable.

2.4 Describe testing in software engineering.

Indicative content

a. Black-box testing
b. White-box testing
c. Grey-box testing
d. Unit testing
e. Integration testing
f. System testing
g. Acceptance testing
h. Smoke and sanity testing
i. Regression testing
j. Functional and non-functional testing
k. Usability testing
l. Security testing
m. Traceability testing

Guidance

Candidates are expected to be able to describe a range of valid range of testing techniques, rather than one single technique. Candidates should also understand that one testing approach is not necessarily an alternative to another, but can be a complementary approach which may uncover a different class of errors to other methods.

2.5 Explain product maintenance.

Indicative content

a. Maintenance debt
b. Dependence on external factors
c. Lifecycle

Guidance

Candidates need to be aware that delivery of a product is not the end of the cycle – it will be followed by maintenance. The idea of maintenance debt, which ties in more closely the cost of maintenance with decisions taken earlier in the maintenance lifecycle, is also important.

2.6 Describe the software product life cycle.

Indicative content

a. Preliminary analysis
b. Systems analysis, requirements definition
c. Systems design
d. Development
e. Integration and testing
f. Acceptance, installation, deployment
g. Maintenance
h. Evaluation
i. Disposal

Guidance

Candidates need to be able to explain and describe the phases that a typical software development exercise will go through.

3. Software development models and methods

Learners will be able to:

3.1 Explain design principles.

Indicative content

a. Transparency
b. Separation of concerns
c. Abstraction
d. Modularity
e. Development by incremental methods

Guidance

Candidates should understand the role of software design and its place within the modelling activity, as it sets the stage for the construction phase – its goal is to create a software model which will take into account (and meet) all user requirements.

3.2 Utilise and demonstrate notations for software components.

Indicative content

a. Syntax
b. Semantics

Guidance

Candidates need to be familiar with techniques for modelling software systems. They should be able to identify the essential features of modelling notations and distinguish between the allowable symbols within notations and the meaning of those symbols.

3.3 Demonstrate Unified Modelling Language (UML) modelling.

Indicative content

a. UML modelling of use cases for a logical/end-user view (e.g. use case diagram)
b. Class diagram
c. Object diagram
d. Activity diagram
e. Sequence diagram

Guidance

Candidates must be able to discuss and demonstrate the use of UML techniques in order to visualise the design of a system.

4. Validation, verification and testing

Learners will be able to:

4.1 Describe product and process visibility.

Indicative content

a. Product visibility
b. Process visibility

Guidance

Candidates will need to be able to explain the concept of visibility and outline ways in which visibility can be achieved for both software products and software processes.

4.2 Explain traceability in software systems and describe the processes.

Indicative content

a. Mapping from requirements to specifications
b. Mapping from specification to design
c. Mapping from design to implementation
d. Testing to ensure traceability
e. Derivation paths
f. Flowdown paths

Guidance

Candidates should be able to identify the steps a software engineer will take to ensure that user requirements identified in a specification are realised in an implementation.

---

5. Software engineering tools and environments

Learners will be able to:

5.1 Demonstrate and explain Computer Aided Software Engineering (CASE) tools.

Indicative content

a. Automated support
b. Semi-automated support
c. Upper CASE
d. Lower CASE

Guidance

Candidates should be able to discuss the ways in which automated tools can aid the software engineer.

5.2 Describe the role of repositories.

Indicative content

a. Package management systems
b. Source control software
c. Software dependencies

Guidance

Candidates should understand the role of repositories in supporting incremental development.

5.3 Explain software reuse and evolution.

Indicative content

a. The reuse landscape
b. Design patterns
c. Generator-based reuse
d. Application frameworks
e. Application system reuse

Guidance

Candidates should be able to identify the way in which software reuse can contribute to the development of new applications. Software should be designed to be reusable in the first instance, as well as making sure to reuse it when the opportunity arises.

---

6. Project management

Learners will be able to:

6.1 Explain how to use project estimating and project planning tools.

Indicative content

a. Algorithmic cost modelling
b. Expert judgement
c. Estimation by analogy
d. Parkinson's Law
e. Pricing to win

Guidance

Candidates need to be able to discuss a range of techniques they might need to employ in order to estimate the amount of effort necessary to produce a software component.

6.2 Describe the management and maintenance of software products.

Indicative content

a. Variety of mechanisms for updating software deployed in customer environments
b. Risks associated with software updates

Guidance

Candidates should be able to describe a range of strategies for updating software products which have been deployed and are operating in user environments. They should be able to discuss the different levels of risks of said strategies and the context in which they should be used.

6.3 Explain the total cost of system ownership.

Indicative content

a. Computer hardware and programs
   i. Network hardware and software
   ii. Server hardware and software
   iii. Workstation hardware and software
   iv. Installation and integration of hardware and software
   v. Purchasing research
   vi. Warranties and licenses
   vii. License tracking compliance
   viii. Migration expenses
   ix. Risk
b. Operation expenses
   i. Infrastructure
   ii. Power
   iii. Testing costs
   iv. Downtime, outage and failure expenses
   v. Diminished performance
   vi. Security
   vii. Backup and recovery
   viii. User training
   ix. Audit
   x. Insurance
   xi. Information technology personnel costs
   xii. Management costs
c. Long-term expenses
   i. Replacement
   ii. Upgrade
   iii. Decommissioning

Guidance

Candidates need to be able to identify all of the issues contributing to the cost of a software product and be able to use these to evaluate different software development strategies.

6.4 Analyse and explain the software life cycle cost modelling.

Indicative content

a. Prevention costs
b. Appraisal costs
c. Failure costs
   i. Internal failure costs
   ii. External failure costs

Guidance

Candidates should understand and have an appreciation for the cost of quality, as well as the cost of a lack of quality, both for users who must try to contend with glitches in software and for the team that has built and now has to maintain it.

6.5 Describe project and product risk management.

Indicative content

a. Schedule flaws
b. Requirements inflation
c. Employee turnover
d. Specification breakdown
e. Poor productivity

Guidance

Candidates should be able to list the main risk factors in producing software products and be able to describe techniques to mitigate them.

Examination Format

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

Type	Four written questions from a choice of six, each with equal marks
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

Candidates will choose four questions from a choice of six. All questions are equally weighted and worth 25 marks.

Recommended Reading

Primary texts

Title: Software Engineering
Author: I. Somerville
Publisher: Pearson
Date: 2015
ISBN: 978-1292096131

Title: Software Engineering: A Practitioner's Approach
Author: R. S. Pressman and B. Maxim
Publisher: McGraw-Hill Education
Date: 2014
ISBN: 978-1259253157

Additional texts and resources

Title: The Mythical Man-Month
Author: F. P. Brooks
Publisher: Addison-Wesley
Date: 1995
ISBN: 978-0201835953

Title: Clean Architecture
Author: R. C. Martin
Publisher: Prentice Hall
Date: 2017
ISBN: 978-0134494164

Title: Beginning Software Engineering
Author: R. Stephens
Publisher: Sybex
Date: 2015
ISBN: 978-8126555376

Title: Effective Project Management: Traditional, Agile, Extreme
Author: R. K. Wysocki
Publisher: Wiley India
Date: 2014
ISBN: 978-8126552207

Title: Peopleware: Productive Projects and Teams
Author: T. Demarco and T. Lister
Publisher: Addison Wesley
Date: 2016
ISBN: 978-0321934113

Title: ISO/IEC/IEEE International Standard for Software Engineering - Software Life Cycle Processes – Maintenance ISO/IEC 14764:2006 (E) IEEE
Author: International Organization for Standardization (ISO)
Publisher: International Organization for Standardization (ISO)
Date: 2006
ISBN: 978-0580465963

Title: Managing Software Debt: Building for Inevitable Change
Author: C. Sterling
Publisher: Addison Wesley
Date: 2020
ISBN: 978-0321948618

Title: International Experiences and Initiatives in IT Quality Management
Author: O. Khan, P. Marchbank, E. Georgiadou, M. Ross, G. Staples and P. Linecar
Publisher: Solent University
Date: 2019
ISBN: 978-1999654924

Title: Debt Metaphor
Author: W. Cunningham
Creation Date: 2009
Accessible at: <https://www.youtube.com/watch?v=pqeJFYwnkjE> [Accessed 09 July 2021]

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 August 2021	Document created

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