BCS Foundation Certificate in Digital Solution Development
Syllabus

Version 1.0
November 2020

This qualification is not regulated by the following United Kingdom Regulators - Ofqual, Qualification in Wales, CCEA or SQA

Contents

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 and Date	Changes Made
V1.0 Nov 2020	1st issue

Introduction

The Foundation Certificate in Digital Solution Development is an introductory level qualification intended to give a broad coverage of the many topics affecting Digital Solution Development (DSD) in a modern enterprise setting. The syllabus seeks to:

- lay the foundation for a Body of Knowledge concerning DSD, and therefore includes a broad range of themes/topics, roles, best practices, techniques and methods that are widely recognised and employed by practitioners in this field.
- cover topics in the development space between Business Analysis and the coding environment.
- be product, technology and platform agnostic as far as possible.

The syllabus emphasises activities in the development part of a digital service lifecycle, with links to the other parts, as appropriate.

Target Audience

The BCS Foundation Certificate in Digital Solution Development is primarily targeted at the following groups:

1) People within roles other than software development who wish to gain a broad understanding of the terminology, concepts, techniques, standards, frameworks and processes involved in the development of modern digital software applications.
2) Developers with entry level coding experience who wish to widen their horizons, preparing themselves for a Solution Architect role.
3) Experienced practitioners involved in the maintenance of legacy software solutions who wish to refresh their understanding of the terminology, concepts, techniques, standards, frameworks and processes involved in the development of modern digital software applications.
4) Graduates in a non-IT discipline who are seeking to work in a consulting capacity.

It is anticipated that this certification will appeal to individuals undertaking a range of job roles including, but not limited to: software developer / software engineer, software tester, business analyst, systems analyst, solution architect, product owner, Scrum master, systems development manager, IT service delivery manager.

Levels of Knowledge / SFIA Levels

This syllabus will provide candidates with the levels of difficulty / knowledge highlighted within the following table, enabling them to develop the skills to operate at the levels of responsibility indicated. The levels of knowledge and SFIA levels are further explained on the website www.bcs.org/levels.

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

Learning Outcomes

Candidates will be able to demonstrate knowledge and understanding of terminology, concepts, techniques, standards, frameworks and processes in the following areas:

1. The context for digital solution development.
2. Digital service definition.
3. User Experience (UX) and User Interface (UI).
4. Digital solution architecture and design.
5. Data and information architecture and design.
6. Quality assurance and quality control.
7. Digital Solution acquisition, deployment and maintenance.
8. Cyber security.

Study Format and Duration

Candidates can study for this certificate in two ways:

- Attending an accredited training course. This will require a minimum of 18 hours of study over a minimum of three days.
- Self-study. Self-study resources include online learning and recommended reading (see syllabus Reading List).

Eligibility for the Examination

There are no specific pre-requisites for entry to the examination, although accredited training is strongly recommended. No experience of software development is required, but around 6 months coding experience on any platform is desirable.

Examination Format and Duration

Type	40 Multiple Choice questions
Duration	60 minutes
Pre-requisites	Accredited training is strongly recommended but is not a pre-requisite.
Supervised	Yes
Open Book	No (no materials can be taken into the examination room)
Pass mark	26/40 (65%)
Calculators	Calculators are not allowed during this examination
Delivery	Online or paper based.

Additional time

For candidates requiring reasonable adjustments

Please refer to the reasonable adjustments policy for detailed information on how and when to apply.

For candidates whose language is not the language of the examination

If the examination is taken in a language that is not the candidate's native/official language, candidates are entitled to:

- 25% extra time
- Use their own **paper** language dictionary (whose purpose is translation between the examination language and another national language) during the examination
- Electronic versions of dictionaries will **not** be allowed into the examination room

Guidelines for Accredited Training Organisations

Each major subject heading in this syllabus is assigned an allocated percentage of study time. The purpose of this is:

1) Guidance on the proportion of content allocated to each section of an accredited course.
2) Guidance on the proportion of questions in the exam.

Courses do not have to follow the same order as the syllabus and additional exercises may be included, if they add value to the training course.

Question Weighting

Syllabus Area	Weighting (%)	Target number of questions
1. The context for digital solution development	10%	4
2. Digital service definition	15%	6
3. User experience (UX) and user interface (UI) design	12.5%	5
4. Digital solution architecture and design	15%	6
5. Data and information architecture and design	12.5%	5
6. Quality assurance and quality control	10%	4
7. Digital solution acquisition, deployment and maintenance	15%	6
8. Cyber security	10%	4
Total	**100%**	**40 Questions**

Trainer Criteria

Criteria	• Hold the BCS Foundation Certificate in Digital Solutions Development. • Have a minimum of 3 years' practical system/solution development experience, a minimum of 2 years' training experience or 1 year with a recognised training qualification.

Classroom Size

Recommended maximum trainer to candidate ratio	1:16

Excerpts from 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 do this. If you are interested in taking out a licence to use BCS published material, you should contact the Head of Publishing at BCS outlining the material you wish to copy and the use to which it will be put.

Syllabus

Learning Objectives

1. The context for digital solution development (10%)

Candidates will be able to:

1.1. Define and distinguish between the terms *digital service* and *digital solution* and recognise some common characteristics of modern digital solutions

1.1.1. Digital service as a use of information technology to solve business problems. Digital Solution as the technology solution employed to provide the service.

1.1.2. The distinction between application software, system software and other forms of software.

1.1.3. Common characteristics of a digital solution in the form of enterprise applications software.
1.1.3.1. Data intensive
1.1.3.2. Highly interactive

1.2. Explain the factors that influence investment in digital solution development

1.2.1. Understand that internal and external drivers lead to the formulation of corporate strategy.

1.2.2. Understand that the implementation of strategy drives the business change lifecycle.

1.2.3. Understand that business change often includes the need for new or improved digital services to the end user, as part of *digital transformation*, and hence digital solution development work will be required.

1.3. Describe the scope of digital solution development

1.3.1. Recognise and describe the stages of a typical Digital Service Lifecycle.
1.3.1.1. Plan
1.3.1.2. Develop
1.3.1.3. Transition
1.3.1.4. Operate
1.3.1.5. Optimise
1.3.1.6. Retire

1.3.2. Recognise what a SDLC Framework is, why it is especially useful in the Develop and Transition stages and what it should consist of.
1.3.2.1. Set of processes
1.3.2.2. Roles and disciplines
1.3.2.3. Deliverables and artefacts
1.3.2.4. Tools, techniques and best practices

1.3.3. Identify the main parameters governing the approach to Digital Solution Development, using an SDLC based on the following models:
1.3.3.1. The need to adopt a contingency approach
1.3.3.2. The Cone of Uncertainty
1.3.3.3. Time-cost-features compromise

1.3.3.4. Continuum from predictive (linear) to adaptive (iterative) approaches
1.3.3.5. Continuum from low ceremony to high ceremony approaches

1.4. Recognise examples of constraints that commonly affect DSD from the following list:
1.4.1. Legal
1.4.2. Financial
1.4.3. Technological
1.4.4. Ethical
1.4.5. Timescales
1.4.6. Organisational policies and standards.

1.5. Recognise and identify the purpose of the following enterprise frameworks, and their relationship with DSD.
1.5.1. Enterprise architecture
1.5.2. Programme and project management
1.5.3. Business analysis and change management
1.5.4. Service operations
1.5.5. Data management and data governance

2. Digital service definition (15%)

Candidates will be able to:

2.1. Recognise the need for Requirements Engineering, its key elements and benefits

2.1.1. Explain the rationale for requirements engineering.
2.1.1.1. Alignment of the digital service to business needs and objectives
2.1.1.2. Completeness of the digital service definition
2.1.1.3. Problem clarification
2.1.1.4. Balancing stakeholder interests
2.1.1.5. Stability of requirements vs. solution
2.1.1.6. Governance of solution acquisition and delivery

2.1.2. Identify the elements of requirements engineering as:
2.1.2.1. Elicitation
2.1.2.2. Analysis
2.1.2.3. Validation
2.1.2.4. Documentation
2.1.2.5. Management

2.2. Identify common sources of requirements and classify different types of requirements

2.2.1. Identify common sources of digital service requirements:
2.2.1.1. Problem statement from the sponsor
2.2.1.2. Decomposition of business processes
2.2.1.3. Business and technical policies and standards
2.2.1.4. Legal and regulatory compliance
2.2.1.5. Stakeholder specific needs

2.2.2. Classify requirements according to the following categories:
2.2.2.1. Service functional requirements
2.2.2.2. Service non-functional requirements

2.2.3. Recognise the following examples of non-functional requirement types:
- 2.2.3.1. Usability
- 2.2.3.2. Security
- 2.2.3.3. Performance
- 2.2.3.4. Accessibility
- 2.2.3.5. Availability
- 2.2.3.6. Reliability
- 2.2.3.7. Recoverability
- 2.2.3.8. Scalability

2.2.4. Understand the link between digital service requirements, key performance indicators and service level agreements.

2.3. Describe the use of the following requirements documentation and modelling techniques.

2.3.1. User stories
- 2.3.1.1. Structure
- 2.3.1.2. Quality criteria (INVEST)
- 2.3.1.3. Epic as an end to end feature requiring further decomposition
- 2.3.1.4. Theme as a set of related user stories

2.3.2. Use cases
- 2.3.2.1. Key components of a use case diagram: system boundary, actors, associations, use cases
- 2.3.2.2. Use of use case diagrams to agree the functional scope of a digital service
- 2.3.2.3. Use case description as an approach to elaborating required system behaviour
- 2.3.2.4. Use case description as a mechanism for elaborating test scenarios

2.3.3. Data and process modelling
- 2.3.3.1. The role of data modelling and data models in driving out requirements
- 2.3.3.2. The role of the data lifecycle in driving out requirements
- 2.3.3.3. The role of process models in driving out requirements

2.4. Explain the purpose of some common requirements management techniques.

2.4.1. Explain the role of a product backlog in managing requirements.
- 2.4.1.1. User stories as product backlog items
- 2.4.1.2. Iteration (sprint) backlog and release backlog as subsets of the product backlog

2.4.2. Recognise the need to track requirements from origin to delivery in a working solution.

2.4.3. Explain the need to refine the iteration backlog.
- 2.4.3.1. Story slicing and splitting
- 2.4.3.2. Defining acceptance criteria

2.4.4. Explain the use of story points and velocity to plan and manage the work of a development team.
- 2.4.4.1. Purpose of the iteration (Sprint) planning meeting
- 2.4.4.2. Story points as a mechanism for the relative sizing of stories
- 2.4.4.3. Velocity as a measure of the capacity of a team to deliver working software
- 2.4.4.4. Burndown as a measure of progress

2.4.5. Describe the application of MoSCoW prioritisation to product, release and iteration backlogs.
2.4.5.1. Use of 'Must Have' to define the minimal viable product (MVP) or increment
2.4.5.2. Distinction between 'Must Have' and 'Should Have'
2.4.5.3. Use of 'Could Have' for contingency
2.4.5.4. Use of 'Won't Have This Time' to confirm out of scope

3. User Experience (UX) and User Interface (UI) (12.5%)

Candidates will be able to:

3.1. Recognise the definition of the following fundamental terms commonly used in UI/UX and explain their significance for UI design.

3.1.1. UI/UX
3.1.2. Style guides
3.1.3. UI paradigm
3.1.4. Metaphor
3.1.5. Idiom
3.1.6. UI pattern

3.2. Recognise and define the following UI/UX principles, centred around usability.

3.2.1. Nielsen's usability heuristic
3.2.2. Simplicity
3.2.3. Navigation
3.2.4. Feedback
3.2.5. Affordance
3.2.6. Intuitive
3.2.7. Tolerance
3.2.8. Consistency
3.2.9. Maximise re-use
3.2.10. Accessibility

3.3. Recognise the following UI paradigms.
3.3.1. Command line interface
3.3.2. Windows, Icons, Menus and Pointers (WIMP)
3.3.3. Desktop interface
3.3.4. Static and dynamic web interfaces
3.3.5. Direct manipulation
3.3.6. VR, AR and MR
3.3.7. Voice and gesture recognition

3.4. Recognise the following examples of UI related models and techniques and their potential use in HCI development.

3.4.1. User discovery and user analysis
3.4.2. Personas
3.4.3. Storyboards/wireframes/prototypes
3.4.4. Task analysis
3.4.5. Customer journey
3.4.6. Site and navigation maps
3.4.7. Data input validation and verification

3.5. Recognise a UI is made up of a group of controls. Identify the type of a given control.
    3.5.1. Classify controls as Command, Data Input, Presentation, Navigation or Feedback controls
    3.5.2. Recognise an example of each type, as given in the guidance notes

4. Digital solution architecture and design (15%)

Candidates will be able to:

4.1. Define architecture and understand the importance of architecture in developing a successful digital solution. Candidates will be able to recognise certain architectural styles and patterns and describe their benefits and drawbacks.

    4.1.1. Definition of architecture.
    4.1.2. The connection between architectural decisions and the realisation of requirements.
    4.1.3. Contrast the effects of 'good' and 'bad' architecture.
    4.1.4. Define a monolithic style of solution architecture and list its benefits/ drawbacks.
    4.1.5. Define a distributed style of solution architecture and list its benefits/ drawbacks.
    4.1.6. Recognise and define the system properties known as ‘cohesion’ and ‘coupling’.
    4.1.7. Explain what is meant by ‘service’, ‘interface’ and a ‘service-oriented’ style of architecture (SOA). Explain how this style benefits distributed systems.
    4.1.8. Recognise what an architecture pattern is and why it is beneficial to use one.
    4.1.9. Recognise the components of an MVC pattern of architecture and describe its merits.
    4.1.10. Recognise the components of a hexagonal pattern of architecture and describe its merits.

4.2. Identify the key elements of a typical contemporary Digital Solution and explain the role that each part plays in the overall architecture.

    4.2.1. Role of networks and the internet
    4.2.2. Role of the Client
    4.2.3. Role of the Server
    4.2.4. Need to link with persistent enterprise and data storage using DBMS; use of caches
    4.2.5. Need to link with Data Warehouse transaction storage through an ETL process
    4.2.6. Use of data pipes, event and job queues
    4.2.7. Role of APIS, middleware and service catalogues/ESB
    4.2.8. Use of partner links
    4.2.9. Role of cloud resources

4.3. Recognise the term 'web service' and the value this represents.

4.3.1. Describe and explain the terms 'web service', 'microservice' and 'API'.
4.3.2. Describe the value of using web services, microservices and APIs.
4.3.3. Explain the significance of the term ‘state’ as it applies to services and application design.
4.3.4. Describe the 2 basic service composition styles – orchestration and choreography, identifying the pros and cons of each.

4.4. Recognise and identify some of the issues that are inherent in a distributed systems architecture, and explain typical design trade-offs and solutions to these issues.

4.4.1. Security
4.4.2. Complexity
4.4.3. Scalability
4.4.4. Failure handling
4.4.5. Concurrency and Latency
4.4.6. Testing challenges
4.4.7. Cloud services as a solution to some of these issues

5. Data and information architecture and design (12.5%)

Candidates will be able to:

5.1. Distinguish between data, information and information systems and recognise distinct views of the components of an information system.

5.1.1. Recognise and explain a definition of data.
5.1.2. Recognise and explain a definition of information.
5.1.3. Recognise and explain a definition of information system.
5.1.4. Distinguish structured from unstructured data.
5.1.5. Explain the concepts of master data, reference data and transaction data.
5.1.6. Recognise 2 views of data processing in information systems
5.1.6.1. OLTP
5.1.6.2. OLAP

5.2. Explain the rationale for the architecture and design of data

5.2.1. Alignment of the application view of data with the corporate data architecture
5.2.2. Discovery of data requirements
5.2.3. Definition and communication of data design
5.2.4. Compliance with regulations and legislation
5.2.5. One version of the truth
5.2.6. Support for Data Analytics
5.2.7. Support for Data Management and Data Governance

5.3. Describe modelling data and information

5.3.1. The main elements of a data model
5.3.1.1. Entities/Entity types
5.3.1.2. Relationship/relationship degree
5.3.1.3. Attributes
5.3.1.4. Metadata/Data Dictionary
5.3.1.5. Models built with the UML Class Diagram notation
5.3.1.6. Models built with the ERD (IE) notation

5.3.2. Relational modelling (normalisation) as a data analysis and design technique.

5.3.3. Dimensional modelling for OLAP applications

5.3.4. Describe the differences between conceptual, logical and physical data models and explain the need for these different views of data.

5.3.5. Modelling data at rest and in motion.

5.4. Explain the relationships between applications software and the information system.

5.4.1. Recognise the four basic operations performed on data (CRUD)

5.4.2. Recognise the following data storage technologies used with digital solutions and the circumstances within which they might be used.
5.4.2.1. Relational technology and the use of SQL
5.4.2.2. Non-relational technology (Big Data and no-SQL)
5.4.2.3. Data warehouse
5.4.2.4. Data lake
5.4.2.5. Data mart/cube
5.4.2.6. Blockchain

5.4.3. Recognise the following data transmission standards used with digital solutions and the circumstances within which they might be used
5.4.3.1. XML
5.4.3.2. JSON
5.4.3.3. EDI
5.4.3.4. YAML

5.5. Explain some of the issues surrounding concurrency of data in multi-user environments, and identify the main strategies for dealing with this

5.5.1. Concept of ACID transactions

5.5.2. Need for transaction controls: commit and rollback

5.5.3. Need for timestamps and locking strategies

6. Quality assurance and quality control (10%)

Candidates will be able to:

6.1. Distinguish between quality assurance and quality control

6.2. Recognise the following quality-oriented activities
6.2.1. Inspection and adaptation
6.2.1.1. Product Review
6.2.1.2. Retrospective and continuous improvement

6.2.2. Refactoring

6.2.3. Adoption of a recognised SDLC

6.2.4. Test planning and execution
6.2.4.1. Static testing
6.2.4.2. Dynamic testing

6.2.5. Adoption of best practices in architecture and software development

6.3. Describe the fundamentals of software testing

6.3.1. Explain why testing is necessary
6.3.2. Distinguish between error, defect and failure
6.3.3. Define the following principles of testing
6.3.3.1. Testing shows the presence of defects, not their absence
6.3.3.2. Exhaustive testing is impossible
6.3.3.3. Early testing saves time and money
6.3.3.4. Defects cluster together
6.3.3.5. Beware of the pesticide paradox
6.3.3.6. Testing is context dependent
6.3.3.7. Absence-of-errors is a fallacy

6.3.4. Explain the concept of test condition, test case and test basis

6.4. Identify a range of common testing practices and processes

6.4.1. Identify the test levels used to organise testing activities within a typical software development initiative.
6.4.1.1. Explain the Agile Test Pyramid
6.4.1.2. Component/unit testing
6.4.1.3. Integration testing
6.4.1.4. System testing
6.4.1.5. Acceptance testing
6.4.1.6. Explain testing quadrants and how they align test levels and testing types

6.4.2. Identify the following test types and explain their purpose
6.4.2.1. Functional testing
6.4.2.2. Non-functional testing
6.4.2.3. Black box testing (behavioural)
6.4.2.4. White box testing (structural)
6.4.2.5. Regression testing

6.4.3. Describe the practice of Test-driven Development (TDD)
6.4.3.1. Writing test cases before starting any development activity
6.4.3.2. Writing automated unit and integration tests
6.4.3.3. Producing components that pass the tests
6.4.3.4. Refactoring code to improve quality and reduce technical debt

6.4.4. Describe the practice of Behaviour-driven Development (BDD)
6.4.4.1. Given-When-Then
6.4.4.2. Writing test scenarios using Gherkin

7. Digital solution acquisition, deployment and maintenance (15%)

Candidates will be able to:

7.1. Describe the benefits and drawbacks of various approaches to software acquisition

7.1.1. Building bespoke software components
7.1.2. Buying commercial software - COTS and MOTS

7.1.3. Hybrid approach: component-based architecture
7.1.4. Decision factors affecting whether to buy or build software
7.1.4.1. Problem uniqueness
7.1.4.2. Cost
7.1.4.3. Time
7.1.4.4. Risk
7.1.4.5. Competitive advantage
7.1.4.6. Training and support
7.1.4.7. Documentation
7.1.5. Building solutions with CRM and ERP platforms or “Best of Breed”
7.1.6. Open source software and frameworks
7.1.7. Cloud-based development
7.1.7.1. Pay per use
7.1.7.2. TCO
7.1.7.3. Scalability
7.1.7.4. Global availability
7.1.7.5. High resilience
7.1.7.6. Agreed service levels

7.2. Describe the following software engineering concepts and practices

7.2.1. Programming paradigms
7.2.1.1. Procedural
7.2.1.2. Functional
7.2.1.3. Object-oriented
7.2.2. Software engineering cycle:
7.2.2.1. Code
7.2.2.2. Compile
7.2.2.3. Build
7.2.2.4. Test
7.2.2.5. Debug
7.2.2.6. Integrate
7.2.2.7. Package
7.2.2.8. Release
7.2.3. Coding standards and examples of best practice in software development
7.2.3.1. Design patterns
7.2.3.2. SOLID principles
7.2.3.3. Code quality metrics
7.2.3.4. Code smells or anti-patterns
7.2.3.5. Managing technical debt
7.2.4. Code management techniques
7.2.4.1. Version control: local, central, distributed
7.2.4.2. Code branching
7.2.4.3. Feature flags/toggles
7.2.4.4. Configuration files
7.2.5. Development environments
7.2.5.1. IDE
7.2.5.2. Integration environment
7.2.5.3. Quality control environment
7.2.5.4. Pre-production environment
7.2.5.5. Live production environment
7.2.5.6. Full stack development

7.3. Describe the key aspects, benefits and considerations of the following practices and techniques

- 7.3.1. Iterative development
- 7.3.2. Incremental delivery
- 7.3.3. Product focus
- 7.3.4. Customer collaboration
- 7.3.5. Self-organising teams
- 7.3.6. Continuous improvement (Kaizen)
- 7.3.7. Kanban
- 7.3.8. Pair programming
- 7.3.9. Agile manifesto values
  - 7.3.9.1. Individuals and interactions over processes and tools
  - 7.3.9.2. Working software over comprehensive documentation
  - 7.3.9.3. Customer collaboration over contract negotiation
  - 7.3.9.4. Responding to change over following a plan

7.4. Identify and describe the following concepts used in the deployment of software:

- 7.4.1. Deployment strategies
  - 7.4.1.1. Direct changeover (big bang)
  - 7.4.1.2. Phased
  - 7.4.1.3. Pilot
  - 7.4.1.4. Parallel running
  - 7.4.1.5. Blue/green deployment
  - 7.4.1.6. Canary release
  - 7.4.1.7. Dark launch
- 7.4.2. Deployment automation and DevOps techniques
  - 7.4.2.1. The DevOps cycle
  - 7.4.2.2. CI
  - 7.4.2.3. CD

7.5. Identify and distinguish between the following types of maintenance and monitoring techniques

- 7.5.1. Service operations activities
- 7.5.2. Types of maintenance
  - 7.5.2.1. Corrective
  - 7.5.2.2. Adaptive
  - 7.5.2.3. Perfective
  - 7.5.2.4. Preventative
- 7.5.3. Application monitoring techniques
  - 7.5.3.1. Health monitoring: heartbeats for internal and external components and other specific activity measures
  - 7.5.3.2. Service logging, collection, aggregation, indexing, analysis and visualisation
  - 7.5.3.3. Application KPIs, for example digital take-up, user satisfaction, completion rate and cost per transaction

8. Cyber security (10%)

Candidates will be able to:

8.1. Explain the importance of having secure systems. Recognise that Cyber Security sits within the wider context of ERM. Identify standards applicable in this area and have an overview level of understanding of the scope of each.

- 8.1.1. Explain the importance of having secure systems, by recognising some common consequences to the enterprise of security breaches:
  - 8.1.1.1. Financial Loss
  - 8.1.1.2. Service outages
  - 8.1.1.3. Reputational exposure
  - 8.1.1.4. Legal consequences
  - 8.1.1.5. Damage to customer confidence
- 8.1.2. Recognise the following hierarchy of security regimes, define the terminology used and understand the key relationships between the concepts involved:
  - 8.1.2.1. Enterprise Risk Management
  - 8.1.2.2. Business Security
  - 8.1.2.3. Cyber Security
  - 8.1.2.4. Application Security
- 8.1.3. Explain the need for security controls and to balance the use of security controls in application software against factors such as risk, cost and usability.

8.2. Recognise and explain the following generic approach used by security experts for managing cyber security risks:

- 8.2.1. Define cyber security risk
- 8.2.2. A generic 4 step approach to managing security
  - 8.2.2.1. Identify
  - 8.2.2.2. Assess
  - 8.2.2.3. Mitigate
  - 8.2.2.4. Monitor

8.3. Identify and describe the purpose of the following range of security related techniques and best practices, which are especially related to digital solution development.

- 8.3.1. Forensics, audit, activity logging.
- 8.3.2. Security Information and Event Management (SIEM)
- 8.3.3. Penetration testing
- 8.3.4. Encryption of data, key and certificate management
- 8.3.5. Identity Access Management (IAM)

8.4. Recognise that every coding platform has inherent security weaknesses and explain the need for secure coding practices.

- 8.4.1. Security weaknesses inherent in the coding platform.
- 8.4.2. Secure Coding Practices (OWASP).
- 8.4.3. Use of Code Analysis tools, integrated with the IDE.
- 8.4.4. Risks arising from the use of 3rd party components.
- 8.4.5. Adherence to recognised Security Principles which apply to secure code application development, in particular web development.

8.5. Recognise that threat modelling takes place as part of software design.

8.5.1. Define the following threat modelling vocabulary and techniques:
- 8.5.1.1. Threat Agent
- 8.5.1.2. Threat Actor
- 8.5.1.3. Threat Target
- 8.5.1.4. Threat Action
- 8.5.1.5. Threat Event
- 8.5.1.6. Attack Vector
- 8.5.1.7. Vulnerability
- 8.5.1.8. Abuse Case

8.5.2. Use of the STRIDE and DREAD models.

8.5.3. Making use of a recognised ‘threat library’ like the OWASP Top 10.

Recommended Reading

BCS Foundation Certificate in Digital Solution Development Syllabus Detailed Guidance

Abbreviations

API	Application Programming Interface
AR	Augmented Reality
BDD	Behaviour Driven Development
CAP	Consistency, Availability and Partition tolerance
CD	Continuous Delivery
CI	Continuous Integration
CIA	Confidentiality, Integrity and Availability
COTS	Commercial Off-The-Shelf
CRM	Customer Relationship Management
CRM	Customer Relationship Management
CRUD	Create, Read, Update, Delete
DBMS	Database Management System
DevOps	A set of practices that combines software development (Dev) and IT operations (Ops) in order to shorten the systems development life cycle and provide continuous delivery of high-quality software.
DFD	Data Flow Diagram
DREAD	**D**amage, **R**eproducibility, **E**xploitability, **A**ffected users, Discoverability
DSD	Digital Solution Development
EDI	Electronic Data Interchange
ERD	Entity Relationship Diagram
ERM	Enterprise Risk Management
ERP	Enterprise Resource Planning
ESB	Enterprise Service Bus
GUI	Graphical User Interface
HCI	Human Computer Interface
IAM	Identity and Access Management
IDE	Integrated Development Environment
IE	Information Engineering
INVEST	Independent, Negotiable, Valuable, Estimable, Small & Testable
ISMS	Information Security Management System
JSON	JavaScript Object Notation
Kaizen	The concept of continuous improvement.
Kanban	A scheduling system for lean production, usually managed using a Kanban board that everyone involved can see and that shows work items, often on cards, moving through the steps of a process.
KPIs	Key Performance Indicators
MOLAP	Multidimensional Online Analytical Processing
MoR	Management of Risk
MoSCoW	Must have, Should have, Could have, and Won't have this time
MOTS	Modified Off-The-Shelf

MR	Mixed Reality
MVC	Model View Controller
MVP	Minimum Viable Product
NCSC	National Cyber Security Centre
OLAP	Online analytical processing
OLTP	Online transaction processing
OWASP	Open Web Application Security Project
REST	Representational State Transfer
ROLAP	Relational Online Analytical Processing
SAFECode	Software Assurance Forum for Excellence in Code
SDLC	Software Development Life Cycle
SIEM	Security Information and Event Management
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SOLID	**S**ingle Responsibility Principle, **O**pen/Closed Principle, **L**iskov Substitution Principle, **I**nterface Segregation Principle, **D**ependency Inversion Principle
Sprint	A usually short, fixed time period during which a team commits to complete certain work. Associated with the Agile and Scrum methodologies.
SQL	Structured Query Language
STRIDE	**S**poofing, **T**ampering, **R**epudiation, **I**nformation Disclosure, **D**enial of Service, and **E**levation of Privileges
TCO	Total Cost of Ownership
TDD	Test-Driven Development
TFD	Test First Development
UI	User Interface
UML	Unified Modelling Language
UX	User Experience
VR	Virtual Reality
WIMP	Windows, Icons, Menus, Pointer
XML	Extensible Mark-up Language
YAML	A recursive acronym for "YAML Ain't Mark-up Language"