Next Generation Higher National Unit Specification

Software Engineering Methods (SCQF level 8)

Unit code:	J7EF 48
SCQF level:	8 (24 SCQF credit points)
Valid from:	session 2023–24

Prototype unit specification for use in pilot delivery only (version 1.0) June 2023

This unit specification provides detailed information about the unit to ensure consistent and transparent assessment year on year.

This unit specification is for teachers and lecturers and contains all the mandatory information required to deliver and assess the unit.

This edition: June 2023 (version 1.0)

Unit purpose

This unit is for learners who have an interest in applying engineering design to the development of software solutions. This is a specialist unit, which is particularly relevant to learners studying software development. While learners do not need prior knowledge of software engineering principles, it is essential that they have experience of designing and developing software solutions. They can evidence this through units such as Software Development at SCQF level 8 or Database Design and Development at SCQF level 8.

During the unit, learners gain knowledge and understanding of the software development lifecycle (SDLC), software models, and the tools and design patterns that satisfy industry standards for the delivery of software projects in a business environment. They discover client requirements and perform requirements analysis to produce a software requirements document. They also select an appropriate software model and create a software design for a given software requirement. They represent this design using Unified Modeling Language (UML) or a similar language. Learners participate in design reviews and create test cases for a given software system.

The unit is included in the framework for the HND Software Development. On completion of the unit, learners may progress to other units in software engineering or software development at SCQF level 8 and above.

Unit outcomes

Learners who complete this unit can:

1 describe software management activities related to the modern software development lifecycle
2 explain the techniques supporting modern software engineering methods
3 define and analyse systems requirements and specify a system design to deliver these requirements
4 define and justify an architectural design style and pattern for a given systems design
5 design UML models to represent structural and behavioural aspects of a software system
6 illustrate various software testing strategies used in real-world projects
7 perform verification and validation of a software system design in relation to the functionality, usability, reliability, performance, and supportability (FURPS) criteria

Evidence requirements

Learners must provide knowledge evidence and product evidence.

Knowledge evidence

The knowledge evidence must demonstrate that the learner has an understanding of the knowledge specified in the ‘Knowledge and skills’ section. Learners can provide evidence in a report or a presentation. Knowledge evidence can also be produced by a question paper that samples from each of the elements of software engineering:

- software development lifecycle process models:
- software requirement specification
- requirements gathering: approaches and outputs
- requirements analysis process:
- non-functional requirements
- system models
- patterns: architectural and design
- reference architectures
- user interface (UI) design
- requirements validation
- design reviews
- software testing
- code quality review
- software quality models

Product evidence

Learners demonstrate that they can successfully apply software engineering methods and tools to a range of real-world problems. They must evidence all the skills listed in the 'Knowledge and skills' section, covering the gamut of methods and processes from establishing software requirements to the design and validation of a software solution.

Learners can produce evidence over an extended period in lightly-controlled conditions or generate it holistically in conjunction with other units in a group award. Evidence produced in lightly-controlled conditions must be authenticated. The Guide to Assessment provides further advice on methods of authentication.

The standard of evidence should be consistent with the SCQF level of the unit.

Knowledge and skills

The following table shows the knowledge and skills covered by the unit outcomes:

Knowledge	Skills
Learners should understand: ♦ software development lifecycle process models: — Waterfall — Agile — Spiral — Prototyping — RAD — Scrum ♦ software requirement specifications — business requirements — system requirements — functional and non-functional requirements — interface requirements ♦ requirements gathering — stakeholders — use-cases — user stories ♦ requirements analysis process — modelling — functions — behaviours ♦ non-functional requirements — security — capacity — reliability — compatibility — scalability — maintainability — usability ♦ software requirements analysis — system constraints — risk	Learners can: ♦ discover requirements from a client consultation and create a requirements document with use cases ♦ define and analyse systems requirements ♦ specify and document a system design to deliver requirements ♦ define and justify an architectural design style and pattern for a given systems design ♦ design UML models to represent structural and behavioural aspects of a software system ♦ illustrate various software testing strategies used in real-world projects ♦ perform verification and validation of a software system design in relation to the FURPS criteria

Knowledge	Skills
Learners should understand: ♦ system models — analysis — design — context — data — object — behavioural ♦ structured analysis — data flow diagrams (DFDs) — DFD levels — data dictionaries ♦ entity relationship diagrams (ERDs): — data objects and entities — data attributes — relationships — cardinality — modality ♦ architectural design ♦ control styles ♦ patterns — architectural — design — idiom ♦ reference architectures (patterns): — data-centred — data flow — call and return — object-oriented — layered ♦ architecture description language (ADL) — requirements — benefits — problems — UML as an ADL — use of XML-based architecture description language (xADL)	

Knowledge	**Skills**
Learners should understand: ♦ patterns — MVC — singleton — observer — decorator ♦ principles of software design — abstraction — modularity — information hiding — stepwise refinement — refactoring ♦ horizontal versus vertical partitioning ♦ concurrency ♦ user interface design ♦ verification and validation ♦ requirements validation — review team — checklist for consistency — omissions — ambiguity ♦ design reviews — conceptual — critical — program (static) ♦ software testing — component — integration — system — test case design — test automation ♦ validation of reliability and security ♦ software quality	

Knowledge	**Skills**
Learners should understand: ♦ ISO/IEC 25010:2011 standard (FURPS) — functional suitability — reliability — operability — performance efficiency — security — compatibility — maintainability — transferability ♦ code quality review — readability — maintainability — coding standards — linting tools ♦ quality models — McCall — Boehm — Dromey	

Meta-skills

Throughout the unit, learners develop meta-skills to enhance their employability in the computing sector.

The unit helps learners develop the meta-skills of self-management, social intelligence and innovation. Learners should develop meta-skills naturally throughout the unit. You should encourage learners to develop a minimum of one area in each of the three categories, but they do not need to cover all the suggested subsection. The following suggestions may help shape delivery and assessment, and vary depending on the chosen topics and assessment method.

Self-management

This meta-skill includes:

- focusing: demonstrating the attention to detail that developing program code and syntax demands and that is crucial to successful coding practices
- adapting: reflecting critically on the processes of software engineering; self-learning
- initiative: displaying independent thinking; demonstrating the self-motivation, responsibility and decision making required at each stage in the selected software engineering methodology

Social intelligence

This meta-skill includes:

- communicating: receiving information; obtaining user requirements and verifying software designs
- collaborating: listening and conveying information
- leading: being a change catalyst

Innovation

This meta-skill includes:

- curiosity: information sourcing; recognising problems and devising solutions
- creativity: demonstrating a maker mentality; being imaginative; visualising
- sense-making: pattern recognition; holistic thinking; careful analysis of requirements
- critical thinking: logical and computational thinking; deconstruction of data objects; judgement

Delivery of unit

You can deliver this unit in conjunction with Software Development at SCQF level 8. This provides practical activities to help understanding of the concepts presented in the unit.

Alternatively, you can build delivery and assessment around an extended case study that offers sufficient complexity to give learners exposure to different approaches. You can expose learners to a range of software requirements, allowing them to consider the range of software engineering processes and experience their application.

Learning should be as learner-centred as possible, and you should take the role of client, coach or mentor as appropriate and provide guidance, resources and advice.

The sequencing of the unit should follow the pattern of most SDLC models:

- requirements gathering
- analysis
- design
- validation and verification
- software production and testing

The time required varies depending on the previous experience of individual learners. Based on 120 hours delivery and assessment time, we suggest the following distribution:

Outcome 1 — Describe software management activities related to the modern software development lifecycle
(15 hours)

Outcome 2 — Explain the techniques supporting modern software engineering methods
(10 hours)

Outcome 3 — Define and analyse systems requirements and specify a system design to deliver these requirements
(30 hours)

Outcome 4 — Define and justify an architectural design style and pattern for a given systems design
(25 hours)

Outcome 5 — Design UML models to represent structural and behavioural aspects of a software system
(15 hours)

Outcome 6 — Illustrate various software testing strategies used in real-world projects
(15 hours)

Outcome 7 — Perform verification and validation of a software system design in relation to the FURPS criteria
(10 hours)

Additional guidance

The guidance in this section is not mandatory.

Content and context for this unit

The general context for the unit is for learners to apply software engineering principles, and to develop a significant software solution to a set of user requirements.

Start the unit with an overview of the SDLC and activities, including requirements gathering and analysis, design, development, testing, deployment, maintenance, and retirement.

You should describe and compare the range of process options for software development, including Waterfall, Agile, Spiral, Prototyping, RAD, and Scrum. You should provide examples of their operation.

Introduce the requirements-gathering process and give learners the opportunity to practice discovering, prioritising, documenting and validating requirements. You should emphasise the importance of documenting requirements. Discuss the relevance of non-functional requirements to the success of a software solution, including performance, security, and usability.

This leads to the analysis of these requirements and their representation using UML or equivalent, and forms the basis for subsequent validation exercises. You should introduce UML as a general-purpose modelling language and present the choice of diagram types in UML, with examples of their use. Diagram types should include sequence, class, state machine, activity, and component.

Learners should apply the structured analysis process to examples of requirements, including decomposition, data modelling, and process modelling. They should see examples of horizontal and vertical partitioning. Learners should become familiar with DFDs and ERDs.

At this stage, learners are ready to verify and validate their system design against functional and non-functional requirements, and other assumptions or constraints. Validating the design involves the client, while verifying that the design is correct involves model-based verification.

Learners should understand that there is a range of architectural styles for software development projects and that the choice of style is determined by functional and non-functional requirements, the technology stack in use, other project constraints, and team expertise. This leads them to consider common architectural styles, including monolithic, microservices, event-driven, and layered. You should introduce the use of an architectural design language, drawing on current practice.

Introduce the concept of a design pattern as a specific repeatable solution to a commonly-occurring problem in software design, and give examples such as Model-View-Controller (MVC), Model-View-View-Model (MVVM), observer and repository.

You should cover the basics of user-experience testing and its contribution to interface design, and the range of strategies for testing, including unit testing, integration testing, system testing, acceptance testing, performance testing, and regression testing. This should lead on to test construction, test cases and automated testing. Learners should experience static testing of some simple code fragments.

You should cover the concept of software quality, including aspects such as FURPS and implementation. For common use, you should introduce McCall’s quality model, and refer to Boehm’s and Dromey's quality models as alternatives.

Your delivery of the unit should engage learners in applying software engineering methods and approaches to real-world problems. You should select these problems to ensure that learners can apply as wide a range of methods as possible. For some practical work, it may be more suitable for learners to work in pairs or other-sized groups.

For validation and verification exercises, you should act as the client or expert.

Practical activities

There are several approaches to helping learners to understand through practical activities, such as linking the unit with Software Development at SCQF level 8. You could also consider the following approaches:

Focus on hands-on projects

Assign practical projects that allow learners to apply software engineering principles in a real-world context. This can include building a software system from scratch, or working on an existing open-source project.

Use real-world examples

Use real-world examples of software engineering problems and solutions to help learners understand how to apply theoretical concepts in practice. Share case studies and success stories from industry to illustrate how software engineering principles are used in the real world.

Include coding exercises

Provide coding exercises that allow learners to practice writing code using the software engineering principles that you are teaching. These exercises can be based on the projects that you assign, or they can be stand-alone exercises that focus on specific software engineering concepts.

Use collaboration tools

Use collaboration tools like Git, GitHub, or Bitbucket to encourage collaboration among learners. This helps them learn how to work together as a team and manage software projects collaboratively.

Invite guest speakers

Invite industry professionals to speak to your class about their experiences working in software engineering. This provides learners with a more realistic understanding of what it is like to work as a software engineer and the challenges they might face.

Overall, the key is to create a curriculum that emphasises hands-on, practical experience and provides learners with opportunities to apply the concepts they are learning in real-world situations.

Software patterns

Here are some practical examples for learners to explore when studying software patterns:

Factory method pattern

Implement a simple web application that creates different types of objects based on user input. For example, the application could create different types of documents (such as PDF, Word, or HTML) based on user input.

Observer pattern

Implement a simple messaging application that allows users to subscribe to different chat rooms. When a user sends a message to a chat room, all subscribers receive the message.

Singleton pattern

Implement a logging component that ensures only one instance of the logger is created throughout the application. This can help to centralise logging and ensure that all logs are captured consistently.

Adapter pattern

Implement a data conversion component that converts data from one format to another. For example, the component could convert data from a comma-separated values (CSV) file to a JavaScript Object Notation (JSON) format.

Decorator pattern

Implement a text formatting component that allows users to apply different formatting options (such as bold, italic, or underline) to text. Learners can apply the formatting options dynamically at runtime.

Software architectures

Here are some practical examples that can help learners understand the concept of software architectures:

Client-server architecture

Build a simple web application that allows users to submit a form and store the data on a server.

Microservices architecture

Build a simple e-commerce application that consists of separate services for catalogue management, order processing, and payment processing.

Layered architecture

Build a simple create, read, update, delete (CRUD) application that separates the user interface, business logic, and data access layers.

Event-driven architecture

Build a simple messaging application that allows users to send and receive messages in real-time.

Service-oriented architecture

Build a simple weather application programming interface (API) that provides weather data to other applications.

Software validation and verification

Here are some practical activities that can help learners understand software validation and verification:

Test plan creation

Provide learners with a sample software system and ask them to create a test plan to verify and validate the system. This can include designing test cases, identifying test data, and determining success criteria.

Code inspection

Ask learners to review a piece of code and identify potential defects and errors. This can help them understand how to use code inspection to identify and correct errors in software systems.

User testing

Conduct a user testing session with a group of learners, using a sample software system. Ask them to provide feedback on the usability, functionality, and reliability of the system. This can help them understand how user testing can validate and verify software systems.

Bug fixing

Provide learners with a sample software system that contains bugs or defects. Ask them to identify and fix the bugs, using a structured approach to debugging. This can help learners understand how the verification and validation process is used to identify and correct errors in software systems.

Requirements review

Provide learners with a set of software requirements and ask them to review the requirements and identify potential inconsistencies or conflicts. This can help them understand how to use the validation and verification process to ensure that software systems meet the needs of stakeholders.

These practical activities can help learners understand the importance of software validation and verification and how these processes ensure the quality and effectiveness of software systems.

Approaches to assessment

You can obtain product evidence by setting a series of tasks that are based on actual user requirements or existing software designs. Learners can gather product evidence in the form of checklists or reports. Learners may also create presentations or demonstrations of solutions to tasks.

However, our preferred method of obtaining product evidence is through an extended case study or project brief that enables learners to demonstrate the full range of skills required in the unit. You must provide feedback to learners that is authentic and constructive to the objectives of their case study or project.

Learners can produce knowledge evidence by a question paper that samples the knowledge evidence detailed in the ‘Evidence requirements’ section. Alternatively, they can produced it in the form of reports or presentations, including research reports.

Equality and inclusion

This unit is designed to be as fair and as accessible as possible with no unnecessary barriers to learning or assessment.

You should take into account the needs of individual learners when planning learning experiences, selecting assessment methods or considering alternative evidence.

Guidance on assessment arrangements for disabled learners and/or those with additional support needs is available on the assessment arrangements web page: www.sqa.org.uk/assessmentarrangements.

Information for learners

Software Engineering Methods (SCQF level 8)

This information explains:

- what the unit is about
- what you should know or be able to do before you start
- what you need to do during the unit
- opportunities for further learning and employment

Unit information

This unit is for learners who have an interest in the application of engineering design principles to the development of software solutions. This is a specialist unit, particularly relevant to learners studying software development. While no prior knowledge of software engineering principles is required, it is essential that you have experience of designing and developing software solutions. You can evidence this through units such as Software Development at SCQF level 8 or Database Design and Development at SCQF level 8.

In the unit, you gain knowledge and understanding of the software development lifecycle (SDLC), software models, and the tools and design patterns that satisfy industry standards for the delivery of software projects in a business environment. You discover and document the business requirements of a client, perform analysis to produce a software requirements document, and validate this against the business requirement.

You select appropriate software models and create software designs for a given software requirement. You represent these designs using a modelling language such as Unified Modeling Language (UML). You participate in design reviews and create test cases for a given software system.

Your knowledge and understanding of software engineering concepts and approaches may be assessed through various means, such as case studies, assignments, and question papers. Your competence in establishing software requirements, conducting an analysis and choosing an architectural style for the software design is assessed through product evidence.

Throughout the unit, you develop meta-skills covering self-management, social intelligence and innovation. You will develop awareness of sustainability issues in the application of computing solutions.

On completing the unit, you have the knowledge and competence to progress to other units in software engineering or in software development methodologies at SCQF level 8 and above.

Administrative information

Published: June 2023 (version 1.0)

Superclass: CB

History of changes

Version	Description of change	Date

Note: please check SQA's website to ensure you are using the most up-to-date version of this document.