Higher National Unit Specification

General information

Unit title: Software Development: Analysis and Design (SCQF level 9)

Unit code: HK6H 36

Superclass:	CB
Publication date:	May 2017
Source:	Scottish Qualifications Authority
Version:	01

Unit purpose

The purpose of this unit is to introduce learners to the use of architecture and design patterns in the software analysis and design process.

The unit reviews object-oriented techniques used in the software analysis and design process, and covers the use of templates for solving problems that occur in many different situations or applications.

Learners will make use of architecture patterns to demonstrate the characteristics and behaviour of applications and they will gain practical experience in the use of creational, structural and behavioural design patterns to show relationships and interactions between classes or objects without specifying the final application classes or objects involved.

The unit is suitable for learners who have previous experience in the use of object-oriented analysis and design techniques (in software development) and wish to further develop their skills in using architecture and design patterns.

On completion of the unit, learners will be competent in planning projects and using architecture and design patterns in software analysis and design.

Having completed this unit, along with the unit *Software Development: Implementation and Testing* (SCQF level 9), learners should be able to progress to the unit *Software Development: Project* (SCQF level 9) or equivalent.

Unit title: Software Development: Analysis and Design (SCQF level 9)

Outcomes

On successful completion of the unit the learner will be able to:

1 Describe object-oriented analysis and design techniques in software development.
2 Plan the development of a software project.
3 Use architecture patterns to demonstrate the characteristics and behaviour of applications.
4 Use design patterns to demonstrate the relationships and interactions between classes or objects.

Credit points and level

2 Higher National unit credits at SCQF level 8: (16 SCQF credit points at SCQF level 8)

Recommended entry to the unit

Entry to this unit is at the discretion of the centre. However, it would be useful if the learner had prior knowledge/skills in applying object-oriented techniques in the analysis and design of computer software. This could be evidenced by possession of the Higher National Unit HA4D 35 *Software Development: Analysis and Design* (SCQF level 8) or equivalent.

Core Skills

Opportunities to develop aspects of Core Skills are highlighted in the support notes for this unit specification.

There is no automatic certification of Core Skills or Core Skill components in this unit.

Context for delivery

If this unit is delivered as part of a group award, it is recommended that it should be taught and assessed within the subject area of the group award to which it contributes.

The Assessment Support Pack (ASP) for this unit provides assessment and marking guidelines that exemplify the national standard for achievement. It is a valid, reliable and practicable assessment. Centres wishing to develop their own assessments should refer to the ASP to ensure a comparable standard. A list of existing ASPs is available to download from SQA's website **(http://www.sqa.org.uk/sqa/46233.2769.html)**.

Equality and inclusion

This unit specification has been designed to ensure that there are no unnecessary barriers to learning or assessment. The individual needs of learners should be taken into account when planning learning experiences, selecting assessment methods or considering alternative evidence.

Further advice can be found on our website **www.sqa.org.uk/assessmentarrangements**.

Higher National Unit Specification: Statement of standards

Unit title: Software Development: Analysis and Design (SCQF level 9)

Acceptable performance in this unit will be the satisfactory achievement of the standards set out in this part of the unit specification. All sections of the statement of standards are mandatory and cannot be altered without reference to SQA.

Where evidence for Outcomes is assessed on a sample basis, the whole of the content listed in the Knowledge and/or Skills section must be taught and available for assessment. Learners should not know in advance the items on which they will be assessed and different items should be sampled on each assessment occasion.

Outcome 1

Describe object-oriented analysis and design techniques in software development.

Knowledge and/or skills

- Object-oriented analysis and design techniques
- Definition of software requirements
- Design of software solutions
- Modelling of software solutions

Outcome 2

Plan the development of a software project.

Knowledge and/or skills

- Traditional and agile methodologies
- Iterative development methodologies
- Requirements gathering and analysis
- Design of software components
- Development of software components
- Testing of software solutions
- Deployment of software solutions

Higher National Unit Specification: Statement of standards (cont)

Unit title: Software Development: Analysis and Design (SCQF level 9)

Outcome 3

Use architecture patterns to demonstrate the characteristics and behaviour of applications.

Knowledge and/or skills

- ♦ Layered architecture patterns for applications that organise components into horizontal layers performing a specific role
- ♦ Event-driven architecture patterns for single-purpose event processing components that asynchronously receive and process events
- ♦ Micro-kernel architecture patterns for product-based applications that allow the addition of application features as plug-ins
- ♦ Microservices architecture patterns for applications consisting of separately-deployed service components
- ♦ Space-based architecture patterns for applications that address and solve scalability and concurrency issues

Outcome 4

Use design patterns to demonstrate the relationships and interactions between classes or objects.

Knowledge and/or skills

- ♦ Creational, structural and behavioural design patterns
- ♦ Creational design patterns to solve software design problems
- ♦ Structural design patterns to solve software design problems
- ♦ Behavioural design patterns to solve software design problems

Evidence requirements for this unit

Learners will need to provide evidence to demonstrate their knowledge and/or skills across all outcomes. The evidence requirements for this Unit will take two forms:

1 **Knowledge (written/oral) evidence** (for Outcomes 1, 2, 3 and 4).
2 **Product evidence** (for Outcomes 2, 3 and 4).

Note that Outcome 1 covers only cognitive competences while Outcomes 2, 3 and 4 cover both cognitive and practical competencies. Each of the knowledge and/or skills items listed in Outcomes 2, 3 and 4 is practical in nature but has an underlying cognitive competence that must be evidenced.

Higher National Unit Specification: Statement of standards (cont)

Unit title: Software Development: Analysis and Design (SCQF level 9)

The written/oral evidence will be the definitions, descriptions and explanations required for Outcomes 1, 2, 3 and 4. The product evidence will be artefacts showing the use of object-oriented techniques to analyse requirements and design and model software solutions as required for Outcomes 2, 3 and 4.

The product evidence (Outcomes 2, 3 and 4) will relate to **at least one problem**. Learners should make use of object-oriented techniques to analyse software requirements, and design and model solutions. The evidence would consist of the requirements analysis and the software design and model. All of the knowledge and skills statements for Outcomes 2, 3 and 4 must be evidenced.

Evidence is normally required for **all** of the knowledge and skills in every outcome. This means that every Knowledge and Skills statement should be evidenced. However, sampling may be used in a specific circumstance (see below).

The amount of evidence should be the **minimum** consistent with the defined knowledge and skills. For Outcome 3, it is sufficient for the learner to use a minimum of **one** architecture pattern. For Outcome 4, it is sufficient for the learner to use a minimum of **one** design pattern.

Evidence may be wholly or partly produced under controlled conditions. When evidence is produced in uncontrolled or loosely controlled conditions it must be authenticated. The *Guide to Assessment* provides further advice on methods of authentication.

There are **no** time limitations on the production of evidence (but see exception below). The evidence may be produced at any time during the life of the unit. Learners may use reference materials when undertaking assessment (but see exception below).

Sampling is permissible when the written/oral evidence for Outcomes 1, 2, 3 and 4 is produced by a test of knowledge and understanding. The test may take any form (including oral) but must be supervised, unseen and timed. The contents of the test must sample **broadly** and **proportionately** from the contents of Outcomes 1, 2, 3 and 4 with approximately equal weighting for each Outcome. Access to reference material is not appropriate for this type of assessment.

The *Guidelines on Approaches to Assessment* (see the support notes section of this specification) provides specific examples of instruments of assessment.

Higher National Unit Support Notes

Unit title: Software Development: Analysis and Design (SCQF level 9)

Unit support notes are offered as guidance and are not mandatory.

While the exact time allocated to this unit is at the discretion of the centre, the notional design length is 80 hours.

Guidance on the content and context for this unit

A free booklet on architecture patterns can be obtained from:

http://www.oreilly.com/programming/free/software-architecture-patterns.csp

The classic reference for design patterns is: Gamma et al (1995). Design Patterns. Massachusetts: Addison-Wesley. ISBN 978-0-201-63361-0.

Another useful reference is Hendrickson, Mike; Loukides, Mike, eds. (2004) Head First Design Patterns. California: O'Reilly Media. ISBN 978-0-596-00712-6.

Numerous examples of the use of design patterns can be found at:

http://www.codeproject.com/KB/architecture/#Design+Patterns

Outcome 1

Learners should be aware of a range of conventional and contemporary approaches to the software development process and the associated analysis and design tools and models. Particular consideration should be given to the Waterfall development approach and the agile development approach.

It should be made clear that agile and object-oriented design are different. Agile software development encompasses a group of software development methods based on iterative and incremental development, whereas object-oriented design is the process of planning a system of interacting objects for the purpose of solving a software problem. The main thing to note in these definitions is the fact that one involves planning and the other uses incremental and emergent development strategies.

Learners should be able to define software requirements using object-oriented analysis techniques. This involves finding and organising objects, object interaction, object behaviour, object internals and common models (use cases and object models).

Unit title: Software Development: Analysis and Design (SCQF level 9)

They should be able to design software solutions using object-oriented techniques. This includes working within implementation constraints (hardware and software platforms, performance requirements, persistent storage and transactions, usability, budgets and time limitations), mapping technology-independent concepts onto implementing classes and interfaces to produce a model of the solution domain and designing software architectures by applying architectural patterns and design patterns. They should be able to model software solutions using object-oriented techniques. This involves modelling dynamic behaviours (business processes and use cases), modelling static structures (classes and components) and using object-oriented modelling languages.

Outcome 2

Agile project management methods focus on people, communications, product and flexibility. They use a range of unique methods that combine to produce an efficient software development process. An agile development model involves the same type of work as a traditional waterfall model: creating requirements and designs, developing the product and integrating the product with other products as required. The product must be tested, any problems fixed, and deployed for use. However, instead of tackling all at once, the product is broken into iterations (smaller segments of the overall project), called sprints.

In the phases of a traditional waterfall development, you move to the next phase only when the previous one is complete. With agile software development, you use an empirical control method — a process of making decisions based on the realities observed in the actual project. Empirical control requires transparency, frequent inspection and adaptation. An empirical approach can be very effective both in developing new products and in enhancing and upgrading existing projects. Frequent and first-hand inspection of the work completed to date allows immediate adjustments to be made as necessary.

In iterative development the development of a large application is split into smaller chunks. Code is designed, developed and tested in repeated cycles. With each iteration, additional features can be designed, developed and tested until a fully functional software application is ready to be deployed.

In a waterfall project, the majority of requirements must be gathered right at the start of the project. This aids the definition of what needs to be delivered, so that everyone knows the scope of the project and has a detailed description of what will be built. In an agile iterative project, the customer does not need to specify all their requirements in detail up front. Instead the broad scope of the project is defined at the start and during each iteration enough requirements are analysed to support the delivery of that iteration's functionality.

Agile design involves the application of certain agile development principles to the design process. The agile approach emphasises people and interactions over processes and tools. This means communicating frequently both within teams and with the customer. Daily scrum meetings allow the whole team to monitor the activities of its members, creating a consistent feedback loop that enables teams to adjust based on what customers are telling them, while checking frequently to ensure their work is functional in the target environment.

The agile development process emphasises the production of on-time and on-budget deliverables and accepts that products can always be tweaked later in the cycle by means of iterations: short, intense periods of production with smaller, more achievable goals that build in further iterations taking place later.

The traditional design aims to present perfect products to clients. However, for complex projects, it is not useful to design for long periods in the abstract without client input. Client demands can change quicker than designers can produce. Adopting an agile approach of involving clients into every phase of the process and producing a constant stream of deliverables can help fix this, as it allows clients to play around with designs as they go.

The agile method of continuous integration has developers integrating code on a daily basis. This takes the mystery out of integration, allowing developers to catch bugs as they arise and either fix them immediately or add them into the backlog for the next iteration.

Frequent testing plays an important role keeping iterations on track. It allows identified problems to be addressed either immediately or in the next iteration. This helps keep the design process on track and helps fuel creativity.

Agile projects make use of continuous deployment, an extension of continuous integration, aimed at minimising the time between developers writing code and this new code being used by live users. This has two major benefits:

- earlier return on investment for each feature developed, reducing the need for large capital investments.
- earlier feedback from users on each new feature as it is released to production.

Developers must be careful to avoid the mistake of optimising for maximum frequency of deployments as this will not, by itself, result in increased quality.

Outcome 3

Learners should know that architecture patterns help define the basic characteristics and behaviour of an application. Some architecture patterns are well-suited to the production of highly-scalable applications while others are geared towards the production of highly-agile applications. Knowledge of the characteristics, strengths and weaknesses of different architecture patterns is needed to choose the one that meets specific business needs and goals.

Without a clear and well-defined architecture, most developers will fall back on the traditional layered architecture pattern (n-tier architecture) which creates implicit layers by separating source-code modules into packages. This can result in a collection of unorganised source-code modules that lack clear roles, responsibilities and relationships to one another.

Applications without a formal architecture are often tightly coupled, brittle, difficult to change and lack a clear vision or direction. It can be difficult to determine the architectural characteristics of the application without understanding the inner-workings of every component in the system. Questions about deployment and maintenance (eg scalability, performance characteristics, ease of modification, responsiveness) can be difficult to answer.

Learners should be able to justify architecture decisions, especially the choice of a particular architecture pattern or approach.

Outcome 4

Learners should know that **creational design patterns** deal with object creation mechanisms, trying to create objects in a manner appropriate to the situation. They involve two basic ideas: encapsulating knowledge about which concrete classes the system uses and hiding how instances of these concrete classes are created and combined.

They should be aware of the following well-known creational patterns:

- **Abstract factory pattern:** provides an interface for creating related or dependent objects without specifying their concrete classes.
- **Builder pattern:** separates the construction of a complex object from its representation so that the same construction process can create different representations.
- **Factory method pattern:** allows a class to defer instantiation to subclasses.
- **Prototype pattern:** specifies the kind of object to create using a prototypical instance, and creates new objects by cloning this prototype.
- **Singleton pattern:** ensures that a class only has one instance, and provides a global point of access to it.

Learners should know that **structural design patterns** ease the design by identifying a simple way to realise relationships between entities. Major Structural Patterns include:

- **Adapter pattern**: 'adapts' one interface for a class into one that a client expects
- **Composite pattern:** a tree structure of objects where every object has the same interface
- **Aggregate pattern:** a version of the Composite pattern with methods for aggregation of children
- **Bridge pattern:** decouple an abstraction from its implementation so that the two can vary independently

Learners should know that **behavioural design patterns** identify common communication patterns between objects and realise these patterns to increase flexibility in carrying out this communication. Major behavioural design patterns include:

- **Chain of responsibility pattern:** command objects are handled or passed on to other objects by logic-containing processing objects
- **Command pattern:** command objects encapsulate an action and its parameters
- **Interpreter pattern:** implement a specialised computer language to rapidly solve a specific set of problems

Unit title: Software Development: Analysis and Design (SCQF level 9)

Guidance on approaches to delivery of this unit

This unit is a component of the PDA Software Development (SCQF level 9). It should be delivered before, or in parallel with the unit *Software Development: Implementation and Testing* (SCQF level 9). Both of the units should be completed before delivery of the unit *Software Development: Project* (level 9).

The outcomes may be delivered in the order in which they are written. They have been written with a learning sequence in mind.

The actual distribution of time between outcomes is at the discretion of the centre. However, one possible approach is to distribute the available time as follows:

Outcome 1: 20 hours
Outcome 2: 20 hours
Outcome 3: 20 hours
Outcome 4: 20 hours

It is anticipated that the required concepts will be introduced by the teacher and reinforced by appropriate examples.

There is significant scope in this unit to illustrate concepts and skills with case studies of analysis and design. The majority of time in this unit will be spent on the practical application of the theoretical aspects of the unit.

Throughout this unit, learner activities should relate to their vocational interests.

Guidance on approaches to assessment of this unit

Evidence can be generated using different types of assessment. The following are suggestions only. There may be other methods that would be more suitable to learners.

A traditional approach to assessment would involve an end of unit test (Outcomes 1, 2, 3 and 4) and a practical assessment (Outcomes 2, 3 and 4).

The end of unit test would sample from the knowledge and understanding contained in Outcomes 1, 2, 3 and 4. The test could consist of selected response or constructed response questions. A selected response test could be composed of multiple-choice questions (MCQs) or multiple-response questions (MRQs), and would be marked and assessed traditionally or online *via* SOLAR. For example, the test could comprise 40 multiple-choice questions, each of which could have four options (A–D), distributed equally across the four outcomes, with an appropriate pass mark such as 60%. This test would be taken, sight-unseen, in controlled and timed conditions without reference to teaching materials. A suitable duration could be 60 minutes. Given the level of this unit (SCQF level 9), the test would comprise questions spanning a range of cognitive skills (including higher level ones involving analysis and synthesis).

Unit title: Software Development: Analysis and Design (SCQF level 9)

Practical assessment (Outcomes 2, 3 and 4) could involve the practical application of the skills taught in each outcome. The exercise could be assessed holistically, without a marking scheme and not assigned a specific score, and given a simple ‘pass/fail’ grade. All of the knowledge and skills would be evidenced in this assessment. There would be no time limitations (beyond the practicality of completing the unit within the scheduled timetable) for this assessment.

A more contemporary approach to assessment could use a web log (blog) to record learning throughout the life of the unit. If this approach is taken, then sampling would not be appropriate. The blog would contain evidence for all knowledge and skills statements. The blog would record, on a daily or weekly basis, the learning that has occurred. It would contain textual definitions, descriptions and explanations as required by the knowledge and skills statements in the outcomes (all outcomes), including hyperlinks and embedded multimedia (audio, graphic or video).

The blog could encompass all outcomes, including Outcome 2, 3 and 4 (which are practical). This could be done by the blog demonstrating how architecture and design patterns could be applied to problems. Given that the blog would, most likely, be completed at various times and locations throughout the life of the unit, some form of authentication would be necessary. There would be no time limitation on the completion of the blog since it would be done on an on-going basis throughout the life of the unit.

Centres are reminded that prior verification of centre-devised assessments would help to ensure that the national standard is being met. Where learners experience a range of assessment methods, this helps them to develop different skills that should be transferable to work or further and higher education.

Opportunities for e-assessment

E-assessment may be appropriate for some assessments in this unit. By e-assessment we mean assessment which is supported by Information and Communication Technology (ICT), such as e-testing or the use of e-portfolios or social software. Centres which wish to use e-assessment must ensure that the national standard is applied to all learner evidence and that conditions of assessment as specified in the Evidence Requirements are met, regardless of the mode of gathering evidence. The most up-to-date guidance on the use of e-assessment to support SQA’s qualifications is available at **www.sqa.org.uk/e-assessment**.

Opportunities for developing Core and other essential skills

There is no automatic certification of Core Skills or Core Skill components in this unit. The unit should aid the development of Computational Thinking skills.

History of changes to unit

Version	Description of change	Date

General information for learners

Unit title: Software Development: Analysis and Design (SCQF level 9)

This section will help you decide whether this is the unit for you by explaining what the unit is about, what you should know or be able to do before you start, what you will need to do during the unit and opportunities for further learning and employment.

The purpose of this unit is to introduce you to project planning and the use of architectural patterns and design patterns in the software analysis and design process. The unit reviews the object-oriented analysis and design techniques used in the software analysis and design process, introduces the main concepts of project planning and covers the use of architecture and design patterns for solving problems that occur in many different situations or applications.

You will gain practical experience in the use of architectural and design patterns to show relationships and interactions between classes or objects without specifying the final application classes or objects involved.

The unit is suitable for learners who have previous experience in the use of object-oriented analysis and design techniques in software development and wish to further develop their skills in project planning and in using architecture and design patterns.

You may be assessed in various ways, including multiple-choice question relating to the theoretical knowledge covered in the unit, and practical exercises applying the analysis and design skills learned.

On completion of the unit, you will be competent in planning projects and in using architecture and design patterns in the software analysis and design process. Along with the unit *Software Development: Implementation and Testing* (SCQF level 9), you should be able to progress to the unit *Software Development: Project* (SCQF level 9) or equivalent.