Higher National Unit specification

General information

Unit title: Software Development: Implementation and Testing (SCQF level 8)

Unit code: HA4G 35

Superclass: CB

Publication date: January 2016

Source: Scottish Qualifications Authority

Version: 01

Unit purpose

The purpose of this Unit is to introduce learners to the use of object-oriented programming techniques, to extend their skills in using algorithms and data structures in program development, and apply a wider range of testing techniques.

Object-oriented programming techniques will include the use of objects and classes and algorithms, encapsulation, inheritance and polymorphism.

Data structures will cover lists, queues, stacks tables and trees, and algorithms will include sorting and searching. Testing will cover Static Testing (verification), Dynamic Testing (validation), Unit Testing, Integration Testing and User Acceptance Testing.

Learners will develop their programming skills by designing, implementing and testing practical solutions using an appropriate software development environment.

On completion of the Unit, learners will have gained knowledge and experience of implementing and testing moderately-complex programs using an object-oriented programming language.

Learners who have completed this Unit could progress to the Unit *Software Development: Implementation and Testing* (SCQF level 9). Learners who have completed both this Unit and *Software Development: Analysis and Design* (SCQF level 8) could progress to the Unit *Software Development: Project* (SCQF level 8).

Higher National Unit specification: General information (cont)

Unit title: Software Development: Implementation and Testing (SCQF level 8)

Outcomes

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

1 Describe programming and testing methods.
2 Apply object-oriented programming concepts.
3 Construct programs that make use of algorithms and data structures.
4 Test programs using a range of approaches.

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 beneficial if learners had prior knowledge/skills in computer programming.

It would be beneficial if learners had some prior experience of the basic concepts of software development, and analysis and design tools that could be evidenced by having achieved the Higher National Unit HA4F 34 *Software Development: Implementation and Testing* (SCQF level 7) 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)**.

Higher National Unit specification: General information (cont)

Unit title: Software Development: Implementation and Testing (SCQF level 8)

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: Implementation and Testing (SCQF level 8)

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 programming and testing methods.

Knowledge and/or Skills

- Describe structured programming constructs
- Describe simple data types, data structures and algorithms
- Describe basic software testing methods
- Describe contemporary programming paradigms

Outcome 2

Apply object-oriented programming concepts.

Knowledge and/or Skills

- Write programs constructed from objects and classes
- Hide internal workings of objects by encapsulation
- Create new classes by inheriting properties and methods from existing classes
- Create a single interface to entities of different types by means of polymorphism

Outcome 3

Construct programs that make use of algorithms and data structures.

Knowledge and/or Skills

- Select or construct algorithms to traverse, sort and search data structures
- Create data structures
- Carry out operations on data structures

Higher National Unit specification: Statement of standards (cont)

Unit title: Software Development: Implementation and Testing (SCQF level 8)

Outcome 4

Test programs using a range of approaches.

Knowledge and/or Skills

- Carry out static testing (verification)
- Carry out dynamic testing (validation)
- Carry out Unit testing
- Carry out integration testing
- Check that software meets specified requirements prior to User Acceptance Testing

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 evidence of cognitive competence (for Outcomes 1, 2, 3 and 4).
2 evidence of practical competence (for Outcomes 2, 3 and 4).

Please 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 needs to be evidenced.

The evidence of cognitive competence will be the definitions, descriptions and explanations required for Outcomes 1, 2, 3 and 4. The evidence of practical competence will be the application of object-oriented programming techniques, algorithms, data structures and testing approaches to specific problems required for Outcomes 2, 3 and 4.

The evidence of practical competence (Outcomes 2, 3 and 4) may relate to **one or more** problems. Candidates should make use of object-oriented programming techniques, algorithms, data structures and testing approaches. The evidence would consist of program code, evidence of successful execution and test documentation. 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 must 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 2, it is sufficient for the candidate to develop **one** program using object — oriented techniques. For Outcome 3, it is sufficient to develop **one** program that makes use of data structures and algorithms. For Outcome 4 it is sufficient to use a range of testing approaches with **one** program. The same program could be used for all three Outcomes or a different program could be used for each Outcome.

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. Candidates may use reference materials when undertaking assessment (but see exception below).

Sampling is permissible when the evidence of cognitive competence 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: Implementation and Testing (SCQF level 8)

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

Useful information about many of the topics covered in this Unit can be found by searching online sources such as Wikipedia.

Outcome 1

Learners should be able to describe the purpose of a range of programming constructs (expressions, sequence, selection, iteration, pre-defined functions and file handling), data types (string, numeric (integer and real) and Boolean variables), data structures (1-D arrays and records (including arrays of records) and sequential files (open, create, read, write, close)). and algorithms (input validation, linear search, find minimum and maximum and count occurrences).

They should be able to describe software testing methodologies including testing methods (static and dynamic, white and black box) testing levels (Unit, integration, component testing), test plan (exceptional, extreme and normal data), debugging techniques (dry runs, walkthroughs, breakpoints, trace tables) and types of errors (syntax, execution, logic).

Learners should know how to develop programs by selecting and using a combination of appropriate constructs, data types, data structures and algorithms. They should also know how to use subprograms and parameter passing.

Learners should also know how to test digital solutions including checking the use of meaningful identifiers and indentation, providing internal commentary, Unit and integration, testing, using own test data and test plan (exceptional, extreme and normal data).

Learners should know that a **programming paradigm** is a fundamental style of computer programming. Major paradigms include **procedural, event-driven** and **object-oriented**. The capabilities of programming languages are defined by their supported programming paradigms. Some programming languages are designed to follow only one paradigm, while others support multiple paradigms.

Higher National Unit Support Notes (cont)

Unit title: Software Development: Implementation and Testing (SCQF level 8)

Outcome 2

Learners should know that object-oriented programming (OOP) is based on the concept of 'objects', which are data structures that contain data, in the form of fields (often known as attributes) and code, in the form of procedures (often known as methods). An object's procedures can access and often modify the data fields of the object with which they are associated. Object-oriented programs are designed by constructing them from objects that interact with one another. Most popular object-oriented languages are class-based, meaning that objects are instances of classes, which typically also determines their type. Major object-oriented languages include Python, C++, Delphi, Java and Ruby.

If a class prevents calling code from accessing internal object data and forces access through methods only, this is known as **encapsulation**. Some languages (eg Java) let classes enforce access restrictions explicitly, buy using the *private* and *public* keywords. In other languages (eg Python) this is enforced only by convention (for example, naming 'private' methods starting with an underscore). Encapsulation prevents external code from being concerned with the internal workings of an object and encourages programmers to put all the code that handles a certain set of data in the same class.

Languages that support classes nearly always support **inheritance**. This allows classes to be arranged in a hierarchy that represents 'is-a-type-of' relationships. For example, class Employee might inherit from class Person. All the data and methods available to the parent class also appear in the child class with the same names. This technique allows easy re-use of the same procedures and data definitions and mirrors real-world relationships in an intuitive way. Classes and subclasses are similar to sets and subsets in mathematical logic.

In object-oriented programming languages, **polymorphism** is the provision of a single interface to entities of different types. A polymorphic type is one whose operations can also be applied to values of some other type, or types. There are several different kinds of polymorphism:

Ad hoc polymorphism: when a function denotes different implementations depending on a limited range of individually specified types and combinations. Ad hoc polymorphism is supported in many languages using function overloading.

Parametric polymorphism: when code is written without mention of any specific type and thus can be used transparently with any number of new types. This is often known as generics or generic programming.

Subtyping (also called subtype polymorphism or inclusion polymorphism): when a name denotes instances of many different classes related by some common superclass. This is often simply referred to as polymorphism.

Higher National Unit Support Notes (cont)

Unit title: Software Development: Implementation and Testing (SCQF level 8)

Outcome 3

Learners should know the range of **data structures** including lists, queues, stacks, tables and trees, commonly used within software development environments and be aware of both static (array) and dynamic (pointer) implementations. They should be introduced to both linear and circular data structures, single and double linked lists, binary trees hash tables.

Operations on data structures should include the creation and deletion of nodes, traversal, sorting, searching and merging.

Learners should be aware of the importance of **algorithm complexity** and be familiar with **big O notation.** They should know that linear O(n) algorithms are preferable to exponential O (n^2) algorithms when operating on large data sets.

You should typically aim for linear O(n) algorithms rather than, say exponential O (n^2). **Searching algorithms** should cover linear, binary and hash table searching techniques to illustrate how the efficiency of searches can be dependent on the underlying data structure.

Sorting algorithms should cover an appropriate range (eg contrast bubble and selection sorts with the merge and quick sorts) to illustrate how the time efficiency can be improved at the expense of increased space complexity.

Outcome 4

Learners should be able to use the following approaches to testing:

Static Testing (verification): testing software manually, possibly using a set of tools. This starts early in the life cycle during the verification process. It does not need a computer as the testing is done without executing the program, eg reviewing, walk through, inspection, etc.

Dynamic Testing (validation): a method of assessing the feasibility of a software program by giving input and examining output (I/O). This requires that the code be compiled and run. Types of dynamic testing include Unit testing, integration testing and acceptance testing.

Unit Testing: a process in which the smallest testable parts of an application, called Units, are individually and independently scrutinised for proper operation. Unit testing is often automated but it can also be done manually.

Integration Testing: tests integration or interfaces between components, interactions to different parts of the system such as an operating system, file system and hardware or interfaces between systems. It is carried out after integrating different components together.

User Acceptance Testing (UAT): is the last phase of the software testing process. During this phase, actual software users test the software to make sure it can handle required tasks in real-world scenarios, according to specifications. UAT is also known as beta testing, application testing or end user testing. While software developers would not normally carry out UAT they should check that the software meets the requirements of the specification prior to handover.

Higher National Unit Support Notes (cont)

Unit title: Software Development: Implementation and Testing (SCQF level 8)

Guidance on approaches to delivery of this Unit

This Unit is a component of the PDA Software Development (SCQF) level 8. It should be delivered after, or in parallel with the Unit *Software Development: Analysis and Design* (SCQF level 8). Both of the Units should be completed before delivery of the Unit *Software Development: Project* (level 8).

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 implementation and testing. 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 of the knowledge and understanding (Outcomes 1, 2, 3 and 4) and a practical assessment of practical abilities (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 comprise a number of multiple-choice questions (MCQs) or multiple-response questions (MRQs), and would be marked and assessed traditionally. For example, the test could comprise 40 multiple-choice questions, each of which could have four options (A–D), distributed equally across all the Outcomes, with an appropriate pass mark. 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 8), the test would comprise questions spanning a range of cognitive skills (including higher level ones involving analysis and synthesis).

Practical assessment (Outcomes 2, 3 and 4) could involve the practical application of the skills taught in each Outcome. A single exercise could be used to cover all three Outcomes or a separate exercise may be used for each Outcome. The exercise(s) 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 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 implementation and testing techniques could be applied to one or more 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 provides opportunities for developing Computational Thinking skills.

History of changes to Unit

Version	Description of change	Date

General information for learners

Unit title: Software Development: Implementation and Testing (SCQF level 8)

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 the use of object-oriented programming techniques, to extend your skills in using algorithms and data structures in program development and let you apply a wider range of testing techniques.

Object-oriented programming techniques will include the use of objects and classes and algorithms, encapsulation, inheritance and polymorphism.

Data structures will cover lists, queues, stacks tables and trees, and algorithms will include sorting and searching. Testing will cover Static Testing (verification), Dynamic Testing (validation), Unit Testing, Integration Testing and User Acceptance Testing.

You will develop your programming skills by designing, implementing and testing practical solutions using an appropriate software development environment.

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 implementation and testing skills learned.

On completion of the Unit, you will have gained knowledge and experience of implementing and testing moderately-complex programs written in an object-oriented programming language. You could progress to the Unit *Software Development: Implementation and Testing* (SCQF level 9). If you have completed both this Unit and *Software Development: Analysis and Design* (SCQF level 8), you could progress to *Software Development: Project* (SCQF level 8).