Higher National Unit specification

General information

Unit title: Software Development: Implementation and Testing (SCQF level 7)

Unit code: HA4F 34

Superclass: CB

Publication date: January 2016

Source: Scottish Qualifications Authority

Version: 02

Unit purpose

The purpose of this Unit is to introduce learners to the principles and practices of computer programming.

The principles of programming include the stages in writing a computer program, the basic constructs of programming, algorithms, data types, data structures and testing techniques. Learners will develop their programming and computational thinking skills by implementing and testing practical solutions using an appropriate software development environment.

The Unit is designed for learners with no previous experience of computer programming. It is suitable for a wide range of learners with a vocational or professional interest in developing programming skills.

On completion of the Unit, learners will have gained knowledge and experience of implementing and testing small-scale programs using a contemporary programming language. Further studies could involve the construction of larger, more complex software products.

Learners who have completed this Unit could progress to the Unit *Software Development: Implementation and Testing* (SCQF level 8). Learners who have completed both this Unit and *Software Development: Analysis and Design* (SCQF level 7) could progress to the Unit *Software Development: Project* (SCQF level 7).

Higher National Unit specification: General information (cont)

Unit title: Software Development: Implementation and Testing (SCQF level 7)

Outcomes

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

1 Describe a range of programming constructs and data structures.
2 Describe software testing methods.
3 Develop programs using a combination of appropriate constructs and data structures.
4 Test programs using a range of methods.

Credit points and level

2 Higher National Unit credits at SCQF level 7: (16 SCQF credit points at SCQF level 7)

Recommended entry to the Unit

No previous knowledge of computer programming is required, but learners would benefit from having previous experience of using computers.

Access to this Unit is at the discretion of the centre, however it is desirable that the learner possesses good communication and problem solving skills as well as the ability to manipulate text and graphical information, gained through either workplace experience or training at an appropriate level. Knowledge and understanding of Information Systems is also desirable.

It would also 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 National Unit DF2Y 11 *Software Development* at SCQF level 5 or equivalent.

Core Skills

Achievement of this Unit gives automatic certification of the following:

Complete Core Skill Problem Solving at SCQF level 6

Core Skill component None

Opportunities to develop aspects of Core Skills are highlighted in the Support Notes for this Unit specification.

Higher National Unit specification: General information (cont)

Unit title: Software Development: Implementation and Testing (SCQF level 7)

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: Implementation and Testing (SCQF level 7)

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 a range of programming constructs and data structures.

Knowledge and/or Skills

- Describe programming constructs, including expressions, sequence, selection, iteration, pre-defined functions and file handling
- Describe data types, including string, numeric (integer and real) and Boolean variables
- Describe data structures, including 1-D arrays and records (including arrays of records) and sequential files (open, create, read, write and close)
- Describe algorithms, including input validation, simple sorts, linear search, find minimum and maximum and count occurrences

Outcome 2

Describe software testing methods.

Knowledge and/or Skills

- Describe types of testing, including static and dynamic testing, white and black box testing
- Describe testing levels, including Unit, integration and system testing
- Describe the contents of a test plan, including exceptional, extreme and normal data
- Describe debugging techniques, including dry runs, walkthroughs, breakpoints and trace tables
- Describe different types of errors, including syntax, execution and logic

Higher National Unit specification: Statement of standards (cont)

Unit title: Software Development: Implementation and Testing (SCQF level 7)

Outcome 3

Develop programs using a combination of appropriate constructs and data structures.

Knowledge and/or Skills

- Construct software from subprograms
- Pass parameters to subprograms
- Construct programs using a range of constructs, including expressions, sequence, selection, iteration, pre-defined functions and file handling
- Select suitable data types, including string, numeric (integer and real) and Boolean variables
- Choose appropriate data structures, including 1-D arrays and records (including arrays of records) and sequential files (open, create, read, write and close)
- Select relevant algorithms, including input validation, linear search, find minimum and maximum and count occurrences

Outcome 4

Test programs using a range of methods.

- Carry out Unit, integration and system testing
- Ensure that identifiers are meaningful
- Indent code to show structure
- Incorporate internal commentary
- Produce a test plan (exceptional, extreme and normal data)
- Prepare own test data
- Ensure that all input data is securely validated
- Debug using a range of techniques including dry runs, walkthroughs, breakpoints and trace tables

Higher National Unit specification: Statement of standards (cont)

Unit title: Software Development: Implementation and Testing (SCQF level 7)

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

The evidence of cognitive competence required for Outcomes 1 and 2 will be the definitions, descriptions and explanations of the list of Knowledge and/or Skills. The evidence of practical competence required for Outcomes 3 and 4 will be the use of programming constructs and structures and testing methodologies applied to specific problems. The evidence for Outcomes 3 and 4 would consist of program code, evidence of successful execution and test documentation.

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 3, the candidate is required to develop as many programs that it takes to demonstrate a practical understanding of all the Knowledge and/or Skills. At this level, this may be achieved by the candidate producing a number of short programs, rather than one complex program. However, this is at the discretion of the candidate. The same programs could be used for Outcome 4 or different programs could be used to carry out the testing methodologies.

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 and 2 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 and 2 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 specification: Statement of standards (cont)

Unit title: Software Development: Implementation and Testing (SCQF level 7)

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.

The suggested time distribution is as follows:

Outcome 1: 20 hours
Outcome 2: 15 hours
Outcome 3: 30 hours
Outcome 4: 15 hours

Guidance on the content and context for this Unit

Outcome 1

This **Outcome** requires learners to describe a range of programming constructs and data structures. The knowledge of programming constructs required by learners is as follows:

- expressions to assign values to variables, return values using arithmetic operations (+, -, *, /, ^, mod) and concatenate strings and arrays using the and operator
- sequence where each event, leads to the next ordered event in a predetermined order by creating modular programs using procedures with parameter passing (value and reference, formal and actual)
- selection including simple and complex conditional statements and logical operators
- iteration including conditional and fixed iteration
- pre-defined functions, with parameters, including mathematical and string functions, eg **INT**, **LEN, RND,** UCase, MID
- file handling to create files, read and write data to and from existing files including inserting, amending and deleting data

Learners should be able to describe data structures including 1-D arrays and records (including arrays of records) and sequential files (open, create, read, write and close). They should also be able to describe simple algorithms including input validation, linear search, find minimum and maximum and count occurrences.

Learners will develop their own algorithms when analysing and designing programs. However, they should also be familiar with a range of standard algorithms including:

- input validation
- linear search
- finding minimum and maximum
- counting occurrences

To allow learners to gain the knowledge of these algorithms, teachers and lecturers could provide the learners with the algorithms in various different contexts and the learners could investigate how they work.

Once the learners have a sound understanding of these standard algorithms, and know and understand how they function within a program, they should develop their own modular programs that make use of these standard algorithms.

Outcome 2

Learners should be able to describe software testing methods. There is a large variety of ways to test software but for the purposes of this Outcome learners require the following knowledge of testing methods:

- testing methods including static and dynamic testing, white and black box testing
- testing levels including Unit, integration and system testing
- test plan including exceptional, extreme and normal data:
  - normal data would fall within the range of data a program is expected to process
  - extreme data tests the boundaries of the range of data a program is expected to process
  - exceptional data is outside the range of data a program is expected to process
- debugging techniques including dry runs, walkthroughs, breakpoints, trace tables
- types of errors including syntax, execution, logic

Learners should know that live data is not suitable for testing as it consists mostly of correct data and is unlikely to allow the program to be tested with sufficient rigour.

For the purposes of this Unit, **Unit testing** refers to the testing of individual components of the program (subprograms or functions) in isolation. **Integration testing** refers to the process of ensuring that the components function correctly together, with particular emphasis on the correct operation of interfaces. **System testing** refers to ensuring that the application as a whole performs as expected.

Syntax errors occur where the compiler finds something wrong with the program, and cannot even try to execute it. For example, there may be incorrect punctuation, or may be trying to use a variable that hasn't been declared. Syntax errors are the easiest to find and correct since the compiler will indicate where the error happened, and its best guess as to what and where it went wrong.

An execution error (sometimes called run time error) occurs when a program is asked to do something that it cannot do, resulting in a ‘crash’. The code contains no syntax or logic errors but when it runs it can't perform the task that it has been programmed to carry out. A common example of a run time error is asking a program to divide by 0. Another example is where an attempt is made to access an item in an array which does not exist, eg item 9 in an array size 8.

A logic error is when a program compiles and runs, but does the wrong thing. The programming environment does not know the program is supposed to do, so it does not provide any additional information to help find the error. To help find a logic error a debugging technique can be used to step through the program and watch what it is doing.

Static testing and **dynamic testing** have been described as the two most fundamental tools for software developing and each plays a different but equally important role in discovering coding errors within a program.

Static testing is often referred to as **verification** and is used to analyse software in a non-runtime environment — when the software is inactive and not in operation. Relevant debugging techniques include **dry runs** and **walkthroughs.**

Dynamic testing is often referred to as **validation** and is performed in a runtime environment while the software is in operation. A **test plan** can be used here where **exceptional, extreme** and **normal data** is identified and tested. With a given input, the software's actual output is compared to its expected output. Relevant debugging techniques include **breakpoints** and **trace tables.**

Both these methods of testing have advantages and disadvantages, eg bugs found during the development process are relatively cheap and easy to fix, making static testing very cost-effective. The costs of dynamic testing can be relatively high but the costs of leaving a bug or unfixed can be much higher. Whilst individual circumstances may benefit from one type of test over another, ensuring software is bug free requires both types of testing.

Black-box testing examines the functionality of a program. Specific knowledge of the program's code and internal structure is not required. The tester is aware of what the software is supposed to do but is not aware of how it does it. This method of test can be applied to Unit, integration and system testing.

White-box testing tests internal structures or workings of a program, as opposed to its functionality. This involves testing the application at the level of the source code and requires knowledge of which line of the code is being executed and the ability to identify what the correct output should be. Although traditional testers tended to think of white-box testing as being done at the Unit level, it is also used for integration and system testing. It can test paths within a Unit, paths between Units and subprograms during integration and system testing. Though this method of testing can uncover many errors or problems, it has the potential to miss unimplemented parts of the specification or missing requirements.

Outcomes 3 and 4

These Outcomes require learners to put into practice the knowledge they gained in Outcomes 1 and 2. Learners will develop programs using a combination of appropriate constructs and data structures and then test their programs using a range of methods. Learners should be aware that it is often necessary to modify existing programs in addition to writing new ones. This can happen due to changes in requirements or to correct newly-discovered bugs.

Higher National Unit specification: Statement of standards (cont)

Unit title: Software Development: Implementation and Testing (SCQF level 7)

Learners should demonstrate all of the following skills while developing programs:

- Use parameter passing
- Use subprograms
- Use programming constructs (including expressions, sequence, selection, iteration, pre-defined functions and file handling)
- Use data types (including string, numeric (integer and real) and Boolean variables)
- Use data structures (including 1-D arrays and records (including arrays of records), sequential files (open, create, read, write and close)
- Use algorithms (including input validation, linear search, find minimum and maximum and count occurrences)

Learners should systematically test each module or sub-program once it has been completed before moving on to the next one and then construct a test plan for the completed solution. Learners should demonstrate all of the following skills while carrying out comprehensive testing of their programs

- Carry out Unit, integration and system testing.
- Use meaningful identifiers (variable names) — learners should ensure that variable, procedure and function names are meaningful and describe what the variable, procedure or function contains, eg a function name should describe what the function does — 'functionX' is not a great function name, 'enterscore' is much better.
- Use indentation — Indentation means moving parts of the code that form the content of constructs such as loops or if statements, to the right so it's easier to see the overall structure of the code. Using indentation makes code much easier to read.
- Use internal commentary — internal commentary describes what is going on in the learner's program. It is good practise to write details of what is happening at regular intervals in code. Having internal commentary makes it easier to see what is going on in a program. Commentary is not required for simple or self-explanatory code.
- Create a test plan (exceptional, extreme and normal data)
- Use own test data
- Use debugging techniques including dry runs, walkthroughs, breakpoints, trace tables
- Ensure that all input data is thoroughly validated and rejected if unsatisfactory

Computational thinking

Computational thinking is recognised as a key skill set for all 21st century learners — whether they intend to continue with computing science or not. It involves a set of problem-solving skills and techniques used by software developers to write programs.

There are various ways of defining computational thinking. One useful structure is to group these problem-solving skills and techniques under five broad headings:

- **Abstraction:** seeing a problem and its solution at many levels of detail and generalising the information that is necessary. Abstraction allows us to represent an idea or a process in general terms (eg variables) so that we can use it to solve other problems that are similar in nature.
- **Algorithms:** the ability to develop a step-by-step strategy for solving a problem. Algorithm design is often based on the decomposition of a problem and the identification of patterns that help to solve the problem. In Computing Science as well as in mathematics, algorithms are often written abstractly, utilising variables in place of specific numbers.
- **Decomposition:** breaking down a task so that we can clearly explain a process to another person — or to a computer. Decomposing a problem frequently leads to pattern recognition and generalisation/abstraction, and thus the ability to design an algorithm.
- **Pattern recognition:** the ability to notice similarities or common differences that will help us make predictions or lead us to shortcuts. Pattern recognition is frequently the basis for solving problems and designing algorithms.
- **Generalisation:** realising that a solution to one problem may be used to solve a whole range of related problems.

Underpinning all of these concepts is the idea that computers are deterministic: they do exactly what you tell them to do. The corollary of this, of course, is that they can be understood. Whilst computational thinking can be a component of many subjects, Software Development is particularly well placed to deliver it. Teachers are encouraged to emphasise, exemplify and make these aspects of computational thinking explicit (at an appropriate level) wherever there are opportunities to do so throughout the teaching and learning of this Course and its Units.

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 before, or in parallel with the Unit *Software Development: Implementation and Testing* (SCQF level 8). Both of the Units should be completed before delivery of the Unit *Software Development: Project* (level 8).

Sequence of delivery of Outcomes

The sequence of delivery of the Outcomes is a matter of professional judgement and is entirely at the discretion of the centre. Some suggested approaches are outlined below.

Outcome 1 and Outcome 3 simultaneously

Teachers may wish to combine practical work with theory using a series of practical tasks that focus on the implementation of one or more programming constructs. Implementation could be accompanied by a written or oral description of how a construct is being used and how it works. For example, when learners are selecting and using appropriate constructs when developing their modular programs, they will need to understand the purpose and function of these constructs. As learners develop their programming skills by selecting and using appropriate constructs, they will also develop a clear understanding of how these constructs work and what they can be used for. This will enable them to read, interpret and describe the code in their programs.

Outcome 1 before Outcome 3

In order to deepen the learners' understanding of advanced concepts and to enhance their ability to describe how programs work it would constitute good practice to discuss the purpose of the range of constructs, data types and algorithms used in this Unit and for learners to practice reading and interpreting code before progressing to a series of practical examples and tasks.

For example, learners could be provided with a range of working programs illustrating particular programming constructs and data types. They could then be asked to describe what is happening in certain sections of code. They would gain familiarity with a range of programming constructs and data types and be able to describe their purpose within the program.

A similar exercise could take place using a range of standard algorithms. Teachers and lecturers could provide the learners with these algorithms and the learners would need to be able to describe how they work.

Once the learners have a sound understanding of the purpose of a range of programming constructs, data types and standard algorithms, and know how they function within a program, they should be well placed to develop their own modular programs that make use of these constructs and standard algorithms in their chosen software development environment. They should be able to apply their knowledge gained in Outcome 1 to select and use the appropriate programming constructs to produce a complete, working program.

Outcome 3 before Outcome 1

Alternatively, it may be preferable to first establish the necessary practical skills, to learn by doing, and meet the requirements of Outcome 3 before moving to formally develop the knowledge and understanding embedded in Outcome 1. For example, learners could develop their programming skills by learning how to program certain constructs using a range of simple data types and 1-D arrays. These activities would allow the learners to fully understand the purpose and function of certain constructs and data types within the programs. Consequently, when learners are then asked to read and describe sections of code within a program in Outcome 1, they would have the necessary knowledge and skills to do this from their experience in coding in Outcome 2.

Higher National Unit specification: Statement of standards (cont)

Unit title: Software Development: Implementation and Testing (SCQF level 7)

Delivery of Outcome 1

In order to meet Outcome 1, learners are expected to develop their knowledge and understanding to describe the purpose of a range of programming constructs and how they work, and are able to describe how a range of standard algorithms work. Teachers can provide some background information relating to aspects of the software to be investigated. Opportunities for learning and teaching activities might include the following.

- Learners could be provided with working programs in order to learn the purpose of a range of programming constructs and how they work within the program. Teachers would describe these constructs and demonstrate how they work inside a program. This would develop the knowledge and understanding required in the learner to be able to read and describe sections of code.
- Learners could be provided with or asked to locate working programs that demonstrate sub-programs, user-defined functions, parameter passing, sequential file operations and variables from any software development environment. They could then be asked to identify and describe sections of code from within these programs.
- A similar exercise could be carried out with standard algorithms, whereby learners are provided with algorithms that exemplify input validation, linear search, finding minimum and maximum values and counting occurrences. Teachers could then demonstrate how these algorithms work by describing each step in the sequence, which would give learners a sound understanding on how these algorithms actually work in practice.

Delivery of Outcomes 2 and 4

The knowledge required for Outcome 2 can be covered at several stages of teaching. For example, it would be possible to cover the knowledge for Outcome 3 alongside the knowledge for Outcome 1 or covered after Outcomes 1, 3 and 4 have been completed.

The skills required for Outcome 4 really need covered before learners start to code their own programs. For example it would seem essential that learners had knowledge of debugging techniques before starting to develop their own programs. As an example of good practice, learners should be encouraged to write code so that it can be read and understood by others and also by the learners themselves if they return to a program after several weeks. To help with this, learners should use meaningful identifiers (variable names), indentation and internal commentary in **all** programs.

In Outcome 4 learners do not need to explicitly demonstrate skills in all of the knowledge they have described in Outcome 2. The knowledge of testing methods including static and dynamic testing, white and black box testing will be tested within the other skills. The knowledge of static and dynamic testing, white and black box testing is an area that would be suitable for group work where learners investigate, evaluate and present their ideas to one another.

Higher National Unit specification: Statement of standards (cont)

Unit title: Software Development: Implementation and Testing (SCQF level 7)

An appropriate balance of teaching methodologies should be used in the delivery of this Unit and a variety of active learning approaches is encouraged, including the following:

- Activity-based learning

  Whole-class, direct teaching opportunities should be balanced by activity-based learning on practical tasks. An investigatory approach is encouraged, with learners actively involved in developing their skills, knowledge and understanding by investigating a range of real-life and relevant programming problems and solutions related to areas of study. Learning should be supported by appropriate practical activities, so that skills are developed simultaneously with knowledge and understanding.

- Group work

  Practical activities lend themselves to group work, and this should be encouraged. Learners engaged in collaborative group working strategies can capitalise on one another's knowledge, resources and skills by questioning, investigating, evaluating and presenting ideas to one another. While 'working as a team' is not specifically identified as one of the skills for learning, life and work for this Course and, therefore, not assessed, it is a fundamental aspect of working in the IT and related industries and so should be encouraged and developed by teachers.

- Problem-based learning

  Problem-based learning (PBL) is another strategy which will support a learner's progress through this Unit. This method may be best utilised at the end of an Outcome or a topic where additional challenge is required to ensure learners are secure in their knowledge and understanding and to develop the ability to apply knowledge and skills in less familiar contexts. Learning through PBL develops a learner's problem-solving, decision-making, investigative skills, creative thinking, team working and evaluative skills.

- Computational thinking Computational thinking is recognised as a key skill set for all 21st century learners — whether they intend to continue with computing science or not. It involves a set of problem-solving skills and techniques used by software developers to write programs.

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 and 2) and a practical assessment of practical abilities (Outcomes 3 and 4).

Higher National Unit specification: Statement of standards (cont)

Unit title: Software Development: Implementation and Testing (SCQF level 7)

The end of Unit test could sample from the knowledge and understanding contained in Outcomes 1 and 2. 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 both 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 7), the test would comprise questions spanning a range of cognitive skills (including higher level ones involving analysis and synthesis).

Practical assessment (Outcomes 3 and 4) could involve the practical application of the skills taught in each Outcome. A single assignment could be used to cover both Outcomes or a separate assignment may be used for each. The assignment(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 periodic 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 3 and 4 (which are largely practical). This could be done by the blog demonstrating how analysis and design 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.

Higher National Unit specification: Statement of standards (cont)

Unit title: Software Development: Implementation and Testing (SCQF level 7)

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

The Unit does provide opportunities for the development of Computational Thinking skills.

This Unit has the Core Skill of Problem Solving embedded in it, so when learners achieve this Unit their Core Skills profile will be updated to show that they have achieved Problem Solving at SCQF level 6.

History of changes to Unit

Version	Description of change	Date
02	Core Skills Component Problem Solving at SCQF level 6 embedded.	20/05/16

General information for learners

Unit title: Software Development: Implementation and Testing (SCQF level 7)

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 Unit is designed for learners with no previous experience of computer programming. It is suitable for a wide range of learners with a vocational or professional interest in developing programming skills.

The purpose of this Unit is to introduce the principles and practices of computer programming. The principles of programming include the stages in writing a computer program, the basic constructs of programming, algorithms, data types, data structures and testing techniques. You will develop your programming and computational thinking skills by 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 small-scale programs using a contemporary programming language. Further studies could include computer games programming or other commercial opportunities in programming. You could progress to the Unit *Software Development: Implementation and Testing* (SCQF level 8). Along with the Unit *Software Development: Analysis and Design* (SCQF level 7), you could progress to the Unit *Software Development: Project* (level 7).

This Unit has the Core Skill of Problem Solving embedded in it, so when you achieve this Unit your Core Skills profile will be updated to show that you have achieved Problem Solving at SCQF level 6.