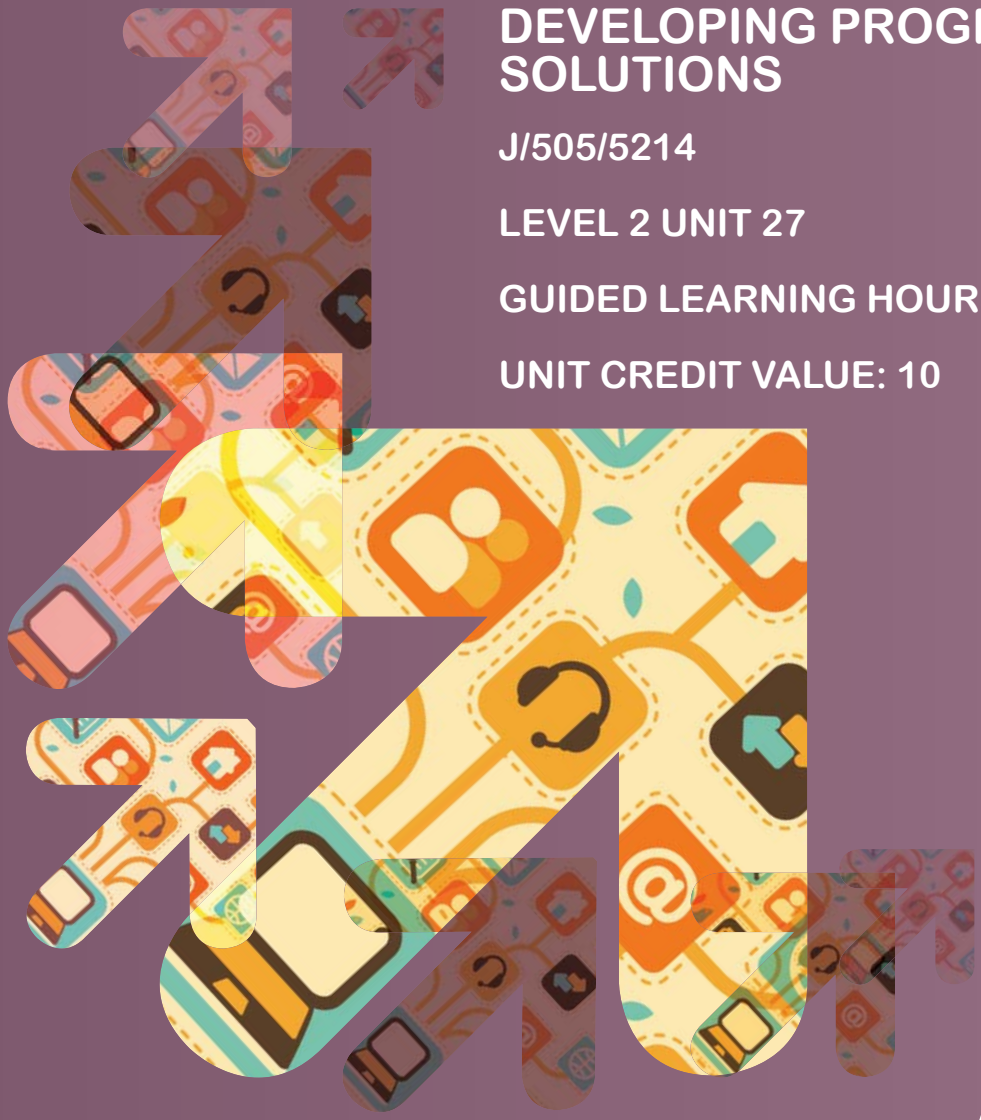
Cambridge TECHNICALS

Accredited

OCR LEVEL 2 CAMBRIDGE TECHNICAL

CERTIFICATE/DIPLOMA IN

IT



DEVELOPING PROGRAMMING SOLUTIONS

J/505/5214

LEVEL 2 UNIT 27

GUIDED LEARNING HOURS: 60

UNIT CREDIT VALUE: 10

OCR

DEVELOPING PROGRAMMING SOLUTIONS

J/505/5214

LEVEL 2

AIM AND PURPOSE OF THE UNIT

This unit will enable learners to create programming solutions in an appropriate development environment, including testing and debugging the code.

On completing this unit learners will know about language options together with the syntax, structure and layout of a programming language. Learners will create a programming solution using an appropriate development environment and complete the processes of testing and debugging the code that has been produced.

ASSESSMENT AND GRADING CRITERIA

Learning Outcome (LO) The learner will:	Pass The assessment criteria are the pass requirements for this unit. The learner can:	Merit To achieve a merit the evidence must show that, in addition to the pass criteria, the learner is able to:	Distinction To achieve a distinction the evidence must show that, in addition to the pass and merit criteria, the learner is able to:
1 Understand appropriate programming languages for a proposed solution	P1 explain the choice of programming languages for a proposed solution	M1 explain how the chosen programming language will be suitable for future expansion or enhancement	
2 Be able to produce programming code for a software solution	P2 outline the structure to be used for programming code to meet a proposed solution.		
	P3 produce programming code for a given specification using appropriate syntax	M2 format the code using appropriate layout and indentation	D1 annotate the code with appropriate comments
3 Be able to debug programming code for a software solution	P4 use debugging techniques to find and correct errors in programming code		D2 enhance the code to improve efficiency and quality of solution
4 Be able to test a software solution	P5 test the functionality of the software solution	M3 implement a change to the software solution based on identified refinements	

TEACHING CONTENT

The unit content describes what has to be taught to ensure that learners are able to access the highest grade.

Anything which follows an i.e. details what must be taught as part of that area of content.

Anything which follows an e.g. is illustrative, it should be noted that where e.g. is used, learners must know and be able to apply relevant examples to their work though these do not need to be the same ones specified in the unit content.

LO1 Understand appropriate programming languages for a proposed solution

- Uses of software and firmware e.g. high level coding solutions, interface between software and firmware, firmware development
- Types of programming language i.e. object oriented, procedural, event driven
- Programming languages
  - High Level (e.g. VB, C++, JavaScript, HTML, XML, CSS, asp.net, PHP, Python, Perl, Ruby)
  - Low level i.e. assembler (dedicated to specific 8/16/32 bit microprocessors and microcontrollers) and key elements e.g. the processes of binary calculations, stack management and interrupt handlers
- Selection of a programming language suitable for a specific coding solution e.g. consideration of target platform, intended use, compatibility, security needs, user needs, maintenance and support

LO2 Be able to produce programming code for a software solution

- Interpreting programming requirements from a software specification e.g. development language and platform, data flow and management, Graphical User Interface (GUI), drivers, libraries, interfaces, algorithms, structure
- Software design principles e.g. identifying components and sub components of the overall system
- Structure of programming code and terminology (e.g. main body, sub routines, kernel, libraries, device drivers, definitions, variables)
  - Indentation and layout of code (e.g. indentation level, use of white space, blank lines)
  - Use of comments to explain processes
- Syntax of programming languages (e.g. uppercase and lowercase characters, naming conventions, file naming and extensions)
- Development environment, folder structure and file naming e.g. single user and multi user environments, what files and folders are used and how they are managed, what file naming conventions are used for source code and executables
- The use of version control when working on software development e.g. numerical systems for successive versions, beta releases, date identifiers

LO3 Be able to debug programming code for a software solution

- Debugging tools and techniques
  - Using a debugger software application e.g. with execution control, disassembly, memory and registers windows
  - Using an emulator e.g. for microcontrollers, android devices
  - Using trace statements e.g. breakpoints, pause/stop/restart, single stepping of instructions
  - Use of monitoring techniques e.g. print instructions
  - Analysis and interpretation of error messages, log files, dump files and stack content
- Applying corrections, improvements and enhancements to optimise code efficiency e.g. different ways of achieving the same results using alternative structure, design approaches, programming routines

LO4 Be able to test a software solution

- Functionality testing i.e. to ensure the developed solution meets the specification and/or client requirements
- Using a test plan i.e. with details of what is being tested and the success criteria
- Recording results of testing i.e. pass/fail against the success criteria, use of appropriate comments
- Identification and implementation of improvements and changes e.g. code changes, re-testing and results
- Documenting and storing the test processes and results

DELIVERY GUIDANCE

This unit will develop the practical skills involved in developing programming solutions. Wherever possible the unit should be delivered effectively within a workshop environment, giving learners a series of exercises, tasks and practical activities. Learners may have completed other units and established some basic knowledge that contributes to this unit, such as the types and features of programming languages. This will need to be developed further so that learners can use their knowledge and combine this with their understanding of a coding requirement in order to make informed choices.

Understand appropriate programming languages for a proposed solution

To introduce this unit the tutor could use examples of IT equipment and explain what software and firmware is used on them. Devices such as printers, routers, smartphones, tablets, robots, industrial automation controllers, games consoles and PCs will provide a range of different platforms. This will establish an understanding of the difference between software and firmware together with examples where software applications interface with embedded firmware.

The tutor could provide information hand outs on a range of programming languages, identifying the different types. Their suitability for use on different platforms should be included with workshops and case studies for how different languages have been used to produce solutions. A series of different categories would be encouraged such as:

- Web technologies and solutions including HTML, XML, CSS, Javascript, asp.net, PHP and Python
- PC based IT systems solutions using VB, C++, Python
- Industrial control, microcontrollers and robotics using VB, C++, Python, PERL, Ruby and assembler languages.
- Applications for solutions that are based on platforms such as Raspberry Pi and Arduino
- Apps for tablets and smartphones

As the basis for further work in this and other units, it will be important to identify the features and capability of different languages to support the development of solutions. Where appropriate, identify examples of SDK (software development kits) for specific purposes.

The tutor should explain the purpose and use of a software requirements specification. Examples of simple and more complex specifications could be used as case studies. A group activity about a particular requirements specification will prompt discussion about what programming languages may be suitable. A quiz (whether short answer or multiple choice) could be used to reinforce the learning.

At this point, learners should be encouraged to research developments in programmes that have been made or are possible within individual languages.

Be able to produce programming code for a software solution

Following on from Learning Outcome 1, the concept and content of a software requirements specification may have been introduced. This should now be developed further to identify how a solution will be designed and implemented. Alongside this, the basic concepts of software design should be covered which are then put into an applied context for a given software specification. One approach would be to begin with simple and short tasks to perform calculations and display the results on the screen. This could then be extended by adding further options and improving the design and layout of the user interface. The use of platforms with audio capabilities will allow the generation of musical tunes and/or audio acknowledgement of instructions and input. At this early stage, the practical activities would be designed to develop knowledge and skills in the use of syntax, structure, indentation and commenting of code.

A series of exercises and projects should become progressively larger so that a wider range of commands and programming structure is used. Each mini project would be introduced with a software requirements specification to reinforce the process of software development. Reference materials in the form of sample code should be used that illustrates good layout. Annotated versions could highlight the use of indentation, white space, blank lines and author's comments. Again it is suggested that these examples cover a range of different programming languages.

Depending on what development environments are available and being used, conventions for managing files and folders should be explained within a workshop activity. Supporting hand outs could also be used to explain naming conventions, file extensions and version control.

Be able to debug programming code for a software solution

This Learning Outcome could be delivered alongside Learning Outcome 2 such that debugging techniques are used with any or all of the programming exercises. Depending on the platforms and languages used, the tools and techniques of debugging should be appropriate to the development environment. For example, if using a PC based visual environment, built in debuggers and tools should be explored in a practical context. Where emulators are available, these may also be used with break points and single step instructions to locate programming errors and interface problems. Debugging techniques may also involve the use of previews and test environments with trace statements and monitoring tools. Where error messages and log/dump files are created, these should be explored in detail so that their practical use is demonstrated. One suggestion here is to have a piece of programming code with known errors on a particular platform. The task would then be to debug the code provided by using a combination of the error messages, log files, memory and register windows as appropriate. A series of these examples could be used across different environments and platforms to develop the debugging skills required. In general, the delivery should be in a workshop with a range of practical activities that is also supported by hand outs, demonstrations and verbal explanation.

Areas for enhancements to the programming code may be identified during this stage of the development process and may be identified using the knowledge gained from their research in Learning Outcome 1. Examples would be simpler coding routines, improved structure or stability to achieve the same objectives, hence improving the efficiency. As above, one suggestion here is to provide an example of a coding solution that is deliberately inefficient. Learners would then need to debug the code and develop an improved solution based on the analysis of information gathered.

Be able to test a software solution

This Learning Outcome could be delivered in conjunction with Learning Outcomes 2 and 3. Functional testing should be completed against the requirements to ensure the software solution meets the customer's or client's needs. The requirements specification used earlier in this unit may form the basis of the testing. Initially and as a learning activity, the tutor may provide a test plan that is relevant to one of the basic software specifications.

This would identify a series of tests and the success criteria, such as in a table format. The tutor will need to provide a working software solution so that learners can test the functionality. Learners could complete the tests on the sample software solution and complete the test plan in their own words. Once learners have developed some skills in carrying out the required tests, they could then move on to generating their own test plan for a different software specification. The tutor should clearly differentiate the testing process as a separate activity which is completed on the finished solution, which is different to any iterative testing carried out during the software development process.

Testing of the software solution may be used with any or all of the programming exercises in Learning Outcome 2 or 3. The tutor could identify examples of changes to the software solution such as the layout and presentation of any output (whether on the display screen or printed). Improvements to the GUI could then be listed and implemented as an outcome. The overall delivery should continue in a workshop setting with a range of practical activities that is also supported by hand outs, demonstrations and verbal explanation.

Learners should be encouraged to store the test results for all their testing into a specific folder with suitable filenames for archive and reference purposes.

SUGGESTED ASSESSMENT SCENARIOS AND TASK PLUS GUIDANCE ON ASSESSING THE SUGGESTED TASKS

Assessment Criteria P1, M1

For P1, learners must explain the choice of programming languages for a proposed solution. The explanation should be produced in response to a clearly defined requirement and could be evidenced using a report or presentation.

For merit criterion M1, learners must explain how the chosen programming language will be suitable for future expansion or enhancement to the proposed solution. The capabilities of the chosen programming language to support these changes should also be described in order to confirm its suitability in the long term. This could be evidenced by a report or presentation and may be an extension to the evidence for the Pass criteria.

Assessment Criteria P2, P3, M2, D1

For P2, learners must outline the structure to be used for the programming code in response to a clearly defined requirement, The structure should be outlined and be appropriate to meeting the clearly defined requirements. Learners could evidence the criteria using a report or presentation.

For P3 learners must produce programming code for a given specification using appropriate syntax. This may be evidenced electronically, on screen or as a printed code listing.

For merit criterion M2, learners must format the code using an appropriate layout, including indentation. This is likely to be evidenced as an enhancement to P3.

For distinction criterion D1, learners must annotate the code with comments in appropriate places, which contain the necessary detail. The comments could include author, version and explanations of what the code is intended to do in a number of key places.

Assessment Criteria P4, D2

For P4, learners must use debugging techniques to find and correct errors in their code. This may be in the form of screen captures, printed listings, error reports, trace statements, 'before' and 'after' code listing that is linked to clearly identified errors, etc. during the debugging process. The type of debugging techniques used will determine which of these is the most appropriate form of evidence. Any errors found and the corrective actions implemented should be noted and documented as part of the process.

For distinction criterion D2, learners must identify areas for improvement in the code to enhance performance or improve quality of the programme and implement these. The evidence may be in the form of 'before' and 'after' code listings. If a section of code has been rewritten to improve efficiency and/or quality, then annotations to the code listings should be used to identify the changes made.

Assessment Criteria P5, M3

For P5, learners must test the functionality of the software solution using a formalised test plan that identifies what is to be tested, the success criteria and the results of testing. The tests should be carried out on the finished solution and make sure the client/software requirements are met. The test plan may be in a document or spreadsheet format and should be based on the requirements specification. Explanatory comments should be added as part of the results of testing.

For merit criterion M3, learners must implement a change to the software solution based on identified refinements. The refinements identified should be clearly stated together with the improvement process. Where appropriate, screen captures or printouts of improvements to the GUI or output should be included. Video evidence of the refinements may be used if appropriate, such as improvements to robotic solutions that involve movement rather than displays. The refinements should be supported by evidence of changes to the code.

SUGGESTED SCENARIOS

Tutors could provide learners with an outline specification for a software development project. This could be based around the audio or video capabilities of single board microcomputers such as the Raspberry Pi. Alternatively a simple arcade style computer game could be produced. Learners should apply basic design principles to create a solution, which is debugged and tested.

RESOURCES

Access to programming languages and software development tools

Suitable hardware and platforms for development

Suitable testing and debugging tools

MAPPING WITHIN THE QUALIFICATION TO THE OTHER UNITS

Unit 14: Computer systems

Unit 17: Customising software

Unit 22: Developing computer games

LINKS TO NOS

5.2 Software Development

CONTACT US

Staff at the OCR Customer Contact Centre are available to take your call between 8am and 5.30pm, Monday to Friday.
We're always delighted to answer questions and give advice.

Telephone 02476 851509

Email cambridgetechnicals@ocr.org.uk

www.ocr.org.uk