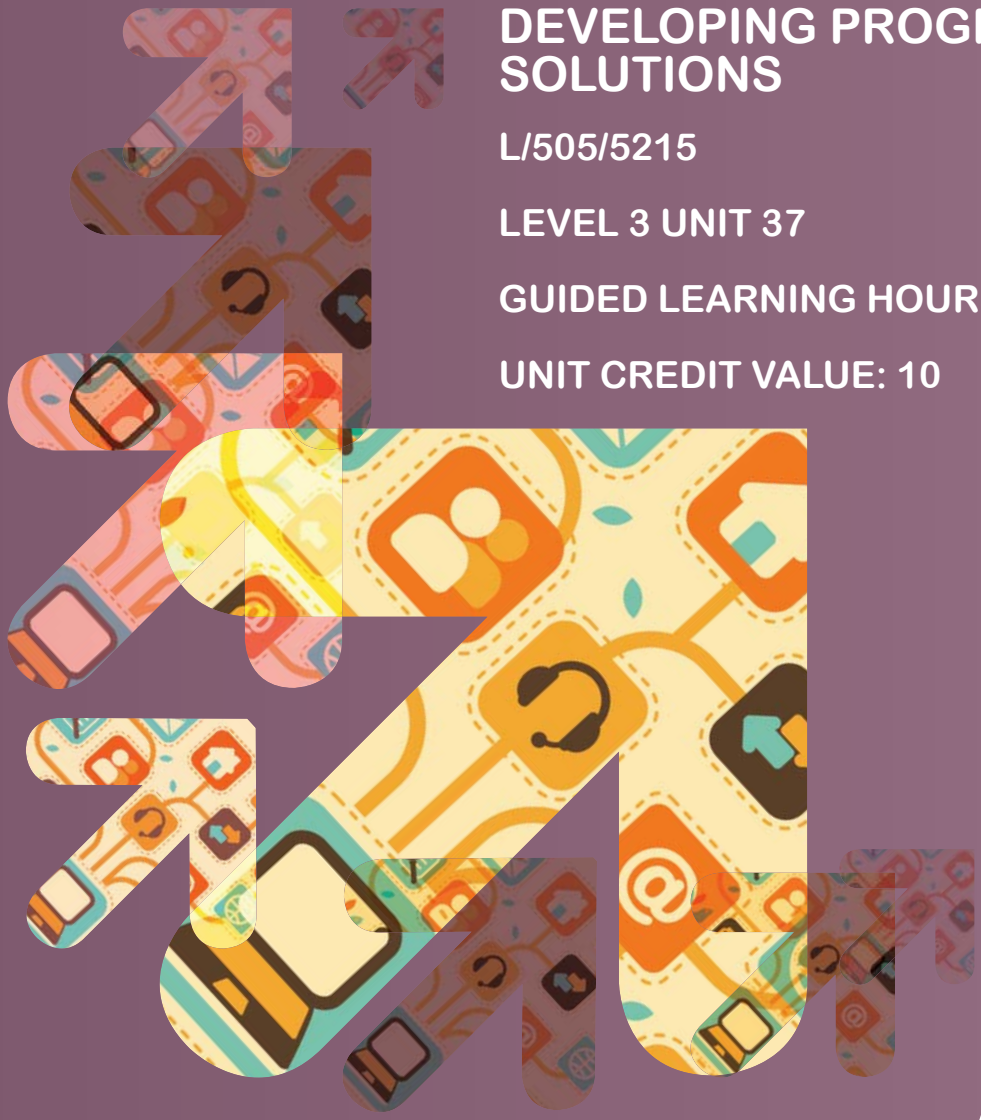
Cambridge TECHNICALS

Accredited

OCR LEVEL 3 CAMBRIDGE TECHNICAL

CERTIFICATE/DIPLOMA IN

IT



DEVELOPING PROGRAMMING SOLUTIONS

L/505/5215

LEVEL 3 UNIT 37

GUIDED LEARNING HOURS: 60

UNIT CREDIT VALUE: 10

OCR

DEVELOPING PROGRAMMING SOLUTIONS

L/505/5215

LEVEL 3 Unit 37

AIM AND PURPOSE OF THE UNIT

This unit will develop the practical skills required to produce functional and effective programming solutions.

On completing this unit learners will know about software development processes, algorithms, validation, verification, optimisation and documentation for programming solutions. Learners will be able to apply this knowledge to develop their own programming solutions. Using a client's requirements specification, learners will be able to design a solution in response. They will then produce a programming solution based on the design using an appropriate development environment. They will test, validate, verify, and optimise the solution. Finally, learners will create and store all relevant test results, manuals and other documentation for future reference.

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 the software development process	P1 describe development environments used to produce programming solutions		
	P2 explain the software development process for a specific project to meet client requirements	M1 produce a software specification for a specific project	
2 Be able to develop a programming solution	P3 produce a design for a programming solution to meet client requirements P4 create a programming solution to meet client requirements P5 apply debugging techniques to programming code to solve errors	M2 format the programming code to meet client requirements	D1 develop algorithms to support a programming solution
3 Understand validation and verification processes	P6 explain where validation and verification processes will be used to support a programming solution P7 produce a verification plan for a programming solution		
4 Be able to produce a functional programming solution	P8 test the programming solution against identified test requirements	M3 implement validation and verification routines to ensure system integrity	D2 Evaluate and optimise programming code, algorithms and routines by using tools and techniques
5 Be able to document a programming solution	P9 Create documentation for a programming solution	M4 create a user guide for a programming solution	

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 the software development process

- **Development environments**
  - Standalone environments eg software development kit (SDK), single user visual environments, Windows developer, environments, emulators
  - Collaborative environments and team working approaches i.e. for projects with several programming developers all working on different components of the same system whether networked, server based or cloud based
- **Choice of language for new software development vs. older languages for maintenance and update purposes e.g. where the time and cost of a complete redesign is unrealistic, consideration for future maintenance, support and expansion**
- **Suitability of languages for different target platforms e.g. development costs, timescales, resources, features, scalability, code efficiency, processing power, security, maintenance**
- **Types of programming language i.e. object oriented, procedural, event driven**
- **High Level programming languages (e.g. VB, C++, JavaScript, HTML, XML, CSS, asp.net, PHP, Python, Perl, Ruby, Pascal, Cobol, Fortran)**
- **Use of low level assembler programming languages specific to 8/16/32 bit microprocessors and microcontrollers e.g. for code efficiency, operating speed, time critical applications**
- **The use of compilers, interpreters and parsers and their use with different programming languages**
- **Software development process**
  - Feasibility
  - Investigation
  - Analysis
  - Design
  - Implementation
  - Maintenance
- **Software specification content (e.g. scope, requirements, platform, data flow, functionality, hardware/software interfaces, user interface, security)**

LO2 Be able to develop a programming solution

- **Software design structure (e.g. use of files, functions, variables, procedures, objects, data types, storage, algorithms, security)**
- **Software solutions (e.g. bespoke/custom, build or buy, off the shelf (OTS), customised off the shelf (COTS))**
- **Use of flowcharts, data flow diagrams, rich picture diagrams e.g. to illustrate the software structure, program flow, component parts, data acquisition, storage and management**
- **The writing and use of algorithms to achieve objectives i.e. different approaches to coding that will achieve the same result e.g. using different methods of sorting data**
- **The writing and use of pseudo code e.g. as an informal language, to show how the programming code and algorithms should be produced, how it is generic across all programming languages, for use when developing solutions for multiple platforms and languages**
- **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)
  - External entities (source and recipient)
  - Containers, I/O, stacks, libraries
- **Syntax and style of different programming languages (e.g. declarations, uppercase and lowercase characters, use of underscore, plurals, prefix, suffix, naming conventions, file naming and extensions)**
- **Coding conventions and house styles (e.g. file organisation, naming conventions, version control and commenting of code for maintenance and upgrade purposes)**

- **Debugging tools and techniques e.g. debugger software tools, using trace statements, monitoring techniques, emulators**
- **Identification of testing requirements**
- **Impact of organisation policies e.g. licensing (GNU GPL, EULA), copy protection considerations, security procedures**
- **The use of version control when working on software development projects e.g. alphanumerical and/or date based file naming systems**

LO3 Understand validation and verification processes

- **Terminology: understanding validation and verification and the differences**
- **Software validation i.e. whether the software meets the users requirements**
- **Software verification i.e. whether the software meets the software specification**
- **Creating validation and verification plans eg what is being checked and at what stage in the development process**

LO4 Be able to produce a functional programming solution

- **Identifying test requirements from a software specification e.g. success criteria, data input, output, user interface**
- **Creating functional test plans e.g. test documents, tables, dates, success criteria, results sections, re-testing requirements**
- **Application of validation and verification techniques and routines e.g. code inspection, static analysis, execution control, dynamic analysis**
- **Impact on testing and system performance when in debug mode e.g. introduction of system delays, timing changes, performance issues and overall functionality**
- **Testing strategies e.g. formal test plans, pass/fail criteria, debuggers, monitoring and tracing, use of in circuit emulator (ICE) for low level code and embedded systems**
- **Testing stages i.e.**
  - Developmental
  - Functional testing (e.g. with client or end user)
  - Beta releases
  - Final sign off
- **Optimisation reports e.g. using built in tools and reporting features**
- **Analysis of code efficiency e.g. for algorithms, time critical applications, operational speed, identification of areas for improvement and/or performance enhancement**

LO5 Be able to document a programming solution

- **Key documentation**
  - Design/structure of the programming solution
  - Programming (ie source) code
  - Test plan with results
  - Storing key documentation in a shared organisational environment or folder for ease of access by others

Analysis of software validation and verification results e.g. does the solution do what was intended, does the solution meet the specification, does it function as expected

- **User guides e.g.**
  - Contents
  - General introduction
  - Installation
  - Operating instructions
  - Configuration
  - Settings
  - Troubleshooting
  - Maintenance
  - Backups
- **Maintenance plans and requirements e.g. identifying regular routines and processes such as data storage, analysis of error logs, reports, security messages, performance monitoring**
- **Housekeeping requirements e.g. clearing out log files, archiving data, performing backups**

DELIVERY GUIDANCE

Wherever possible the unit should be delivered effectively within a workshop environment, giving learners a series of exercises, tasks and practical activities that build on knowledge and complexity of solutions. 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 client's software requirement in order to make informed choices.

Understand the software development process

To introduce this unit the tutor could discuss a range of different software development environments. These should be demonstrated, by, for example using a PC and projector, highlighting key features and functions. Learners would then be able to experiment and explore these environments using their own computer. Supporting hand outs on programming languages and what development environments they are used in should be provided in order to develop a broad understanding of their use. Shared workspaces and collaborative/team working environments should be included in addition to standalone environments aimed at single users or developers. Learners should work together in a small group to create a programming solution in order to experience the team working approach to software development.

Handouts, presentations and research activities can be added to build knowledge on the use of compilers, interpreters and parsers with different programming languages. A minimum of two case studies should be discussed. One of these should be for a new programming requirement and one for an upgrade or enhancement to an existing system. A good example here would be to use a case study for a system that was written in an older language such as Cobol, Pascal or Fortran. The tutor could oversee a group discussion on the pros and cons of continued upgrades to an older system versus the costs, timescales and risks associated with a complete re-write.

An activity to compare and contrast different languages and development environments should be completed with examples of how they were used to produce solutions. These comparisons should consider the availability of features within the development environments that will affect the development and rollout timescales, in addition to the cost of a commercial (i.e. non educational) license. A further comparison of high level languages and low level (assembler) should be covered, either as a discussion or research activity. The occasions where assembler programming still has some benefits can be shortlisted such as in applications where known precise timing and execution of instructions is needed.

The tutor should explain and illustrate the software development process, such as using a flowchart. A case study for a specific project that includes reference to these key stages should be discussed and explained. The process should follow through all stages of the design, development, validation, verification, debugging, testing and systems integration. The introduction to this may be in the form of the initial client requirements, which is further supported by an example of a detailed software specification. Where possible, the original software specifications for the case studies should be used so that learners can develop their understanding of the processes, requirements and decisions made to meet a given requirement. An alternative would be to use some simulated material as long as it demonstrates a real world process and is not unrealistic within the sector.

Be able to develop a programming solution

The use of a flowchart would be one way to introduce the overall structure. A verbal explanation of the component parts should then allow investigation in more detail, eventually identifying the code and syntax used. A 'top down' approach is encouraged since at this stage an overall understanding of the structure and design is being developed rather than just building up larger coded solutions.

Separate workshops should be completed on the topics of pseudo code and algorithms. Having introduced the concepts in a presentation, exercises should be completed individually to create pseudo code solutions for different scenarios and requirements. Peer reviews could also be used on the pseudo code to ensure they are clear and effective in illustrating what is needed, which would be used later in the process by programmers and developers. The writing of algorithms could also be produced using pseudo code as a starting point.

A workshop on the use of good practice in the layout of programming code should be completed. Quizzes could be used to cover knowledge of syntax, indentation, variables etc. although these should also be put into practical use in the production of programming code. Peer reviews could be used to assess whether the use of comments in a piece of programming code is sufficient to explain what the code is intended to do.

The workshop activities should continue in the form of exercises to design a solution that would satisfy a client requirement. The process could be delivered by using a software requirements specification that could be met using a relatively short programming solution. Sessions on writing short pieces of code to produce individual programming solutions in different environments could be completed although a team approach is also encouraged for at least one requirement.

The use of a document that defines organisational guidelines (i.e. house styling) with regard to formatting and layout of programming code would be encouraged. This document could also include naming conventions, file extensions and version control policy. Exercises and projects to become progressively more complex that require different algorithms and design structure in both single user and multi user environments.

Understand validation and verification processes

This Learning Outcome could be delivered alongside Learning Outcome 2 such that validation and verification (V&V) techniques are used with any or all of the programming exercises. The overall delivery should continue in a workshop environment with a range of practical activities to validate and verify coded solutions.

The tutor could introduce the concepts of V&V through a presentation. This should identify the differences between validation and verification together with some of the main techniques used. This could be followed up with an initial exercise to inspect a short piece of code to see if it does what is intended. Having introduced some of the techniques, the tutor could then identify when and where these are used. A workshop activity to create V&V plans should then be completed. Having established the basic knowledge and understanding of V&V, learners would then be in a position to apply the techniques to their own coding projects as part of the next Learning Outcome.

Be able to produce a functional programming solution

The introduction to testing can be completed in the form of a presentation by the tutor. This should identify the purpose of testing, strategies used, the different test stages throughout the development and the format and content of test plans. A workshop activity on how to create a test plan would need to explain how suitable tests are identified from the initial requirements specification. One of the specifications used earlier in this unit could be used for this.

As a starting point, an example of a detailed test plan could be circulated for reference and then a group discussion used to identify tests for a different requirements specification. It would be beneficial if this specification was already familiar to the learners so that they already have a broad understanding of the system. Using this approach, the tutor becomes a facilitator for the learners to develop a test plan as a group activity.

Examples of test plans can be produced in the form of hand outs which are to be completed by each learner (i.e. with their results of testing). Alternatively an electronic version (e.g. word processed document or spreadsheet) may be provided as the test plan for learners to complete and save in an appropriate format. Version control and time/date of testing should be incorporated into the filename for record purposes here.

The process of testing a programming solution may be used with any or all of the programming exercises in Learning Outcome 2 or 3. If not already covered, it is recommended that these include a range of different types of IT solution such as:

- **web/cloud based PC solution**
- **embedded controllers or bespoke hardware**

Testing in different environments and stages of the development should be included. This will assist a broad range of understanding test requirements when developing IT solutions. In addition to the developmental stages of testing an activity should be included that simulates the final signing off with the client. This could be for each learner's programming solution and use either peers or the tutor acting as the client. The learners would need to demonstrate the solution, record the results of testing (which should be a pass at this stage) and request the client to sign the test plan as being complete. This is a summative activity for the unit and any programming solutions developed, which is a formal process in commercial project development.

Following on from the understanding of validation and verification in LO3 it is suggested that a range of algorithms are provided for analysis. The algorithms would have the same intended outcome but use different programming approaches. Learners would be able to apply validation and verification techniques (as appropriate to the development environment and function) to identify which meet the users requirements and which meet the software specification. Intentional problems could be embedded into these

algorithms although they should not be coding errors that require debugging (which is a different process).

Opportunities for the optimisation of programming code should be identified when creating solutions and testing system performance. Built in analysis tools and reporting can be used that are part of the development environment. Examples used for optimisation exercises may again include different algorithms and/or have clear opportunities for an improved structure.

The analysis of a software system could be demonstrated using a PC and projector, with a 'live' activity to run the analysis and optimisation tools and view the corresponding reports. Copies of the reports could be printed for reference purposes. A discussion of the content would allow learners to annotate their own copy so that they can identify key elements and areas for improvement. This will assist in the interpretation of reports on their own software solutions in their subsequent analysis.

Be able to document a programming solution

Having completed the development and testing, the final stage is to document the system producing user manuals, maintenance plans and identifying any housekeeping tasks. The tutor should emphasise the importance of storing all the relevant test results and documentation in a presentation with perhaps a case study of what could go wrong if these are not done. The importance of storing these in a location that can be accessed by others should be included. Learners should then be given the opportunity to create a user manual for their software system and store all the supporting information using suitable file names and folder structures.

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

Assessment Criteria P1, P2, M1

For P1 learners must describe development environments used to produce programming solutions. This may be in the form of a report or presentation, and should describe the features and functions of a range of development environments.

For P2 the learners must explain the software development process for a specific project to meet client requirements. The teaching content identifies a list of key elements that should be included. This may be a document combined with P1, and the learner should identify the planned development and the criteria that will form the basis of the development.

For merit criterion M1, learners must produce a software specification for a specific project, which should provide enough detail for the design of the final solution as outlined in the teaching content.

Assessment Criteria P3, P4, P5, M2, D1

For P3, learners must produce a design for a programming solution to meet client requirements. It may assist the learner if the solution is based on the software specification from M1 if appropriate. The design should be evidenced in the form of a document that describes the structure of the proposed solution which may also be supported by a presentation. This design should also include a test plan, which may be in a table or spread sheet format and should include the test requirements as identified in the teaching content. The type of testing should be clearly identified and must be one of the stages found in the teaching content (for Learning Outcome 4) that is appropriate to the nature of the programming solution.

For P4 learners must create a programming solution to meet client requirements using an appropriate programming language and code. This may be evidenced electronically, on screen or as a printed code listing.

For P5 learners must apply debugging techniques to programming code to solve errors. This may be documented using screen captures if appropriate and included in a report on the processes followed. Additional forms of evidence may include printed listings, error reports, trace statements etc., which should be stored as part of the overall project documentation.

For merit criterion M2, learners must format the programming code to meet client requirements using an appropriate layout, indentation, use of white space and include annotations of their code as appropriate. The comments may include author, version and explanations of what the code is intended to do in a number of key places. The formatting conventions should follow a style which may have been identified to the learner or by the learner.

For distinction criterion D1, learners must develop two or more algorithms to support the programming solution. These algorithms should be clearly identified in the programming code. They should also be formatted as required as per merit criterion M2.

Assessment Criteria P6, P7

For P6 learners must explain where validation and verification processes will be used to support the programming solution. Learners should clearly demonstrate their understanding of both validation and verification processes. This could be evidenced by a report or a presentation, and should clearly identify where each will be applied throughout the development process.

For P7, learners must produce a verification plan for the programming solution that identifies what techniques will be used and at what stage. This could be evidenced by a report or a presentation. This may use evidence from P2 and P3 as a basis, and may be generated alongside P6 but needs to be clearly documented separately.

Assessment Criteria P8, M3, D2

For P8, learners must test the programming solution against identified test requirements. This will require the use of a test plan that would have been identified and/or produced earlier as part of the design solution in P3. The test plan may be in a table or spread sheet format and should include the test requirements as identified in the teaching content. The type of testing should be clearly identified and must be one of the stages found in the teaching content that is appropriate to the nature of the programming solution. Explanatory comments should be added as part of the results of testing. The test plan and results should be stored.

For merit criterion M3, learners must implement validation and verification routines to ensure system integrity. This process should be documented, identifying what is being done and what the results were. This may be evidenced in the form of a V&V report document, which could also be in a suitable format for storing in line with organisational guidelines.

For distinction criterion D2, learners must evaluate and optimise programming code, algorithms and routines by using tools and techniques. The evaluation should identify areas for optimisation and improvement to enhance performance. Separate sections may look at the programming code, any specific algorithms and/or routines. The results of the evaluation and any changes should be documented with the improvements and benefits clearly stated.

Assessment Criteria P9, M4

For P9, learners must create all 'Key documentation' as identified in the Teaching Content and store these in a suitable shared workspace location. This documentation may have been created as each learning outcome was completed. Where learners have also achieved merit criterion M3 (ie the use of validation and verification techniques) then the results of this should also be stored, together with other key documentation produced as part of the system development.

For merit criterion M4, learners must create a user guide for a programming solution This should be a separate document, formatted appropriately for distribution to the client and its users. Key sections should include an appropriate range from the list identified in the Teaching Content. This may also include supporting documentation outlined in P9.

RESOURCES

Software tools and environments including access to a range of programming software, compilers, interpreters, debuggers and analysis tools

Systems software to produce designs, flowcharts, test plans, V&V plans

MAPPING WITHIN THE QUALIFICATION TO THE OTHER

Unit 3: Computer systems

Unit 10: Developing computer games

Unit 13: Installing and upgrading software

Unit 26: Web server scripting

LINKS TO NOS

5.2 Software Development

5.3 IT/Technology Solution Testing

5.4 Systems Integration

5.5 IT/Technology Systems Installation, Implementation and Handover

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