

THE INSTITUTE OF COMMERCIAL MANAGEMENT

SUBJECT SYLLABUS



Programming Project

Unit code: PP-0908

Level: Diploma

Credits: 20

Unit leader: PY

Pre-requisites:

Main Aims of the Unit:

The aim of the project is to facilitate the application of programming skills in the development of a real-life situation (normally related to local business) and to extend the student's experience of programming.

The student will identify a specific complex problem and produce a working solution to that problem.

Main Topics of Study:

Students will build on the knowledge gained in earlier units and this project will enable the demonstration of a range of knowledge and skills including:

- Project identification
- Analysing a real-life situation
- Designing the system in relation to specific elements such as input, output, file structures and procedures
- Developing a computer program in a suitable programming language
- Incorporating test procedures for the program
- Producing appropriate documentation

Learning Outcomes for the Unit:

At the end of this Unit, students will be able to:

1. Analyse a given business problem or issue
2. Design an appropriate system to resolve the identified problem(s)
3. Develop a program using an appropriate high-level language
4. Design and develop appropriate documentation for the system
5. Evaluate the complete process and provide recommendations for further enhancements

The numbers below show which of the above module learning outcomes are related to particular cognitive and key skills:

Knowledge & Understanding	2, 3, 4
Analysis	1, 5
Synthesis/Creativity	1, 5
Evaluation	5
Interactive & group Skills	-
Self-appraisal/Reflection on Practice	5
Planning and Management of Learning	1-4
Problem Solving	1, 2, 3
Communication & Presentation	4
Other skills (please specify)	-

Learning and teaching methods/strategies used to enable the achievement of learning outcomes:

Learning takes place on a number of levels through lectures, class discussion including problem review and analysis. Formal lectures provide a foundation of information on which the student builds through directed learning and self managed learning outside of the class. The primary activity will be student-centred and will involve practical self-managed practical work .

Assessment methods weightings which enable students to demonstrate the learning outcomes of the Unit:

Program and documentation: 100%

Indicative Reading for this Unit:

Main text

Programming manuals

Alternative Texts & Further Reading:

Guideline for Teaching and Learning Time (10 hours per credit)

Lectures / Seminars / Tutorials / Workshops: 50 hours

Tutorial support includes feedback on assignments and may vary by college according to local needs and wishes.

Directed learning: 50 hours

Advance reading and preparation / Class preparation / Background reading / Group study / Portfolio / Diary etc

Self managed learning: 100 hours

Working through the given real-life problem and designing and developing a working solution will take up the bulk of the learning time. In addition students are expected to engage with the tutor and other students and to undertake further reading using the web and/or libraries.

Overview

There is no examination for this module. It is tested totally by submitting documentation. The documentation should include all 18 sections listed below. These have all been defined more clearly to assist the candidates and centres in knowing what is required. Candidates must be made aware that the nature of this documentation makes it almost IMPOSSIBLE TO WRITE IT UP AFTER ALL IS COMPLETED. For instance, the testing process requires FULL PRINTED EVIDENCE and unless testing is ongoing and systematic, it will be difficult to produce full documentation later. THIS DOCUMENT should be covered in detail as part of class-work and be given to each candidate when DEVELOPING the programs.

The aim of the project is to test a candidate's ability to apply programming skills in the development of a REAL situation (normally related to business) and extend him/herself beyond the classroom programming exercise stage.

The project must produce a working solution to the problem. This problem should be of a NON-TRIVIAL nature; avoid projects like payroll which can be very involved. Past projects on payroll have been naïve simplifications, unrealistic and unworkable in the real world where a commercial payroll package may have hundreds of different possible and complex adjustments to an employee's monthly salary.

ALL CANDIDATES AT A CENTRE MUST CHOOSE DIFFERENT PROJECTS.

Any appropriate computer language may be used. However, if a method is adopted where code is generated AUTOMATICALLY, then a listing of this code MUST be included and it must be FULLY ANNOTATED showing which part of the coding deals with which aspects of the project. THIS IS A PROGRAMMING PROJECT, NOT A DATABASE PROJECT.

The submitted project work MUST contain all the documentation relating to the production and testing of the programs. In particular, items 12 to 15 below are all components of the testing process and each should show how it relates to particular parts of the other sections. Previous submissions by candidates have included either no testing or minimal. For instance, if the project is about a customer ordering system, it is not satisfactory to have one or two customers and one or two different products – this does not test the accessing of files adequately nor does it show realistic billing which could spill over onto a second page. The tests MUST also be put into context. e.g. For a given test, manually write on the printout to which test in the test plan it refers, which set of test data was used to produce it and to which test it relates in the test log. Where output is on the screen, screen prints should be provided. Only if these sections are handled correctly is the candidate likely to score highly.

Sequence of events

- 1. Start of Course.** Select a suitable project to solve. It should really contain at least TWO data files. All candidates at a given centre must choose DIFFERENT projects.
- 2. Read the documentation requirements.** It is essential that the candidate knows what is expected BEFORE any significant work is started.
- 3. In the first weeks of the course.** Complete an ICM approval form and submit it to the Institute. ICM will return this quickly with comments. The aim of this is to ensure that the project is neither TOO simple nor TOO difficult. If ICM considers it too difficult and would involve too much time and effort, it will either advise changing or suggest that only a part of the project be completed. Some previous projects have been so trivial they could not pass however good the documentation might be.
- 4. During the course.** Solve the problem through a systematic development method, test the work and document it throughout. NOTE. There must be printed evidence of full testing. See items 12 to 15 in the list below.
- 5. End of course.** Check that you have documented ALL elements from the attached list. This list has been detailed more fully to clarify the requirements more precisely. Put your work into order – preferably the order of the list and submit to the Institute. DO NOT SEND DISKETTES.

Documentation

Marks	Explanation
1. 5	Definition of the problem and limitations of the programs. ALL programs have limitations and never completely solve the problem – identify these. Explain WHY the program is being written and for whom. DO NOT include a list of thanks to tutors/friends etc - this has nothing to do with the running of the program, wastes time and will be ignored.
2. 5	Objectives of the program. This is not asking for AIMS. Objectives are <u>measurable</u> . A program either produces an invoice or it does not. AIMS such as “to make the data entry easy for the user” are not measurable and therefore should not be included.
3. 10	Analysis of the current situation. Explain in detail how the current system works. Include its shortcomings and any additional user needs.
4. 5	Alternative solutions. Identify alternative ways in which the problem could be solved. Some alternative ways are listed below but they need to be fully explained and justified: <ul style="list-style-type: none"> i buy a ready written package (if it exists – so you must identify it). Why write payroll programs when a package can be bought cheaply and working within hours? This is almost certainly the cheapest method and so there must be good reasons to reject it. ii develop using a database system. iii develop in one of several languages. <p>State why your chosen method IS the most appropriate and why the others are rejected.</p>
5. 10	Data input. List in data entry order, the specifications of each data set. It is useful to give each data set a meaningful name – e.g. 1. customer registration data. Data should be treated as a group of fields rather than many single items – e.g. all data for customer registration data or all data for an order item. For each field in each data set, include (i) fieldname, (ii) data type, (iii) any restrictions on it and (iv) full validation. A standard form could be devised.
6. 10	Output. Show a sample of every type of output whether on screen or on the printer. If the project is over-complex, this could present a real problem.
7. 10	Files. Identify all files involved. For each file state: <ul style="list-style-type: none"> (i) filename (ii) method or organisation (iii) method of access – there may be more than one access method depending on which aspect of the processing is used – differentiate between these clearly (iv) fieldnames (v) datatypes of each field (vi) field size. These last three items can best be tabulated in a standard form.
8. 15	Process specification. Describe ALL the processes in turn. For each state: <ul style="list-style-type: none"> (i) its purpose (ii) name the data input and output produced (iii) files involved and how each are accessed for THIS process (iv) where this process occurs in relation to others – e.g. after A and before C where A and C are named processes.
9. 20	Algorithm. This must use a structured approach. More emphasis will be placed on this section for projects where some of the code is produced automatically by the system. Pseudo-code is preferred.
10. 20	Annotated structured listing. A full listing of the code should be given. It should follow the algorithm CLOSELY. The code should contain the programmer’s notes to explain how a particular feature is performed. Hand written comments should be added if these notes are incomplete. Program syntax must be totally correct. Any syntax errors will invalidate any testing applicable to that section of code and testing marks WILL BE REDUCED.
11. 5	Variables list. This must include EVERY variable used in the program except those input from fields of a file. Each must show <ul style="list-style-type: none"> (i) which module/section uses that variable, (ii) variable type (iii) size (iv) why it is needed. <p>AVOID statements like “To hold customer number”. Every variable “holds” something and in this case, WHY does the program need to have the customer number?- this is the reason for the variable’s existence. There should be some system in your list – list either by program module/section (this may result in duplication) or perhaps alphabetical (but this must then show which module(s) use the variable).</p>
12. 5	Test Plan. This is an outline showing the order in which processes/sections of code are tested. For instance ADD A NEW CUSTOMER should be fully tested before attempting any orders for that customer, amending customer details and certainly before deleting a customer record. Number each proposed test for easy cross-referencing.

13. 15 **Test Data.** This MUST be comprehensive. Unless a significant number of data records are on file, the testing will not show how a particular record is located or allow for multiple items such as a customer ordering many different products.
For each set of data, explain WHY it is present. e.g. If discounts are offered, then there must clearly be data to test for a customer who earns (i) NO discount, (ii) discount at one level (iii) discount at higher levels. In addition the test data must ensure that discounting does kick in at the right point by having test data AT the changeover point and immediately below it. Cross-reference each set to the Test Plan.
Testing must be FULLY documented. No documented testing will lose ALL the marks allocated for sections 12,13, 14 and 15 – nearly 25% of the project.
14. 20 **Testing.** Include printed evidence for ALL significant tests (see the notes for section 15). For each output, write in by hand, (i) a brief explanation of the purpose of the test (ii) cross reference to the test plan (iii) mark/comment on printouts of all items which are correct and which were not previously tested (iv) mark/comment on all items which are incorrect. For screen outputs, produce screen prints. For instance, as a guide, to show that a new customer has been added to a file, produce a suitable report BEFORE the new customer showing the customer details NOT present and the same print after the new customer has been added – highlight the differences by hand with a brief comment.
15. 5 **Log of Testing.** After each test, record in table form, (i) date and time (ii) the test being performed (cross-referenced to the Test Plan) (iii) the outcome of the test – either faults or processing assessed as being correct (iv) first thoughts about the remedy for correcting any observed fault.
NOTE: It is not wrong to have tests which fail – all professional programmers make mistakes or overlook some feature. It IS however wrong to ignore faults that occur. There is no need to record details of a test that is a complete failure if this test is immediately followed by one that corrects some of these problems. Past test logs seem to show the candidate as being a brilliant programmer - the whole program completed in three runs and no errors! Is this likely?
16. 25 **User manual.** This should ideally be a SELF CONTAINED SECTION of the documentation describing (i) the program in outline, (ii) its options and limitations, (iii) sample inputs and outputs (iv) errors/problems which could occur and how to rectify them (v) operating instructions. If any part of this user manual should ALSO be elsewhere in the whole documentation, it MUST be here but in its other locations, a cross reference to the page number in the user manual can be given.
17. 10 **Technical Documentation.** This is for the maintenance programmer and relates to (i) any special methods employed, (ii) formulae used, (iii) data structures used such as matrices/linked lists, (iv) comments about the use of files, (v) back up procedures.
18. 5 **Appraisals.** This can be in two forms. If the program has been written for another user, that user should give a CRITICAL appraisal of its use and any problems should be noted. If there is no available end-user, the candidate can perform a self-appraisal but it must be CRITICAL. Are (i) there any features which could be added later or improved further, (ii) any features still not working etc... This appraisal is about the program, not the process of developing it so do not include problems you had with machine access etc. A claim that the program works fully will be discounted if there is no testing to prove it. In short, writing a few sentences to fill in this section is pointless unless it says something meaningful.

TOTAL 200 marks which will be halved to produce a percentage.

Overall Comment:

The candidate will gain more credit and marks for an honest assessment which shows that problems/errors have occurred and the steps taken to put them right. This still applies if the submitted project still has some logic errors and therefore only some of the submitted project is completed. This recognises that the creation of programs is not an exact science but requires means of putting things right. A candidate who appears to claim that everything works first time is not showing recognition of this process. There is also no point including sections of documentation that state very little and are merely there to try to satisfy the examiner.

Note that the user manual should be self-contained. It need not be detachable but the pages, if given to the user, would be all that the user needed. Therefore, other parts of the documentation should cross-reference into the user manual.

Forms:

Forms can be used to describe many items in this documentation. This can either be created by the candidate or supplied by the centre itself. However, if a centre-supplied form is not as appropriate as it should be perhaps because it was designed for a slightly different purpose, then the candidate should adapt it or design his/her own form.

Suitable forms could be used for details of:

1. Whole project progress time chart to show when each stage is completed.
2. Input data – one for each named and different SET of data.
3. File contents – one for each file.
4. Process specification.
5. Variables list.
6. Test Plan.
7. Test data including the purpose of that data – one for each set of data.
8. Test log with space to cross-reference to other test related items.