Computing coursework help

Choosing a project

A good project choice is critical to the success of this coursework. Although the choice of project is up to you there are some guidelines which must be followed. These are to help you choose a project which is manageable but will also meet the requirements for the top grades –

- There must be an end user who will be willing to take an active role on the project
- The project must not be too simple! It MUST have a substantial coding element to it
- The project should not be too ambitious!
- You must fully understand what the user wants and understand the area of business they represent.

The end user

The end user must take an active role in the project. You will need to interview them, get them to test your program and they must be willing to do a bit of writing! It is very likely you will be in active contact with the user over the period of a year. You must be able to get in contact with them quickly either face to face, email or over the phone.

It works better if the end user is known to you or your family. Parents, older siblings, uncles / aunts or friends of the family are normally more willing to give you the time needed to implement a system. It also helps if they have their own business or are self-employed as they will normally not have any custom built software due to the cost. Users who work for small companies may also fall into this category.

Your user must also require a system! It is possible to re-design part of an existing system for them but this is more difficult. The normal categories for a system are –

- Converting a paper based system into a computerised system
- Automating parts of a current system
- Implementing a new system for a need which is currently not met
- Upgrading an old legacy system.

When talking to potential users you may wish to ask the following questions –

- Is there any part of your system / work which is paper based?
- Is any part of your job repetitive?
- Do you use an old or very slow system?

Note - DO NOT select a user who works for a large company! They will normally have a working system or the new system will be highly complex.

Note 2 – NEVER try and integrate more than one piece of software together, especially if it is legacy software or not office based. These projects tend to go tits up!

What makes a good project

This list IS NOT exhaustive! It is merely to give you an idea of what you could do.

- An interactive website / content managed website
- Booking systems (either online or app based)
- Anything which requires data to be stored in categories (like school resources)
- Automated email system / systems which generate emails
- Customer / order management systems

What makes a bad project

These projects tend to go south very quickly!

- Online stores (due to complexity)
- Games (hard to justify a end user and hard to get documentation marks on)
- Games! (they are also hard to program!!)
- Anything flash based or Microsoft Office based (again hard to get marks on)
- Anything which has a large GUI! (Becomes hard to manage and is time consuming without getting you any extra marks)
- Address books or anything which stores data in just one or two tables.
- Anything involving concurrency or threads!!
- Expert systems.

Scope of the project

Projects vary in size from trivial to highly complex. The key to a good project is to pick one which is in the middle. It is also important to bare in mind your skill level. If you are a seasoned veteran when it comes to programming you can be more ambitious. In order to decide if you can do a project you need to do have an idea on how you could make it. This is where your teacher can guide you!!

One thing to remember is that large ambitious projects can be cut down to make them more manageable. It is much harder to add extras to a small simple project.

Understand what the user wants

You will, later on, perform fact finding to find out exactly what the new system will do. However that takes time and if you find out at the end of that process that the project will not work then you will have wasted a lot of time! We need to look at the feasibility of the project.

You must have a good idea of what the user wants before you start analysing the system. You should be able to answer the following questions

1) What sector does your user work in?
2) What problems do they currently face?
3) Which of the problems are you going to solve?
4) How does their business work?
5) Who is the project aimed at?

This means you will have to discuss, at length, how your user works and what they do. Never simply get a rough idea and then hope it will all fit into place. The more time spent on this will mean less time is needed later on.

Grade boundaries

These are UMS boundaries. That means that if you get, for example, 60 raw marks you may end up with only 55 UMS or maybe 65 UMS! As such this section is merely a guide. You should focus on getting near the middle of any grade boundary in order to secure that grade.

Grade	Mark needed
A*	72
A	64
B	56
C	48
D	40
E	32
U	< 32

Note – A* can only be achieved if you get over 72 on the coursework AND 108 on the F453 exam. You must get 90% on your A2 units to be eligible for A*.

Problem definition

What is it?

This is where you find out what problems/issues your user has and decide on which ones you are going to try and fix. It is NOT how you are going to fix the problem.

What you need to include

- An overview of who your user is.
    - What sector do they work in?
    - How many people work for them?
    - What is their current ICT infrastructure?
    - What is their level of competence in ICT?
- The origins of data or where does information come from? It could be from emails, phone, face to face, scanning or even MICR or OCR.
    - Where does the data come from?
    - How is it CURRENTLY stored?
    - Is it backed up?
- The problems the user currently is facing. This should be focused on the area you are looking at creating a system for. There is no point writing about all of the financial issues if you are going to be focusing on marketing!
    - What are the main problems?
    - How do they affect the business?
    - Does it cost time / customers or is it a limitation to expansion?
- What happens next? You must state how you are going to proceed. This tends to be a interview with the main user but could be any combination of fact finding.

Mark descriptions

3 marks	Excellent description with all elements present.
2 marks	Some description of both the stages of study and end user involved.
1 mark	Vague description of the end user or area for development.

Top mark boundary

- There will be no confusion over what the business does or how the business works.
- Technical terms used by the business will be explained or will be included in a glossary.
- Clear description of problems which are not vague or generic. Every problem must be specific to the company.
- Clear origins of data specifying exactly where the data came from and how it is used.
- Clear statement of how analysis will be conducted.

Middle mark boundary

Will contain MOST of the following-

- Clear description of business and how it works.
- Technical terms used by but not always explained.
- Problems are specified but may not be detailed or specific.
- Clear origins of data specifying exactly where the data came from and how it is used.
- Clear statement of how analysis will be conducted.

Low mark boundary

Will include a brief description of some of the following –

- Description of business and how it works.
- Problems are specified but may not be detailed or specific.
- Statement of how analysis will be conducted.

Top tips

It is critical that, at this stage, you fully understand how the business works. If you take your time to understand this then the rest of the project becomes MUCH easier.

You should aim to get 2-3 marks on this section as it is a good indication on how well you understand the whole problem. Projects which get less than 2 tend to be poorly defined and poorly understood.

Analysis

What is it?

Analysis is the fact finding part of the system. You will already have had an informal meeting with the user in order to write the problem definition. Future meetings will be more formal and much more detailed. You will be expected to do at least 2 different fact finding methods in order to get a good understanding of the system.

What you need to include

- A set of questions covering the details of how the current system works and what improvements are required.
- A transcription of the interview. We cannot accept a recording!
- The interview signed by the user
- One other form of fact finding (document collection is normally the best)
- Hardware and software requirements of the proposed solution
- A requirements specification

Interview questions

Your questions must get the required detail. That means questions such as “what do you want in your new system?” or “what are your current problems?” will not suffice. In fact you already know most of this from the problem definition! What you need to find out is the detail required in order to fully implement the system.

You need to find out the following –

- What input will be required and how does the user want the data entered?
- What output the user wants to see? This includes reports, printouts and onscreen information.
- What searches need to be done or what analysis is required?
- What is the workflow? How does data progress from input to output?
- Details on each problem specified in the problem definition.
- Does the system require any security arrangements?
- Is backup needed?

Workflow

A work flow / use case specifies how data will move through a system. Consider a mail order company. When an order arrives a number of things can happen. It could be that the item is out of stock and stock needs to be ordered. It could be that stock has already been ordered but has not yet arrived. The status of a order will change as it moves through the system from allocated to packing and finally dispatched. At different stages different things must happen.

As you can see this information is critical in order to make a working system! A question such as “What problem do you have with ordering?” would not get this detail! The current business processes MUST be found out and documented. In order to find this out a question such as “How does an xxxx progress through the current system? What states can it be in?” would start to pick up this detail.

It is important to remember that a user will not venture this information unless asked!! This will not be out of malice but because they will not realise you need that information.

Hardware and software requirements

You need to specify exactly what software and hardware the user is going to need in order to implement your system. This includes the minimum PC requirements (CPU speed, memory size and HDD space needed). You should also include any software you use to develop the program in. That will give the moderator an idea of how you went about making the system.

You need to research this section carefully. You must also decide on the programming language you are going to use and what API’s you will need to make the system. Your teacher will be able to point you in the right direction. You ARE NOT allowed office based applications.

Requirement specification

The requirement specification is little more than a big list of what your new system will do. This list MUST be detailed. A lot of students will write requirements such as

- Will need to be able to add new information
- Will need to delete information
- Will need to edit
- ... and so on!

This is not detailed and could describe any system! You need to split down these types of requirements. For example we could split the “add new information” requirement into –

- Register new customer
    - Validate their email address with a confirmation email
    - Ensure that this customer has not already signed up with the same email address
- Take a new order and set the correct status depending on current stock levels
    - Set to be “new order” if stock exists
    - Set to be “waiting stock” if it is waiting stock
    - Set to be “order new stock” if stock does not exist and has not been ordered.

You should also refer each requirement back to your fact finding so we can see why each requirement is needed. If you have added a requirement which cannot be found in your fact finding then you will lose marks. It makes it easier for both you and the moderator to be specific about why each requirement has been included.

You can use diagrams to aid your explanations. Describing workflow, for example, is much easier as a state diagram. Although they are not essential they are very helpful.

Mark descriptions

9–11 marks	Excellent user involvement with detailed recording of the user's requirements. All other items must be present, showing a thorough analysis of the system to be computerised. A detailed requirements specification, including full justification for the approach and hardware and software requirements, has been produced.
6–8 marks	Good user involvement and recording of the data collection methods. Most of the necessary items have been covered. However, one or two items have been omitted. A requirements specification is present with some attempt to justify the approach based on the results of the investigations but with some omissions, eg hardware and software requirements.
3–5marks	Some evidence that an attempt has been made to identify the end-user requirements and some recording of it has been made. Attempts at some of the other items have been made. An attempt has been made to develop a requirement specification but with little attempt to justify this based on the results of the investigation.
1–2 marks	Some elements have been discussed but with little or no user involvement.

High mark boundary

- Clear user involvement and detailed fact finding.
- Detailed requirement specification with no omissions
- All requirements are linked back to the fact finding and are fully justified.
- Clear hardware and software requirements.
- The chosen hardware / software method has been compared with another potential solution and justified.

Middle mark boundary

- User has been involved but fact finding may have omissions.
- Requirements specification has been produced but will have omissions and lack detail.
- Some justification has been made on why requirements have been chosen.
- Hardware / software requirements may be missing or lack detail.

Low mark boundary

- An interview has been made and transcribed but lacks detail and only gathers superficial information.
- Little or no attempt to justify requirements.
- No hardware / software requirements
- A basic / minimal requirements specification has been produced.

Top tips

- It does not hurt to have two interviews!!
- If you do not understand something the user has said... FIND OUT!!!
- Get used to matching your requirements back to your interview. If you have a requirement which cannot be matched back then simply have another interview!
- Use state and Use Case UML diagrams where possible. It cuts down the amount of words you need to write and will help you understand the system.
- Mistakes made in this phase will cause major problems later. Do not move onto design till you and your teacher are 100% happy.

Nature of solution

What is it?

This is where you look at the implementation of the system and decide on the aesthetics and storage of the system. Essentially you are looking at data structure design and input/output design.

Data dictionary

At this point you are not deciding on what data needs to be stored. The first step is to go through the requirements specifications and write down all of the data items which may need storing. This is best done in a table like the one below –

Customer table

Name	Data type	Size	Description
Surname	Text	40	Store customer surname
Customer ID	Integer	4	Each customer will have a unique number – **Primary Key**

As you are listing them try and assign them to tables. Once you have created the data dictionary you then need to normalize your database. You should refer to your lesson notes for this. Tables should then be written in the following form -

CUSTOMER(Customer ID, Surname, forename, DOB, email, phone number)

The next step is to create an ERD diagram to show the relationships between the different tables. Your final data structure design will include –

- ERD diagram
- Overview of tables
- The data dictionary

You do not have to show the steps of normalization nor do you get any marks for doing it. However if you do not normalize your database then you will get a lot of problems during implementation!

Input / output design

As you will most likely be producing a GUI you will need to design each screen before you implement them. This follows the exact same idea as what you did for F452 designing an interface. You will decide on how to make the users experience efficient and complete.

Note – You must include online help! You will potentially lose a lot of marks if you miss out online help.

When designing you need to -

- A screen number / code / name so it can be easily referenced
- Create a mock up of the screen with
    - Online help
    - User interface controls
- Include validation (where appropriate) for each GUI component.
- Describe what the buttons do including any buttons which take you to other forms.

Note – To speed things up design your forms in whatever you will be developing in. That way you only have to do it once!

Mark descriptions

5–6 marks	A clear set of objectives with a detailed and complete design specification, which is logically correct. There is evidence to show that the end user has seen and agreed these designs. There are also detailed written descriptions of any processes/modules and a clear, complete definition of any data structures. The specification is sufficient for someone to pick up and develop an end result using the software and hardware specified in the requirements specification.
3–4 marks	The major objectives of the new system have been adequately summarised, *but omissions have been made*. There is a brief outline of a design specification, including mock-ups of inputs and outputs, and the process model has been described (including a diagram: structure diagram, data flow diagram or system flowchart). There is some evidence that the end user has seen these designs. However, there is a lack of completeness with omissions from the process model, inputs and outputs. Data structures have been identified but there may be inadequate detail.
1–2 marks	Some vague discussion of what the system

High mark boundary

- The user has signed and accepted the design
- There are no omissions in the design. Everything in the requirement specification has been included.
- Data structure is logically correct. A ERD diagram has been included.
- Each screen has been explained in detail including validation and what each function will do.
- The design is complete enough for another developer to pick up and implement.
- Online help is clearly evident
- Processes have been summarized in diagrams. (This is done in the next section!)
- Hardware and software requirements are clearly evident and are correct

Middle mark boundary

- All of the main features have been designed but some omissions have been made.
- Diagrams of the main processes have been produced but may lack detail.
- User has signed to say they have accepted the design
- Data structures have been designed but there may be some problems with normalization or missing fields.

Low mark boundary

- Screen designs have been poorly designed and some major omissions.
- Data structure design may be missing or may lack detail.
- There is some idea of how the system will work but not enough to produce a working system

Top tips

- Make sure you use Word's headings feature. It makes life a lot easier!
- Each form should have it's own section. Use heading 2 or 3 to help lay this out.
- Try and avoid drawing lines on top of screenshots. It tends to look messy!
- Use bullet points for writing validation rules or use a table.
- Do not start the form design until you have had your data structure design checked by your teacher!
- Have a nice clear section where the user can sign. Wait till the end of the project to get it signed as you will most likely be making tweaks!

Algorithms

What is it?

In this section we will focusing on the process / algorithm design. The system must be suitably complex to have at least one algorithm which is NOT linear. That means that it should have some form of loop. This is a very loose definition but hopefully will give an idea of how it should work. Some examples could be –

- Calculate the total cost of salary and companies contribution to NI and pensions.
- Using priorities allow some orders to be processed before others.

Class diagram

You need to create a class for the following items. The next sections will explain what methods and attributes each class will have-

- Every form
- Every database table
- A database controller

You then will add relationships if classes will use each other. For example if one form opens another then there should be a relationship between the two.

Note – Strictly the class diagram is not needed. However it will help design modular code. No real developer starts programming without one! Netbeans allows you to generate code from your class diagrams which will speed up development!

Form classes

Attributes

- Each component will be a private attribute of the class (unless it will not change or perform a function)
- Any data which needs to displayed should have it's own attribute.

Methods

- Any processing which must occur (like sorting a list of names or calculating a total)
- Methods to handle mouse clicks or changing GUI elements (Always passed an EVENT class)
- Draw / paint method – this will most likely be generated by the IDE

GUI elements respond to events. Below are some common events you may wish to deal with. Note if you want something to happen when they are clicked you MUST have a method to deal with it!

- Button – Click event
- Slider / spinner – change value event
- Scroll bar – change event
- Text box – on focus / loose focus / change value event

Database table

Attributes

- All fields in the table must be attributes

Methods

- Insert, edit and delete methods

Database controller

Attributes

- Link to the database or however the data is being stored (it could be a file)

Methods

- A generic query / insert method which accepts SQL as text (only if saved into a database)
- All searches required by the system
- Insert, delete and edit methods.

The idea is that the database table classes will use the database controller to handle the actual saving and loading. The database table classes are effectively a model of the database and are used to make it easier to pass around data from one form to another.

Sequence diagrams

You now need to create a sequence diagram for all of the main algorithms in your system (the main processing). Simple or trivial processing should not be included.

As you have already decided on the methods in your class diagram this section should be straight forward. It also will make the pseudo code VERY easy to produce and also will aid in implementation. Not bad really!

Pseudo code of main algorithms.

If you have done your job correctly with sequence diagrams this will simply be a case of converting the diagrams into pseudo code. You should use the same function calls as specified in the sequence diagrams.

Note – for 5/5 marks you need to do some test runs of your pseudo code. This is best done with trace tables.

Summary of what should be included

- Class diagram of forms
- Class diagram of database (this can be included on the form diagram but may get confusing)
- Sequence diagram for all main algorithms
- Pseudo code of all main algorithms.
- Trace tables of pseudo code to test correctness

Mark boundaries

5 marks	A complete set of algorithms with evidence to show that they have been assessed by the candidate to show that they will meet the design specification. (Evidence should show how these algorithms form a complete solution and that they have been tested for functionality using appropriate techniques.)
3–4 marks	A complete set of detailed algorithms covering the system as specified.
1–2 marks	Some vague algorithms detailing how the system will be developed.

High mark boundary

- You have a complete set of pseudo code listings for each and every main algorithm in the system.
- You have a complete set of sequence diagrams.
- Your pseudo code is correct and performs the required tasks.
- Trace tables show that the pseudo code works.

Middle mark boundary

- You have a complete set of pseudo code listings for each and every main algorithm in the system.
- You have a complete set of sequence diagrams.
- Your pseudo code is correct and performs the required tasks.

Low mark boundary

- Not all algorithms have been included
- Algorithms may not be correct or may be vague

Top tips

- Think very carefully about HOW each of your features will work.
- Your teacher will help you come up with algorithms. Do not be afraid to ask!
- Give your poor old teacher some thinking time as well! It best to catch them outside of lesson.
- You must include pseudo code and not just write it in English!
- You can use flow charts instead of sequence diagrams. However sequence diagrams help you revise UML which is needed for your exams!

Test Strategy

What is it?

Before you start implementing the system you must decide on how to test it. Using your design you need to consider what tests must be run to ensure that the program can be deemed working. It will normally take the form of a table.

Testing layout

All of your tests must be in the following table. You will have one for every form in your system.

Test num	Description of test	Test data	Expected outcome	Actual outcome
1	Ensure that login works for a standard user	**Username** – Bob **Password** - doody	Will take the user to main form (form 1)	

1. **Test num** – Each test is numbered so it can be easily referenced
2. **Description of test** – You say exactly what you are testing and what you are testing for.
3. **Test data** – What must be entered onto the form for this test
4. **Expected outcome** – What should happen
5. **Actual outcome** – left blank for now!

Each form must have its own table like this. To make life easier for you it is best to lay this out in sections. Each form will have its own section and you will include -

- Copy the input form design image
- The testing table

The input form is not needed but it should make things a bit easier.

What should be tested

You need to test the following –

- Main functionality (for example adding a user, calculating correct totals etc)
- Validation rules
- GUI items act correctly
- Extreme test cases
- Abnormal test cases

You will end up with a lot of tests! This is a good thing! There are a lot of marks over the project for testing so although there are only 5 marks here, later on there are many more marks to be had!

Mark boundaries

5 marks	A detailed test strategy and plan covering all aspects of the system with data to test under normal, extreme and abnormal circumstances.
3–4 marks	A detailed test strategy and a plan covering several aspects of the system but with inadequate data to effectively test the system, eg data covers only normal circumstances or covers only a limited part of the design specification.
1–2 marks	A vague discussion of how the system might be tested.

High mark boundary

- All functionality from the requirements specification is tested at least once.
- Each validation rule is tested at least once.
- Each form has extreme and abnormal tests on the MAIN fields only

Middle mark boundary

- All functionality from the requirements specification is tested but there may be some of the smaller functions un-tested
- Most validation rules are tested.
- Some extreme or abnormal testing.

Low mark boundary

- Some testing used on most of the forms.
- Testing will be brief or limited.

Top tips

- Spending time on this section will save time later on!
- Testing carries marks in this section, development and evaluation. It is critical it is done in detail.
- Use separate tables for each form. This makes managing the test much easier!
- Be specific about test data. This makes test repeatable which is very important when tracking down bugs. Random testing is hard to repeat!

Software development

What is it?

This is where you actually make the software! Your selection of project will heavily influence how implementation will occur. However the documentation side of things will all go in a similar fashion.

Order to develop in

A big project needs to be planned out! Below is a VERY rough guide on the order you should do things.

- Database structure
- Test data
- Database controller + database classes
- Form which contains some of the main features
- All other forms in order of most important to least important.

Your forms will most likely be designed but will not do anything. Implementation of these will involve adding the code to get them working!

It does not need to be perfect!

Due to the size of these projects and the potential complexity it is very likely that not all features will be implements and that bugs will exist. This is not a problem and is very normal. As long as you can justify why things have not worked or have not been implemented then you will not be docked marks. Phew!

However for top marks on implementation you do need to complete everything.

Proper coding standards

When coding remember to –

- Use comments!
- Use indentation
- Use functions where possible
- Use sensible variable names
- Follow your class diagram!

Alpha testing

To get the top marks you need to test as you go. You will do this naturally as you will want to make sure what you have done works! However you also need to prove that you have done it! When you complete a section of work run your tests from your test strategy to ensure it works. Then when you document it you will have for EACH form –

- Code listing
- What the code does / additional information
- Test run / alpha test
- Any bugs found

Note – Any bugs you find document! Write down when something goes wrong! This is proof that you are testing your code! If you have no bugs then we will not believe you! No one has no bugs when they program!

How to document

You will have the following sections

Database implementation

- Screen shot of every table
    - Make it clear if any validation rules are implemented on the database
    - Also make it clear which field is the primary key!
- Screen shot of test data (which should be sensible!)
    - You do not need much test data. About 3-5 rows will do!
- Short description of each table. This could be copied from design.

Form implementation

Note – one of these PER form

- Code listing
    - Maintaining the syntax colouring would be helpful but not critical
- What the code does / additional information
    - You can /should reference design here!
- Test run / alpha test
    - Copy your tests for that form and then run them!
- Any bugs found
    - Screenshots help here but are not always necessary. DO NOT include syntax errors!

Mark boundaries

13–16 marks	There is complete evidence showing how the solution was developed using suitable alpha testing at each stage to inform the process. The modular code is fully annotated indicating clearly the purpose of each section and the interrelationship between the sections. The developed solution fulfils all of the design specification.
9–12 marks	Program listings are provided in the form of printouts. Data structures are illustrated as part of the listings where appropriate, detailing their purpose. There is sufficient annotation evident to illustrate how the solution was developed for a particular purpose and indicate the purpose of sections of code. The code will be modular and there will be good evidence to show how testing was used during the development process to inform each stage. The developed solution fulfils the design specification but there are some minor flaws in the solution.
5–8 marks	Program listings are provided in the form of printouts. Data structures are illustrated as part of the listings where appropriate, detailing their purpose. There is some annotation evident to illustrate how the solution was developed and some limited evidence that some testing took place during development. The developed solution has significant flaws and only partially fulfils the design specification. The code may be linear but with some annotation indicating how the code relates to the problem and some limited evidence of alpha testing.
1–4 marks	Program listings are provided in the form of printouts but with no annotation or evidence of alpha testing. The developed solution does not fulfil the design brief

Note – annotation can mean comments!

High mark boundary

- Data structures are clearly evidenced and annotated
- Code is modular (uses functions + classes)
- Code has clear annotation for every function, class and relevant structures
- The code fully fulfils the design brief
- Code has been well tested

Middle mark boundary

- Data structures are clearly evidenced and annotated
- Code is modular (uses functions + classes)
- Code has clear annotation for every function, class and relevant structures
- The code fulfils the design brief but will have minor flaws or omissions
- Code has been tested during development and bugs have been found

Low mark boundary

- Data structures are c evidenced
- Code is annotated but may not be in detail.
- The code only partially fulfils the design brief
- Limited testing evidenced

Very low mark boundary!

- Code listings are evidenced but not annotated (or limited annotation)
- The code does not full fill the design brief

Top tips

- Leave yourself 2-3 months to produce the implementation. It is really important you do not try and rush it!
- If you know you will have difficulties producing a bit of code see your teacher BEFORE you get to it. That way they can point you in the right direction or write some sample code for you. Waiting for help will be a bottle neck you will want to avoid!
- Make sure you document any logic or runtime errors you get. Even if you just jot it down in rough first. It will help you document the code later.
- The internet is your friend! Use it!

Testing

What is it?

This is testing you do once the implementation has been completed. It is broken into two parts

- Alpha testing
- End user testing

Alpha testing

You will have already done some testing already during development. This alpha test is done right at the end of the implementation. To do this you must –

- Copy the test strategy
- Fill out the “actual outcome” column by running each test
- Take screenshots of the main tests to show they worked
- ... or any tests which fail!
- You should not screen shot EVERY test. Just the main features!

When you take a screen shot label it with the test number that it relates to.

End user testing

Your end user should test the system using your test strategy and just using the system for real. They then should write a short statement saying

- They used the system
- What features it implemented which they are happy with
- What problems / bugs they find

It is critical that they are not too nice here! We expect bugs and problems so your user should clearly state any issues they have with it. They then should sign the statement.

Mark boundaries

11–14 marks	The testing covers as many different paths through the system as is feasible, including valid, invalid and extreme cases. The testing covers all aspects of the design specification and the test plan from the design section. There is clear evidence of end-user testing.
8–10 marks	There is evidence of testing covering most aspects of the design specification but with omissions, eg test data does not include erroneous data for all tests or there is limited evidence of end-user testing.
5–7 marks	There is limited evidence of testing based on a badly developed test plan with clear omissions. There is no description of the relationship between the test plan and the testing in evidence.
1–4 marks	A collection of hardcopy test run outputs with no clear link to the test plan and covering few aspects of the system. No evidence of end-user testing.

High mark boundary

Note – if you did a good test strategy then this will be easy marks!

- All main functionality has been tested
- Abnormal and extreme cases have been tested
- All validation rules have been tested
- Clear evidence of end-user testing

Middle mark boundary

- All main functionality has been tested
- Abnormal and extreme cases will not of been tested (or limited testing)
- limited evidence of end-user testing

Low mark boundary

- Limited testing – not all functions are testing
- No end user testing

Top tips

- If you have a test which adds data to the database the proof of the test is that the data is inside your database(!). That means you should screenshot the database rather than the form.

Documentation

What is it?

This section will get you to create a simple user guide. It will also assess your on-screen help which is why it was so important to add it.

Note – If you have failed to complete the system you can still get full marks on this section as long as you base the user guide on the design!

What does it contain?

Note – This should be a standalone document!

- Title page
- Table of contents
- Hardware + software requirements
- Installation guide
- Getting started / step by step guide to main features
- Backup
- Troubleshooting guide

What does each section contain?

Hardware and software requirements

You should have already produced this during analysis so it will be a copy and paste job! You will need to state –

- The minimum specification of the PC which can run your software
    - CPU speed
    - Memory size
    - Hard drive space
    - *Note* – the above will be based on your pre-requisite software such as the Java virtual machine or .net libraries. Or even tomcat or apache.
- Any other specific hardware (you can ignore mouse/keyboard etc)
- Software requirements including
    - Host OS
    - Any interpreters / libraries which need to be installed
    - Any other required software

Ensure you format it into its own section.

Installation guide

It is unlikely you will have made a install wizard for your software (if you have well done!). You need to explain how to install the software onto the target. This includes installing any pre-requisite software first. For example a Java program would need Java virtual machine installed which would need to either be downloaded or included on the install media.

You also must clearly state how the user will get the final product. Will it be on CD or memory stick?

You must say –

- Where any pre-requisite software can be found
- How to install this software
- Where to copy your work to (what needs to be copied)
- How to start it off.

It may be worth making some simple BAT files (or script files) to do this to make the install guide simpler. They are easy to make. Something like

```
cp –r myProjectFolder ~/myProject
```

Would, on linux, copy a folder called “myProjectFolder” and place it in the users directory in a folder called “myProject”. Windows would be something like

```
Copy –r myProjectFolder c:/myProject
```

Note – I do not know if the above windows command is correct. I do not use windows and I am too lazy to test it! It is something like that!

Getting started / step by step guide

Note – Only do the MAIN features otherwise you will be here for a while!

For each of the main features –

- Copy the screen design or final screenshot of that form
- Say how to use it in a step by step manner.
- Comment on expected validation rules.

You will have seen any piece of software or hardware will have a getting started guide. The best way to do this section is to have a look at one of those and emulate it.

Backup

Here you should describe how the system should be backed up and how often. If your system is automated then explain this. Otherwise say exactly what files need to be saved in order to back up the data. A step by step guide with screen shots would be helpful.

Troubleshooting guide

This should be in the form of a table as shown below –

Problem	Symptoms	Solution
Program will not start	• Nothing appears when program launched • Error message stating "unknown command Java"	• Install Java from the install media • Check that you have copied over all files / recopy them
Can not log in	• Error message appears saying "invalid logon"	• Have you got a username or password from your administrator? • Check your password is correct • Ensure caps lock is off • Request a password reset.

As you can see the problems are clearly stated with symptoms and solutions. You do not need to write loads in this section but you do need to be specific!

You should have at least 4-8 entries in this table. Errors can range from unable to add things to the database, problems loading the program or logging in or common validation error messages.

Mark boundary

8–10 marks	Candidates will provide detailed and accurate documentation. The documentation will be well presented, in a structured and coherent format. The documentation will cover all aspects of the system, with no omissions, including installation, typical use, troubleshooting, and backup. The on-screen help and supplementary documentation makes a complete guide to the solution and is well presented and easy to follow. Subject-specific terminology will be used accurately and appropriately. There will be few, if any, errors of spelling, grammar and punctuation.
4–7 marks	Candidates will provide clear documentation. The documentation will be well presented. There is clear on-screen support to enable the end user to use the system. The supporting documentation and on-screen help is well presented and covers most aspects of the system operation with only one or two omissions, eg troubleshooting or backup. Some subject-specific terminology will be used. There may be occasional errors of spelling, grammar and punctuation.
1–3 marks	Candidates will provide superficial documentation, with weak supplementary user documentation covering few aspects of the system. The information will be poorly expressed and limited technical terms will be used. Errors of grammar, punctuation and spelling may be intrusive.

High mark boundary

- Stand alone document with table of contents and title page.
- The document uses clear sections and they are in a logical and coherent order.
- All elements are present.
- On screen help is clear and well presented. The guide and on screen help combined should be sufficient to explain all of the main tasks of the system.
- Clear backup guide
- SPAG

Middle mark boundary

- Stand alone document with table of contents and title page.
- The document uses clear sections and they are in a logical and coherent order.
- Most elements are present but may be missing small sections such as trouble shooting.
- On screen help is clear and well presented. The guide and on screen help explains most of how the system is used.
- SPAG with the occasional error.

Low mark boundary

- Stand alone document
- The document is vague and does not explain the system.
- Most elements are missing or poorly completed.
- On screen help is poor or missing.
- Poor SPAG

Top tips

- Make sure everyone of your main features has a step by step guide.
- Use screenshots to help explain the features.
- Assume the person using your system is a bit thick. Computers seem to make fools out of us all at some point!
- Make sure you do this in a separate document!

Degree of success

What is it?

This is where you explain how well the project has done in detail. As long as you can justify yourself you can still get full marks on evaluation without completing the final system.

Copy the requirements

In order to get full marks you need to state the degree of success for every requirement. The easiest way to do this is to copy the requirements and then make a statement on each one.

You need to state how well it was done and provide evidence to back up your claims.

If it worked…

You need to provide proof via –

- Which tests prove your program works?
- Statement from end user (which comes in the next section or in end user testing)
- Screen shots or designs.

You then need to evaluate the degree of success. Something which works may still have problems. You need to be critical of your own work. For example rather than saying "I can easily add a new customer to the database." You can say "My tests show that a new customer can be added. The form used is intuitive and has clear onscreen guidance to aid the user. However the validation rules could be tightened up a bit as it does not verify if the user has entered a valid email address ."

If it did not work or if it was not completed…

You need to be able to justify why something does not work or was not implemented. Statements such as "I ran out of time" do not cut the mustard! You need to justify why you ran out of time (which obviously cannot be that you left it to the last minute!).

Common reasons include –

- Technical problems learning the language or API.
- What I wanted to do was not possible in the software I chose.
- I had to do it differently / manually.
- Delays in previous technical issues meant that this had to be dropped (low priority)

Notice the last comment does say "I ran out of time" but does so in a way which justifies why.

Mark boundaries

3–4 marks	A full discussion, taking each objective mentioned in **(b) (i)** and explaining the degree of success in meeting them (indicating where in the project evidence can be found to support this), or reasons why they were not met.
1–2 mark	Some discussion about a number of objectives, but some omissions or inadequate explanation of success or failure.
0 marks	No discussion present.

High mark boundary

- Each objective is discussed
- Comments say how WELL a object was completed rather than just saying it was completed
- Evidence to back up claims
- Explanations for objectives not complete are well justified.

Middle mark boundary

- Not all objectives discussed
- Explanations are brief and lack evidence or justification.

Low mark boundary

No discussion present.

Top tips

- Imagine you are in a argument and you have to prove that you had completed a feature. You need as much evidence as you can to back up your claims. This is the essence of evaluation!
- Be critical on your work. You get LESS marks if you say your work has very few problems! All programs have issues.

User response

What is it?

This is acceptance testing. The user will write a statement saying how happy they are with the system and state any issues they may have with the system produced. You again will be assessed on your onscreen help in regards to how user friendly the system is.

What the end user needs to say

- What features they like about the system
- Would they be able to use the system in its current form?
- What problems did they encounter that needs fixing?

Ask your end user to be specific about which features they like or have issues with.

Response to end user statement

You should respond to your user's statement by explaining how the problems that the user encountered could be rectified. You need to explain how these faults could be fixed by giving detailed suggestions. This section will be big or small depending on how many faults there are!

Mark boundary

3 marks	The user indicates that the system could be used but there are some faults which need to be rectified. The candidate provides a detailed discussion of how these inadequacies may be dealt with. OR A fully functional user-friendly system has been produced. The user indicates that the system fully meets the specification given in section **(a)**, and there are no known faults in the system.
2 marks	The system is, in the main, user-friendly, but there is room for improvement (eg no on-screen help has been provided). The user indicates that the system could be used but there are some faults which need to be rectified. The candidate has made some limited attempt to discuss how these inadequacies may be dealt with.
1 mark	The system does not meet the design specification and the end user is not able to make use of the system. The candidate

High mark boundary

The user specifies that there are no faults to the system AND it is clear that a well developed solution has been produced.

OR

The user is overall happy but has identified some faults. These faults are discussed and possible solutions detailed.

Middle mark boundary

- The system is user friendly but may lack onscreen help.
- The user has stated that small problems exist with the developed solution
- These problems are discussed but not in any great detail.

Low mark boundary

The system is unusable by the end user.

Desirable extensions

What is it?

This is more of a classic evaluation. The title of "desirable extensions" is misleading. It is really asking for you to say what the good and bad points are and then state possible improvements.

Good and bad points

You need to state what are the good and bad points of your system. Like in any evaluation you need to be specific and really should focus on specific objectives. So rather than saying "it is easy to use" you can talk about specific forms which are easy to use. What makes them easy to use (like on screen help and validation rules).

The bad points should be areas of weakness noticed by your end user. You MUST NOT talk about objectives which were not implemented. Just focus on what parts of the system you IMPLEMENTED and you will be fine.

Possible extensions

Note – DO NOT say that objectives you have not completed are extensions. They are not!

Extensions are "wish list" ideas. Things which could be done but would of been impossible to complete. Things such as implement a online shopping basket and checkout service or automated fingerprint recognition. Extensions must

- NOT be part of the original specification
- Be desirable and useful (i.e. not just eye candy!)
- Be practical and offer value to the user.

You must also suggest how the extension could be implemented. This includes what hardware and software it would need and a rough idea how it could be implemented. This means that you should not include really complicated extensions!

Mark boundary

3 marks	The candidate clearly portrays the good and bad points of the system indicating the limitations, possible extensions and how to carry out the extensions.
2 marks	The candidate clearly identifies good and bad points and any limitations.
1 mark	The candidate identifies the obvious good points of the system and possibly some bad points or limitations.

High mark boundary

- Well chosen good and bad points have been stated.
- Well chosen extensions with explanations on how to implement the extensions.

Middle mark boundary

- Well chosen good and bad points have been stated.

Low mark boundary

- Has stated the good parts of the system but not many / any bad points.

Appendix A - Deadlines

Date	What must be completed
25/06/2010	Chosen a user and discussed this with Mr Hamflett
09/07/2010	First draft of problem definition
16/07/2010	Second draft of problem definition
10/09/2010	Analysis interview completed and first draft of requirements specification
17/09/2010	Second interview completed and second draft of requirements specification
01/10/2010	Data structure design (with ERD diagrams completed)
15/10/2010	First draft of input design
5/11/2010	Second draft of input design
19/11/2010	First draft of system diagrams (DFD/UML/Flow chart)
3/12/2010	Second draft of system diagrams
17/12/2010	Pseudo code for main algorithms
14/01/2011	Updated pseudo code
28/01/2011	Test strategy including extreme, normal and abnormal tests
11/02/2011	First programming milestone (will be agreed with students)
11/03/2011	Second programming milestone (will be agreed with students)
01/04/2011	All programming completed
22/04/2011	Testing completed
29/04/2011	Documentation completed
06/05/2011	Evaluation and final coursework completed

Above are the deadlines for coursework. These should be considered as the LATEST a piece of work should be completed by. The last few weeks are very tight. Do not fall behind on this coursework! It is too big!

Appendix B – Mark grade summary

Problem definition – Max 3

3 marks	Excellent description with all elements present.
2 marks	Some description of both the stages of study and end user involved.
1 mark	Vague description of the end user or area for development.

Analysis – Max 11

9–11 marks	Excellent user involvement with detailed recording of the user's requirements. All other items must be present, showing a thorough analysis of the system to be computerised. A detailed requirements specification, including full justification for the approach and hardware and software requirements, has been produced.
6–8 marks	Good user involvement and recording of the data collection methods. Most of the necessary items have been covered. However, one or two items have been omitted. A requirements specification is present with some attempt to justify the approach based on the results of the investigations but with some omissions, eg hardware and software requirements.
3–5marks	Some evidence that an attempt has been made to identify the end-user requirements and some recording of it has been made. Attempts at some of the other items have been made. An attempt has been made to develop a requirement specification but with little attempt to justify this based on the results of the investigation.
1–2 marks	Some elements have been discussed but with little or no user involvement.

Nature of solution – Max 6

5–6 marks	A clear set of objectives with a detailed and complete design specification, which is logically correct. There is evidence to show that the end user has seen and agreed these designs. There are also detailed written descriptions of any processes/modules and a clear, complete definition of any data structures. The specification is sufficient for someone to pick up and develop an end result using the software and hardware specified in the requirements specification.
3–4 marks	The major objectives of the new system have been adequately summarised, *but omissions have been made*. There is a brief outline of a design specification, including mock-ups of inputs and outputs, and the process model has been described (including a diagram: structure diagram, data flow diagram or system flowchart). There is some evidence that the end user has seen these designs. However, there is a lack of completeness with omissions from the process model, inputs and outputs. Data structures have been identified but there may be inadequate detail.
1–2 marks	Some vague discussion of what the system will do, with a brief diagrammatic representation of the new system.

Algorithms – Max 5

5 marks	A complete set of algorithms with evidence to show that they have been assessed by the candidate to show that they will meet the design specification. (Evidence should show how these algorithms form a complete solution and that they have been tested for functionality using appropriate techniques.)
3–4 marks	A complete set of detailed algorithms covering the system as specified.
1–2 marks	Some vague algorithms detailing how the system will be developed.

Test strategy – Max 5

5 marks	A detailed test strategy and plan covering all aspects of the system with data to test under normal, extreme and abnormal circumstances.
3–4 marks	A detailed test strategy and a plan covering several aspects of the system but with inadequate data to effectively test the system, eg data covers only normal circumstances or covers only a limited part of the design specification.
1–2 marks	A vague discussion of how the system might be tested.

Implementation – Max 16

13–16 marks	There is complete evidence showing how the solution was developed using suitable alpha testing at each stage to inform the process. The modular code is fully annotated indicating clearly the purpose of each section and the interrelationship between the sections. The developed solution fulfils all of the design specification.
9–12 marks	Program listings are provided in the form of printouts. Data structures are illustrated as part of the listings where appropriate, detailing their purpose. There is sufficient annotation evident to illustrate how the solution was developed for a particular purpose and indicate the purpose of sections of code. The code will be modular and there will be good evidence to show how testing was used during the development process to inform each stage. The developed solution fulfils the design specification but there are some minor flaws in the solution.
5–8 marks	Program listings are provided in the form of printouts. Data structures are illustrated as part of the listings where appropriate, detailing their purpose. There is some annotation evident to illustrate how the solution was developed and some limited evidence that some testing took place during development. The developed solution has significant flaws and only partially fulfils the design specification. The code may be linear but with some annotation indicating how the code relates to the problem and some limited evidence of alpha testing.
1–4 marks	Program listings are provided in the form of printouts but with no annotation or evidence of alpha testing. The developed solution does not fulfil the design specification. There is some evidence of system development.

Testing – Max 14

11–14 marks	The testing covers as many different paths through the system as is feasible, including valid, invalid and extreme cases. The testing covers all aspects of the design specification and the test plan from the design section. There is clear evidence of end-user testing.
8–10 marks	There is evidence of testing covering most aspects of the design specification but with omissions, eg test data does not include erroneous data for all tests or there is limited evidence of end-user testing.
5–7 marks	There is limited evidence of testing based on a badly developed test plan with clear omissions. There is no description of the relationship between the test plan and the testing in evidence.
1–4 marks	A collection of hardcopy test run outputs with no clear link to the test plan and covering few aspects of the system. No evidence of end-user testing.

Documentation – Max 10

8–10 marks	Candidates will provide detailed and accurate documentation. The documentation will be well presented, in a structured and coherent format. The documentation will cover all aspects of the system, with no omissions, including installation, typical use, troubleshooting, and backup. The on-screen help and supplementary documentation makes a complete guide to the solution and is well presented and easy to follow. Subject-specific terminology will be used accurately and appropriately. There will be few, if any, errors of spelling, grammar and punctuation.
4–7 marks	Candidates will provide clear documentation. The documentation will be well presented. There is clear on-screen support to enable the end user to use the system. The supporting documentation and on-screen help is well presented and covers most aspects of the system operation with only one or two omissions, eg troubleshooting or backup. Some subject-specific terminology will be used. There may be occasional errors of spelling, grammar and punctuation.
1–3 marks	Candidates will provide superficial documentation, with weak supplementary user documentation covering few aspects of the system. The information will be poorly expressed and limited technical terms will be used. Errors of grammar, punctuation and spelling may be intrusive.

Degree of success – Max 4

3-4 mark	A full discussion, taking each objective mentioned in **(b) (i)** and explaining the degree of success in meeting them (indicating where in the project evidence can be found to support this), or reasons why they were not met.
1–2 mark	Some discussion about a number of objectives, but some omissions or inadequate explanation of success or failure.
0 marks	No discussion present.

Users response – max 3

3 marks	The user indicates that the system could be used but there are some faults which need to be rectified. The candidate provides a detailed discussion of how these inadequacies may be dealt with. OR A fully functional user-friendly system has been produced. The user indicates that the system fully meets the specification given in section **(a)**, and there are no known faults in the system.
2 marks	The system is, in the main, user-friendly, but there is room for improvement (eg no on-screen help has been provided). The user indicates that the system could be used but there are some faults which need to be rectified. The candidate has made some limited attempt to discuss how these inadequacies may be dealt with.
1 mark	The system does not meet the design specification and the end user is not able to make use of the system. The candidate briefly discusses these issues in terms of their project management.

Extensions – Max 3

3 marks	The candidate clearly portrays the good and bad points of the system indicating the limitations, possible extensions and how to carry out the extensions.
2 marks	The candidate clearly identifies good and bad points and any limitations.
1 mark	The candidate identifies the obvious good points of the system and possibly some bad points or limitations.