

---

SCHOLAR Study Guide

# **SQA Advanced Higher Computing**

## **Unit 2: Developing a Software Solution**

---

**David Bethune**

Heriot-Watt University

**Andy Cochrane**

Heriot-Watt University

**Ian King**

Heriot-Watt University

**Interactive University**

Edinburgh EH12 9QQ, United Kingdom.

First published 2005 by Heriot-Watt University

This edition published in 2006 by Interactive University

Copyright © 2006 Heriot-Watt University

Members of the SCHOLAR Forum may reproduce this publication in whole or in part for educational purposes within their establishment providing that no profit accrues at any stage, Any other use of the materials is governed by the general copyright statement that follows.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, without written permission from the publisher.

Heriot-Watt University accepts no responsibility or liability whatsoever with regard to the information contained in this study guide.

SCHOLAR is a programme of Heriot-Watt University and is published and distributed on its behalf by Interactive University.

**British Library Cataloguing in Publication Data**

Interactive University

SCHOLAR Study Guide Unit 2: Computing

1. Computing I. Developing a Software Solution

ISBN 1 904647 86 3

Typeset by: Interactive University, Wallace House, 1 Lochside Avenue, Edinburgh, EH12 9QQ.

Printed and bound in Great Britain by Graphic and Printing Services, Heriot-Watt University, Edinburgh.

**Part Number 2006-1379**

## Acknowledgements

Thanks are due to the members of Heriot-Watt University's SCHOLAR team who planned and created these materials, and to the many colleagues who reviewed the content.

**Programme Director:** Professor R R Leitch

**Series Editor:** Professor J Cowan

**Subject Directors:** Professor P John (Chemistry), Professor C E Beevers (Mathematics), Dr P J B King (Computing), Dr P G Meaden (Biology), Dr M R Steel (Physics), Dr C G Tinker (French)

**Subject Authors:**

**Biology:** Dr J M Burt, Ms E Humphrey, Ms L Knight, Mr J B McCann, Mr D Millar, Ms N Randle, Ms S Ross, Ms Y Stahl, Ms S Steen, Ms N Tweedie

**Chemistry:** Mr M Anderson, Mr B Bease, Dr J H Cameron, Dr P Johnson, Mr B T McKerchar, Dr A A Sandison

**Computing:** Mr I E Aitchison, Dr P O B Holt, Mr S McMorris, Mr B Palmer, Ms J Swanson, Mr A Weddle

**Engineering:** Mr J Hill, Ms H L Jackson, Mr H Laidlaw, Professor W H Müller

**French:** Mr M Fermin, Ms B Guenier, Ms C Hastie, Ms S C E Thoday

**Mathematics:** Mr J Dowman, Ms A Johnstone, Ms O Khaled, Mr C McGuire, Ms J S Paterson, Mr S Rogers, Ms D A Watson

**Physics:** Mr J McCabe, Mr C Milne, Dr A Tookey, Mr C White

**Learning Technology:** Dr W Austin, Ms N Beasley, Ms J Benzie, Dr D Cole, Mr A Crofts, Ms S Davies, Mr A Dunn, Mr M Holligan, Dr J Liddle, Ms S McConnell, Mr N Miller, Mr N Morris, Ms E Mowat, Mr S Nicol, Dr W Nightingale, Mr R Pointon, Mr D Reid, Dr R Thomas, Dr N Tomes, Ms J Wang, Mr P Whitton

**Cue Assessment Group:** Ms F Costigan, Mr D J Fiddes, Dr D H Jackson, Mr S G Marshall

**SCHOLAR Unit:** Mr G Toner, M G Cruse, Ms A Hay, Ms C Keogh, Ms B Laidlaw, Mr J Walsh

**Media:** Mr D Hurst, Mr P Booth, Mr G Cowper, Mr C Gruber, Mr D S Marsland, Mr C Nicol, Mr C Wilson

**Administration:** Ms L El-Ghorr, Dr M King, Dr R Rist,

We would like to acknowledge the assistance of the education authorities, colleges, teachers and students who helped to plan the SCHOLAR programme and who evaluated these materials.

Grateful acknowledgement is made for permission to use the following material in the SCHOLAR programme:

To the Scottish Qualifications Authority for permission to use Past Papers assessments.

The financial support from the Scottish Executive is gratefully acknowledged.

All brand names, product names, logos and related devices are used for identification purposes only and are trademarks, registered trademarks or service marks of their respective holders.



---

# Contents

<b>1</b>	<b>The Advanced Higher Project</b>	<b>1</b>
1.1	Introduction . . . . .	2
1.2	Unit, Project and Course . . . . .	2
1.3	Record Keeping - the Record of Work . . . . .	3
1.4	Choosing a suitable project . . . . .	4
1.5	Group Projects . . . . .	6
1.6	Making up your mind . . . . .	7
1.7	Summary . . . . .	8
1.8	Revision Questions . . . . .	9
<b>2</b>	<b>Analysis</b>	<b>11</b>
2.1	Prior Knowledge and Revision . . . . .	12
2.2	Introduction . . . . .	13
2.3	The Project Specification . . . . .	13
2.4	Project Planning . . . . .	17
2.5	Research . . . . .	22
2.6	Selecting strategies . . . . .	23
2.7	Resources . . . . .	25
2.8	Summary . . . . .	26
2.9	Revision Questions . . . . .	27
<b>3</b>	<b>Design</b>	<b>29</b>
3.1	Prior Knowledge and Revision . . . . .	30
3.2	Introduction . . . . .	31
3.3	User Interface Design . . . . .	31
3.4	Design Methodologies . . . . .	33
3.5	Design Notations . . . . .	33
3.6	Summary . . . . .	37
3.7	Revision Questions . . . . .	37
<b>4</b>	<b>Implementation</b>	<b>39</b>
4.1	Prior Knowledge and Revision . . . . .	40
4.2	Introduction . . . . .	41
4.3	Creating a test plan . . . . .	41
4.4	Implementing your design . . . . .	43
4.5	Keeping your Record of Work . . . . .	44
4.6	Summary . . . . .	44
<b>5</b>	<b>Testing</b>	<b>45</b>
5.1	Prior Knowledge and Revision . . . . .	46

5.2	Introduction . . . . .	47
5.3	Carrying out your test plan . . . . .	47
5.4	Evidence of Testing . . . . .	48
5.5	Rectifying Errors and Bugs . . . . .	49
5.6	A User Questionnaire . . . . .	49
5.7	Summarising your results . . . . .	51
5.8	Summary . . . . .	52
<b>6</b>	<b>Project Report</b>	<b>53</b>
6.1	Prior Knowledge . . . . .	54
6.2	Introduction . . . . .	54
6.3	Evidence required for Unit Assessment . . . . .	54
6.4	Evidence required for Course Assessment . . . . .	56
6.5	User and Technical Documentation . . . . .	58
6.6	Evaluation Report . . . . .	59
6.7	Summary . . . . .	61
	<b>Glossary</b>	<b>62</b>
	<b>Answers to questions and activities</b>	<b>63</b>
1	The Advanced Higher Project . . . . .	63
2	Analysis . . . . .	65
3	Design . . . . .	66
4	Implementation . . . . .	67
5	Testing . . . . .	68

---

# Topic 1

## The Advanced Higher Project

---

### Contents

1.1	Introduction . . . . .	2
1.2	Unit, Project and Course . . . . .	2
1.3	Record Keeping - the Record of Work . . . . .	3
1.4	Choosing a suitable project . . . . .	4
1.5	Group Projects . . . . .	6
1.6	Making up your mind . . . . .	7
1.7	Summary . . . . .	8
1.8	Revision Questions . . . . .	9

### ***Learning Objectives***

*After Studying this topic, you should be able to:*

- *describe how the project fits into the Advanced Higher Computing course*
- *explain what evidence you need to produce for assessment*
- *describe the problem for which you are going to develop a software solution*

## 1.1 Introduction

You may have already begun the Advanced Higher Computing course, probably working on the Software Development unit, or you may be right at the start.

This Scholar unit will help you to work your way through the Coursework project that you have to do as part of your course. It is quite unlike any other units that you have studied so far in Computing. Instead of having to learn and understand lots of new facts, concepts and skills, this unit is about you using the knowledge and skills you already have, to analyse a problem, then design, implement and test a software solution to that problem.

In this first Topic, you will discover how the project fits into the course, what records and evidence you will be required to produce, and then you will choose a suitable problem to tackle.

## 1.2 Unit, Project and Course

The Advanced Higher Computing course, like all other courses, is made up of 2 mandatory units and a choice of one out of 3 optional units:

Mandatory Units	Optional Units
Software Development	Computer Architecture
Developing a Software Solution	Artificial Intelligence
	Computer Networking

To complete the course, you need to:

- pass each unit
- pass the course assessment

The course assessment is in 2 parts:

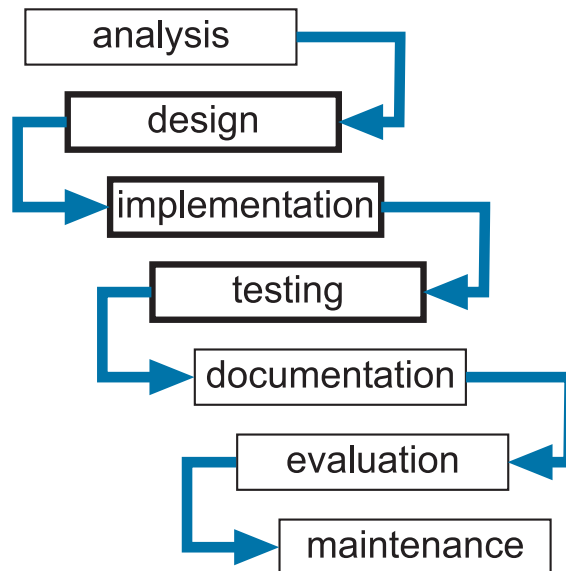
- a final exam on all 3 units, worth 120 marks
- a report on your coursework project, worth 80 marks.

As you work through this unit, you will

- a) learn about managing a project,
- b) choose a problem for your own project
- c) complete the project
- d) generate evidence for the practical Outcome for this unit,

- e) produce a project report for the course assessment

There are several stages to work through in any software development project. You know them already:



You will work your way through each of these stages as you follow these notes.

### 1.3 Record Keeping - the Record of Work

As you work your way through this unit, you should keep a record of work.

This is important both to keep track of what you have done, and because it will be required as evidence for assessment. If you don't keep a record of work, you could fail the unit and also get very low marks for the coursework.

So, what format should the record of work take?

It is probably best to use an A4 loose-leaf folder, with dividers for the different sections.

The sections required to pass the unit will be the first 4 stages of the development process: analysis, design, implementation and testing.

The next 2 stages - documentation and evaluation are required for the course assessment. Maintenance isn't assessed, so you can leave it out.

#### Preparing your Record Of Work folder

- Get a loose-leaf folder.
- Obtain or make a set of dividers.
- Label the dividers with the 6 stages.
- Insert them into your folder.



15 min

As you work through your project, you will gradually fill up the Record of Work folder. Into it, you will insert notes and diagrams on analysis and design, listings of program code, screen dumps and tables of testing, and many other items.

It is a good idea to get into the habit of putting the date and your name (or initials) on every page that you put into your Record of Work.

The Record of Work is not meant to be a work of art! You don't need to spend time on making it look pretty, or rewriting or typing rough notes. It should simply contain the actual working documents you create as you go along.

At the end of the project, you will be required to create a more formal report covering some aspects of your work, but you don't need to be concerned about that at this stage.

Don't make the mistake of thinking that the formal report is all that is required. Your assessor may want to see the Record of Work, so it is important that you maintain it, and keep it up to date.



Figure 1.1:

## 1.4 Choosing a suitable project

The first thing you need to do is to choose a suitable project.

Your project can be based on any computing problem to which you can design and implement a software solution.

It can be on **any** computing problem at all, but it should:

- be at an appropriate level for Advanced Higher
- build on learning from the mandatory units
- be achievable within 40 hours.

### What does "at an appropriate level" mean?

It means that the problem you choose to tackle couldn't simply be solved by using standard coding and algorithms that you learned for Higher Computing. You will need

to choose something that requires more complex ideas, coming from what you have already learned during the Advanced Higher course, or from research that you carry out as part of your project.

For example, in the Software Development unit in Advanced Higher, you have learned about

- file handling
- sort algorithms
- 2-d arrays

You should choose a project which includes at least one of these constructs.

Your tutor will advise you about whether or not your ideas would be appropriate.

### Brainstorming Project Ideas

Discuss some possible ideas for projects. Write them all down as you think of them.



15 min

### Considering Possible Project Ideas

Consider each possible project topic below, and discuss whether or not it might be suitable with your teacher and other students.



15 min

You can check your answers at the end

#### Q1:

1. a simulation of a simple card game with a graphical user interface and an element of artificial intelligence developed using a high level programming language;
2. a simulation of a simple card game using only code modules easily available in the public domain;
3. A simulation of a multi-player complex card game, with realistic graphics, able to be played over a local area network.
4. an encryption / decryption program using only a simple scrambling algorithm such as one which would be within the reach of a Higher candidate;
5. an encryption / decryption program involving file handling and complex key algorithms;
6. an implementation of a natural language interpreter based on a very limited vocabulary and simplistic approach to grammar rules;
7. an implementation of a natural language interpreter using a declarative language and based on a recognised grammar and parsing technique;
8. an implementation of a natural language interpreter using a declarative language and based on a recognised grammar and parsing technique, and which can respond effectively to any input.
9. A computer aided learning package using only simple multimedia authoring tools;
10. A computer aided learning package using some scripting and saving user data to a filing system.

11. a computer aided learning package which demonstrates a simple assembly language emulator for teaching the computer systems unit and incorporating a number of commonly used op-codes and addressing modes;
12. a network application to allow peer-to-peer chat facilities;
13. a web site which does not include advanced techniques such as scripting;
14. a web site which incorporates advanced techniques such as the development of user interactivity, form filling or a front end to a database system making extensive use of PHP, Java or PERL;
15. an expert system for a careers database which does not include advanced rules or other constructs available within an expert system shell.
16. an expert system developed without the aid of an expert system shell.

If you consider that these are all a bit too complicated and daunting, here are a few more ideas that you could use, all of which meet the criteria for an AH Coursework Project.

- a lottery program, which selects 7 random numbers, sorts them into order and displays them graphically
- a multiple choice test which records each user's score and saves them to a file
- any simple game, which generates a player's score, allows this to be saved to a file, then displays a sorted high score table
- an implementation of "Who wants to be a millionaire", using questions and answers stored in data files, and a 2-d array to store the amounts won after each question
- a simple implementation of "Eliza" using data files to store vocabulary



15 min

### **Narrowing down your possible projects**

Now return to your list of possible projects that you "brainstormed" earlier. Go through them one by one and consider whether they are suitable. You may need to adapt them - either adding some complexity, or reducing an over-ambitious idea.

Now you need to go away and think about it! Which of the ideas you have considered interests you the most? Is it possible within 40 hours? Is it complex enough, but not so complex that it is unrealistic? Do you have access to suitable hardware and software?

## **1.5 Group Projects**

It is possible to attempt a group project rather than an individual one.

There are some advantages - you can share and discuss ideas with others in the group.

There are also some disadvantages - you have to be able to divide up the problem so that your tutor can easily identify and assess each person's contribution to the group. There is also the risk that if one person is not "pulling their weight" or drops out midway through the project, it might affect everyone else's progress.

If you think a group project might be possible, you will need to discuss it very carefully with your tutor and the others in the group before you begin.



## 1.7 Summary

- You have begun to manage and carry out a software development project
- Your project will be assessed - it is needed to pass the unit, and it also contributes 40% of your overall award (A, B, C or D) for the course.
- You should have chosen a project which is complex enough for Advanced Higher, but achievable within 40 hours and with the hardware and software that you have available.
- You should have started to keep a Record of Work throughout the project.

## 1.8 Revision Questions

**Q2:** To pass Advanced Higher Computing:

- a) you must pass 3 units, complete a project and sit an exam
- b) you must complete 5 units
- c) you can choose any 3 out of 5 units
- d) you must complete 2 mandatory units, and either a project or an optional 3rd unit

**Q3:** The project:

- a) is only required to pass the unit, it doesn't count for the course award
- b) is required to pass the unit and counts 50% of the course award
- c) is required to pass the unit and counts 40% of the course award
- d) must be based on one of the optional units

**Q4:** The topic for your project:

- a) must be chosen from a list published by the SQA
- b) is your own choice, but it must be suitably complex
- c) will be decided by your teacher
- d) is your own choice, and there are no limits to your choice

**Q5:** A Record of Work

- a) is an optional extra that you might want to produce
- b) is something that you put together after you have completed your project
- c) is useful to you, but not required for assessment
- d) will help you keep track of progress, and must be maintained for assessment

**Q6:** Your project topic must be based on

- a) a software solution to a computing problem
- b) the high level language that you used for the Software Development unit
- c) an investigation of some area of Computing hardware that interests you
- d) a problem which has never been solved before



---

## Topic 2

# Analysis

---

### Contents

2.1	Prior Knowledge and Revision . . . . .	12
2.2	Introduction . . . . .	13
2.3	The Project Specification . . . . .	13
2.3.1	Statement of requirements / project proposal . . . . .	14
2.3.2	Scope and boundaries . . . . .	15
2.3.3	Functional requirements . . . . .	16
2.3.4	Specification Review . . . . .	17
2.4	Project Planning . . . . .	17
2.4.1	Sub-tasks . . . . .	18
2.4.2	Time planning . . . . .	20
2.4.3	Monitoring and management . . . . .	21
2.5	Research . . . . .	22
2.6	Selecting strategies . . . . .	23
2.7	Resources . . . . .	25
2.8	Summary . . . . .	26
2.9	Revision Questions . . . . .	27

### Learning Objectives

After studying this topic, you should be able to:

- describe the main elements of the analysis stage of the software development process
- describe and create a problem specification
- explain the terms scope and boundaries, and functional requirements
- describe and create a project plan
- explain the importance of identifying sub-tasks, setting a realistic time-scale and applying appropriate project management techniques
- carry out research and select appropriate strategies for a project

## 2.1 Prior Knowledge and Revision

You should already know the seven stages of the software development process: Analysis - Design - Implementation - Testing - Documentation - Evaluation - Maintenance:

**Analysis** is the first stage, and involves clarifying exactly what is required. Often this means beginning with a vague problem description or project proposal, and by applying various techniques and activities, turning it into a precise program specification. This is agreed between the client and the developer before any further work is done on the project.



### Revision

**Q1:** The first 3 stages of the software development process, in order, are:

- a) Analysis, Design, Implementation
- b) Design, Analysis, Implementation
- c) Documentation, Evaluation, Maintenance
- d) Analysis, Implementation, Testing

**Q2:** The purpose of analysis is to:

- a) turn a vague program specification into a precise problem description
- b) design the overall structure of the program
- c) turn a vague problem description into a precise program specification
- d) decide whether or not a program needs to be developed

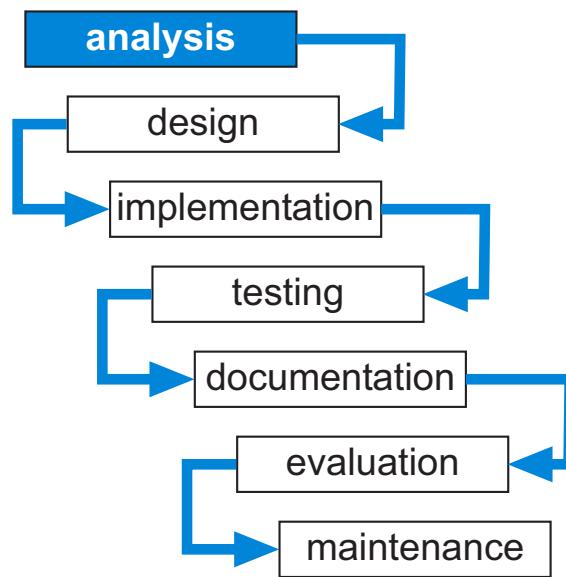
**Q3:** The program specification is:

- a) written by the client
- b) agreed between the client and the developer
- c) written by the developer
- d) the starting point of the analysis stage

**Q4:** Careful analysis of a problem:

- a) is only required for major software development projects
- b) is only used by learners - real program developers jump straight to the design stage
- c) is useful to you, but not required for assessment
- d) should be the starting point for any software development, large or small.

## 2.2 Introduction



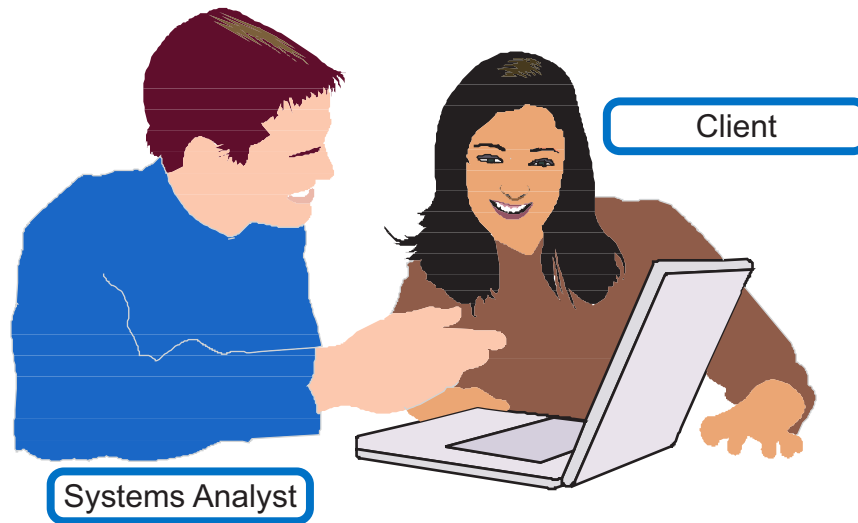
In this Topic, we will develop the ideas you already know about the analysis stage of the software development process. You will learn more about what happens during the analysis stage, leading to a precise program specification. You will also learn about project management, and the various important aspects of a project plan.

You will then apply this to your own Advanced Higher Computing project. As you do so, you will produce items of evidence to file in your Record of Work.

## 2.3 The Project Specification

The purpose of the analysis stage is to develop a precise program or **project specification**. The starting point will usually be either an existing system which is to be improved in some way, an outline project proposal, or a rough description of a new system required by a customer or client.

The task of refining this into a precise specification is carried out by a **systems analyst** working closely with the client. As you already know, the systems analyst (and his/her team) will use a variety of techniques to clarify the original proposal, until a precise specification can be agreed between the analyst and the client.

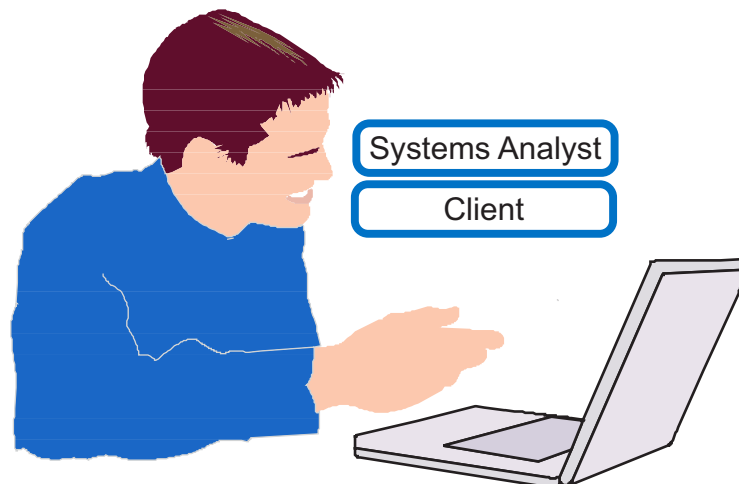


This precise specification should include:

- A clear and detailed statement of the scope and boundaries of the problem / project
- A list of the precise functional requirements

We will consider these in detail next.

In the context of your own Computing project, you will be both the client and the analyst. You have already written down a project proposal, which gives a rough description of what your project will be about. Your next task will be to clarify that into a specification.



But first, we should take a closer look at what the specification should include.

### 2.3.1 Statement of requirements / project proposal

This is what you already have: the initial statement of requirements from the client. It is enough to get the analysis process started, but not nearly detailed enough to begin design work. In a "real-life" scenario, it will lead to many meetings between the client and the systems analysts, and may involve a **feasibility study** to ensure that the program / project is possible within the client's budget and time constraints.

For your project, this will have taken the form of a discussion between yourself and your tutor. Your tutor may have had to point out to you that your original idea was too ambitious, or that the required resources were not available, or perhaps that you could extend your idea into something more interesting or challenging.

Now it is time to turn your project proposal into a project specification

### 2.3.2 Scope and boundaries

It is very important at the outset to establish clearly the scope and boundaries of the project. Scope and boundaries are opposite sides of the same "coin". Between them, they give a precise description of the extent of the project.

Here is a typical statement about scope and boundaries. You will find similar statements by searching the web. "The project scope states what will and will not be included as part of the project. Scope provides a common understanding of the project for all stakeholders by defining the project's overall boundaries."

One way of thinking about it is:

- the scope clarifies what the project must cover
- the boundaries clarify what the project will not cover.

For example, suppose your project was to develop an expert system giving students guidance on job opportunities which they should consider after graduating from University.

The scope of the project would be to create an expert system. Then it would be necessary to describe the range of jobs and degrees that would be included in the system, the level of information that would be output by the system (does it suggest contact addresses as well as simply job types), the types of questions that the user will be asked. Does it cover all degrees, or is it only for students with Computing Science degrees, and so on ... All these things will define the **boundaries** of the system.

Sometimes it is also helpful to spell out exactly what will NOT be covered. So, for example, a clear statement could be made which states that the system will NOT cover advice on jobs for those with medical and veterinary degrees, or jobs overseas.

The scope and boundaries could also refer to technical issues. For example, they might state that the resultant system will run on any computer capable of running any version of Windows after Windows 3.1, but not on any other operating system.

#### **Why is it important to clarify the exact scope and boundaries?**

**Real world answer:** Proper scope definition is critical to a project's success. It establishes the boundaries of what the project will and will not accomplish. The scope statement eliminates any confusion or ambiguity that might still exist after considering the project's goal, objectives and high-level deliverables statements. Poorly defined scope leads to "scope creep", which means that the project's objectives change as it progresses. These changes inevitably lead to increased work effort, which in turn causes project delays, cost overruns, poor team morale and/or customer dissatisfaction.

**AH project answer:** Proper scope definition is essential to ensure that you embark on a realistic project. If you don't define the scope and boundaries, you won't know when





There are 3 important aspects to a project plan. These are:

- dividing the overall task into manageable sub-tasks
- estimating realistic times for carrying out these tasks
- setting up a system for monitoring progress and managing your time

We will look at each of these separately.

### 2.4.1 Sub-tasks

In "real life" software development projects, this stage not only requires identifying the different tasks to be carried out, but also allocating these task to members of the project team, or to groups within the team. Unless you are attempting a collaborative project, you will be doing all of these yourself.

As a first step, you can divide the project up into sub-tasks corresponding to the 7 stages of the software development process:

1. **Analysis** - already done
2. **Design**
3. **Implementation**
4. **Testing**
5. **Documentation**
6. **Evaluation**
7. **Maintenance** - not required for Advanced Higher

Almost certainly, you will need to add in **research** at an early stage - probably before the design stage. All the stages down to testing are required for the unit assessment. Documentation and evaluation will be included in the project report for course assessment.

So the list now looks like:

1. **Analysis** - already done
2. **Research**
3. **Design**
4. **Implementation**
5. **Testing**
6. **Project Report**

The next stage is to break these down into smaller tasks. These will depend on the specific nature of your project. Here are some ideas which might be appropriate:

**Research**

- investigating different ways of implementation (e.g. different programming languages or software)
- finding out how to implement (whatever)
- looking at similar products and considering the best approach
- selecting a strategy / programming language / software development environment

**Design**

- drawing sketches of the user interface
- creating dataflow and structure diagrams
- writing pseudocode for modules

**Implementation**

- creating user interface forms
- creating test data and a test plan
- coding main structure
- coding section 1
- coding section 2
- etc ...
- module testing and debugging

**Testing**

- component testing
- acceptance testing

**Project Report**

- collating evidence from record of work
- writing user documentation
- writing technical documentation
- writing an evaluation report



30 min

### Listing Project sub-tasks

Create a table - with 5 columns.

Label the columns (from left to right) **sub-task**, **time**, **target date**, **completed**, **comments**.

In the left hand column, list all the sub-tasks for your project, in the order you would expect to carry them out. It should look like this:

Sub-task	Time	Target Date	Completed	Comment
Project proposal				
Specification				
Research				
...				
...				
etc				

Get your tutor to check this before you go ahead.

### 2.4.2 Time planning

Now that you have mapped out the work that is in front of you, you need to think about time-planning.

For a real-life software development project, this is absolutely vital, as the main cost for most projects is staff time. Suppose a project is being tackled by a team of 25 analysts, programmers and other staff, on an average monthly pay of £2000. The project plan estimates that the project will take 2 years to complete. The staffing costs will be £2000 x 25 staff x 24 months = £1.2 million. The project manager (from past experience!) decides to build in a 20% allowance for project slippage, so goes to the finance director asking for a staffing budget of £1.4 million.

Unfortunately, for a variety of reasons, the project takes 8 months longer than expected, even with 4 new team members added during the last 8 months to speed thing up. The final staffing costs come to £1.7 million, an overspend of £300,000. In addition, the client claims a £500,000 discount for failure to complete the project on time, so the company ends up almost £1 million worse off than expected.

In your case, no-one is paying you for doing your Advanced Higher project. However, if your time-planning is unrealistic, you may lose vital marks as you may run out of time to complete the work.

So, now it is time to plan your time!



30 min

### Estimating Project Timescale

You have a table listing all the sub-tasks.

**Stage 1:** Go down this table, and estimate the time required for each sub-task and enter it in the second column. The times for the first 2 items will not be estimates, as you have already done them!



After a while it will start to look like this:

Table 2.1:

Sub-task	Time	Target Date	Completed	Comment
Project proposal	2 hours	24 Aug	24 Aug	
Specification	2 hours	26 Aug	26 Aug	
Research into ...	6 hours	4 Sept	9 Sept (partly)	Most done, wrote off for more info
Selecting Method	1 hour	5 Sept	10 Sept	
Design of user interface	2 hours	7 Sept		
etc				

In this example, the project is already behind the time plan by a few days, and the research has not been completed.

Does it matter?

It depends!

You should have built in some extra time for unforeseen difficulties. If so, you should be able to catch up later. Keep an eye on any slippage, though, so that it doesn't become unmanageable. If you start to get too far behind, then you will need to take corrective action. That could mean spending some extra time for a couple of weeks to catch up, or discussing with your tutor whether you need to amend your proposal. The action you take will depend on your situation and the reasons for any delay.

On the other hand, you might find you are ahead of schedule. Good! But don't become complacent. Something later on might hold you up, so you will be glad to have some time in hand.

Monitoring and managing are very important. Use the table for monitoring. Don't cheat. Be honest with yourself. And use the information to make good management decisions.

## 2.5 Research

Research is not meant to be a major part of your AH project, but you will almost certainly need to do some. The types of information you may need to uncover could include:

- a programming technique that you will need
- comparing and selecting an item of hardware to buy
- searching a module library for an existing module you could use
- finding out the needs of potential users of your system
- considering alternative programming languages which you could use

- finding out if a similar project has been undertaken

There are many other possibilities. The methods you use will depend on the particulars of your project. Some important things to remember include:

- be methodical about your research
- keep records of where you look and what you find
- don't waste time

## 2.6 Selecting strategies

Once you have carried out research, you will be ready to make some decisions about strategies you will use. It is easy to rush straight on with the project without thinking carefully about the best way forward.

For example, if your project requires some high level language programming, you will have to decide which programming language and environment is most appropriate to use.

If your project involved creating web pages, you will need to make a decision about which web page editor to use, or whether to code directly in HTML.

You may have to decide whether to implement your solution by developing software from scratch using a high level language, or by adapting an existing application using a scripting language.

Whatever decisions you have to make, you must be able to justify them. That is, you have to be able to say, "I considered X, Y and Z, and chose Y because ..." . Such decisions should be based on sensible criteria.

Here is an example of using criteria to select a strategy - in this case, a choice is to be made between Hyperstudio (a multimedia authoring tool) and 2 high level languages (Visual BASIC and C++)

This first-pass analysis leaves you with no options. So, there is now a need to do some further research to identify if there are other options that might be considered. If this still leaves you with no options, you should discuss this with your tutor and either select a different project or build time into your plan to allow for a learning period.

Criteria	Hyperstudio	Visual BASIC	C++
<b>Development tools</b>	Supports drag and drop for the interface. A text editor is used in enter Hyperlogo scripts.	Visual Basic has a number of sophisticated editing tools. Code can be formatted and different colours can be used to highlight remarks	Extensive text editing and debugging tools are available. Modules can be linked to provide ease of adding and removing different sections of code.
<b>Documentation</b>	Manual plus extensive on-line support via the HyperStudio web site.	Very limited printed documentation but the on-line library is extensive.	The manuals are not available in school and the on-line documentation is extremely difficult.
<b>Previous experience</b>	I have had no previous experience of using HyperStudio and so would have to spend a lot of time learning how to use the package and how to write code in the scripting language.	Visual Basic has only just become available in the school and I am a bit worried about how well I can use something that has not been used before in school.	I would like to learn to use C++ as it would be very useful in the future. The main problem is that I have never used it before and would need to spend a lot of time

The final choice:

Criteria	Hyperstudio	Visual BASIC	C++
<b>Viability: Which environment will I use?</b>	HyperStudio has many advantages but the scripting language will take too long to learn for this project.	Although I would like to use Visual Basic I feel that it is not a good idea due to the fact that we have only just got it in the school and there will be a lot of learning involved to get to a stage where I can do this project.	Not a viable option due to the lack of resource materials and my lack of experience with this type of language.

## Selecting a strategy

If you have some important decisions to make about the way you will solve your problem:



30 min

- list the options
- decide on the criteria
- use the criteria to evaluate the options - a table like the one above is a good idea
- make a decision which you can justify

Note: you may find that you have to go back to do some more research to complete this activity.

File your table of criteria and your decision on your Record of Work. Keep any research notes and file them in your Record of Work, which should include:

Table Of Contents	
	<u>TitlePage</u>
	<u>Project Proposal</u>
	<u>Scope &amp; Boundaries</u>
	<u>Functional Requirements</u>
	<u>Time Plan</u>
	<u>Selecting a Strategy</u>
	<u> </u>
	<u> </u>
	<u> </u>
	<u> </u>
	<u> </u>
	<u> </u>

## 2.7 Resources

In "real life" projects, there are also usually resource management issues to consider. The project manager will have a set budget within which to work. It is important to consider, at the start, all the resources which may need to be purchased, and ensure that there are enough funds to buy them all. Where new resources are required, consideration must be made of when they will be required, so that they can be ordered in advance. Some resources may take several weeks between ordering and delivery, so it is vital that they are ordered early so that the project is not delayed.

This might apply to your project too. Use these questions as a checklist:

- Do I already have everything I will need?
- Do I need extra software?
- Do I need extra hardware?
- Can I afford to buy these?

- How long will they take to arrive?
- How long will it take to get new software installed?



### Making a resource list

Make a list of all the resources you will require. Include:

- Hardware
- Software
- Other resources

Alongside each item, put a tick if you have it already.

For each item without a tick, decide what you are going to do, and note it down as an action point.

Tick off these action points as you complete them.

File this in your Record of Work, and keep it up to date.

Your Record of Work should now include:

Table Of Contents	
	<u>TitlePage</u>
	<u>Project Proposal</u>
	<u>Scope &amp; Boundaries</u>
	<u>Functional Requirements</u>
	<u>Time Plan</u>
	<u>Selecting a Strategy</u>
	<u>Resource List</u>

## 2.8 Summary

- The main elements of the analysis stage of the software development process include creating a project specification, project planning, carrying out research and selecting strategies.
- The problem specification is agreed between the client and the systems analyst before proceeding with any work on the project
- The problem specification should include scope and boundaries, and functional requirements
- A project plan includes division of the task into sub-tasks, allocating time to these tasks, and a system of monitoring progress

- Research may be required in order to select and justify appropriate strategies for a project
- It is important to identify all resources that may be required

## 2.9 Revision Questions

**Q5:** The project specification:

- a) is a formal document agreed between the client and the systems analyst
- b) is a rough outline of the project written by the client
- c) is the starting point from which a project proposal is developed
- d) is one of the functional requirements of the project

**Q6:** Scope and boundaries:

- a) describe the functional requirements of the project
- b) are part of the project plan
- c) should be included in the project specification
- d) should be clarified before the analysis stage

**Q7:** Project planning involves

- a) defining the scope and boundaries of the problem
- b) identifying sub-tasks, times and target dates
- c) working out a detailed schedule which cannot be changed
- d) listing the functional requirements of the solution

**Q8:** Selecting a strategy

- a) does not need to be done until the implementation stage
- b) is important in "real life" projects, but unimportant for Advanced Higher
- c) is important, but does not need to be based on formal criteria
- d) should be based on research and defined criteria

**Q9:** Your Record of Work

- a) already contains several items including a specification and project plan
- b) should be a formal record which you write up at the end of the project
- c) should only contain the project specification at this stage
- d) is an optional extra which you may or may not find useful



---

## Topic 3

# Design

---

### Contents

3.1	Prior Knowledge and Revision . . . . .	30
3.2	Introduction . . . . .	31
3.3	User Interface Design . . . . .	31
3.4	Design Methodologies . . . . .	33
3.5	Design Notations . . . . .	33
	3.5.1 Data Flow Diagrams . . . . .	33
	3.5.2 Structure Diagrams . . . . .	34
	3.5.3 Pseudocode . . . . .	35
3.6	Summary . . . . .	37
3.7	Revision Questions . . . . .	37

### **Learning Objectives**

*After studying this topic, you should be able to:*

- *describe the main elements of the design stage of the software development process*
- *describe aspects of good user interface design*
- *create a user interface design for your project*
- *explain and exemplify top-down design and stepwise refinement*
- *exemplify the use of data flow diagrams, pseudocode and structure diagrams*

### 3.1 Prior Knowledge and Revision

You should already know the seven stages of the software development process: Analysis - Design - Implementation - Testing - Documentation - Evaluation - Maintenance:

**Design** is the second stage, coming after analysis and before implementation. The starting point of the design stage is the project specification produced during the analysis stage. This specification is a clear, agreed description of the project, including scope and boundaries and all the functional requirements. From this, the project team can begin to design a solution to the problem.



#### Revision

**Q1:** The first 3 stages of the software development process, in order, are:

- a) Design, Analysis, Implementation
- b) Analysis, Design, Implementation
- c) Documentation, Evaluation, Maintenance
- d) Analysis, Implementation, Testing

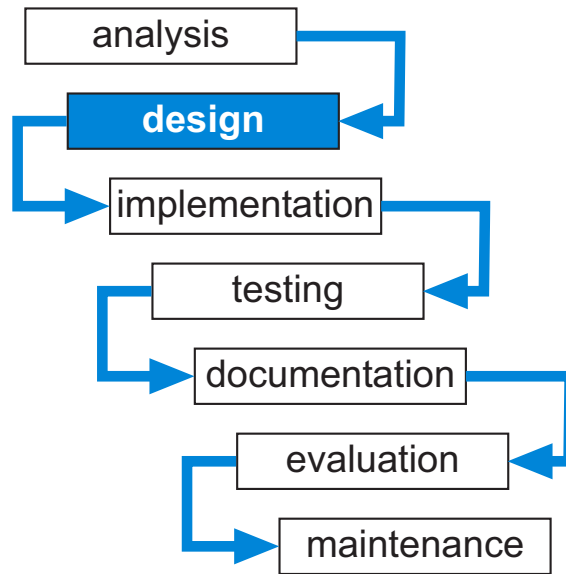
**Q2:** The starting point for the design stage is:

- a) the project proposal written by the client
- b) the project proposal agreed between client and analyst
- c) the detailed specification agreed between client and analyst
- d) the initial meeting between client and analyst

**Q3:** The design stage is:

- a) an optional extra, which can be omitted for simple projects
- b) essential, even in small projects
- c) important in "real-life" project, but not required for Advanced Higher Computing
- d) often combined with implementation

## 3.2 Introduction



In this Topic, we will develop the ideas you already know about the design stage of the software development process.

You will then apply this to your own Advanced Higher Computing project. As you do so, you will produce items of evidence to file in your Record of Work.

There are two main aspects of software design to consider. You will need to **design the user interface** of your software, and you will need to **design the structure of the software** (the algorithm).

## 3.3 User Interface Design

During the design stage, you will need to make decisions about the user interface of your application.

As you already know, there are 3 main classes of User Interface:

- command driven interfaces
- menu driven interfaces
- graphical user interfaces (GUI)

It is often assumed that GUIs are the best. However, that will depend on the context for your application. In some cases, a command line interface may be more appropriate. The choice of interface type will depend, not only on the user group of your application, but also on your ability and experience. Visual languages and authoring tools certainly make GUIs much easier to develop than they used to be.

Once you have decided on the type of interface, there are many guidelines available for "good interface design". Some of these are listed below.

Good interfaces should:

- be as simple as possible
- give clear prompts to the user
- be consistent
- make sensible use of colour
- allow the user to "undo" the last action
- provide on-line help

You can find many more lists in textbooks and on web sites. Different sources emphasise different aspects. You will be aware of many of these criteria from your own experience.



10 min

### Evaluating user interfaces

Think about some user interfaces you have used recently. They don't need to be computer-based. For example, you might think about some domestic appliances.

- Make a list of 4 different user interfaces (2 computer-based, 2 non-computer-based)
- For each one write 2 good aspects of the interface
- For each one, write 2 improvements which could be made to the interface
- Compare your answers with another student, or discuss them with your tutor



120 min

### Designing the user interface

For your project, design all aspects of the user interface.

This may involve sketching how you want the screen to look while the application is running. Include as much detail as possible - colours, objects, icons, text (size, colour, font), interactions.

Label these designs, and file them in your Record of Work which should now include:

Table Of Contents	
	<a href="#">TitlePage</a>
	<a href="#">Project Proposal</a>
	<a href="#">Scope &amp; Boundaries</a>
	<a href="#">Functional Requirements</a>
	<a href="#">Time Plan</a>
	<a href="#">Selecting a Strategy</a>
	<a href="#">Resource List</a>
	<a href="#">Interface Design</a>

## 3.4 Design Methodologies

The standard design methodology for developing software is called top down design with stepwise refinement. You should apply this methodology to your AH Project.

There should be nothing new to learn here - just a reminder of what you learned in the Higher Software Development unit, and have been applying to all your software development work in Higher and Advanced Higher. The use of top down design with stepwise refinement becomes even more important now that you are about to begin a major software development project.

### Reminder

Top down design starts with an overall picture or concept of what the program should do. From this starting point, the designer can break the overall task up into smaller tasks. These smaller tasks, in turn, can be broken down into sub-tasks, and so on .... Eventually detailed design is applied to each sub-task. Step by step, the fine detail is worked out. The process is called stepwise refinement.

## 3.5 Design Notations

Once again, there are no new ideas here. You should have been using at least 2 design notations for all software development work through Higher (and earlier) and into Advanced Higher.

The two design notations that you are most likely to have used are:

- **Pseudocode**
- **Structure charts**

You may also have used **data flow diagrams**.

Here is a quick reminder of all three of these:

### 3.5.1 Data Flow Diagrams

A data flow diagram may be used in both the analysis and design stages of the software development process. It simply shows the data going into and out of any program module. Data flow diagrams can be drawn for a whole program or for any module within the program.

For example, Figure 3.1 shows a data flow diagram for a simple program to process pay and tax for an employee:

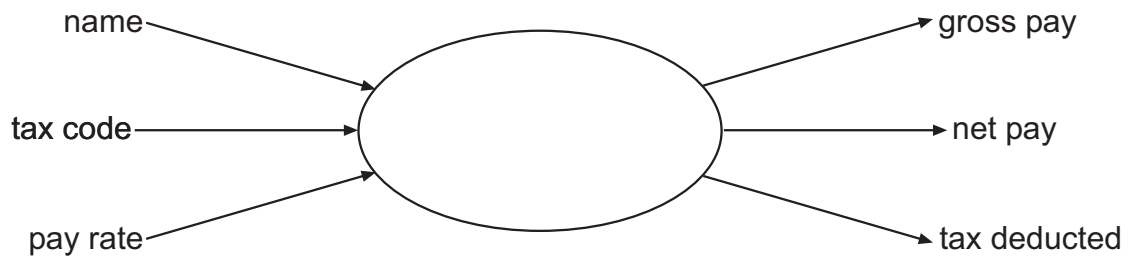


Figure 3.1:

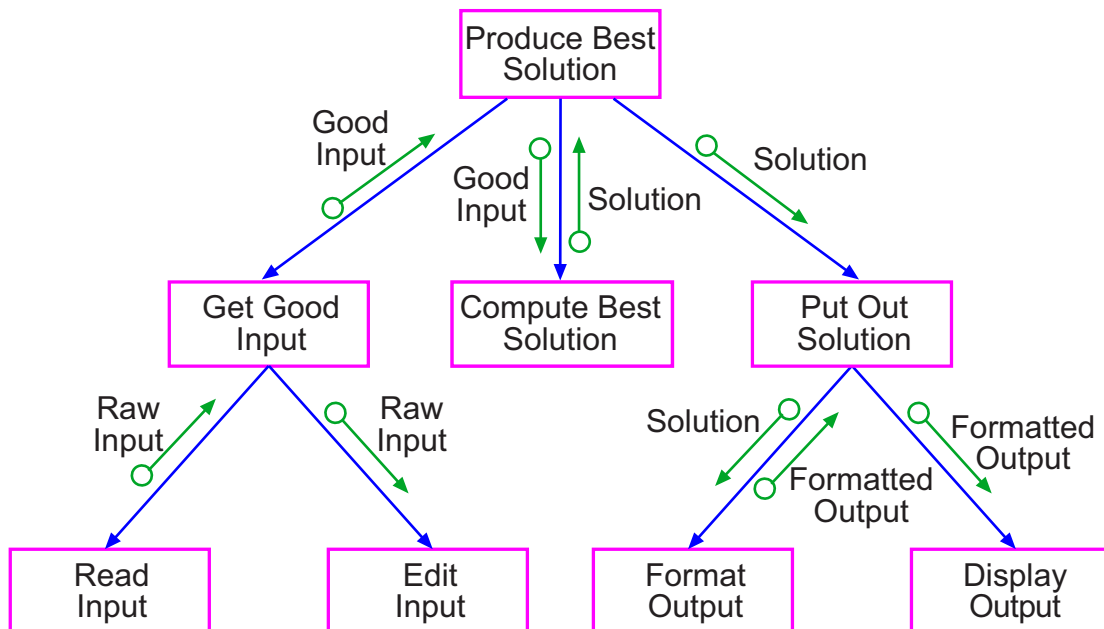
This overall data flow diagram helps the designer to think about all the data that will be input and output from the program. This, in turn, will help in the design of the user interface.

Data flow diagrams can be drawn for each module, to help the designer plan the parameters and variables which will be inputs and outputs of each module.

### 3.5.2 Structure Diagrams

During the process of top down design, it is often useful to show how the modules and sub-modules are related using a diagram. There are many variations to be found in different textbooks. Some use special symbols or different shaped boxes to indicate different types of sub-program. It doesn't really matter, so long as the diagrams you draw are helpful to you in designing the program.

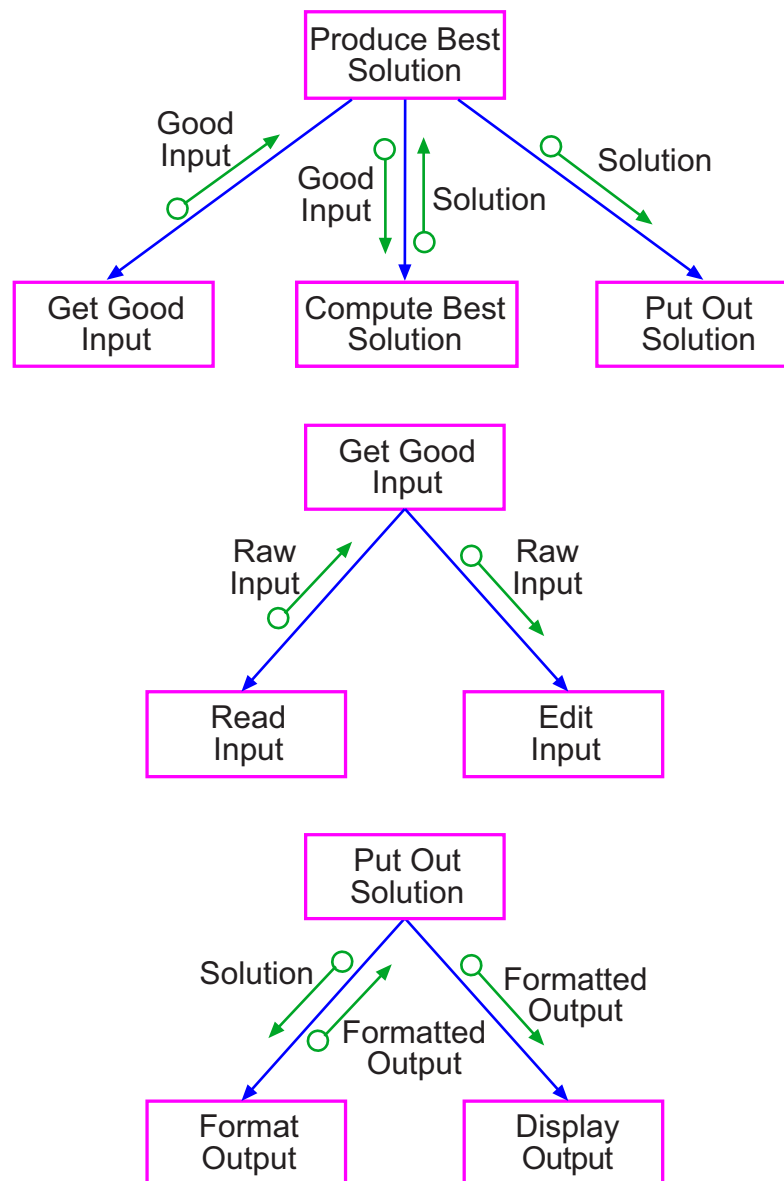
Here is an example of a structure diagram for a program:



The diagram shows that the main program is sub-divided into 3 modules. Each module is named. Two of these modules are further sub-divided into 2 sub-modules. The diagram also shows clearly the flow of data between modules. In fact, the structure diagram is really several data flow diagrams interconnected.

A structure diagram for a whole program can become very complex. If so, you can draw a top level structure diagram, showing only the main program and its first level

sub programs. Further structure diagrams can be used for each of the first level sub-programs.



Some people think in pictures and diagrams. If you are that type of person, you should use structure diagrams during the design phase of your project.

Some people find it easier to use words only. Pseudocode is an alternative design notation using words rather than diagrams.

### 3.5.3 Pseudocode

Pseudocode is another way of describing the design of a program. It uses a hybrid language, using some words of normal English, and some words of the intended programming language which will be used to implement the program.

Here is the same program design as in the previous section, described using pseudocode rather than a structure diagram:

Produce best solution:

- |     |                       |                      |                       |
|-----|-----------------------|----------------------|-----------------------|
| 1   | Get good input        |                      | out: good input       |
| 2   | Compute best solution | in: good input       | out: solution         |
| 3   | Put out solution      | in: solution         |                       |
| 1.1 | Read input            |                      | out: raw input        |
| 1.2 | Edit input            | in: raw input        |                       |
| 3.1 | Format output         | in: solution         | out: formatted output |
| 3.2 | Display output        | in: formatted output |                       |

This contains exactly the same information as the structure diagram. Some people find the diagram more useful in helping them to design a solution, other prefer the pseudocode. You can use either!

However, many people find it best to use a combination of both design notations. They may use a structure diagram for the top level design, breaking the program down into modules which can be implemented using a single sub-program, then use more detailed pseudocode to design the algorithm within each sub-program. Detailed pseudocode might look like this:

```

1. counter = 0
2. prompt the user for the search value
3. set pointer to start of list
4. do
5.   compare search item to list(position)
6.   if equal then
7.     increment count
8.   end if
9.   move to next position in the list
10. until until counter > 15
11. report number of occurrences

```

Note: some designers number the lines of pseudocode, some don't - again it is a matter of what you find helpful.



240 min

### Designing your program

Now is the start of the difficult bit! You need to design the solution to your project. You should use top down design with stepwise refinement as your methodology. You can use any combination of data flow diagrams, structure diagrams and pseudocode as your design notation.

All your design work should be kept and filed in your Record of Work, which should now include:

Table Of Contents	
	<a href="#">TitlePage</a>
	<a href="#">Project Proposal</a>
	<a href="#">Scope &amp; Boundaries</a>
	<a href="#">Functional Requirements</a>
	<a href="#">Time Plan</a>
	<a href="#">Selecting a Strategy</a>
	<a href="#">Resource List</a>
	<a href="#">Interface Design</a>
	<a href="#">Program Design</a>

Remember to update your project plan, showing that you have completed the design stage of your project.

You have completed the design phase of your project, and are now ready to move on to implementation.

### 3.6 Summary

- design can begin once the analysis stage has established a clear project specification
- the design stage involves designing the user interface and the program algorithm
- good user interface design is very important
- the algorithm will be designed using top-down design with stepwise refinement
- data flow diagrams, pseudocode and structure diagrams are all useful design notations
- data flow between modules or sub-programs should be shown clearly at the design stage
- evidence of design must be included in your Record of Work

### 3.7 Revision Questions

**Q4:** During the design stage,

- a) the user interface must be created
- b) the user interface and program structure/algorithm must be designed
- c) data flow between modules can be ignored
- d) actual coding can begin

**Q5:** The design methodology which should be applied is:

- a) structure diagrams or pseudocode
- b) data flow diagrams
- c) top down design with stepwise refinement
- d) stepwise design with top down refinement

**Q6:** Structure diagrams and pseudocode are

- a) design methodologies
- b) design notations
- c) useful tools but not required for your record of work
- d) aspects of the program specification required before design begins

**Q7:** Data flow between modules

- a) can only be shown on a structure diagram
- b) can only be shown using pseudocode
- c) does not need to be shown on either structure diagrams or pseudocode
- d) can be shown on either structure diagrams or pseudocode

**Q8:** Your Record of Work

- a) should now contain a specification, a project plan and evidence of design
- b) is an optional extra which you may or may not find useful
- c) should only contain the project specification at this stage
- d) is a formal record which you write up at the end of the project

---

## Topic 4

# Implementation

---

### Contents

4.1	Prior Knowledge and Revision . . . . .	40
4.2	Introduction . . . . .	41
4.3	Creating a test plan . . . . .	41
4.4	Implementing your design . . . . .	43
4.5	Keeping your Record of Work . . . . .	44
4.6	Summary . . . . .	44

### ***Learning Objectives***

*After studying this topic, you should be able to:*

- *describe and create a test plan*
- *complete the implementation of your design*

## 4.1 Prior Knowledge and Revision

You should already know the seven stages of the software development process: Analysis - Design - Implementation - Testing - Documentation - Evaluation - Maintenance:

Inexperienced programmers might be tempted to plunge directly into **implementation**, but by now you should understand the importance of spending time on analysis and design before beginning implementation.

If the analysis and design has been done effectively, implementation should be straightforward. Once completed, the product can be tested and then documentation created to support users.

Sometimes, however, during implementation, problems occur which may lead to further analysis, or changes in design. Similarly, faults found at the testing stage may require certain aspects of implementation to be changed. This is why the software development process is often described as an iterative process. Figure 4.1 is a simplification of reality. Still, it makes sense to follow the stages in their correct order, and reduce the need to review earlier stages.

Implementation means getting involved with the computer. Choices made at the design stage are now turned into reality. This may mean using a high level programming language or some other type of development environment, like a multimedia authoring package.



### Revision

**Q1:** Implementation

- a) is the first stage of the software development process
- b) follows analysis and design in the software development process
- c) is the final stage of the software development process
- d) follows testing in the software development process

**Q2:** Time spent on analysis and design

- a) is time wasted
- b) is important for AH Computing, but unimportant for "real life" projects
- c) can save time at the implementation stage
- d) slows down the whole development process

**Q3:** During implementation

- a) you will require to make further design decisions
- b) you will use a software development environment
- c) you will evaluate the solution to your problem
- d) you will certainly use a high level programming language

## 4.2 Introduction

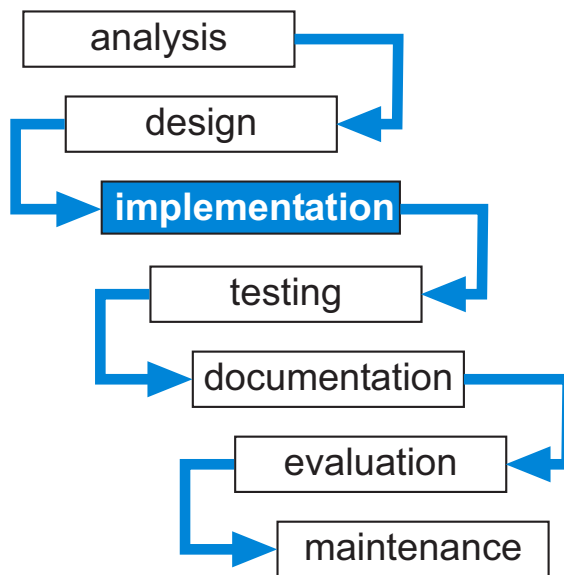


Figure 4.1:

Most of this section is about YOUR project.

From the analysis and design that you have carried out, you are now ready to implement your solution. But there is one more thing to consider - the creation of a test plan.

## 4.3 Creating a test plan

You may be surprised to find that we're talking about testing now, at the start of the implementation stage. According to the 7 stages of the software development process, testing comes **after** implementation, not before it. This is correct! You can't test the program before it has been implemented.

However, a test **plan** should be created at this stage, although the plan won't be carried out until later. In "real life" major software development projects, the test plan is a lengthy and detailed document. For example, look at <http://www.epri.com/eprisoftware/processguide/testplan.html>. There you will see what a test plan could include.

For your project, the **test plan** can be less ambitious!

What should be covered?

You already know that testing should be

- systematic (following a logical order)
- comprehensive (covering every function defined in the functional specification)

The program should be tested with

- normal test data
- extreme test data (boundary conditions)
- exceptional test data

And testing can be at different levels:

- module testing (each sub section of the program tested separately as it is developed)
- component testing (groups of modules tested together)
- beta (acceptance) testing (final testing of the completed program)

The exact details of the test plan will depend very much on your program, and how it is structured, and how you will implement it. However, the guidelines above will help you to structure your test plan. It should be systematic and comprehensive, and cover all three types of test data.



30 min

### Creating a test plan for your project

List all the sub-programs within your design.

List all the functional requirements.

Put these in tables, like these:

Sub-program	Tested	Comment

Functional requirements	Tested	Comment

Most of the sub-programs can be tested individually as they are implemented. As you do so, tick them off, and add any comments (like, "works correctly" or "OK, but could be tidied up if time allows" or "doesn't work, will need to get some help").

The functional requirements are more likely to be tested at the end, once you have completed all the implementation.

Some modules and functions either do the job or not. They don't depend on input data. However, many will need to be tested with a range of test data.

For any aspects of your program which will need to be tested with different types of test data, make up test data tables, giving reasons for your choice of data, and space to fill in results:

Testing ... (insert function or module being tested here)					
Type	Test Data	Reason	Expected result	Actual result	Comment

Table Of Contents	
	<a href="#">TitlePage</a>
	<a href="#">Project Proposal</a>
	<a href="#">Scope &amp; Boundaries</a>
	<a href="#">Functional Requirements</a>
	<a href="#">Time Plan</a>
	<a href="#">Selecting a Strategy</a>
	<a href="#">Resource List</a>
	<a href="#">Interface Design</a>
	<a href="#">Program Design</a>
	<a href="#">Test Plan</a>

## 4.4 Implementing your design

Finally, it is now time to start implementing your design! Check back to your project plan to see what order you have planned to follow. You may decide to alter the order a little, now that you have a clear design to follow. Keep an eye on the time and target dates.

Remember to follow all the techniques of good programming you have learned in Higher and Advanced Higher Software Development. These include

- including internal commentary
- using meaningful identifiers (variables and procedure names)
- using parameter passing and local variable rather than global variables
- modular programming (use existing modules if they exist)

You can save yourself a lot of work later on, if you keep a record of your implementation as you go along.

## 4.5 Keeping your Record of Work

You will probably spend 20 hours or more on implementation, spread over several weeks or months. **You should keep your record of work** up-to-date at all times. Why not set aside 10 minutes at the end of every session to update your Record of Work.

In your Record of Work, you should keep

- printouts / hard copies of your program at each stage of development - make sure you label them carefully, with the date and version number; annotate them with any problems or notes about further work required;
- notes of any unit testing you carry out - these can be anything from simple comments "it works!" written on a program listing to annotated screen dumps showing part of a program working correctly with given test data

You should also make sure that both your project plan and test plan are filled in as you go along.

Your Record of Work will be like a scrap book or diary showing what you have done. It may be required as evidence for unit assessment, so don't forget about it. On the other hand, it is not meant to be a work of art or a formal project report, so don't waste time re-typing things or making them look pretty.

Table Of Contents	
	<a href="#">TitlePage</a>
	<a href="#">Project Proposal</a>
	<a href="#">Scope &amp; Boundaries</a>
	<a href="#">Functional Requirements</a>
	<a href="#">Time Plan</a>
	<a href="#">Selecting a Strategy</a>
	<a href="#">Resource List</a>
	<a href="#">Interface Design</a>
	<a href="#">Program Design</a>
	<a href="#">Test Plan</a>
	<a href="#">Implementation Notes</a>
	<a href="#"> </a>
	<a href="#"> </a>
	<a href="#"> </a>

## 4.6 Summary

- a test plan should be created before implementation begins
- a test plan describes what will be tested, how it will be tested, and when it will be tested
- good programming techniques should be applied during implementation
- the record of work should be maintained throughout the process
- module testing during implementation should be recorded against the test plan

---

## Topic 5

# Testing

---

### Contents

5.1	Prior Knowledge and Revision . . . . .	46
5.2	Introduction . . . . .	47
5.3	Carrying out your test plan . . . . .	47
5.4	Evidence of Testing . . . . .	48
5.5	Rectifying Errors and Bugs . . . . .	49
5.6	A User Questionnaire . . . . .	49
5.7	Summarising your results . . . . .	51
5.8	Summary . . . . .	52

### ***Learning Objectives***

*After studying this topic, you should be able to:*

- *carry out a test plan*
- *produce a user questionnaire*
- *summarise test results*
- *rectify errors and bugs*

## 5.1 Prior Knowledge and Revision

You should already know the seven stages of the software development process: Analysis - Design - Implementation - Testing - Documentation - Evaluation - Maintenance:

You also know that the software development process is **iterative**. That means that sometimes, you have to repeat stages to get it right. This is particularly true of implementation and testing.

You know that testing should be both systematic and comprehensive. Any software should be tested under a range of conditions, and using normal, extreme and exceptional test data.

Testing should be based on a **test plan** created before implementation. The test plan should include module testing, component testing and beta (acceptance) testing.



### Revision

**Q1:** Testing

- a) is the final stage of the software development process
- b) follows implementation in the software development process
- c) takes place before implementation in the software development process
- d) is sometime called maintenance

**Q2:** Test data designed to test the software under boundary conditions is called

- a) normal test data
- b) acceptance test data
- c) extreme test data
- d) exceptional test data

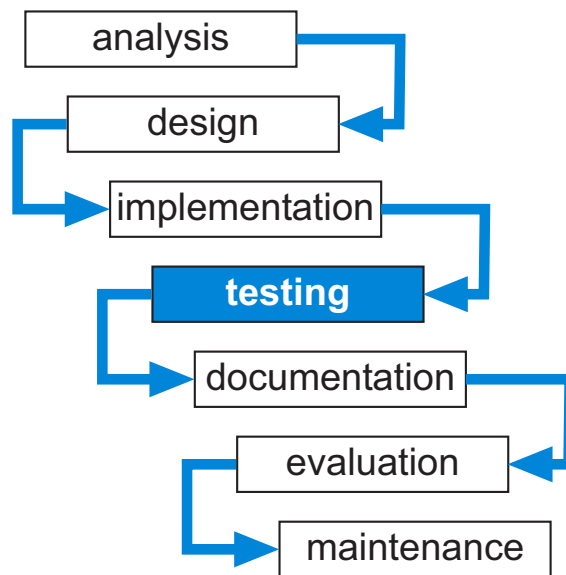
**Q3:** The testing of individual sub-programs as they are implemented is called

- a) component testing
- b) module testing
- c) beta testing
- d) comprehensive testing

**Q4:** Comprehensive testing means

- a) testing of every possible combination of input data
- b) testing every function defined in the functional specification
- c) getting users to test the final product
- d) testing with exceptional test data

## 5.2 Introduction



Once again, most of this section is about YOUR project.

You should now have a completed project, and a test plan that you devised earlier. You will apply this test plan to your project. You should already have tested each module or sub-program as you were implementing them; now you are going to test all the functional requirements in a systematic way.

## 5.3 Carrying out your test plan

Refer to your test plan in your Record of Work. It will look something like this (except that you should already have filled in the first columns (lists of sub-programs and functional requirements). Also you have probably filled in the other columns of the first table, showing that you have tested each sub-program (and corrected any bugs that you found). If there are any of these sub-programs that you haven't tested, do so now.

Sub-program	Tested	Comment

Functional requirements	Tested	Comment

Now you must carry out systematic testing against all the stated functional requirements which you have listed above.

Where appropriate, you should test using a range of test data (normal, extreme and exceptional).



30 min

### Devising test data

If you have not already done so, create test data for each functional requirement. Usually, the best method is to create a table to display your test data and the results, like this:

Testing ... (insert function or module being tested here)					
Type	Test Data	Reason	Expected result	Actual result	Comment
<i>Normal</i>	<i>5, 10, 15</i>	<i>Common input</i>	<i>"use widget B54Xa2"</i>	<i>"use widget B54Xa2"</i>	<i>OK</i>

In some cases, a table of testing is not really appropriate. The main thing is that you show what you are testing, and you keep a record of results. Remember to fill in the "expected result" column BEFORE you carry out the test(s).

## 5.4 Evidence of Testing

For your Record of Work, you should keep some evidence of testing. In most cases, the tables of results are sufficient evidence. However, it is also useful to have some "real" evidence in the form of screen shots. In particular, you should obtain screen shots to illustrate what appears on the screen at any stage of your program. So perhaps you might generate screen shots of the opening or title page, of an input screen, and of the output of the program (or similar for your program). Note that there is no requirement to create screen shots of **every** test you carry out! You may test many functions of the program with many different sets of test data, but only need 3 or 4 screen shots to illustrate how the program looks on screen.

All evidence of testing - tables of results, print outs, screen shots - should be filed in your record of work, and carefully labelled to explain their relevance.

Table Of Contents	
	<a href="#">TitlePage</a>
	<a href="#">Project Proposal</a>
	<a href="#">Scope &amp; Boundaries</a>
	<a href="#">Functional Requirements</a>
	<a href="#">Time Plan</a>
	<a href="#">Selecting a Strategy</a>
	<a href="#">Resource List</a>
	<a href="#">Interface Design</a>
	<a href="#">Program Design</a>
	<a href="#">Test Plan</a>
	<a href="#">Implementation Notes</a>
	<a href="#">Evidence of Testing</a>
	<a href="#"> </a>
	<a href="#"> </a>

## 5.5 Rectifying Errors and Bugs

As a result of testing, you have almost certainly uncovered some errors or bugs in your software. If so, you should correct these as they are discovered. Keep a record of any corrections you make - simple notes written on the earlier print-outs may be enough. If an error requires a major rewrite on any part of the software, print it out and file it in your record of work, with a hand-written note on the print-out explaining why you have made the changes. You should implement some logical use of version numbers, and make sure you label all print-outs accordingly.

Beware of the possibility of correcting a bug causing a new error in some part of the software that you have already tested. If you have applied the principles of modular programming, this should be unlikely to happen, but it is usually a good idea to repeat some earlier tests just to be on the safe side.

Sometimes, students think that they should hide and/ or destroy all evidence of bugs or errors, and simply keep evidence of the completed (perfect!) program. This is not recommended! Your tutor will be awarding marks for your ability to correct errors, so **keep the evidence** to show that you have managed to fix any problem that occur.

## 5.6 A User Questionnaire

So far, you have carried out module testing and systematic and comprehensive testing of the functions of your program. You have tested your own program; as a result, you may have been "blind" to any weaknesses. At this stage, you are ready to carry out some acceptance (or beta) testing. That means asking some potential users to test the software for you.

In "real life" software development projects, beta testing can be very extensive. For example, Microsoft distributed half a million beta versions of its Office 2003 software for potential users to try and test.

Many software companies take a more restricted approach to beta testing, and ask users to apply to be beta testers. See, for example, the following web sites:

<http://www.cambridgesoft.com/about/betatesting.cfm>

<http://thematrixonline.warnerbros.com/web/beta/signup.jsp>

You could allow a group of your class-mates to test your software for you. Your teacher may also be willing. Another possibility is to use the SCHOLAR forum to find students in other centres who might be willing to carry out some beta testing for you.

The simplest way is to give your tester(s) a copy of the software, and ask them to try it out and report any errors, bugs or weaknesses that they find. This approach is useful, but it is often more effective to give the testers some structure- pointing them to what you want them to test.

One useful way to do this is to create a **questionnaire**. Creating a questionnaire is not as easy as it might seem. Here is a site which contains loads of useful hints:

<http://writing.colostate.edu/references/research/survey/pop4a.cfm>

However, you have only a limited time, so here are some summary hints:

- give the tester some guidance on which parts of the software to test
- ask a limited number of clearly worded questions (see below)
- keep your questionnaire to a single sheet of paper
- set a date for its return

Here are some example questions from a beta testing questionnaire on the web: (you will find the complete list at [www.supermemo.com/archive/beta/Beta2002q.htm](http://www.supermemo.com/archive/beta/Beta2002q.htm))

1. What operating systems did you test on? What browser versions?
4. What are the top three bugs that have not yet been removed?
7. What are your top three dislikes about SuperMemo 2002?
8. Did you read on-line documentation? What should be changed in the structure? Or in style?
9. Are you aware of any documentation errors?
14. What is the most unclear concept/component of SuperMemo 2002?



60 min

### Beta Testing

Create a short user questionnaire.

Ask your tutor to check it before you use it.

Find some suitable testers.

Provide them with a copy of the questionnaire and software.

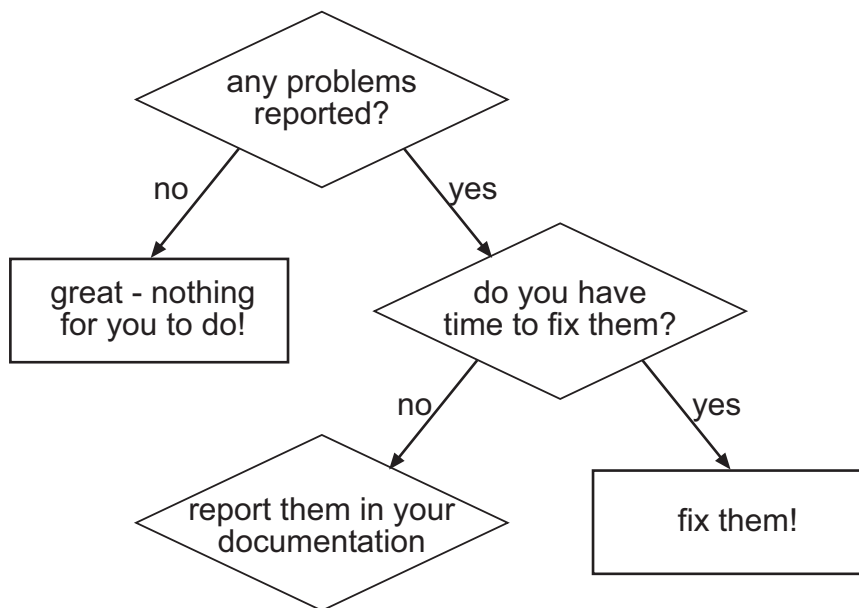
Set a deadline for return of the questionnaire.

When they are returned, file them in your Record of Work.

Table Of Contents	
	<a href="#">TitlePage</a>
	<a href="#">Project Proposal</a>
	<a href="#">Scope &amp; Boundaries</a>
	<a href="#">Functional Requirements</a>
	<a href="#">Time Plan</a>
	<a href="#">Selecting a Strategy</a>
	<a href="#">Resource List</a>
	<a href="#">Interface Design</a>
	<a href="#">Program Design</a>
	<a href="#">Test Plan</a>
	<a href="#">Implementation Notes</a>
	<a href="#">Evidence of Testing</a>
	<a href="#">Beta Test Results</a>
	<a href="#"> </a>

After the questionnaires come back to you, you may (or may not) need to take action.

Here is a flow chart of what you might need to do



## 5.7 Summarising your results

You should now have completed the testing

- module testing during implementation
- systematic testing of the functionality (by you)
- acceptance/beta testing (by someone else)

In your Record of Work, you should have

- your test plan
- the completed tables of modules and functions
- tables of testing with different types of test data
- the results of any beta testing

### Summary of Testing

You should now summarise the results of testing. This could be quite brief, referring to the documents listed above as references.



30 min

You should summarise

- the types of testing you carried out
- any significant bugs or errors that you discovered, and have now rectified

- any significant bugs or errors that you have been unable to rectify
- any minor weaknesses that you have decided do not need to be dealt with

In total, this should be no more than 1 page of A4. Add this to your Record of Work.

Table Of Contents	
	<a href="#">TitlePage</a>
	<a href="#">Project Proposal</a>
	<a href="#">Scope &amp; Boundaries</a>
	<a href="#">Functional Requirements</a>
	<a href="#">Time Plan</a>
	<a href="#">Selecting a Strategy</a>
	<a href="#">Resource List</a>
	<a href="#">Interface Design</a>
	<a href="#">Program Design</a>
	<a href="#">Test Plan</a>
	<a href="#">Implementation Notes</a>
	<a href="#">Evidence of Testing</a>
	<a href="#">Beta Test Results</a>
	<a href="#">Summary of Testing</a>

You have now completed all the stages required to pass the unit.

## 5.8 Summary

- testing should follow implementation, and be based on the test plan
- testing should be comprehensive and systematic
- testing should include module testing during implementation
- all functional requirements should be tested using a range of test data
- acceptance (beta) testing should be carried out by independent testers
- errors and bugs identified during testing should be rectified and/or documented
- a summary of testing should be produced

---

## Topic 6

# Project Report

---

### Contents

6.1	Prior Knowledge . . . . .	54
6.2	Introduction . . . . .	54
6.3	Evidence required for Unit Assessment . . . . .	54
6.4	Evidence required for Course Assessment . . . . .	56
6.5	User and Technical Documentation . . . . .	58
6.6	Evaluation Report . . . . .	59
6.7	Summary . . . . .	61

### ***Learning Objectives***

*After studying this topic, you should be able to:*

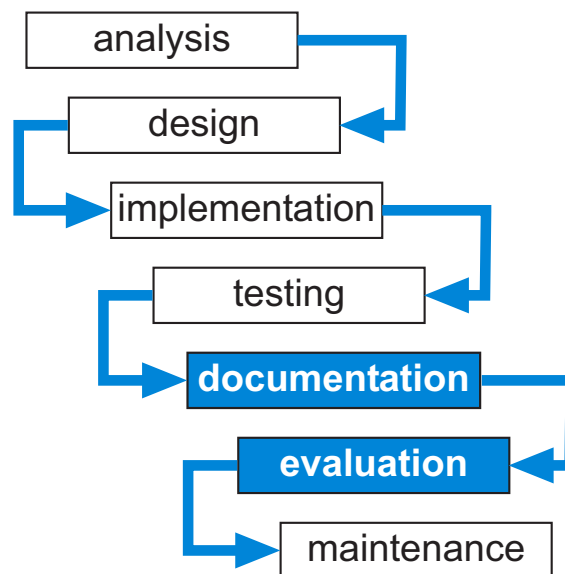
- *give your tutor the evidence required for Unit assessment*
- *write user and technical documentation*
- *evaluate your project*
- *give your tutor the evidence required for course assessment*

## 6.1 Prior Knowledge

You should already know the seven stages of the software development process: Analysis - Design - Implementation - Testing - Documentation - Evaluation - Maintenance:

You should be at the stage where you have completed the first 4 stages for your Advanced Higher Project. The next 2 stages are **Documentation** and **Evaluation**. Both of these are required for the **course assessment**.

## 6.2 Introduction



Once again, most of this section is about YOUR project.

If you have completed the work laid out in the previous 5 topics, you should by now have completed your project. You have probably spent 30 - 40 hours on this work. Your Record of Work should now be complete, including evidence of analysis, design, implementation and testing.

All that is left to do, is to produce the **final report** that is required for course assessment. This should take no more than 10 hours.

This topic will show you what is required.

## 6.3 Evidence required for Unit Assessment

The evidence required for Unit Assessment is simply your Record of Work. If you have been following the instructions in previous topics, and keeping your Record of Work up to date throughout all the stages of development of your Project, then you should have nothing more to do!

### Checking your Record of Work for Unit Assessment Evidence

Use this checklist to make sure your Record of Work contains all the evidence required for unit Assessment:



30 min

Stage	Evidence required	Check
Analysis	Problem specification, including <ul style="list-style-type: none"> <li>• scope and boundaries (Topic 2.2.2)</li> <li>• list of functional requirements (Topic 2.2.3)</li> </ul>	
	Evidence of planning, including <ul style="list-style-type: none"> <li>• list of sub-tasks (Topic 2.3.1)</li> <li>• notes justifying selection of strategy (Topic 2.5)</li> </ul>	
Design	Pages from record of work, showing <ul style="list-style-type: none"> <li>• annotated sketches of user interface design(s) (Topic 3.2)</li> <li>• pseudocode and/or structure diagrams (Topic 3.4.2/3)</li> </ul>	
Implementation	Annotated program listings and screen shots, showing <ul style="list-style-type: none"> <li>• development of software through various stages (Topic 4.4)</li> <li>• final complete version of software</li> <li>• use of modular programming techniques</li> <li>• use of complex languages structures</li> </ul>	
Testing	Evidence of testing, including <ul style="list-style-type: none"> <li>• systematic test plan (Topic 4.2)</li> <li>• normal, extreme and exceptional test data (Topic 5.2)</li> <li>• results of testing (tables and/or annotated screen shots) (Topic 5.3)</li> <li>• summary of test results (Topic 5.6)</li> </ul>	

Hopefully, you were able to tick off all these items. If any are missing, you will need to return now to the appropriate topic, and produce the required evidence.

Notes:

1. You may also have some additional items in your Record of Work. That's fine - don't get rid of them! They may be needed for Course Assessment (see next sub-topic)
2. DO NOT waste time typing up neat copies of any of the above! The Record of Work is a working document. There is no extra credit given for professional presentation.
3. There is no "pass mark" for Unit Assessment. If you have adequate evidence (as listed above) of analysis, design, implementation and testing of a software solution to an appropriate level of problem, you will pass the unit.

## 6.4 Evidence required for Course Assessment

For Course Assessment, you must provide a Project Report which includes

- the problem specification
- evidence of project planning
- evidence of a completed solution (preferably files on a CD, plus hard copy of coding / data files, screen shots)
- user documentation and technical documentation
- an evaluation report



### Creating a CD

Copy all the completed program files and data files on to a CD, and label it with your name, date, school and project title.

**Checking your Record of Work for Course Assessment Evidence**

30 min

Stage	Evidence required	Check
Analysis	Problem specification, including <ul style="list-style-type: none"> <li>• scope and boundaries (Topic 2.2.2)</li> <li>• list of functional requirements (Topic 2.2.3)</li> </ul>	
Analysis	Evidence of planning, including <ul style="list-style-type: none"> <li>• list of sub-tasks (Topic 2.3.1)</li> <li>• time schedule (Topic 2.3.2/3)</li> <li>• resources required (Topic 2.6)</li> </ul>	
Implementation	Evidence of completed software, including: <ul style="list-style-type: none"> <li>• program files on CD (Topic 4.4)</li> <li>• data files on CD (if appropriate)</li> <li>• hard copy of program (and data) files</li> <li>• screen shots of aspects of completed software</li> </ul>	

You should have all this evidence (and more) in your Record of Work. If not, go back to the appropriate sub-topic and produce the required evidence.

Notes:

1. For course assessment, your tutor will allocate marks to all of these items. There is no "pass mark", but the total mark out of 60 (however high or low) is added to your exam mark out of 140 to give a total out of 200, on which your overall grade (A,B,C,D or fail) is calculated. Ask your tutor to show you the marking scheme, so that you can see what marks are allocated for, and ensure you haven't missed anything important.
2. If any items of evidence are missing, and you cannot produce them, you will not "fail". You will simply lose marks and risk getting a lower grade for the course.
3. As you will see from the marking scheme, some of the marks (10 out of 60) are for "process skills" - analysis, decision making, research, time management, use of correct terminology. Your tutor will base these on observation during the project as well as on the final evidence.

Finally, there are 3 items that you need to produce that you DON'T already have in your Record of Work. These are:

- User Documentation
- Technical Documentation
- Evaluation Report

You will create these as you work through the next two sections.

## 6.5 User and Technical Documentation



180 min

### User Documentation

For your Advanced Higher project report, you should produce a User Guide, approximately 1-2 A4 pages in length, which includes:

- a description of the features of your software (what your software can do)
- a description of the user interface (how to use the software) - you may want to include some labelled screen shots to illustrate this.
- some help and troubleshooting information (this could be in the form of FAQs based on the responses to your user questionnaire)

Notes:

1. There are 10 marks allocated for the User Documentation (2 for each of the points listed above, and 4 for "clarity and presentation")
2. The User Documentation can be presented as a word processed document, or as a slide presentation or as web-pages
3. The help and troubleshooting may be incorporated within your software as on-line help
4. However, note that the marks are for the **content** and **clarity** of the information, rather than any fancy presentation style.

File your completed User Documentation in the Documentation section of your Record of Work.

## Technical Documentation

For your Advanced Higher project report, you must also produce Technical Documentation. This will be no more than 1 A4 page in length, and should include:



90 min

- software requirements (what Operating system and other software is required)
- hardware requirements (is there a minimum amount of RAM or processor type/speed, or does your software require any particular peripherals)
- list of files provided (this should list and describe all the files on the CD)

Notes:

1. There are 10 marks allocated for Technical Documentation (2 for each of the points listed above, and 4 for "clarity and presentation")
2. The Technical Documentation can be presented as a word processed document, or as a slide presentation or as web-pages
3. However, note that the marks are for the content and clarity of the information, rather than any fancy presentation style.

File your completed Technical Documentation in the Documentation section of your Record of Work.

Table Of Contents	
<a href="#">TitlePage</a>	<a href="#">User Documentation</a>
<a href="#">Project Proposal</a>	<a href="#">Technical Documentation</a>
<a href="#">Scope &amp; Boundaries</a>	
<a href="#">Functional Requirements</a>	
<a href="#">Time Plan</a>	
<a href="#">Selecting a Strategy</a>	
<a href="#">Resource List</a>	
<a href="#">Interface Design</a>	
<a href="#">Program Design</a>	
<a href="#">Test Plan</a>	
<a href="#">Implementation Notes</a>	
<a href="#">Evidence of Testing</a>	
<a href="#">Beta Test Results</a>	
<a href="#">Summary of Testing</a>	

## 6.6 Evaluation Report

Only one more item is required - the Evaluation Report.

You are required to evaluate

- your finished product using 7 criteria
- your own process skills using 2 criteria



120 min

## Evaluation Report

For your Advanced Higher project report, you should produce an Evaluation Report, approximately 1-2 A4 pages in length.

**The first section** is an evaluation of your finished software. You should write 7 brief paragraphs, critically evaluating your software in terms of

- **fitness for purpose** (does it do everything as specified; if not, why not; does it include any extra features)
- **user interface** (is it easy to use and learn; could it be improved in any way)
- **robustness** (does the program cope with external errors and unforeseen mishaps during execution)
- **reliability** (does the program produce correct results for all inputs)
- **portability** (will it run under different OS; what modifications might be required to do this)
- **efficiency** (does it waste processor time, RAM or storage space)
- **maintainability** (what steps have been taken to ensure it is easily maintained)

**The second section** is an evaluation of your application of process skills.

You should already have evidence in your Record of Work. All you need to do is write a brief critical evaluation of your performance, referring to the appropriate section(s) of your Record of Work, under each of the following headings:

- consideration of alternative strategies
- time management

Notes:

1. There are 20 marks allocated for the User Documentation (2 for each of the points listed above, except for "alternative strategies" which gets 4 marks.
2. It pays to be honest! If your time management wasn't good, say so, and give your reasons. You get marks here not for the time management itself, but for your own **critical evaluation** of your performance.
3. The Evaluation Report should be presented as a word processed document, with references to the evidence in your Record of Work.
4. Once again, note that the marks are for the **content** and **clarity** of the information, rather than any fancy presentation style.

File your completed Evaluation Report in the Evaluation section of your Record of Work.

Table Of Contents	
<a href="#">TitlePage</a>	<a href="#">User Documentation</a>
<a href="#">Project Proposal</a>	<a href="#">Technical Documentation</a>
<a href="#">Scope &amp; Boundaries</a>	<a href="#">Evaluation Report</a>
<a href="#">Functional Requirements</a>	
<a href="#">Time Plan</a>	
<a href="#">Selecting a Strategy</a>	
<a href="#">Resource List</a>	
<a href="#">Interface Design</a>	
<a href="#">Program Design</a>	
<a href="#">Test Plan</a>	
<a href="#">Implementation Notes</a>	
<a href="#">Evidence of Testing</a>	
<a href="#">Beta Test Results</a>	
<a href="#">Summary of Testing</a>	

## 6.7 Summary

You have now completed the Advanced Higher project, and should have a folder containing the following evidence:

For Unit Assessment:

- a record of work providing evidence of analysis, design, implementation and testing

For Course Assessment:

- a project specification and evidence of project planning for Course Assessment
- a CD with your completed software and any associated files
- User Documentation
- Technical Documentation
- an Evaluation Report

## Glossary

### Data flow diagrams

A type of diagram used in the design of software systems, A program or module is represented by an oval shape. Arrows are used to represent any data flowing in or out of the module.

### Feasibility study

At a very early stage in any project development, a feasibility study is carried out to judge whether or not the project can be achieved, within any known constraints. These constraints include time, finance and availability of appropriate hardware, software and expertise.

### Functional requirements

A list of all the functions which the finished program or system must be able to do. This forms a main part of the specification.

### Project specification

A document agreed between the systems analyst and the client during the analysis phase, which defines clearly all the functional requirements of the systems to be developed. It forms a binding legal agreement between the developer and the client.

### Pseudocode

A design notation used in the design of computer programs. It consists of a combination of ordinary English and programming language specific words. Statements may be numbered to assist coding.

### Structure charts

A type of diagram used as a design tool. Modules, sub-programs or individual steps are represented by boxes, linked by lines. Data flow between boxes is shown by labelled arrows.

### Systems analyst

Usually an experienced programmer, who is responsible for the analysis and design phases of software development. The Systems Analyst, working with the client, develops the project or system specification, on which the design and implementation are based.

### Test plan

A plan devised before implementation of software, listing all the tests to be carried out, the test data to be used, and the expected outcomes.

## Answers to questions and activities

### 1 The Advanced Higher Project

#### Considering Possible Project Ideas (page 5)

##### Q1:

1. (Yes) This would be a good project. You will need to use advanced features of the HLL you choose, possibly including 2-D arrays, file handling and a good user interface.
2. (No) If it can be done using only code easily available in the public domain, you won't be able to show the assessor that you have had to design and implement complex algorithms yourself. However, it would be OK if you used a combination of public domain code and your own coding.
3. (No) This is TOO complex! It would take you much more than 40 hours. You need to be a bit more realistic.
4. (No) If it could be done by a Higher candidate, then it is too straightforward for Advanced Higher. But it is a good idea. Can you think how to extend it a bit to make it complex enough for AH?
5. (Yes) This is based on the same area as the last example, but it involves file handling, which you have only learned about in the AH course.
6. (No) NLP is an interesting topic to consider, especially if you have done the AI unit at Higher. But if it only involves a very limited vocabulary and simplistic grammar, it could be done using the basics of Prolog you learned at Higher.
7. (Yes) This is better. You are going to implement a proper grammar, with some real-life complexity in it. The size of the vocabulary isn't really much of an issue. If it works for a small vocabulary, it should extend to a larger one. Your research will also benefit you if you are doing the AI optional unit.
8. (No) This is over the top! To create a system to handle ANY input would require considerably more skill than you have and certainly more than the 40 hours you have at your disposal. You need to cut it back a little!
9. (No) If your CAL package could be created using a basic presentation package or similar, then it is nearer Int 2 level than AH. Don't be put off, though. A CAL implementation is a good idea, but you will need to specify a more complex scenario. For example ... ?
10. (Yes) That's better! If you need to use a scripting language behind the scenes, and are going to be involved in file handling, then it becomes an AH problem.
11. (Yes) And here's another good CAL idea. This time, you will need to show an understanding of some low level language concepts. This will also benefit you if you are going to choose the Computer Architecture unit. Don't make it too complicated though - you only have 40 hours.
12. (Yes) Here's another good idea, especially if you have done Computer networking at Higher. The only problem you might have to consider is whether the school/college network will allow you to test your application.
13. (No) A straightforward web site won't give you the opportunity to demonstrate any AH software development knowledge or skills, even if you choose to write it in pure HTML. But that doesn't mean that a web site is a no-go area ...

14. (Yes) This web site is much more suitable, as it will involve you in some new programming techniques, and in file handling.
15. (No) This is potentially a good idea for the AI people, but if it can be implemented with simple rules in an expert system shell, then it is too simple.
16. (Yes) This is more like it. You would need to use a declarative language, or perhaps even a procedural language, and some fairly complex algorithms.

**Answers from page 9.**

**Q2:** a) you must pass 3 units, complete a project and sit an exam

**Q3:** c) is required to pass the unit and counts 40% of the course award

**Q4:** b) is your own choice, but it must be suitably complex

**Q5:** d) will help you keep track of progress, and must be maintained for assessment

**Q6:** a) a software solution to a computing problem

## 2 Analysis

### Revision (page 12)

**Q1:** a) Analysis, Design, Implementation

**Q2:** c) turn a vague problem description into a precise program specification

**Q3:** b) agreed between the client and the developer

**Q4:** d) should be the starting point for any software development, large or small.

### Answers from page 27.

**Q5:** a) is a formal document agreed between the client and the systems analyst

**Q6:** c) should be included in the project specification

**Q7:** b) identifying sub-tasks, times and target dates

**Q8:** d) should be based on research and defined criteria

**Q9:** a) already contains several items including a specification and project plan

### **3 Design**

#### **Revision (page 30)**

**Q1:** b) Analysis, Design, Implementation

**Q2:** c) the detailed specification agreed between client and analyst

**Q3:** b) essential, even in small projects

#### **Answers from page 37.**

**Q4:** b) the user interface and program structure/algorithm must be designed

**Q5:** c) top down design with stepwise refinement

**Q6:** b) design notations

**Q7:** d) can be shown on either structure diagrams or pseudocode

**Q8:** a) should now contain a specification, a project plan and evidence of design

## **4 Implementation**

### **Revision (page 40)**

**Q1:** b) follows analysis and design in the software development process

**Q2:** c) can save time at the implementation stage

**Q3:** b) you will use a software development environment

## 5 Testing

### Revision (page 46)

**Q1:** b) follows implementation in the software development process

**Q2:** c) extreme test data

**Q3:** b) module testing

**Q4:** b) testing every function defined in the functional specification