8. Systems Analysis and Design

Systems life cycle

Life Cycle	Objectives	Examples Methodology
Analysis	Different methods of researching a situation Establishing the inputs, outputs and processing in the existing system Recording information about the current system Identifying problems with the current system Identifying suitable hardware and software for a new system Identifying the user and information requirements	Observation, examination of documents, questionnaires, interviews Identify the sources and volume of input data and collection methods. Identify the input documents currently in use. Determine frequency addition/deletion of records. Identify manual and computer procedures necessary to achieve the current output. Data flow diagrams (Level 0 DFD – context diagram and Level 1 DFD – current system), system flowcharts Observation, examination of documents, questionnaires, interviews Analysing required outputs, storage and processing requirements Collating the interview transcripts, questionnaires and documents
Design	Specifying the required hardware and software Designing data collection forms, screen layouts Designing report layouts and screen displays Designing validation routines Designing the required data/file structures and programming specifications;	Volume of data will determine the choice of output devices. The order that data will be output will affect choice of storage devices. These will depend on the user requirements as well as output required from system and file structures The content and presentation of reports, layouts and screen displays will depend on the requirements of the users The form of input and file structures will determine these The data structures and programming will depend on the types of processing and input and output structures.
Development and testing	Creating data structures, program modules Testing strategies Improvements that could be needed as a result of testing	Testing each module with normal/live data Testing each module with abnormal and extreme data Testing whole system Amend structures, program modules in the light of the results of testing
Implementation	Identifying the different methods of system implementation	Parallel running, direct changeover, implementation and pilot running
Documentation	Designing and developing elements of technical documentation Designing and developing elements of user documentation	Developing systems documentation (results of systems analysis, what is expected of the system, overall design decisions, test plan and test data) Developing program documentation (description of the software, purpose of the software, input data formats, output, flowcharts, program listing, notes to assist future modifications) A guide to simple elements of use of the software and hardware making up the system
Evaluation	Evaluating a new system in terms of the efficiency, ease of use and appropriateness of the solution	Using test results to evaluate the solution. Obtaining feedback from the user. The results of this evaluation are used to identify limitations Using the identification of the limitations to make improvements

Systems Analysis

A **computer system** consists of the hardware, the software, the data, the people and the ways they all interact.

When a company needs a computerised solution to a problem, they call on a **systems analyst** to oversee the creation, development, testing and implementation of a new computer system....and possible further improvements....

More correctly....

........**systems analysis** involves a detailed study of an existing system and specifications of a new system...

........and **systems design** involves the design of the new system (hardware, software, data and people)

One of the jobs of the systems analyst is to make sure the new system is...

....**cost-effective** - the new system should generate more money than is spent on installing and running the new system!

....**completed on time**

....completed **within a budget** agreed with the management of the company.

The System Life Cycle

The stages of creating a new computer system is called a **system life cycle**.

1. Definition of the problem.

Without a detailed definition of the problem we are trying to solve, people are often unsure of what it is they are trying to achieve.

-**aims** and **objectives** of new system must be stated
eg cost reductions, better service to customers; greater volume of business transactions etc

The final results will be measured against these to decide the effectiveness of the solution.

2. Feasibility Study

A preliminary investigation is essential to determine whether a project is technically and economically feasible. Some projects may not be suitable for computer solution.

Is technology currently available?
Is it **socially** feasible - will there be a need for redundancies, retraining?
Will it be cost effective? Development costs and running costs need to be balanced against benefits such as reduced costs, better customer service etc

A written report may be presented to management for a decision on whether to proceed.

Data Flow Diagrams

A computer system needs to be written down for others to understand. One way of representing a system is to use a **data flow diagram**.

This type of diagram shows how the data flows through the system, and what data stores are used. It does not define what type of data storage is used, or how the data is stored. this type of detail can be determined at a later stage.

Symbols used :

Entity - data source or data destination - ie where the data comes from or where it goes to...eg timesheets, customers, accountant,etc...

Process - an operation performed on the data.

Data store - this could, for example, represent a file held on disc or magnetic tape, a batch of input documents or a report.

Data flow - the arrow represents movement of data between entities, processes or data stores. The arrow should be **labelled** to describe what data is involved.

When drawing data flow diagrams, you should stick to the following conventions:

- do not draw data flow lines directly between **data stores** and **external entities**. There should be a **process box** between them to show the operation performed.
- most processes will have **input** data and **output** data...so state what they are.
- label the data flow lines so that it is clear what data is being transferred.

Problem analysis

Top-down Design

When confronted by a large task........... break it down into a number of smaller tasks. (If you have to move a mountain...... move one boulder at a time!).

Top-down design is the technique of breaking down a problem into a number of smaller problems. Each of these is broken down into even smaller problems and so on..... until each problem is sufficiently simple to be solved easily.

These 'small' problems are referred to as **modules** and should be as **self-contained** as possible....this means that each module can be solved on its own without depending on the solution of other modules.
Top-down design is sometimes called **stepwise refinement**.

Non-programming example:

The problem :	Put on a musical show
Break it down	Choose cast Rehearse Perform show

The first steps might be broken down again....

Choose cast	Choose singers Choose speaking parts Choose dancers
Rehearse	Select times to rehearse Develop a rehearsal schedule Arrange use of a hall Rehearse
Perform show	Hire costumes Hire lighting and sound Sell tickets, Performances

and these could be broken down again.....

Structured system design

Structure

This word '**structure**' keeps appearing.....what is it and why do we need to keep talking about it?

It really means...the way in which different parts are arranged and how they are linked together.

When designing a system it is important that the **modules** are simple enough to be easily understood by all the people who are going to be working on it...

....if it is easy to understand, then it is easy to create the system and to solve problems...

....if it is easy to solve problems, then there will be fewer **errors**.

Lack of structure means a confused, complex and error-riddled system which is difficult to sort out!

Modules in Programming

Top-down design is useful for tackling large programs.

In a program the modules are called **subroutines** (**procedures** and **functions** in Pascal).

Advantages of a modular approach to programming :

- A small problem is easier to solve than a large one!
- The modules can be **shared** out between a team of programmers.
- Each module can be **coded** and **tested** separately.
- Easier to **debug** if it is not working properly.
- If changes need to be made it is easier to maintain small modules.
- A library of modules can be stored for use in other programs.
- A large project becomes easier to monitor and control.

Hints for programming

- Always try to write subroutines that are as 'self-contained' as possible..(they do not depend on others)
- Use **local** variables wherever possible.
- Use **parameters** to make the subroutines as useful as possible - parameters are the interface to other subroutines and should be selected carefully.

Remember..a well-designed program...

- ...is more likely to satisfactorily solve the intended problem,
- ...will be less prone to errors
- ...will be completed quicker as sections would not need to be re-written etc
- ...will be easier for other programmers to understand if they need to alter it at a later stage.

Specification of the system

When a new system is designed the first step is to specify the **system configuration** - that is the **hardware** needed and the **software** which will be installed,

..then it has to be defined in terms of its......

Input	**Data capture** - How is the data collected? - automatic eg bar codes, swipe cards, MICR etc.. - data capture forms **User interface** - Command line; menus; GUI; **Input screens** - Screen layouts need to be designed. **Methods of data entry** - automatic; keyed in; **Data Verification**? - Data entered twice; only accepted if two versions identical.
Data Storage	Definition of **data** - what fields;what field types, lengths etc Will data be stored as a **database** or as **files**? Definition of **database tables** or of the **files** What **storage medium** - CD? disc? tape?
Processing	**Hardware**? choice will depend on - - volume of data - number of users - location of users (Same room or spread around the country) - security considerations Data **Validation** - what methods? Range checks; type checks;

	length checks etc... What **software** should be used? If programs need to be written - - High or Low Level language? - Which Language? (Pascal, C++, Visual Basic etc)
Output	Type of output? - printed; sound; visual etc Output screen displays Printed reports will need to be designed.
Other considerations:	
Recovery procedures	These will need to be defined. What to do if there is an error.
Clerical procedures	Usually documents. eg What happens to incoming orders? Outgoing invoices? etc
Documentation	All systems need to be documented. - to set specifications for the project. (Must be kept up-to-date) - to ensure the system can be maintained after completion. - "*as time progresses and the needs of the system change, without the ability to be maintained properly, the software will become outdated and useless. Changes of staff within a company may mean that no-one involved in the original design or programming of a system is still with the company. 90% of programming is in updating existing software.*"

Human/Computer Interfaces

An **interface** is where one object meets another. When designing a computer system, consideration has to be given to how a person will interact with a computer. There may be a number of design factors...

- what is the workplace like? ..noisy? dirty? busy?
- what are the IT skills of the users like?
- security considerations

There are three main types of **user interface**:

Command driven	A user has to type in a **command**. An expert user can generally do this quickly and can tailor a command to exactly suit requirements.

	Example : **DOS** is a command line interface To get a list of all files on a floppy disc, a user would type: **DIR a:** To print the list across the screen rather than down he would type: **DIR a: /w** **Batch files** of a number of these commands may be created. A disadvantage to this type of interface is that a user will need to learn the commands and spell/type them in correctly.
menu driven	Options are displayed as a hierarchy of **menus**..selecting one menu option may lead to another menu and so on... There is no need to learn commands but a user would need to understand the terms in the menus, and it can be quite difficult sometimes to find the exact option you want ..especially if it is an option in a submenu of a submenu of a submenu! **Example**: Some DOS based programs use menus eg Borland **Pascal**
GUI - Graphical User Interface	Small pictures (**icons**) are used to represent selections and a user would click on them with a mouse pointer. These are usually combined with a system of menus. Sometimes called a **WIMP** (**W**indows, **I**cons, **M**enus and **P**ointers) environment. **Example**: Microsoft **Windows** is an example of a GUI **Advantages** : • No great level of IT skill needed • Easily recognisable and understandable icons lead to fast selection • No need to learn any commands

Changeover

Changing from an old system to a new system can be done.....

	Direct changeover The old system is stopped and the following day the new system is used. This could be disastrous if there are errors in the new system!

	Parallel running The old system and the new system are run together for a period of time. The old system is stopped only when it is certain that the new system is running correctly. **Advantage** : no disasters if there are errors in the new system **Disadvantage** : a lot of duplication of work.
	Piloting The new system could be tried by part of the company - eg one branch. If all goes well, then the rest of the company can change to the new system
	Phased change Part of the new system is used, and when all problems have been solved and it works then another part is implemented and so on...

Maintenance

A new computer system is up and running...but that is not the end of the story..

All computer systems require **maintenance**. This may be for a number of reasons :

perfective	All systems can be improved... programs may be **improved** to run faster, give better response time to customers etc..
adaptive	there may be **changes** in the company. All businesses expand or contract....number of customers may increase......number of products sold may increase...new branches opened..etc...
corrective	**problems** may arise which have been hidden for a while and then suddenly arise. These would need to be corrected.

A good and effective system design will make sure that any maintenance is done effectively and should minimise the chance of any errors arising.

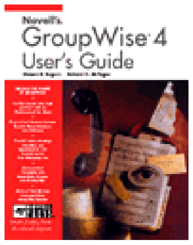
Programmers who have to change other people's programs can do it easier, in less time and with fewer mistakes if the program is well-designed......and if there is good program documentation.

Documentation

When systems are changed it is not usually done by the same people who originally designed and created it. In order to change a system, a person would need to know exactly how it works...hence the need for good documentation. A lack of understanding may lead to a new system which has errors or to a system which is not effective.

(*It is estimated that 90% of programmers are involved in changing existing software rather than creating new programs*)

Documentation of a system will include :

1	**Technical Manual**	**systems specifications** - description of systems - data flow diagrams (or similar) **algorithm (program) specifications** - algorithm flowcharts (or pseudocode) - program listing - lists of variables used (*most programs are self-documenting; comments are added to the listing explaining the steps used in the program. Meaningful variable names are also used.*)
2	**User Manual (Operating Instructions)** 	- software **installation** procedures - details for **starting** the program. - details for setting **security** (passwords etc) - details of discs/tapes required. - clerical procedures - data **preparation** (batching / hash totals etc) - how to enter data(which fields; codes used; how to enter dates etc) - details and samples of **reports** which may be printed. - **backup** procedures to be followed. - **recovery** procedures in the event of hardware failure.