

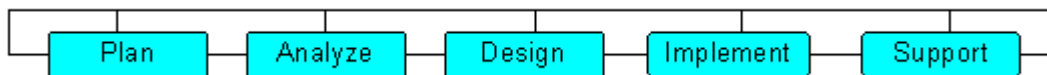
## Revision Notes

### *What is systems analysis?*

- A systematic approach to identifying problems, opportunities and objectives
- Analysis of organisational information flows
- Presentation of complex information to non-technical staff
- Design of information systems

Systems analysts also act as business consultants, solve problems, act as agents for change, communicate with people at all levels, understand and keep abreast of technical developments and build appropriate relationships with the user community.

### The Systems Development Life Cycle



CASE tools can assist in this process. Good project management is critical as problems picked up after systems design cost much more to solve than those picked up before then. Systems analysis and design needs to be as thorough as possible. Note that the process is cyclical.

### **Project Analysis**

A project is a one-of-a-kind product, with concurrent/consecutive activities and time/budget limits. We must allocate resources and coordinate personnel. Project analysis enables us to calculate estimates of the total time to complete the project for a given resource level, scheduled start/finish times for each activity, a list of critical tasks which cannot be delayed, maximum delay time for noncritical activities. It also gives us information to guide resource reallocation and tools to monitor project progress.

#### *Starting a project*

- Appoint a Project Manager (PM)
- Specify project goals
- Specify project deadlines
- Gather resources
- Commit workers

#### Project Phases:

1. Collection and analysis of task (activity) data
2. Project planning and scheduling
3. Execute/monitor the project plan

#### *Collection and analysis of task (activity) data*

- Decompose the project into tasks.
- A task is an identifiable unit of work with a clear beginning and end
- Identify precedence relationships between tasks
- Estimate the time required to complete each task at the current resource level

Sensible estimates of time and resources required for each task are important in order to determine project feasibility and as a yardstick for progress measurement - allowing the project manager to set targets. They also allow the exploration of different possible ways in which a project could be completed. Any estimate is usually better than none at all.

## Revision Notes

### *Tools for task time estimation*

- Experience is the most accurate and expensive method. It requires experience of each task
- Analogy
  - o Using documentation from previous projects
  - o Projects should be technically similar
  - o Suitable comparators are hard to come by
- Estimating task times by Calculation – CoCoMo (Constructive Cost Model)
  - Developed by Boehm from a detailed study of many software development projects
  - $PM = 2.4 KDSI^{1.05}$
  - PM = Person-months effort
  - KDSI = expected number of source instructions
  - Elapsed time =  $2.5 PM^{0.38}$
  - To what extent it helps is questionable

Document your estimates - review previous estimates as the best possible learning tool – and the assumptions you used to calculate them. Get multiple viewpoints (through multiple methods, other personnel). Aim for only sufficient accuracy - but not too much

### *Time/Cost Tradeoffs*

In order to reduce overall project duration, we must reduce the duration of critical tasks. We may 'crash' activities, by giving them more resources in order to reduce overall project time. In the process of crashing we reduce the times of each critical activity in turn, by at most the smallest float of a non-critical activity, recalculating at each step, until we have the desired project duration. It may be possible to reduce resources allocated to non-critical activities without increasing overall project duration.

### *Probabilistic Task Times*

Another technique gives probability distributions to each task time. This allows us to answer questions such as "What are the chances of being finished in 30 days?" The assumptions are that a) task times behave like a beta distribution and b) Optimistic, Pessimistic and Most Likely task times must be known with considerable accuracy. Hence this is only useful when task times on a project are known to considerable accuracy - e.g. for repetitive tasks.

### *Implementation Issues*

Fast computer software is available for "what-if" analysis. Computer software can decompose large projects into, and keep track of, subprojects. Task times should be reviewed and updated throughout the project. Similar techniques can be used for some repetitive activities such as maintenance.

### *Project recipe*

1. Allocate project manager, personnel and resources
2. Develop task list, task times and predecessors
3. Draw activity-on-arrow network
4. Find Earliest and Latest Node times
5. Find ES, EF, LS, LF and floats for each activity
6. Find the critical path
7. Reallocate resources and resolve if necessary
8. Carry out probabilistic analysis if desired
9. Monitor and change task time estimates throughout the lifetime of the project



## Revision Notes

### **Interviewing**

Interviewing is good for getting information about opinions, feelings, goals (organisational, personal and system) and things you didn't know you needed to know. However, it is bad for actually gathering data.

#### *Interview Preparation*

- Decide objectives
- Read background material
- Decide whom to interview
- Set up the interview
- Decide interview structure
- Write down questions you will ask and pointers for questions you might ask

Open questions, e.g., "How do you generate schedules?" are hard to analyse and make it hard to keep focussed on interview objectives as well as possibly unsettling interviewees. However, they may tell you things you didn't know you needed to know and tell you about further possible projects.

Closed Questions, e.g. "How many courses do you teach?" may settle an interviewee as they are easy to analyse and to maintain focus. Observation and use of data may be more efficient but questions such as these can be a useful source of unbiased opinion, when you know the facts.

#### *Question Pitfalls*

Leading questions: "How bad is the current system?"

Double questions/poor structure: "How often does the system fail, and how do you fix it when it does?"

Stay focussed. If you are unlikely to be able to make use of the answer don't ask the question!

#### *Question Structure*

Pyramid: starting with closed questions and working toward open-ended questions

Funnel: starting with open-ended questions and working toward closed questions

Diamond: starting with closed, moving toward open-ended, and ending with closed questions

#### *Starting the Interview*

- Who you are
- Why you want to interview them
- What objectives you want to achieve from the interview
- That the information they give is confidential and how it might be used

Subsequent greetings should discuss an issue from or summarise the previous interview.

During the interview, introduce each line of questioning. Probe into answers ("Why?") - shows that you are listening, good for keeping interview focus and double-checks information and notes.

#### *Taking notes*

- Not too many
- Maintain eye contact
- Go back over important points
- Use of sound recorders

End the interview with an open "anything else" question, discuss next steps and arrange follow-up.

## Revision Notes

### *Alternatives to Interviewing*

JAD (Joint Application Design) is a collaborative approach to requirements gathering through short, intensive workshops to find requirements with the help of several users and technical facilitators.

PD (Participatory Design) is a user-focussed technique to improve the workplace.

RAD (Rapid Application Development) gives quick product delivery, through

- Rapid prototyping
- Integrated CASE tools
- SWAT (Specialists With Advanced Tools) Team
- Interactive JAD
- Timeboxing (setting a deadline by which an objective must be met, rather than describing when a task must be completed)

Phases:

Requirements Planning → User Design → Construction → Cut-over (switching to the new system)

### *Questionnaires*

- For wider coverage
- Need to know the right questions (through interviews and background)
- Focus on objectives
- Don't ask unnecessary questions
- Keep it short + facilitate response
- Use both open and closed questions
- Beware of the "halo effect" (the impression formed in one question carries into the next question)
- Scales (Nominal: 'Which one?'; Ordinal: 'Best, Good, Average, Poor'; Interval: '1-5 with 1 as ...')

Interviewing will always be one of the most important ways to find out organisational and system goals  
There are alternatives JAD/PD etc. Questionnaires can reach a wider audience but lack flexibility

## **Prototyping**

Prototyping is a useful tool for quickly gathering specific user information in order to find out what needs to be found out for novel or high-risk systems. It incorporates users' innovations, suggestions, reactions and priorities and may be used to build better project plans and to get user buy-in.

### *When and What to Prototype*

- Early – during feasibility and business requirements
- Throughout the project (as an alternative to an SSADM-like approach)
- At decision points - when the future shape of a system is unclear
- When risks are significant
- For decision support systems
- In small chunks, implemented quickly (at most a few days for each chunk)
- For good design of input/output

### *Types of Prototype*

Patched-up prototypes – complete functionality but inefficient.

Nonworking scale models – look right, but don't do anything

First full-scale models – show the final system before deploying it in large numbers.

Selected Features Prototype – features *some* functionality of the final system.

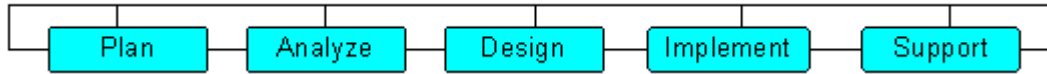


- Early identification of major problems in business requirements analysis
- Can terminate an unworkable plan early
- Can address user needs more closely and quickly
- Increases user feelings of project ownership
- Can develop a prototype into a full system
- Reduces the requirement for user training



- May become difficult to manage project
- May raise or lower user expectation
- Prototype may come to be regarded as a full system
- Expensive, but prototyping tools (e.g. ACCESS/EXCEL are making it cheaper)

## Systems Analysis Methodologies and Tools



The life cycle produces a useful framework for putting in tasks, but is not a substitute for good project management (recall that the cost of changes increases enormously throughout the project).

Alternatives exist (usually more open-ended):

- 'b' model
- 'V' model
- Incremental model
- Spiral model

However, a useful plan is not open-ended.

### *Structured Systems Analysis and Design Methodology (SSADM)*

(developed by Letchworth and Burchett Management School (LBMS) and Central Computer and Telecommunications Agency (CCTA) - a UK government agency)

SSADM provides a

- "to do" list to structure development
- analysis/design tools and methods

It has been a legal requirement that SSADM be used on certain (UK government) projects.

### *SSADM Stages*

- Stage 0: Feasibility
- Stage 1: Requirements Analysis
- Stage 2: Business Systems Options
- Stage 3: Requirements Specification
- Stage 4: Technical Systems Option
- Stage 5: Logical Design
- Stage 6: Physical Design

Within each stage there is a sequence of numbered steps which in turn are divided up into numbered tasks (230 in all). It is rare that all tasks need to be performed. Starts after initiation and ends before any software is written.

For example, Stage 6: Physical Design consists of:

- Step 610: Prepare for Physical Design
- Step 620: Create Physical Data Design
- Step 630: Create Function Component Implementation Map
- Step 640: Optimise Physical Data Design
- Step 650: Complete Function Specification
- Step 660: Consolidate Process Data Interface
- Step 670: Assemble Physical Design

And Step 620: Create Physical Data Design consists of:

- Task 20: Identify the required entry points and distinguish those that are non-key
- Task 30: Identify the roots of the physical hierarchies
- Task 40: Identify the allowable physical groups for each non-root entry
- Task 50: Apply the least dependant occurrence rule
- Task 60: Determine the block size to be used
- Task 70: Split the physical groups to fit the required block size.
- Task 80: Apply the product specific data design rules for the target DBMS

### *Perspectives*

SSADM is a data driven method – the data is fixed and processes change. There are three viewpoints:

- How the data items in the system move through it
- How the various data items are related to each other
- How any one of the data items changes over time

### *Tools*

It offers 3 separate techniques for these three viewpoints:

- Dataflow diagrams ( DFD )
- Logical Data Modelling ( LDM )
- Entity Life Histories ( ELH )

Good CASE tools exist to support these and to allow speed up of code writing. There are good CASE tools for I/O. Structured English is also used.

### *Requirements Analysis*

#### 1: Current System Analysis

- Interviews
- Questionnaires
- Observation
- Examination of documents

Produce models of the current system (if time permits)

- DFDs, LDMs, Data Dictionaries, ELHs, current system analysis document

#### 2: Requirements Analysis

##### Requirements Analysis Document

- What does the business need from the system?
  - o Current systems analysis + Interviews (and/or JAD/PD approaches)
- Prioritised list of possible features
- Division into
  - o “must have” features/functions and
  - o “would like to have” features/functions
- Important discussion/decision document
- Some reflection on system/data ownership

##### *Systems Design*









- Using modelling tools
  - o Dataflow diagrams (DFD)
  - o Logical Data Modelling (LDM)
  - o Entity Life Histories (ELH)
  - o Structured Englishto design system interactions and processing
- Consider different technical options
- First logical, then physical design
- Down to the level of individual code modules

Structured systems analysis approaches provide a start for, but do not replace, project management. They assume a logical sequence of tasks, are well supported by modelling tools and software, but are inflexible and paper-heavy and must be treated with care. They deal well with big projects, less well with small ones.

**Data Flow Diagrams (DFDs)**

A DFD is one of the most important analysis tools for analysing data-oriented systems. It gives a simple way of representing information flows, a means of representing both business processes and system solutions and a means of system scoping. It is often better than English narrative.

*Symbols*

	<b>Gane &amp; Sarson</b>	<b>SSADM</b>	<b>Naming</b>	
<b>External Entity</b>			A single noun	External entities are sources or sinks of data lying outside the context of the system. Externals don't speak to each other.
<b>Data movements</b>			The type of data being moved	A data flow can be thought of as a pipeline through which packets of data of known composition flow. Data flows must be an input or output of a Process Box.
<b>Process</b>			Unique ID number	Transforms incoming flows into outgoing flows.
<b>Data store</b>			Unique ID number "Client" <i>not</i> "Client Data"	Data stores can hold permanent, temporary, historical or extract data.

N.B. a double line means duplicate.

*Logical* DFDs show how the business operates. Processes are independent of the system implemented. *Physical* DFDs show how the system operates (or will be implemented). Processes show how the system will be implemented.

CASE tools (including Select SSADM) allow us to

- Draw and edit attractive diagrams easily
- Automatically maintain parent/child numbering and relationships
- Document processes, data stores, data flows and external entities
- Automatically generate text documents

**Context Diagram**

- All external entities
- A single process "The System"
- Data flows to and from external entities

Explode to produce...

**Diagram 0**

- Major activities within the context diagram's single process
- External entities
- Major (non-transient) data stores

**Child Diagrams and Levels**

- Each process can itself be decomposed into a child DFD
- Decompose to the level where we can write a detailed specification for each process
- Numbering of child processes is logical, e.g. Process 2's child processes are numbered 2.1,2.2 etc. 2.1's child processes are 2.1.1,2.1.2, etc.
- Context diagram is level 0. Its child diagram (Diagram 0) is on Level 1. A child diagram of a level 1 process is level 2 etc.



## Revision Notes

### Common Errors

- All processes must have both input and output data
- Data direction arrows incorrectly oriented
- Data stores and external external entities cannot directly connect with each other, in any combination.
- Incorrect or unclear labelling
- Cluttered diagrams - aim for 3-9 processes
- Omitting data flows
- Inputs and outputs for each parent process must correspond exactly to those on child diagrams (except for minor flows such as error messages)
- Putting technical process details on a logical DFD

### *Physical DFDs*

Generally, the sequence of events is:

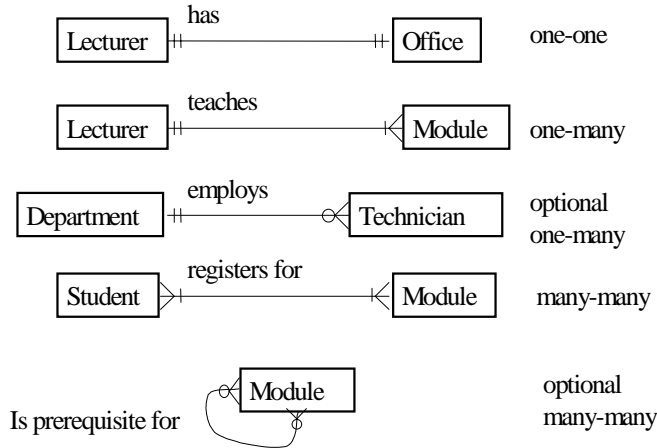
- Produce a DFD of the current system
- Add in the new features required of the system
- All of the above processes and flows are logical, but now we:
- Derive a physical DFD by deciding on how each process and data flow will be implemented.  
Note that we need to decide implementation for both manual and IT-supported process

### Partitioning into Manual Procedures and Computer Programs

- Identify manual procedures
- "Batch" IT processes only interact with data stores and other IT processes
- Group processes into programs
- Different user groups need different programs
- Processes occurring at different times need different programs
- Similar tasks can be grouped in the same program for efficiency, ease of use or simplicity
- Separation may be required for security

**Databases and Entity-Relationship (E-R) Data Modelling**

See G52DBS Database Systems revision notes. Note the slightly different relationship notation:



**Ethnomethodology**

Ethnomethodology is the methodology of people, a social science developed in 1960's California. It is a reaction against formalism in sociology, interaction and work and can be considered as a separate subject, studying human behaviour at a 'meta' level, i.e., how do people show how they orient to each other in interaction?

*How do 'ethno' studies get done?*

- Ethnography
  - o not to be confused with ethnomethodology
  - o involves hanging around 'collecting data'
  - o data may be notes, documents, audio/video
- Conversation analysis & related approaches
- Audio/video data
- Other approaches
  - o Participatory Design
  - o Interviewing
  - o Questionnaires
  - o Business Process Reengineering & Workflow
  - o Joint Application Design

Ethnomethodology focuses on the 'core' processes/data of business, using methods which gloss data - what gets done over ways it gets done. This may lead to missing something in generating requirements.

- |   |  |
|---|--|
| <ul style="list-style-type: none"> <li>✓ - Appearance of rigour in analysis</li> <li>- Appearance of sensitivity within company</li> <li>- Cheap</li> </ul> | <ul style="list-style-type: none"> <li>✗ - Time constraints</li> <li>- Generalisability</li> <li>- Cannot answer some questions</li> </ul> |
|---|--|

The 'real work' that gets done is just as important as its modelled counterpart. Studies can sensitise systems designers to key 'micro' issues in work. What people say they do is not always representative of what they do.



## Process Specification

### *Specification Methods*

- Structured English
  - o Similar to pseudo code
- Decision Tables
  - o Similar to truth tables
- Decision Trees
  - o Similar to flow diagrams

### Structured English

- Used for simple decisions
- Express logic as sequences, decisions, cases or iterations
- Use capitalised keywords (IF, WHILE, etc)
- Indent blocks to show nesting
- Use words from data dictionary (Item, price, etc)
- Careful use of some words (“and”, “or”, “greater than”)

### *Sequences*

Blocks of instructions that contain no branching:

```
print value
add value to total
print total
```

### *Decisions*

Only execute the following statements if a condition is true; otherwise jump to an alternative statement

```
IF price is less than credit
THEN print “You have credit”
ELSE print “You have no credit”
ENDIF
```

### *Cases*

Decisions with mutually exclusive cases

```
IF day is 5
THEN print “Saturday”
ELSE IF day is 6
THEN print “Sunday”
ELSE print “Weekday”
ENDIF
```

### *Iterations*

Blocks of statements that are repeated

```
DO WHILE there are items remaining
    process next item
ENDDO
```

Revision Notes

*Example*

```

calculate days late as current date minus due date
IF days late is greater than zero
  THEN calculate fine as (4 + days late)% of item's RRP
  IF fine is greater than item's RRP
    THEN fine is 110% of item's RRP
  ENDIF
ELSE
fine is zero
ENDIF
  
```

Decision Tables

Use for more complex decisions

Conditions and Actions	Rules
Conditions	Condition Alternatives
Actions	Action Entries

Conditions and Actions	Rules			
	1	2	3	4
Has four sides	Y	Y	N	N
Sides are equal length	Y	N	Y	N
Square	X			
Rectangle		X		
Equilateral triangle			X	
Right-angled triangle				X

*Method*

- Determine the number of conditions
  - o Mutually exclusive conditions are combined
- Determine the number of actions
- Determine the number of condition alternatives
- Insert X where a rule requires an action
- Combine rules where alternatives are redundant
- Check for impossibilities and contradictions
- Reorder the table if it will make more sense

Conditions and Actions	Rules							
	1	2	3	4	5	6	7	8
County is UK	Y	Y	Y	Y	N	N	N	N
Country is Ireland	Y	Y	N	N	Y	Y	N	N
Country is Greece	Y	N	Y	N	Y	N	Y	N
Use pounds				X				
Use euros						X	X	

→

Conditions and Actions	Rules		
	1'	2'	3'
County is UK	Y	N	N
Country is Ireland	N	Y	N
Country is Greece	N	N	Y
Use pounds	X		
Use euros		X	X

→

Conditions and Actions	Rules		
	1'	2'	3'
County	UK	Ireland	Greece
Use pounds	X		
Use euros		X	X

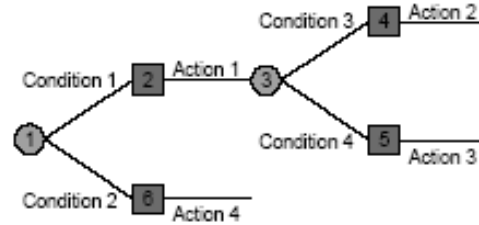
→

Conditions and Actions	Rules	
	1'	2''
County	UK	-
Use pounds	X	
Use euros		X

Revision Notes

Decision Trees

Use for complex decisions  
 Complex branching  
 Sequence of ordered decisions

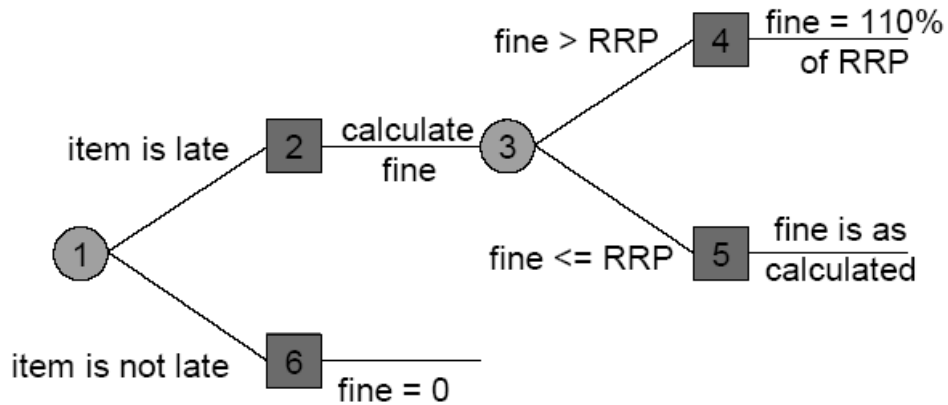


*Method*

1. Determine the order of all conditions and actions
2. Build the tree from left to right, ensuring that all alternatives are complete before moving to the right

*Example*

“If an item is returned late, a fine is incurred until the fine exceeds the item’s RRP, at which point the borrower must pay a fine to the value of 110% of the item’s RRP.”



**Systems Proposal**

Hardware

*Inventory*

If not already available, determine the type, status, age, projected life, physical location, responsible party, and ownership of the equipment.

*Workloads*

Assess current and future workloads using a sample to evaluate the existing and proposed systems. For each task consider the method, personnel involved, cost, frequency and requirements (human and computer)

*Evaluation*

Find equipment that meets the needs, using inventory and workloads. Create *benchmarks* – simulate workloads on various hardware (including current) to estimate average transaction time, capacity of the system, CPU idle time, memory size, etc.

*Acquisition*

	Advantages	Disadvantages
Buying	<ul style="list-style-type: none"> <li>■Cheaper long-term</li> <li>■Upgradeable</li> <li>■Full control</li> </ul>	<ul style="list-style-type: none"> <li>■High initial cost</li> <li>■Becomes outdated</li> <li>■Full responsibility</li> </ul>
Leasing	<ul style="list-style-type: none"> <li>■No capital tied up</li> <li>■Cheaper than renting</li> </ul>	<ul style="list-style-type: none"> <li>■No ownership</li> <li>■More expensive than buying</li> </ul>
Renting	<ul style="list-style-type: none"> <li>■No capital tied up</li> <li>■Upgrade options</li> <li>■Maintenance included</li> <li>■Insurance included</li> </ul>	<ul style="list-style-type: none"> <li>■No ownership</li> <li>■Expensive long-term</li> </ul>

Software

*Performance*

- i) Effectiveness: Can it perform the required/desired tasks?  
Have the output screens been well-designed?  
Is the capacity enough?
- ii) Efficiency: Is the response time fast enough?  
Are the input/output methods efficient?  
Are the storage and backup methods efficient?

*Usability*

- Is the user interface good enough?
- Can the interface be customised?
- Are there help menus available?
- Is the feedback informative?
- Does it handle errors well?

## Revision Notes

### *Flexibility*

- Are there input/output options?
- Is it compatible with other software?

### *Documentation*

- Is the documentation sensibly organised?
- Are there any tutorials?
- Is there a FAQ?

### *Manufacturer Support*

- Is there a technical support helpline?
- Are there downloadable product updates?

## Costs and Benefits

### *Predicting*

If historical data is available, base predictions on models rather than what-if types of analysis. If historical data is not available, use:

- Sales estimates
- Customer demand estimates
- Delphi studies (developed by experts)
- Create scenarios

Modelled predictions are:

- Conditional (variables are linked)
  - Correlation
  - Regression
  - Econometrics
- Unconditional (variables might not be linked)
  - Graphical judgement (line of best fit or 'least squares')
  - Least squares
  - Moving averages:

Seasonal or cyclical patterns often distort lines of best-fit. By calculating moving averages this problem can be overcome. For example, using the average sales over 4 years throughout a 20 year period, year by year, rather than using just the annual sales over 20 years.

### *Identifying*

- Tangible Benefits
  - o can be measured in terms of the money they bring in or save
  - o generally arise from reducing the employee time required to complete tasks
  - o examples: increasing the speed of processing information, making resources more accessible/readily available
- Intangible Benefits
  - o difficult to measure in terms of the money they bring in or save
  - o examples: increased accuracy, improved decision making, increased employee satisfaction, maintaining a good image

### Revision Notes

- Tangible Costs
  - o can be accurately projected
  - o easily determined
  - o examples: equipment (such as computers), resources, systems analyst's time, programmer's time, other employee's salaries
- Intangible Costs
  - o difficult to estimate and sometimes unknown
  - o must be included
  - o examples: declining company image due to increased customer satisfaction, bad decision making due to inaccessible information, external incidents effecting the market (e.g. wars, natural disasters)

### *Comparing*

Techniques for comparing costs and benefits include:

- Break Even Analysis
  - o when do the costs of the current system exceed the costs of the proposed system?
  - o use when proposed system needs to be justified in terms of cost rather than benefits or benefits do not improve a great deal with the proposed system.
- Payback
  - o how long will it take the benefits of the proposed system to pay back its costs?
  - o use when tangible benefits make a convincing case for the proposed system
- Cash-Flow Analysis
  - o how long until profit is made, and how long until revenue exceeds the initial investment?
  - o use when proposed system is expensive relative to the size of the company or when the company would be significantly affected by a large drain on funds.
- Present Value Analysis
  - o when will the proposed system's total benefits outweigh its total costs?
  - o use when the payback period is long or the cost of borrowing money is high

### Writing the Systems Proposal Document

- i) *Cover Letter* to the management and IT department. Keep it brief, friendly and to the point and include a list the people involved in the study, a summary of the objectives and the time and date of the presentation (if relevant).
- ii) *Title Page* should include the name of the project, the names of the systems analysts and the date the proposal was submitted. The title should clearly explain the content of the proposal, but some imagination is acceptable.
- iii) Only include a *Table of Contents* if the proposal exceeds ten pages. Be consistent in style and content.
- iv) The *Abstract* (Executive Summary) should be written last. Between a paragraph and a page in length it should include a summary of the proposal (wwwwh), the analyst's recommendations and the management's actions.
- v) *Outline of Systems Study*. Provide information about who or what was studied and the methods that were used. Include questionnaires, interviews, sampling of data, observation and prototyping.
- vi) *Detailed Results of Systems Study*. Describe what has been discovered regarding the current system through the methods described in the previous section. Include conclusions made during the study, the types and rates of errors, the current and projected system capacity, how work is currently handled and problems and opportunities.

## Revision Notes

- vii) **Alternative Systems.** Describe two or three alternative solutions to address the highlighted problems. One alternative should suggest to keep the current system. Include (for each alternative) the costs and benefits, advantages and disadvantages and a clear wording of actions to management
- viii) **Recommendations.** Give a definite professional opinion of which alternative is the most workable. This should be a logical conclusion from the previous section. Include reasons to support the decision
- ix) **Summary.** Restate the points outlined in the abstract and stress the importance, feasibility and value of the recommendations.
- x) **Appendices.** Include information that might be interesting to individuals, but not necessarily for understanding the study and proposal. Also graphs and charts, a summary of phases undertaken and relevant correspondence.

The writing style should mirror that of existing publications in the organisation. It should be easily comprehensible, but not condescending, containing enough detail to make informed decisions without being overwhelmed. Minimise the number of references and do not use footnotes.

Use appropriate white space to break up sections of the report. Leave equally-sized margins (except when binding) of about one inch. Using headings and subheadings allows the reader to logically follow the structure of the report.

Tables and graphs are an important way to communicate information about the current and proposed systems. All figures should be interpreted with a text description to integrate them into the proposal. Integrate tables into the text body and try to keep large tables all on one page. Try not to exceed two tables per page and number tables consecutively. Use descriptive and meaningful table names, labelling each row and column. Use footnotes to explain detailed information.

Integrate graphs into the text body, limiting the number of graphs per page to one unless making a comparison. Number graphs consecutively as figures, using descriptive and meaningful figure names. Label each axis, line, bar, or pie segment including a key to distinguish and identify different lines, bars etc.

When the systems proposal report is complete, select who should receive it and personally deliver the report.

### Presenting

#### *Structure*

- Less than 30 minutes
  - o 4-6 main points from abstract - recommendations and summary
  - o then add introduction and conclusion
- More than 30 minutes
  - o as above, but cover more main points and in greater detail

The Introduction should hook the audience in – use something to capture their attention and make them want to listen. The conclusion should reiterate the introductory points without repeating them word for word. Try to keep questions until the end - answering questions during the presentation make it more relaxed, but less formal. A difficult question before the end could put you off for the rest of the presentation or worse, put off the audience, if it is answered badly.