Module Handbook

Graduate Diploma in Information Technology

Module CA593

User Interface Development

TABLE OF CONTENTS

1. Introduction To User Interface Development

1.1 What is HCI?

- HCI is a term used to either mean *Human-Computer Interaction* or *Human-Computer Interface*. The former is the more common usage, though in the USA *Computer-Human Interaction* (CHI) is sometimes used, as is *Man-Machine Interface* (MMI).
- HCI is the study of the interaction between people, computers and tasks. It is principally concerned with understanding how people and computers can interactively carry out tasks, and how such interactive systems are designed.
- HCI involves the development and application of principals, guidelines and methods to support the design and evaluation of interactive systems.
- An important concern of HCI is the demands made by the computer on peoples knowledge and understanding, and on the amount of problem solving and learning required.
- The user interface is more than what the person can see, touch or hear. The user interface includes the concepts the user the user needs to know about the computer system and how it can be used to carry pout different tasks.
- Interaction involves task-orientated dialogues between the user and the program running the computer. The program must communicate to the user the results and feedback of activities that are being carried out, as well as requests for further actions. The user must instruct the program and enter data.
- HCI is a multi-discipline field; this means that it uses knowledge from other areas in order to come to its own conclusions. It has much to learn from other areas though the amount of input from each field varies according to the area of HCI and perhaps depending on the individual. The material is often complex and contradictory because human beings are highly complex and some would say contradictory too. Subjects that provide an input into HCI include: Computer Science, AI, Anthropology, Ergonomics, Linguistics, Philosophy, Art, Sociology, Design, Psychology, Engineering, Physiology. See figure 1.1.

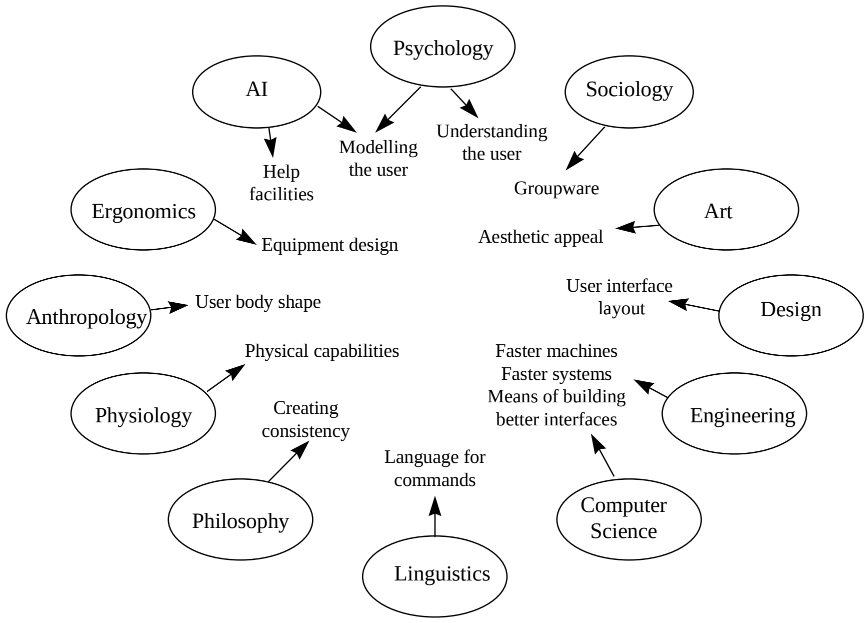


Figure 1.2 - The various disciplines and their contribution

- The main factors that should be taken into account in HCI design are shown in figure 1.2.

Organisational Factors Training, job design, politics, roles, work organisation		**Environmental Factors** Noise, heating, lighting, ventilation
Health & Safety Factors Stress, headaches, musculo-skeletal disorders	**The User** Motivation, enjoyment, satisfaction, experience level	**Comfort Factors** Seating, equipment layout
User Interface Input devices, output devices, dialogue structures, use of colour, icons, commands, graphics, natural language, 3D, multimedia		
Task Factors Easy, complex, novel, task allocation, repetitive, monitoring, skills		
Constraints Costs, timescales, budgets, staff, equipment, building structure		
System Functionality Hardware, software, applications		
Productivity Factors Increase output, increase quality, decrease costs, decrease errors, decrease labor requirements, decrease production time		

Figure 1.2 - Factors in HCI

1.2 WHERE DO WE FIT IN?

- The course is "*User Interface Development*". Its a subset of HCI.
- You already know how to design and write programs → so the purposes of this course is to learn about those aspects of the user interface which we - *the system designer* - can and should actively design.
- This involves the designer in understanding users and what they want to do and in understanding the limitations and potential of the tools available for implementation. It is concerned with the tools and techniques which can be used by software designers in order to promote a user-orientated approach to the design and implementation of the user interface.

1.3 WHY IS THE USER INTERFACE IMPORTANT?

- The user interface is the main point of contact between the user and the computer system - it is the part of the system that the users sees, touches, hears and communicates with.
- Depending on the users experience with the interface, the system may succeed or fail in helping the user carry out a task.
- The types of problem caused by poor user interface design include reduced user productivity, unacceptable learning times and unacceptable error levels, all these factors leading to frustration and potential rejection of the system by the user.
- In general developers of applications software (such as stock or order control systems) dedicate between 50% / 70% of program code to manipulation of the user interface. Thus the time and cost incurred in the development of the user interface is very significant.

- Graphical User Interfaces (GUI) have brought quantifiable benefits to users and organisations. Users make less mistakes, feel less frustrated, suffer less fatigue and are more able to learn for themselves about the operation of new packages than users of non-graphical or character-based user interfaces.
- However, GUIs are more difficult to design, as user interaction is more complex.
- Systems which support groups of users working together (e.g. CSCW) are capable of bringing productivity gains to organisations. The user interface to GroupWare systems are normally graphical, but with the added complexity of supporting communication between group members and common objects. From a software designers point of view, these are more difficult class of systems.
- It is the duty of the system designer(s) to understand the feelings of the user and to make the system so easy and natural to use that the user is able to concentrate on the task alone.

1.4 HCI AND THE SOFTWARE DESIGNER

- The human-computer interface can be thought of as the point at which the activities of the user within an organisation interact with the technical computer system developed by the designer (you!) - see figure 1.3.
    - Users inhabit their own world where they form part of an organisation, made up of groups trying to meet management objectives. Thus the role of the computer system is to help each individual and group in their work.
    - The designer inhabits a technical world where it is difficult to understand the concerns of the user. The designer may not belong to the same organisation, thus the designers organisation may have different objectives.
    - The design process is inevitably a political one because the management view of what will support the users in their work may not be the same as the users own view.
    - Users inhabit a world that is constantly changing, due not only to external factors such as competitors or trade regulations, but also to internal factors such as changes technical strategy or management personnel. Similarly the use of computers systems is not static - users adapt their use of them according to their changing needs.

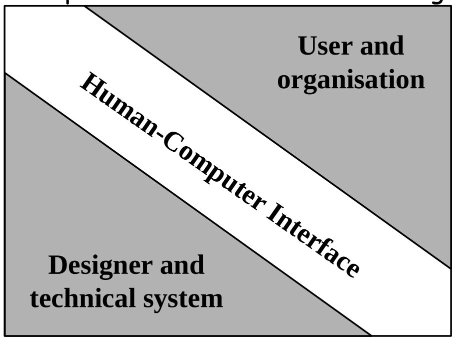


Figure 1.3 - The Human-Computer Interface

- Who builds user interfaces? - Ideally a team of specialists:
    - Graphic designers
    - Interaction / interface designers
    - Technical writers
    - Marketing personnel
    - Test engineers
    - Software engineers

1.5 THE HCI LIFECYCLE

The traditional view of software engineering characterises the development of software as consisting of a number of processes that are produced in a linear fashion. A common example is the Waterfall model - figure 1.4.

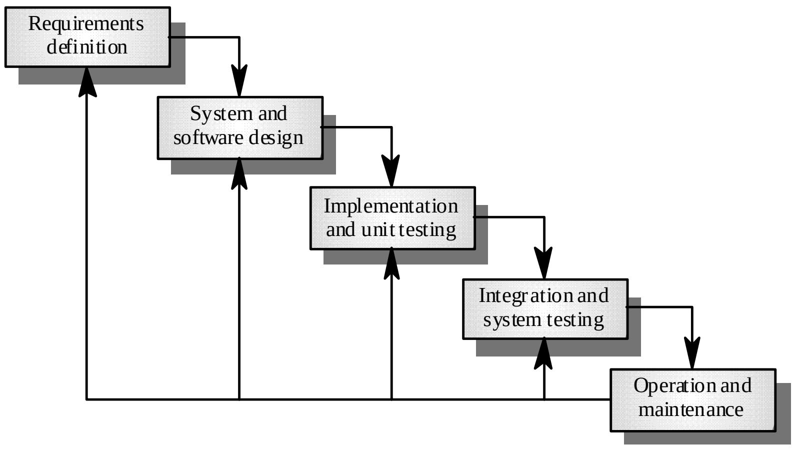

Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

Figure 1.4 - The Waterfall model

How to design and build user interfaces - figure 1.5 & 1.6.

- Design cycle
- User-centred design
- Task analysis
- Rapid prototyping
- Evaluation
- Programming
- Iteration

Figure 1.6 shows the main stages of the HCI design lifecycle

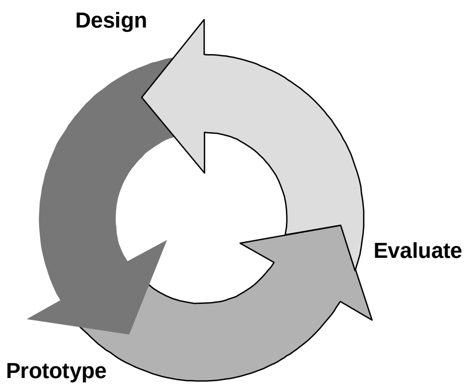


Figure 1.5 - UI design cycle

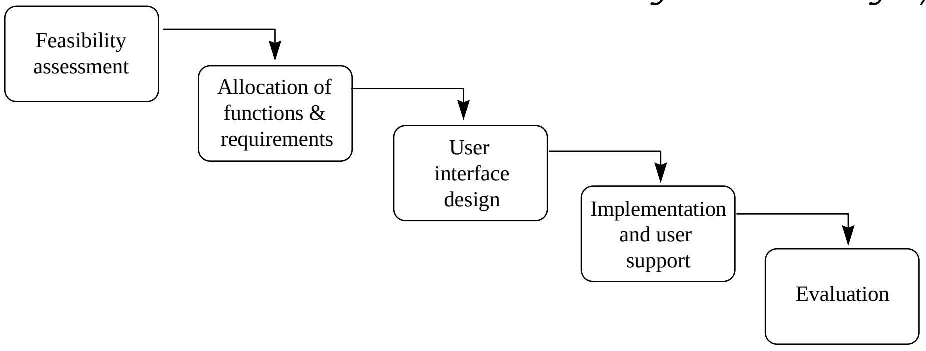

Feasibility assessment
Allocation of functions & requirements
User interface design
Implementation and user support
Evaluation

Figure 1.6 - Stages of HCI Lifecycle

- Some interesting related articles:
  - A.Dix, J.Finlay, G.Abowd and R.Beale, Human Computer Interaction (2/E), Prentice Hall
  - C.Faulkner , The Essence of Human-Computer Interaction, Prentice Hall
  - B.Shneiderman, Designing the User Interface: Strategies for Effective Human Computer Interaction, Addison Wesley Longman

2. Designing the User Interface

2.1 Introduction

- Well designed user interfaces provide a good match between the users task needs, skill level and learning ability and will lead to satisfied and productive users. A good user interface will be easy to learn and easy to use and will encourage users to experiment and try out new features within the system without getting frustrated. Well designed user interfaces will also help considerably towards selling or encouraging the adoption of the computer system, either to managers or to users themselves.
- In order to understand what is meant by a 'good' or 'well-designed' user interface, we need to have some understanding of the classes of user interface commonly available and of their appropriateness for given situations. Interaction between end users and the system is achieved through a interactive dialogues. There are a number of different classes of interactive dialogue and each of these has advantages and disadvantages depending on the situation in which they are used. The are five major classes of user interface are:

2.2 Classes of user interface

- Interaction between end users and the system is achieved through a interactive dialogues. There are a number of different classes of interactive dialogue and each of these has advantages and disadvantages depending on the situation in which they are used. The five major classes are:

1. **Command language** - dialogues are those in which the user types instructions to the computer in a formally-defined command language. E.g. `mv file1 file2` is a UNIX command for copying file1 into file2.
    - The advantages of this approach is that it is very flexible, allowing users to create their own commands; it supports user initiative and it appeals to 'power users', typically to software and systems developers.
    - Command language usually require significant level of training and a high degree of memorisation.
2. **Natural language** - interfaces are those in which the users command language is a significant, well designed subset of some natural language such as English.
    - Natural language interfaces are typically easy to learn, although they often require considerable typing skills on the part of the user. They can also be slow to use if the system is unclear as to the exact meaning of the user request and has to seek clarification.
    - However, natural language systems are increasing in sophistication and a great deal of research and development work is currently being undertaken, particularly in the area of speech recognition.
3. **Menu systems** - allow the user to issue commands by selecting choices from a menu of displayed alternatives.
    - Menu systems are popular since they reduce learning time, reduce the number of keystrokes necessary and structure decision making.
    - Most of the currently available 4GL environments provide design tools which will support the development of menu-based interfaces.

4. **Form filling dialogues** - are those in which the user enters data by filling in fields in one or more forms displayed on the screen.
   - The user of forms on the screen considerably simplifies data entry and requires very little training to use. Forms management tools, similar to those available for menus, can be found within 4GL environments.
5. **Direct manipulation interfaces** - are those in which the user manipulates, through button pushes and movements of a pointing device such as a mouse, a graphic or iconic representation of underlying data.
   - Most direct manipulation interfaces use window systems or environments, in which the users screen is divided into a number of possibly overlapping rectangular areas, each of which handles a specific function.
   - Direct manipulation interfaces represent task concepts visually, are easy to lean and use, encourage exploration or experimentation with the system features and generally result in a high level of user satisfaction.
   - Such interfaces are traditionally difficult to design and to program. However, most of the user interfaces standards currently being put forward are based on direct manipulation interfaces.

- Choosing the most appropriate class of user interface to match the needs and expectations of the users is an important aspect of good user interface design. For any given class of user interface a number of design decisions must be made by the interface designer, particularly in terms of what information should appear on the screen, how much information, in what order, what type of error messages, where error message should be displayed, etc.
- To assist in making the "right" design decisions and achieving a good user interface, a number of design guidelines are available.

2.3 PRINCIPLES OF GOOD DESIGN

- Much guidance is available in the literature in the form of interfaces design guidelines or principals. The major guidelines common of many of the texts can be summarised into five categories:

1. **Naturalness** - A natural dialogue is one which does not cause the user to alter significantly his/her approach to the task in order to interact with the system.
   - In this case the ordering of the dialogue is important. The ordering of the user input, for example, should be geared towards the normal order of working of the user rather than whatever is easier for the programmer. This requires careful study of the tasks the user undertakes before, during and after each interactive session.
   - The use of language which is natural to the user is also important. Jargon may be desirable, provided that it is the jargon of used every day in the users department/company and not that used in the computer department/software house.
   - Phrasing should be self explanatory - for example, words such as 'print', 'end' and 'copy' have obvious meanings, whereas 'pip' (the CP/M keyword for copy) and 'mv' (the UNIX keyword for rename) do not.
   - Use of non-standard abbreviations should be avoided since they slow down word recognition and introduce unnecessary stress.

2. **Consistency** - of dialogue ensure that expectations which the user builds up through using one part of the system are not frustrated by idiosyncratic changes in the convention. For example - in one system 'list' usually meant display on the screen, but occasionally meant output to a printer.
    - Consistent layout for screens which fulfil a similar function ensures that the user knows where to look for instructions and norms.
3. **Non-redundancy** - A non-redundant dialogue requires the user to input only the minimum information for the system's operation.
    - A user should not be asked to give information which can be automatically derived by the system or which has been entered previously. Default values can be used to minimize the amount of input.
    - Similarly redundant information should not be output. Too much information on one screen is detrimental to the clarity of the screen and will delay the user unnecessarily when they try to 'spot' a particular field or item.
4. **Supportiveness** - The supportiveness of a dialogue refers to the amount of assistance the dialogue provides to the user in running the system.
    - It has three major aspects: the quantity and quality of instructions provided; the nature of the error messages produced and the confirmation of what the system is doing. Instructions to the user are provided by both the systems prompts and any addition help facility.
    - Error messages should be helpful and not obscure, for example, 'syntax error' is not at all helpful. At very least the system should tell the user what caused the syntax error, for example 'missing ; on line 42'.
    - Inputs should be confirmed: if their acceptance will result in an irreversible action.; if a code has been entered and the user has to check the associated description or when confirmation of complete details is desirable.
5. **Flexibility** - of a dialogue refers to how well it can cater for or tolerate different levels of user familiarity and performance. This depends largely on the skill and expertise of the user in relation to a given situation
    - Different types of dialogue may be used in different situations; for example, a hierarchical menu structure for use by a first time user may be navigated using commands and parameters once the user becomes more experienced.

2.4 COMMERCIAL EXAMPLE - IBM

- The following design principles are published by IBM and are combination of traditional wisdom with extensions to address the evolution of future interfaces. These design principles are based on IBM's experiences in user interface design, on the design experiences of others, and on insights from linguistics and psychology. They have extended these design principles to address evolving interfaces that will provide a more friendly appearance and behaviour in the future as well as the blossoming popularity of the Internet and the World Wide Web.

- **Simplicity: Don't compromise usability for function**
- Keep the interface simple and straightforward. Users benefit from function that is easily accessible and usable. A poorly organised interface cluttered with many advanced functions distracts users from accomplishing their everyday tasks. A well-organised interface that supports the user's tasks fades into the background and allows the user to work efficiently.

- Basic functions should be immediately apparent, while advanced functions may be less obvious to new users. Function should be included only if a task analysis shows it is needed. Therefore, keep the number of objects and actions to a minimum while still allowing users to accomplish their tasks.

- **Support: Place the user in control and provide proactive assistance**
- To give users control over the system, enable them to accomplish tasks using any sequence of steps that they would naturally use. Don't limit them by artificially restricting their choices to your notion of the "correct" sequence.
- The system should also allow users to establish and maintain a working context, or frame of reference. The current state of the system and the actions that users can perform should be obvious. Users should be able to leave their systems for a moment or a day and find the systems in the same familiar state when they return. This contextual framework contributes to their feeling of stability.
- Most users perform a variety of tasks, being expert at some and novice at others. In addition to providing assistance when requested, the system should recognize and anticipate the user's goals, and offer assistance to make the task easier. Ideally, assistance should provide users with knowledge that will allow them to accomplish their tasks quickly. Intelligent assistance is like the training wheels on a bicycle - at some point, most users will want to take them off and go forward on their own. The assistance should allow them to become independent at some point when they choose to be so.

- **Familiarity: Build on users' prior knowledge**
- Allow users to build on prior knowledge, especially knowledge they have gained from experience in the real world. A small amount of knowledge, used consistently throughout an interface, can empower the user to accomplish a large number of tasks. Concepts and techniques can be learned once and then applied in a variety of situations. Users should not have to learn new things to perform familiar tasks. The use of concepts and techniques that users already understand from their real world experiences allows them to get started quickly and make progress immediately.
- The metaphors used in today's user interfaces tend to be inadequate when compared to the real world. Through the use of visuals and interaction techniques that more closely resemble users' real world experiences, there should be little need to continue reliance on such metaphors.
- In the past, designers tended to invoke a principle of consistency when no single design alternative appeared to be the best answer. By choosing to be consistent with something the user already understands, an interface can be made easier to learn, more productive, and even fun to use.
- Avoid the tendency to employ consistency without understanding your users, their tasks, and their shared experiences. When choosing a dimension within which to be consistent, seek to understand what the user expects and be consistent with those expectations. Providing a familiar experience is the ultimate use of consistency in which a truly intuitive interface will result.

- **Obviousness: Make objects and their controls visible and intuitive**
- Where you can, use real-world representations in the interface. Real-world representations and natural interactions (direct action) give the interface a familiar look and feel and can

make it more intuitive to learn and use. Icons and windows were early attempts to draw on user experiences outside the computing domain. As we move toward real-world representations, reliance on such computer artifacts should decline.

- In an object-oriented interface the objects and concepts presented to users parallel familiar things from the real world; for example:
  - Trash can - when we throw things away we usually use some type of trash receptacle or "trash can". An object on the desktop displayed as a trash can communicates to users that it is a place for discarding things. It should look like the real object rather than like an abstract container, and the user should be able to show its contents in a meaningful way.
  - Telephone - the actions we take with telephones are so familiar to most of us that they require little thought. A telephone object on the desktop indicates to users that it will allow them to perform phone-related tasks, and users will expect it to behave like the real thing.
  - The controls of the system should be clearly visible and their functions identifiable. Visual representations provide cues and reminders that help users understand roles, remember relationships, and recognise what the computer is doing. For example, the numbered buttons on the telephone object indicate that they can be used to key in a telephone number.

- **Encouragement: Make actions predictable and reversible**
- A user's actions should cause the results the user expects. In order to meet those expectations, the designer must understand the user's tasks, goals, and mental model. Use terms and images that match users' task experience, and that help users understand the objects and their roles and relationships in accomplishing tasks.
- Users should feel confident in exploring, knowing they can try an action, view the result, and undo the action if the result is unacceptable. Users feel more comfortable with interfaces in which their actions do not cause irreversible consequences.
- Even seemingly trivial user actions, such as deselection or moving objects, should be reversible. For example, a user who spends several minutes deliberating and selecting individual files to be archived from a group will be very upset if all the files are accidentally deselected and the deselection cannot be undone.
- Avoid bundling actions together, because the user may not anticipate the side effect. For example, if a user chooses to cancel a request to send a note, only the send request should be cancelled. Do not bundle another action, such as deletion of the note, with the cancel request. Rather than implementing composite actions, make actions independent and provide ways to allow users to combine them when they wish.

- **Satisfaction: Create a feeling of progress and achievement**
- Allow the user to make uninterrupted progress and enjoy a sense of accomplishment. Reflect the results of actions immediately; any delay intrudes on users' tasks and erodes confidence in the system. Immediate feedback allows users to assess whether the results were what they expected and to take alternative action immediately. For example, when a user chooses a new font, the font of all applicable text, or of sample text, should change immediately. The user can then decide if the effect is what was desired and, if not, can change it before switching attention to something else.

- Offer a preview of the results of an action when it would be inconvenient for a user to apply the action and then reverse it. For example, if a user wants to bold, underscore, and use Helvetica font in certain places throughout a document, provide a sample part of that document with those changes applied, allowing the user to decide if that is the right action to take. This saves the user a lot of time by not having to reverse the action that's been applied to an entire document and enhances the user's confidence in the system.
- Avoid situations where users may be working with information that is not up-to-date. Information should be updated immediately or refreshed as soon as possible so that users are not making incorrect decisions or assumptions. If, for some reason, the results of a refresh cannot be displayed immediately, the situation should be communicated to users. This becomes especially important in networked environments where it is more difficult to maintain state between networked systems dynamically. For example, most Web browsers display a completion percentage in the information area so that users know the progress of the graphics loading process.

- **Availability: Make all objects available at all times**
- Users should be able to use all of their objects in any sequence and at any time. Avoid the use of modes, those states of the interface in which normally available actions are no longer available, or in which an action causes different results than it normally does.
- Modes restrict the user's ability to interact with the system. For example, one of the most common uses of modes in menu-driven systems is the modal dialog box (such as "Print" and "Save as") used to request command parameters. Modal dialogs tend to lock users out of their system; to continue, users must complete - or cancel - the modal dialog. If users need to refer to something in an underlying window to complete the dialog, they must cancel the dialog, access the information they need and re-invoke the dialog.

- **Safety: Keep the user out of trouble**
- Users should be protected from making errors. The burden of keeping the user out of trouble rests on the designer. The interface should provide visual cues, reminders, lists of choices, and other aids, either automatically or on request. Humans are much better at recognition than recall. Contextual and hover help, as well as agents, can provide supplemental assistance. Simply stated, eliminate the opportunity for user error and confusion.
- Users should never have to rely on their own memory for something the system already knows, such as previous settings, file names, and other interface details. If the information is in the system in any form, the system should provide it.
- Two-way communication may be necessary at times to allow users to clarify or confirm requests, or to remedy a problem. In the past, many interfaces have treated communication with users as primarily one-way, computer-to-user. The communication should be interactive - as rich in presentation and interaction capabilities as the rest of the interface. It should present relevant information, provide access to related information and help, and allow users to make task-specific decisions to continue. For instance, spell check, as designed in some systems, highlights potentially misspelled words as users work, allowing them to either select a new word or continue to work until they reach a point where they can go back and validate the potentially misspelled words.
- Adopt the following design perspective: users know what they want to accomplish, but sometimes they find it difficult to express their desires using the objects and actions

provided, and the system is unable to recognise their request. Two-way communication may be used to help users reach their goals.

- **Versatility: Support alternate interaction techniques**
- Allow users to choose the method of interaction that is most appropriate to their situation. Interfaces that are flexible in this way are able to accommodate a wide range of user skills, physical abilities, interactions, and usage environments.
- Each interaction device is optimised for certain uses or users and may be more convenient in one situation than another. For example, a microphone used with voice-recognition software can be helpful for fast entry of text or in a hands-free environment. Pen input is helpful for people who sketch, and mouse input works well for precisely indicating a selection. Alternative output formats, such as computer-generated voice output for foreign language instruction, are useful for some purposes. No single method is best for every situation.
- Users should be allowed to switch between methods to accomplish a single interaction. For example, allow the user to swipe-select using the mouse, then to adjust the selection using the keyboard. At the same time, users should not be required to alternate between input devices to accomplish what they perceive as a single step or a series of related steps in a task. For example, it would be tedious to require the use of a mouse for scrolling while editing text from the keyboard. Users should be able to complete an entire useful sequence through the same input device.
- Providing a range of interaction techniques recognises that users are individuals with different abilities and situations. The differences include disabilities, preferences, and work environments.

- **Personalization: Allow users to customise**
- The interface should be tailorable to individual users' needs and desires. No two users are exactly alike. Users have varying backgrounds, interests, motivations, levels of experience, and physical abilities. Customisation can help make an interface feel comfortable and familiar.
- Personalising a computer interface can also lead to higher productivity and user satisfaction. For example, allowing users to change default values can save them time and hassle when accessing frequently used functions.
- In an environment where multiple users are using a shared machine, allow the users to create their own system personality and make it easy to reset the system. In an environment where one user may be using many computers, make personalization information portable so the user can carry that "personality" from one system to another.

- **Affinity: Bring objects to life through good visual design**
- The goal of visual design in the user interface is to surface to the user in a cohesive manner all aspects of the design principles. Visual design should support the user model and communicate the function of that model without ambiguities. Visual design should not be the "icing on the cake" but an integral part of the design process. The final result should be an intuitive and familiar representation that is second nature to users.
- The following are visual design principles that promote clarity and visual simplicity in the interface:

- Subtractive design - reduce clutter by eliminating any visual element that doesn't contribute directly to visual communication.
- Visual hierarchy - by understanding the importance of users' tasks, establish a hierarchy of these tasks visually. An important object can be given extra visual prominence. Relative position and contrast in colour and size can be used.
- Affordance - when users can easily determine the action that should be taken with an object, that object displays good affordance. Objects with good affordance usually mimic real world objects.
- Visual scheme - design a visual scheme that maps to the user model and lets the user customise the interface. Do not eliminate extra space in your image just to save space. Use white space to provide visual "breathing room."

2.5 More Design Guidelines

2.5.1 Some guidelines on the use of colour

- The correct use of colour in screen design often causes difficulty for software designers since they often lack the training and visual perception of the graphic designer or artist. There are some major points on how to approach the inclusion of colour into a design, the some specific guidance concerning the amount of colour, background colours, user needs and choice of colour
    1. 'get is right in black and white' and then add colour sparingly.
    2. Develop a plan of when, where and why colour should be used. The plan should cover the 'suite' of screens/windows that are to be designed. Be consistent.
    3. Test colour designs using palettes available on the target computer systems - colour designs on paper may not transfer well to the system. Colours which work well together on one platform may not work well on another.
- Amount of colour
    - Use the minimum amount of colours and not more than 3 or 4 per screen.
    - Do not overuse colours. The benefits of colour as an attention getter, information grouper and value assigner are lost if too many colours are used.
- Background colours
    - Use background colours in large blocks.
    - Group related items by using a common background colour.
    - Use bright colours for emphasis and weaker colours for background areas.
    - Not all colours are equally readable. Extreme care should be exercised with text colour relative to background colours. As a rule, the darker colours such as blue, magenta and brown make good background colours.
- User needs
    - Use colour coding consistent with user expectations.
    - Similar colours should denote similar meanings.
    - To avoid frequent refocusing and visual fatigue, extreme colour pairs such as red and blue or yellow and purple should be avoided.
    - Older users may need brighter levels to distinguish colours.
    - Colour blind users may not be able to distinguish some colour combinations, for example, read and green should be avoided.

- Choice of colours
  - Brightness and saturation draw attention.
  - Link the degree of colour change to event magnitude.
  - Colours change appearance as ambient light levels change.
  - Opponent colours go well together. Yellow and blue are good combinations.
  - The user could be allowed to choose their own preferred colour combinations, which may be helpful in cases of visual handicap.
- These are suggestions only - there are too many variable in colour display, colour copying and human interpretation to make hard and fast rules. Plan the use of colours, experiment on the target system and test designs with users.

2.5.2 OTHER GUIDELINES

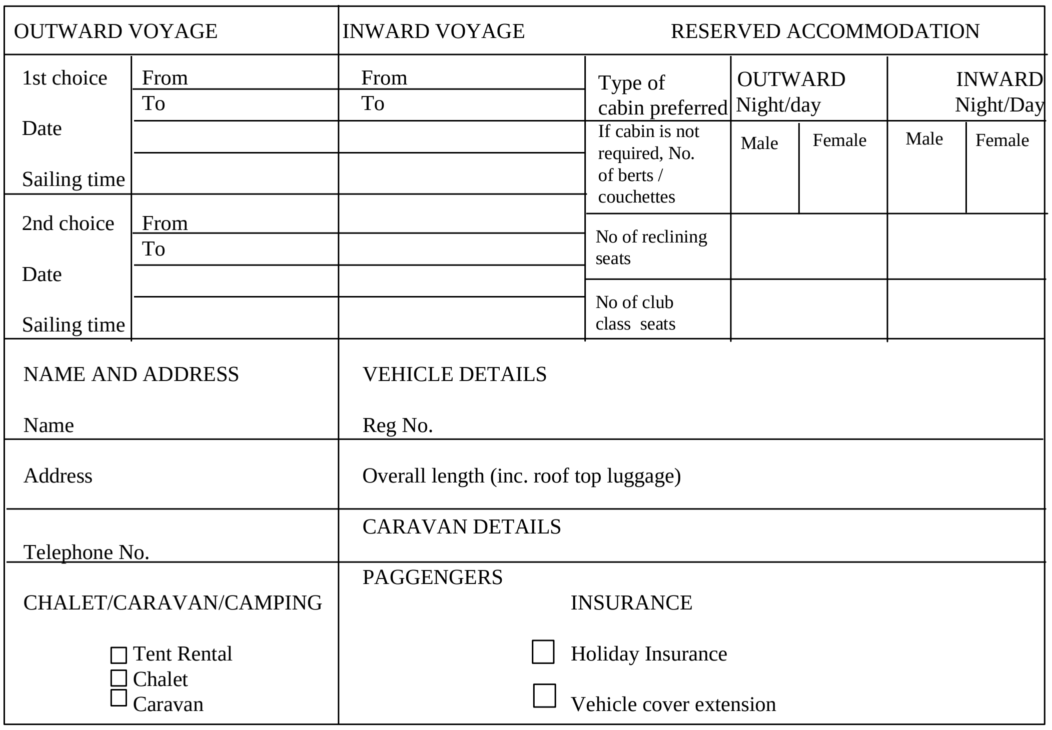
- **Menu design**
  - Menu lists should always be in logical order and in logical groupings
  - Hierarchical menus should be in logical groupings
  - Each menu should permit only one selection by the user
  - Each new option should be on a new line
  - Selection should be by pointing device where possible
  - Wording of menu options should be commands not questions
    - Selection should be by initial letter, not arbitrary code
- **Information presentation**
  - Use upper and lower case (as in road signs)
  - Use normal conventions, e.g. accountants easily recognise negative values which are displayed in red or in brackets.
  - Caption names should be as brief as possible but with meaningful abbreviations.
  - Captions should be positioned in a natural and consistent physical relationship to the corresponding data fields.
- **Data entry**
  - Explicit entry: users should be asked to check data before entry
  - Explicit movement between fields: users should tab or Return or some other key to move explicitly from field to field
  - Explicit delete: users should always be required to confirm any request for deletion.
  - Provide an undo: whenever possible allow users to backtrack to a 'previous state' i.e. undo their last action
- **Positioning of text**
  - Leave approximately half the total screen (window) black
  - Every screen (and window) should be self contained
  - There should be an obvious starting point - usually top left - then proceed left to right and top to bottom.
  - The same information should be displayed in a consistent and predictable relative position on the screen throughout the application.
- **Design of windows - The advantages of multiple windows include:**
  - Allowing users access to multiple sources of information
  - Allowing information to be viewed from different perspectives, e.g. debugging a program
  - Allow the user to examine the same information at different levels of detail

- Allowing the system to attract the users attention, e.g. by displaying a new windows in the middle of the screen
- Allowing the user to control multiple concurrent tasks in an environment where multitasking is provided

- **Design of windows - The disadvantages of multiple windows include:**
  - The dangers of 'overcrowding' on the screen
  - Distraction from the task in hand by causing the user to manipulate the interface in order to obtain the information required
- **Design of windows - Some guidelines**
  - The contents of a window should form a logically relate group
  - The borders of each window should be clearly delimited
  - Avoid filling the screen with a multiplicity of small windows
  - Windows should appear initially in a consistent position and have a consistent size
  - The default position of each window should be adjusted to reflect users preference
  - The contents of each window and of each screen should reflect a logical ordering and consistent format
  - The spatial positioning of windows on the screen should reflect a logical ordering
  - Use of colours across the whole screen should be minimal and consistent
  - Allow 'poping-up' of windows to attract users attention

2.6 Evaluate design using the principles

- The five principles of good design can also be used to evaluate existing user interfaces. For example figure 2.2 shows a form which is completed by travel agents wishes to book a ferry. Figure 2.3 shows a screen design which was used in the automated booking system. This screen design is obviously unsatisfactory, but why? The five principles can be used as a basis for identifying the problems.



OUTWARD VOYAGE		INWARD VOYAGE	RESERVED ACCOMMODATION				
1st choice	From	From	Type of cabin preferred	OUTWARD Night/day		INWARD Night/Day	
	To	To	If cabin is not required, No. of berts / couchettes	Male	Female	Male	Female
Date							
Sailing time							
2nd choice	From		No of reclining seats				
	To						
Date			No of club class seats				
Sailing time							

NAME AND ADDRESS Name	VEHICLE DETAILS Reg No.
Address	Overall length (inc. roof top luggage)
Telephone No.	CARAVAN DETAILS
CHALET/CARAVAN/CAMPING ☐ Tent Rental ☐ Chalet ☐ Caravan	PAGGENGERS INSURANCE ☐ Holiday Insurance ☐ Vehicle cover extension

Figure 2.2 - Ferry reservation form

- Conduct an evaluation of the user interface using the principles above. These will be discussed during the lecture.

```
FAST FERRIES LTD

            OUTWARD VOYAGE              INWARD VOYAGE      RESERVED ACCOMMODATION
1ST CHOICE FROM STN TO SAM  FROM DIP TO WEY    CABIN       OUT   DAY    IN   NIGHT
                1105   891031            2230   891222                 BTHS/CHTS              1M 2F

1ST CHOICE FROM STN TO SAM  FROM DIP TO STN    RECLIN      0             0
   COMFIRMED 1105 891031    2330 891222        CLIUB SEATS 0                  0
NAME AND ADDRESS                               VEHICLE DETAILS
NAME MRS J.BLOGGS                              REGNO 99D4427
ADDRESS 10 GLASNEVIN WAY                       OVERALL LENGTH 3.4M   HEIGHT Y
            DUBLIN 9                           MOTORCYCLE REGNO
TELEPHONE 087212762                            PASSENGERS: ADULTS 3  CHILDREN 0
CHALET/CARAVAN/CAMPING                         INSURANCE
TENT N                                         HOLIDAY Y
CHALET N                                       VEHICLE N
CARAVAN Y

CONFIRM?
```

Figure 2.3 - Screen design for figure 2.2

2.7 Design Models

- Models facilitate understanding users, analysing complex systems, and describing effective designs. The use of three models contributes to the design of easy-to-use computer systems: the user's conceptual model, the designer's model, and the programmer's model. Here we provide an understanding of the three models, including how they are used and the relationships between them.
- Here we see three models that are relevant to the design and implementation of a user interface. Each model provides a different perspective on the interface, beginning with the end user's perspective, and including the designer's perspective, and the implementing programmer's perspective.
  - The user's conceptual model represents what the user thinks is happening and why
  - The user interface designer's model describes what the user is intended to experience
  - The programmer's model describes implementation details

2.7.1 THE USER'S CONCEPTUAL MODEL

- The user's conceptual model of a system is a mental image that each user subconsciously forms as he or she interacts with the system. People create mental models by putting together sets of perceived rules and patterns in a way that explains a situation. A typical person cannot draw or describe his or her mental models and in many situations the person is not even aware that these mental models exist.
- A mental model does not necessarily reflect a situation and its components accurately. Still, a mental model can help people predict what will happen next in a given situation, and it serves as a framework for analysis, understanding, and decision-making.
- The user's conceptual model is based on each user's expectations and understanding of what a system provides in terms of functions and objects, how the system responds when the user interacts with it, and the goals the user wants to accomplish during that interaction. These

expectations, understandings, and goals are influenced by the user's experiences, including interaction with other systems, such as typewriters, calculators, and video games.

- Because each user's conceptual model is influenced by different experiences, no two conceptual models are likely to be the same. Each user looks at a user interface from a slightly different perspective.
- The problem for the interface designer is to design an interface that users find predictable and intuitive when each user is approaching the interface from a different perspective. To come as close as possible to matching users' conceptual models, designers should find out as much as they can about users' skills, motivations, the tasks they perform, and their expectations. This process involves: using
  - resources such as task analysis, surveys, customer visits, and user requirements lists
  - incorporating information that users provide into the user interface design
  - conducting usability tests

2.7.2 THE DESIGNER'S CONCEPTUAL MODEL

- The interface components and relationships intended to be seen by users and intended to become part of each user's conceptual model are described in the designer's model. This model represents the designer's intent in terms of components users will see and how they will use the components to accomplish their tasks.
- The designer's model identifies objects, how those objects are represented to users, and how users interact with those objects. User oriented objects are defined in terms of properties, behaviours, and relationships with other objects. Differences in properties and behaviours are the basis for class distinctions, such as the distinctions between folders and documents. Relationships between objects affect how they are used in accomplishing users' tasks. For example, users can use folders to contain and organise memos, reports, charts, tables, and many other classes of objects. Users can discard an object by dragging and dropping the object's icon on a wastebasket icon, and users can print an object by dropping the object's icon on a printer icon. These actions are logical in that they maintain real-world relationships between objects.
- If the designer's model closely matches a user's conceptual model, the user should learn quickly and apply knowledge correctly in new situations. In other words, the user will feel the interface is intuitive. Designers can help users to develop a closely matching conceptual model by creating a clear and concise designer's model. A designer's model is clear and concise when it has made a minimum number of distinctions among objects, the distinctions are clear and useful to users, and they are consistently conveyed throughout the interface.
- For the designer's model to be consistent with the user's conceptual model, the designer must know the users, their tasks, and their expectations. If designers do not understand their users, the interface will not behave as users will expect it to. If the system does not behave as users expect it to, their conceptual models will be different from the designer's model and misunderstandings will occur. Users can lose confidence in the reliability of their conceptual model, and thus in the system itself, when these misunderstandings occur. If users form an incorrect conclusion or a superstition to explain an inconsistency, they may try to apply it elsewhere in the system. This can lead to further misunderstandings and distrust of the system.

2.7.3 THE IMPLEMENTOR'S CONCEPTUAL MODEL

- The implementor's model describes the system internals used to implement the designer's model. The implementor's model includes details relevant only to programmers and others who develop the product.
- For example, the designer's model might include a directory object that consists of people's names, addresses, office numbers, and so forth. The implementor's model of the directory object might consist of records in a file, with one record for each directory entry; or, it could be a complex organisation of multiple records from multiple files.
- These implementation details from the implementor's model should not be evident in the designer's model and are therefore transparent to users.

- Some interesting related articles:
- D.Diaper, Task Analysis for Human Computer Interaction, Ellis Horwood
- Clayton Lewis and John Rieman , Task-Centred User Interface Design - A Practical Introduction, On-line shareware book – see URL on module web page or directly with authors ftp site ftp://ftp.cs.colorado.edu/pub/distribs/clewis/HCI-Design-Book/

3. *UNDERSTANDING USER NEEDS*

3.1 INTRODUCTION

- Developing an understanding of users, what they want and what they will do as a result of using the proposed system is fundamental to the software designer. Without this understanding the designer will not be in a position to specify and design a "usable" user interface.
- This is a complex problem with may questions
- *What should the system do in order to meet the needs of the user?*
- *Who is the user?*
- *Will the person now doing the job the system is intended to support be the one to use the system, or will this job be fundamentally changed by the system?*
- Some would argue that the above questions are not questions for the software designer to ask and in fact a detailed specification of requirements should have been developed before the designer becomes involved.

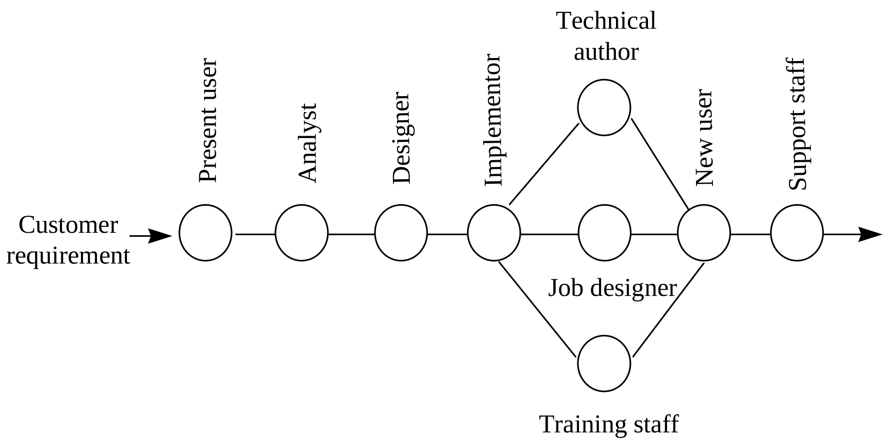


Figure 3.1 - Linear communications structure causes problems for designers & users

- The traditional approach to understanding users needs is to think of the systems analyst as responsible for "eliciting" requirements from users. Figure 3.1 shows this traditional *linear* human communication structure.
- A major problem with this approach is that is often leads to what is referred to as the "over the wall" syndrome, where each person will only accept responsibility for their part of the system and no one is responsible for the overall success of the system.
- A problem with the linear communications structure is that the human perception of what the system should do changes as it is passed from one person to the next - *Chinese Whispers effect.*
- Thus it is argued that for the designer to achieve a 'good' user interface design, participation in the process of understanding users and evolving requirements is essential.
- In order to design an effective user interface, it is necessary to answer the basic questions:
  - Who is the user?
  - What is the task?
  - What is the environment in which the system will operate?
- Often 'system requirements' do no pre-exist as such. What does exist, however, is a perceived need for change and what has to be developed is a vision of that change.

3.2 Understanding users needs

- Before a designer can begin system design, they must have an understanding of the of the future system that needs to be designed.
- How can the designer know what the future system is before deign begins?
- The designer needs to have a 'vision' of the future system and knowledge and understanding of the target users, what they do now and what they will do in the future and knowledge of the technological options available for building the future system.
- The designer needs to acquire three areas of knowledge before beginning the design task:
  1. Knowledge of users present job
  2. Knowledge of the possible technological options
  3. Knowledge of the future system, including the proposed new jobs

Figure 3.2 shows a simplified view of the system design process. It illustrates the fact that knowledge of the users present job and of the technological options acts as input to the system design and that the output is the proposed new system.

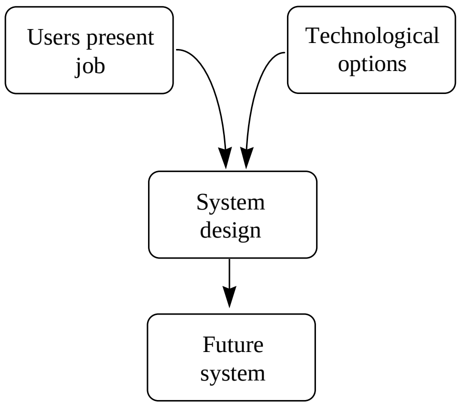


Figure 3.2 - Areas of knowledge

3.3 What is User-Centred Design?

- How do designers come up with an interface that does what you want, and doesn't make you waste time doing what it wants? Easy-to-use software doesn't just happen. It requires focusing on the product's potential users from the very beginning, and checking at each step of the way with these users to be sure they will like and be comfortable with the final design. The User-Centred Design (UCD) process starts by forming a multi-disciplinary UCD project team. This team will work with the product's users throughout the design process and beyond. So the first thing that the UCD team must figure out is: Who will be using the product?

- Once this target audience has been identified, representative users can be recruited to work with the team. These users help establish the requirements for the product by answering questions such as:
  - What do you want the product to do for you?
  - In what sort of environment will you be using the product?
  - What are your priorities when using the software? For example, which functions will you use most often?
- The answers to these questions start the process of user task analysis.
- Another important set of issues concern the product's competition, which includes not only other products but also any other means the target users have for completing their tasks. Again, users are consulted to help designers understand how to make their product competitive:
  - How are you doing these tasks today?
  - What do you like and dislike about the way you've been getting your tasks done?
- When the users' task requirements and the competing methods are understood, the design can start to take shape. A trial set of objects and views is designed to support the main user tasks.
- To test the design so far, the team puts together a preliminary version (or prototype). Prototypes can be as simple as pieces of paper with proposed screen designs sketched on them, or so developed that they look like finished products, but most prototypes fall somewhere between these extremes. A prototype may not have all the function that will be in the product, but it has enough to test some part of the design. Test participants recruited from the target audience try out the prototype, and their task performance, reactions, and comments help the designers decide what to keep and what to change about the design. The design goes into a cycle of modification and re-testing until it meets functional and usability criteria.
- At this point, a pre-release, or beta, version of the product may be constructed and distributed to a restricted set of users for their evaluation. Unlike the test prototypes, this version should have all the function planned for the actual product. It also can contain extra software to record usage information, such as how often the users refer to Help or run into problems with the product. The information gathered from users of the beta release can help the UCD team fine-tune the product for its formal release.
- Finally, the tweaking stops and the product is released. But the user input doesn't end there. Users participate in benchmark assessments in which the product is rated against both the users' requirements and its competitive products. Customer service also records and tracks any problems reported by users. The problem reports help the designers know what to improve in the next iteration of the product.
- Throughout the entire development process and beyond, users play a critical role in the design of easy-to-use products. After all, who knows more about which products are easy to use than the people who use them?

3.3.1 User-Centred Design Principles

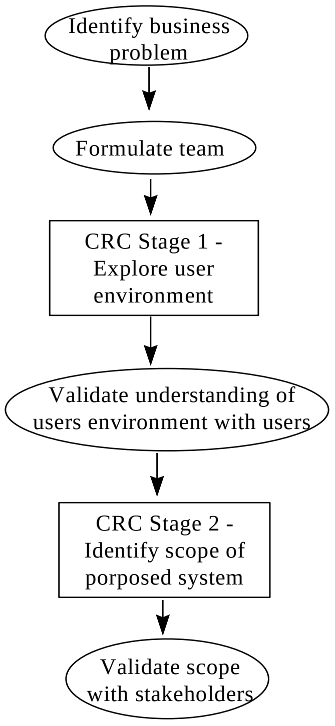
- Meeting the ease of use challenge is largely a matter of adhering to the following principles. For each principle, the goal is to involve users - to ask the right people the right questions. Putting yourself in their shoes is a sure way to put your product at the front of the pack.
- **Set business goals**. Determining the target market, intended users, and primary competition is central to all design and user participation.
- **Understand users**. A commitment to understand and involve the intended user is essential to the design process. If you want a user to understand your product, you must first understand the user.
- **Assess competitiveness**. Superior design requires ongoing awareness of the competition and its customers. Once you understand your users' tasks, you must test those same tasks against competitive alternatives and compare their results with yours.
- **Design the total user experience**. Everything a user sees and touches is designed together by a multidisciplinary team. This includes the way a product is advertised, ordered, bought, packaged, maintained, installed, administered, documented, upgraded and supported.
- **Evaluate designs**. User feedback is gathered early and often, using prototypes of widely ranging fidelity, and this feedback drives product design and development.
- **Manage by continual user observation**. Throughout the life of the product, continue to monitor and listen to your users, and let their feedback inform your responses to market changes and competitive activity.

3.3.2 User-Centred Design Process

- The goal of User-Centred Design process is to ensure that the final product fulfils the users' wishes and needs. To achieve this goal, the first step is to form the multidisciplinary UCD Project Team. The project team includes representatives of the fields of visual or industrial design, human factors, information development, marketing, project management, service and support, technology architecture, and user interface design.
- The Project Team then solicits user input throughout the design process. Below is a description of the six stages in the UCD design process, and some possible methods for gathering user input during each stage.
- **Market Definition**. Define the target audience, identify competitors, and determine the core user needs and wishes that must be fulfilled for the product to succeed.
- Typical methods: ask members of proposed target audiences to rate their levels of interest in a new product or product enhancement; ask target users to list and prioritise their needs and identify current solutions they use and prefer
- **Task Analysis**. Identify and understand the users' goals and tasks, the strategies they use to perform the tasks, the tools they currently use, any problems they experience, and the changes they would like to see in their tasks and tools.
- Typical methods: ask users to list and prioritise tasks; observe users accomplishing their tasks
- **Competitive Evaluation**. Determine the design strengths and weaknesses of the competition.

- Typical methods: ask users to complete the same tasks using different products and assess their overall satisfaction with each one; ask them to list the strengths and weaknesses of products in order of importance
- **Design and Walk-through**. Using the results from task and competitive analyses, create alternative proposed solutions, solicit feedback through design walk-through sessions with users, and choose a solution based on user input.
- Typical methods: ask users to evaluate "lo-fi" prototypes such as simple sketches
- **Evaluation and Validation**. Periodically solicit user feedback on the evolving design, and iterate the design based on analysis of users' experiences with it.
- Typical methods: observe users accomplishing important tasks with a working prototype
- **Benchmark Assessment**. Run a head-to-head benchmark assessment against the competition to verify that the product has met its primary objectives. If a third-party company conducts the benchmark study, positive results can become important selling points in product promotions.
- Typical methods: ask users to complete the same tasks using different products and assess their overall satisfaction with each one; ask them to list the strengths and weaknesses of products in order of importance
- For any single product, the process is usually recursive. For instance, periodic Benchmark Assessments typically uncover changes in the market and new user needs, which leads to a new Market Definition, and the process begins anew.

3.4 CRC – An example


Identify business problem
Formulate team
CRC Stage 1 - Explore user environment
Validate understanding of users environment with users
CRC Stage 2 - Identify scope of porposed system
Validate scope with stakeholders

- Figure 3.3 gives an overview of the CRC method. The rectangular boxes represent face-to-face meetings or workshops. Normally the meeting is supported by a trained facilitator, who guides the team through the main steps of the method and encourages all the stakeholders to participate. The oval boxes represent activities which must take place either before or after a workshop. These usually involve some subset of the stakeholders consulting with others outside the immediate team.

3.4.1 Identify business problem

- It is assumed that there will be some motivation for proposing that a future system be developed. That motivation may come from some specific business need, or the commissioning organisation is planning ahead and the business problem may refer to some future need. In other cases it will only be a small incremental improvement that is needed.
- If the commissioning organisation is a software house or computer company it may be that they are in possession of some innovative technology and need to identify whether target users do have a need for that technology and be able to describe what that need is.
- In all cases a business need is identified, whether it be an improvement in customer services, a future need, a small incremental improvement or the ability to use a certain technology.

- There are many different motivations for proposing that a future system be developed. What is important to the application of the CRC approach is that the business need is articulated and described in such a way that it can be shared with other team members.
- User needs are investigated within the context of the business problem.

3.4.2 Formulate team

- This will normally involve the project manager or initiator and the facilitator in identifying the stakeholders and the requirements capture team.
- Ideally, between six and nine stakeholder representatives will participate in the CRC process.
- The team should draw from each of the four categories of stakeholders.

3.4.3 CRC stage 1 - Explore user environment

- Exploring the users environment means that the requirements capture team must collectively investigate the organisational setting the target users are in and identify and described what target users do.
- The term 'explore' is used because the team is encouraged to 'find out' afresh, to share knowledge about users and to set aside preconceptions about what users need.
- They also assess the likely costs and benefits of change from the users point of view and produce a document recording the shared view of the users environment - figure 3.4.

1. Management summary - (including the business case and a brief description of the proposed system) 2. Organisation / workgroups 2.1 Workgroup control sheets 2.2 Organisation chart 2.3 Workgroup table 2.4 Workgroup description checklists 3. Generic users 3.1 Generic users controls sheets 3.2 Generic users description checklists 7. Consolidation 7.1 State of credibility 7.2 Further investigations	4. Tasks 4.1 Task control sheets 4.2 Task hierarchy 4.3 task description checklists 5. Objects 5.1 Object control sheet 5.2 Object structures 5.3 Object description checklists 6. Interactions 6.1 User / task / object interactions 6.2 Initial list of requirements & attributes 8. Worth proceeding? 8.1 User / stakeholder perspective 8.2 Business perspective 8.3 Plan of action 9. Conclusion

Figure 3.4 - List of contents of the user document

3.4.4 Validate understanding of user environment with users

- Representative users do participate in the workshop but other users will need to be consulted or interviewed to ensure that the team has reliable information about all those users who may be affected by the system.
- After Stage 1, but before Stage 2, the information recorded in the 'user document' should be validated and then expanded or updated where necessary.

- The techniques for validation will depend on the specific problem. For a generic product, further market research may be needed; for a bespoke system, specific user interviews may be necessary.
- It is important to note that at this stage of the development process, highly detailed knowledge of all users tasks may not be necessary. The team needs to have enough reliable information to be able to decide which users, which tasks and which objects need to be computer supported and to decide what extent that support should be. That is they need enough information to decide the scope of the proposed system.

3.4.5 CRC STAGE 2 - IDENTIFY THE SCOPE OF THE PROPOSED SYSTEM

- This usually involves a two day face-to-face meeting of the requirements capture team and the use of checklists. The scope of the proposed system must now be determined at a number of levels.
  - Firstly the stakeholders must decide which jobs are to be affected and what the role of the system should be in supporting each of those jobs. The likely acceptability of this proposed change should then be considered. In addition, for each work role identified, an initial task model is produced, which helps to clarify and consolidate the understanding of the team with respect to specific roles.
  - Secondly, the team is asked to consider which objects from the user environment - that is those contained in the user document - are likely to be of interest to the system. i.e. which objects will the system need to hold information about, which will it need to interact with and which will remain entirely in the user domain.

1. Management summary
   (including the business case and a brief description of the proposed system)
2. The human requirements
   2.1 Description of the objectives of the client organisation
   2.2 List of stakeholders
   2.3 List of key workgroups, users and their objectives
3. The high level functional requirements
   3.1 List of work roles to be supported and why
   3.2 Description of each work role in terms of users, objects and tasks
4. The detailed functional requirements
   4.1 Consolidated list of objects to be supported
   4.2 Descriptions of each object and associated users tasks
5. The quality attributes
   May include usability, reliability, performance, security, etc.
6. Organisation and user assistance requirements
   6.1 Documentation requirements
   6.2 Training requirements
   6.3 User support
   6.4 Human computer interface requirements
7. The technological requirements and constraints
   7.1 Known hardware requirements and constraints
   7.1 Known software constraints (user or supplier)

Figure 3.5 - List of contents of the initial requirements document

- The scope of the proposed system is determined by the extent of support for the work roles and by the list of objects the system will need to support. In addition, both the scope of the system and the list of requirements are reviewed by each of the major stakeholders to make sure their needs are met. Once the scope of the system is decided and the list of requirements reviewed, the team is asked to identify and agree upon usability targets for the proposed system. The outcome from this stage is an 'initial requirements document' containing an agreed set of requirements for the proposed system. Figure 3.5 shows a typical list of contents.

3.4.6 Validate scope with stakeholders

- The scope of the proposed system should be validated.
- A range of techniques could be used as this point depending upon the type of system under consideration. Appropriate techniques include use of questionnaires, interviewing users, building mock-ups, throw away prototypes or holding focus groups.
- Once the scope of the system has been agreed and documented, resources, timescales, tasks, milestones and deliverables can be evolved and the software designer can then proceed with the detailed design.

3.5 Task analysis and allocation of function

- Task analysis is the process of analysing the way people perform their jobs and is important to the software designer because a major part of the design will focus on supporting the jobs people do. A number of techniques exist for task analysis exist, each of which tends to place emphasis on one aspect.
    - *Task decomposition* looks at the way a task is split into subtasks, and at the order in which tasks are performed.
    - *Knowledge based* techniques look at what users need to know about the objects and actions involved in a task and how that knowledge is organised.
    - *Hierarchical* task analysis is used to decompose tasks into subtasks without any specific reference to the order in which tasks are performed.
- In addition to task analysis, the designer should also be concerned with analysis of the objects people use when carrying out their tasks.
- The user carries out a task on an object and can be thought of as the user/object/task triangle in figure 3.6.

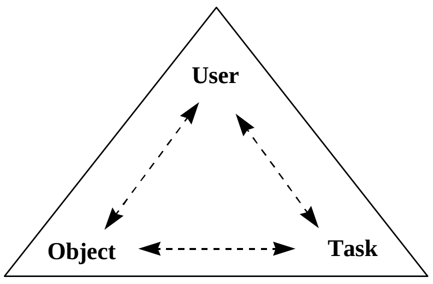


Figure 3.6- user/object/task triangle

3.5.1 Hierarchical task analysis

- Is used in CRC stage 2. The first step is to identify the **invariant tasks** which users carry out. Those tasks which will not change even though the means of achieving them may change. They are fundamental to the organisation.

- Identifying invariant tasks enables the software designer to see which aspects of the users job are fundamental and which can be subject to change - i.e. which aspects can they apply their creative design skills. By taking this approach to task analysis we are more likely to achieve an innovative design for the new system as opposed to simply automating the current way of working.

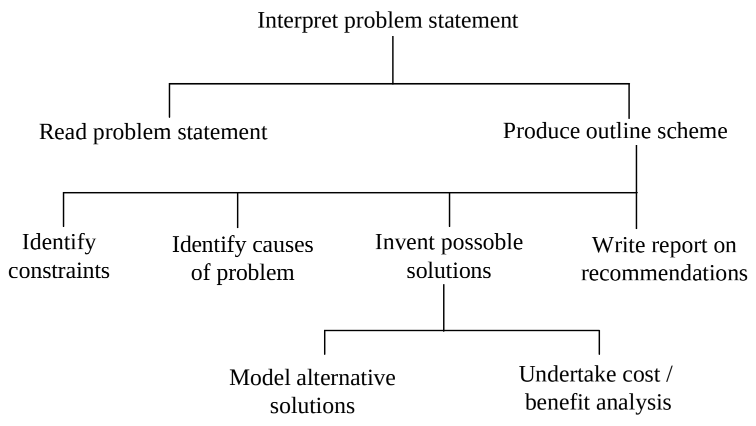

Interpret problem statement
Read problem statement
Produce outline scheme
Identify constraints
Identify causes of problem
Invent possoble solutions
Write report on recommendations
Model alternative solutions
Undertake cost / benefit analysis

Figure 3.7- Example task hierarchy

3.5.2 Allocation of Function

- Allocation of functions between human and computer can be thought of as a spectrum of possible allocations types from the human carrying out a task with little or no support from the computer system, to the computer system carrying out a task wit little or no reference to the human - figure 3.8.
- **Sharing** of tasks is only one issue associated with function allocation. Who or what **controls** the task is also important, as is the type of support the system provides, i.e. **passive** or **active**.
- Making explicit decisions about the type of function allocation is important at this stage in the analysis, since it can be used to assess likely reactions of the various stakeholders to the system. This prevents potential hostile **reactions** from users, managers or other groups.

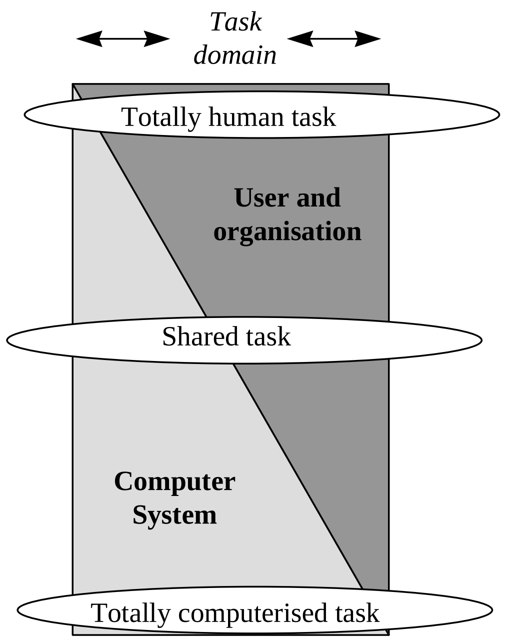


Figure 3.8- Allocation of function

- Function allocation is also important to help assess the alternative **costs and benefits** of the system to the users and stakeholders.
- Some examples of allocations types are:

1. The system replaces the user
2. The system carries out the task, the user provides input only
3. The system provides expert advice, the user inputs parameters and receives the results
4. The user carries out the task, the system controls the rate at which the task is carried out.
5. The system provides passive assistance
6. The system provides active assistance
7. The system provides general task support

3.6 SUMMARY

- The role of the system in supporting the users task should be explicitly decided through the designer considering a range of possible alternative allocations of function.
- Once the role of the system and the role of the user have been decided, the next stage in the HCI lifecycle is to identify those subtasks which will appear at the user interface and to specify the user interface requirements.

- Some interesting related articles:
- Karat, C. (1997). Cost-justifying usability engineering in the software life cycle. In Helander, M., Landauer, T., and Prabhu, P. (Eds), Handbook of Human-Computer Interaction. Elsevier Science, Amsterdam.
- Karat, C. (1990). Cost-benefit analysis of usability engineering techniques. Proceedings of the Human Factors Society. Orlando. Fl.
- Norman, D. (1998). The Invisible Computer: Why Good Products Can Fail. MIT Press, MA.
- Moore, G. (1991). Crossing the Chasm. Harper Business, NY.
- Schlesinger, L.A., and Heskett, J.L. (1991). A service driven service company. Harvard Business Review, 69, 5, 71-81.
- Wildstrom, S. (1998). A computer user's manifesto. In Technology and You, Business Week, Sept. 28.

4. GRAPHICAL USER INTERFACE

4.1 INTRODUCTION

- Graphical User Interfaces (GUI) bring quantifiable benefits to users and organisations. Studies in the mid 90s showed that experienced users of GUI make fewer mistakes, feel less frustrated, suffer less fatigue and are more able to learn for themselves about the operation of new packages than users of non-graphical interfaces.
- The development of applications with GUI differs from the development of traditional approaches in both the technology that is used and in the design itself. It is more difficult to acquire the skills required to design a GUI that will provide user with a natural and intuitive interface. The main tasks are the creation of the specification both of the functions provided by the interface and of its behaviour, and the design of the graphical displays.

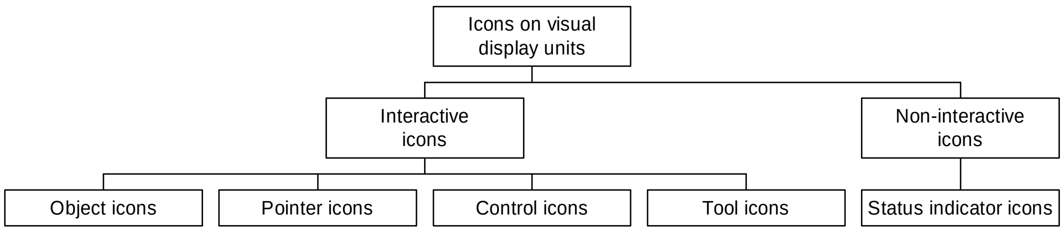
- The purpose of this section is to introduce some of the basic concepts of GUIs and to introduce some of the tools available to aid their effective design and implementation.
- One of the main goals of a GUI is to create the illusion of objects that can be manipulated; for example, moved across a screen and discarded into a 'waste bin'. GUIs can use an object-orientated paradigm where the user indicates an object first an then gives the command.
- Most GUIs have a pointing device, usually a mouse; a bit-mapped display, with a WYSIWYG (What You See Is What You Get) screen where what you can see on the screen is what you get on the printout; windows which can be resized; on-screen menus which appear or disappear under the pointer control; icons, which represent files, folders and devices, and widgets such as dialog boxes, buttons and scroll bars.
- From a users point of view, a GUI has three main elements which contribute to the 'look and feel':
  - **The visual appearance** – This includes items such as the position and layout of the menu and scroll bars, the design of the icons used to represent applications and files, and the shape and size of the mouse controller.
  - **Behavior** – The behavior of an interface is the way in which it responds to actions taken by the user. For example, a double click on a file will result in that file being activated.
  - **Metaphor** – The term 'metaphor' is used to describe the analogy used in the design and implementation of the GUI. The use of metaphor can help users to foresee the consequences of their actions.

4.2 Design of Icons

- An important component of the visual appearance of the GUI is the icons. There have been various attempts to classify icons. For example, the original Xerox Star classification separated icons into data and function icons. A data icon is defined as representing an object on which actions can be performed, for example, documents, folders and files and anything that can be done to one data icon can be done to all, for example copy, move, delete. Function icons, in contrast, represent objects which perform actions, for example, drawers, in-trays, calculators. Most function icons will accept any data icon; for example, the in-tray receives a document and places it in a stack of documents.
- Other researchers distinguish between form and function:
  1. **Resemblance icons** – These depict the underlying referent through an analogous image.
  2. **Exemplar icons** – These depict a typical example of a general class of object.
  3. **Symbolic icons** – These convey the underlying referent at a higher level of abstraction than the image itself.
  4. **Arbitrary icons** – These bear no resemblance to the referent.
- The design of icons is still (relatively) poorly understood, although most current methods advocate user participation in the design of the icon. Some recommend getting the user to 'doodle' in an effort to facilitate visualisation of some aspect of the user environment. Tools which allow icons to be drawn rapidly and prototyped with the user are also available to assist the designer.
- Icons are highly pictorial representations and can be used to represent actual objects such as documents, folders or waste bins. Specific operations can be represented through analogy; for example, the in-tray represents the process of mail arriving for the user, while the out-tray is used to send outgoing mail. Icons can also abstract representations of system

states, such as the hourglass icon to indicate that the system is processing data and the user must wait.

- The ISO suggests as part of its ISO/IEC CD 11581-1.2 1993 standard that there are two main types of icons: interactive icons and non-interactive icons. Interactive icons represent the objects, pointers and tools which mediate user interaction with the software application. Non-interactive icons are normally status indicators.
- Figure 4.1 shows the ISO classification of icons and indicates the parts where icons are further described.



Part 1	Part 3	Part 4	Part 5	Part 6
Document	Selection	Pushbuttons Fill		Hourglass
Folder	Text	Radio buttons	Eraser	Watch
Cabinet	Graphics	Check boxesBox		Thermometer
etc.	etc.	etc.	Pencil	etc.

Figure 4.1 – ISO classification

4.3 USE OF METAPHORS

- Graphical user interfaces were initially targeted at office system users. In an attempt to provide an interface which was natural, consistent and supportive of such users, designers looked at analogies in the clerical performance of similar tasks. For example, spreadsheets are based on the metaphor of the ledger book. Three examples of metaphor are:
- **The physical object metaphor**
  - Here items are represented by the physical object and exhibit real-world behaviors. Thus files are sheaves of paper or folders, directories are drawers in a filing cabinet, while clerical operations involve physical actions on these objects.
  - The physical object metaphor seeks to eliminate the dichotomy between the syntax and semantics of the computer system by modelling the interface on the natural human performance of the task.
- **The desktop metaphor**
  - This suggests that the interface should provide the user with a similar type of flexibility as the desktop; for example, there should be access to several information sources and a variety of formats such as pictures or graphs. The system should permit easy access and should facilitate swapping from one task to another. Tools such as calculators, notepads, diary's, etc. should be available.
- **The travel holiday metaphor**
  - This has been used within learning environments and is based upon the users understanding of tours, guides and navigation. Examples are CAL or CSCL environments.
  - This metaphor should be used with great care since too strict an adherence to a metaphor could lead to an unnecessarily restricted interface.

4.4 GUI DESIGN – AN OBJECT-ORIENTATED APPROACH

- GUIs use an object-action paradigm where the user indicates an object first and then gives the command. Thus when considering the design of graphical user interfaces it is natural to adopt an object-orientated approach.
- Figure 4.2 shows an overview of the procedure which can be followed when taking an object-orientated approach to interface design.
- **Stage 1 Understanding users**
  - This has already been covered. However, two important inputs to this process are the workgroup table and the job issues for each workgroup. These provide descriptions of what the users do now and a basis for identifying objects.
- **Stage 2 List objects associated with all users and workgroups**
  - This has already been covered at an initial level. This is an important stage because any given system will only support one set of objects, no matter how many different users or workgroups use the system. Different users will carry out different actions upon the same object.
  - The tasks of the primary and secondary workgroup should be considered first. These tasks should be set out in a list for each workgroup and all the nouns underlined. Some of these nouns may not be of any relevance to the proposed system but at this stage everything should be included. The outcome of this stage will be a long list of potential objects.

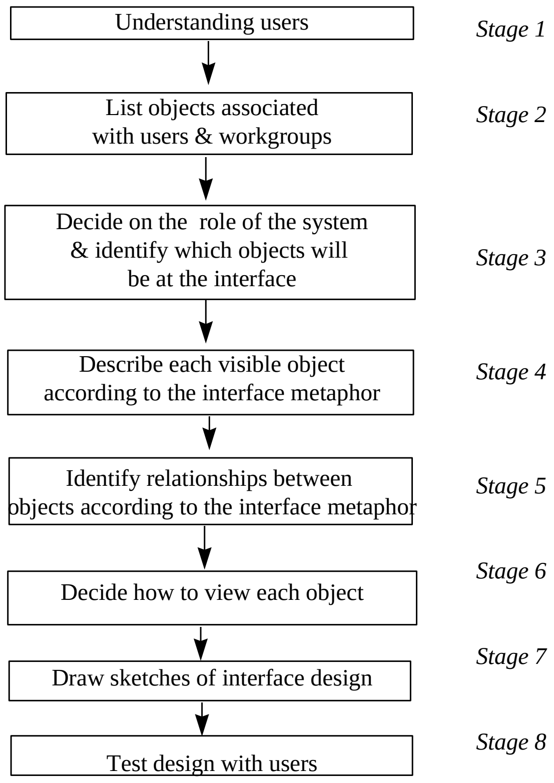

Understanding users
Stage 1
List objects associated with users & workgroups
Stage 2
Decide on the role of the system & identify which objects will be at the interface
Stage 3
Describe each visible object according to the interface metaphor
Stage 4
Identify relationships between objects according to the interface metaphor
Stage 5
Decide how to view each object
Stage 6
Draw sketches of interface design
Stage 7
Test design with users
Stage 8

Figure 4.2 – Stages in object orientated user interface design

- **Stage 3 decide on role of the system and identify visible objects**
  - This is concerned with deciding the role of the system, which object should be supported by the system and which objects will remain as they are now.
  - *3-1 Decide on the role of the system* - There should be some general statement about the role of the system in supporting the users and workgroup, as discussed in section 2.4
  - *3-2 Classify objects* - Once the role of the system has been decided, it is then possible to classify the objects.
    - Objects classified as 1 are to be fully automated and will not be visible at the user interface.
    - Objects classified as 2 will be visible at the user interface and the user will be able to carry out actions on them.
    - Objects classified as 3 will be external to the system and will remain totally under the users responsibility.
    - At this stage, some duplication of objects can normally be identified and thus the list may be reduced.

  - *3-3 Identify aggregations, different names and reports* - Next only those objects with a classification of 1 or 2 need be considered and a list should be drawn up showing just these objects.
    - Aggregations - upon inspection some of these objects may be seen to be attributes of other objects and hence can be aggregated into one object. For example, name, address and phone could be aggregated into customer.
    - Different names - in some cases two objects may be the same object, but there names may be different.
    - Reports - some objects may simply be outputs or reports from the system which can be generated as a result of user actions on other objects. For example, a receipt for payment may have been listed as a object initially but the important object may be the payment, and the user can ask the system to issue a receipt. There is no need to treat the receipt as an object in its own right.
  - *3-4 Revised list of objects* - The outcome of stage 3 is a revised list and usually much shorter list of objects.
- **Stage 4 describe each visible object**
  - The next stage is to describe all the visible objects, that is those objects with classification of 2. Obviously the designer must be constantly reassessing whether this is the 'correct' set of objects, as there is no single answer.
  - *4-1 'Now' and 'proposed' descriptions* - Each object can be described according to its 'now' and 'proposed' characteristics. They are described in terms of who has access to the object, who is responsible for the management of the object and its representation and quality.
  - *4-2 Descriptions according to metaphor adopted* - In addition, each object can be described according to the interface metaphor being adopted.
- **Stage 5 identify relationships between objects**
  - The next stage is to identify relationships between objects, in terms of which object is associated with another object, which one contains another object and which one is sent to another object.
  - The nature of the relationships will also depend of the interface metaphor adopted. In the main, however, this stage involves understanding how the user will interact with the system.
- **Stage 6 decide how to view each object**
  - Next the designer must decide how each object should be viewed in order to give the user the best access to the objects and the information they contain.
  - The designer should produce lists detailing the contents of each menu bar, each pull down menu, each for, table or list. Id addition, icons should be described in terms of allowable users actions and their consequences.
- **Stage 7 draw sketches of the interface design**
  - Hand draw sketches of the interface design or rapidly put-together sketches are useful. Their main purpose is to discuss the proposed system design with the user. The design will probably change as a result of the discussion and so the designer should not spend too much time on drawing a draft design.
- **Stage 8 test the design with users**
  - User reaction should be sought concerning the choice of objects, the views of each object, the allowable user actions on each object and the representation of each object. The design should be modified to reflect user reaction.

5. Evaluation

5.1 Introduction

- It is now generally understood that computer systems should be designed to be 'easy to use'. This is a major factor in marketing. However, what 'easy to use' actually means is less well understood.
- When a product is marked as 'easy to use'. Does it mean it is easy for anyone to use, in any situation, regardless of what they are doing or what knowledge and skills they possess. Has the product been tested for ease of use and if so, with whom and under what circumstances.
- What does easy to use mean? Does it mean that the system will be easy to learn for first time users? Does it mean that experienced users will be able to use the system very quickly? Does it mean that the system is actually effective in supporting users in their job. Do it mean that users enjoy using the system.
- In the HCI community the term usability is preferred to 'ease of use'. The definition of usability includes ease of learning, ease of use, flexibility of use, effectiveness of use and user satisfaction with the system.
- **Usability:**
  - Usability is the effectiveness, efficiency and satisfaction with which specified users can achieve goals in a given environment.
  - Effectiveness is the accuracy and completeness with which users achieve specific goals.
  - Efficiency is the accuracy and completeness of goals achieved in relation to resources expended.
  - Satisfaction is the comfort and acceptability of using the system
- Objectives in usability evaluation
  - How do we know that the product is usable? How can we guarantee usability? We can do this by:
    - defining usability through metrics
    - setting planned levels for usability attributes
    - incorporating user-derived feedback into the design process and redesign
      - repeating all of the above until usability levels are met

5.2 Viewpoints - user, Customer and Designer

- Figure 4.1 shows the customer usage cycle. At each stage in the cycle the customer will have different requirements with respect to the usability of a system.
- As the 'needs assessment' stage, the customer will have identified the need for a new system to support some business requirement. Here the supplier will need to demonstrate the system to the customer and it is critical that that the users can see an effective demonstration of key features. Given that the customer purchases the system, the next stage is installation. The user must unpack and install the system and it is critical that this can be done successfully and in the shortest possible time. Once the system is installed the users must learn to use the system features in order to perform their job. Thus at the 'introduction and training' stage the system must be easy to learn and any supporting materials must be consistent with the software and training courses.

- At the 'limited usage' stage, users will be left alone to use the system and will be expected to perform their jobs effectively. Thus ease of use and speed of completing most frequently-performed tasks become critical. At the next stage, 'full usage', the users will be expected to make full use of all the system features provided and exploit the full potential of the system. Here it is critical that users can explore system features which they are unfamiliar with and that they can find out from the system itself how best to exploit capabilities.
- Once the users have fully exploited the features of the system and their job needs change and develop, the customer goes through a period of 'evaluation'. The limits of the current system have been reached and the customer is looking towards the next stage of computer support, and the next purchase.

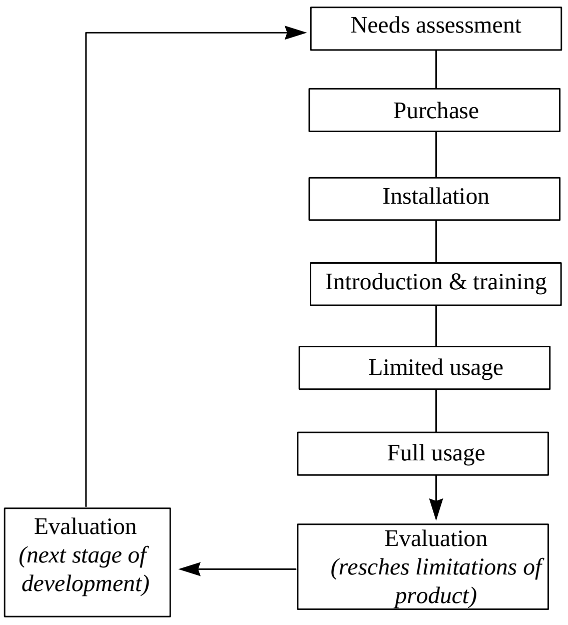


Figure 5.1 – The customer usage cycle

- At each stage in the usage cycle there are different usability requirements. For example, at the introduction stage, ease of learning if important and at the full usage stage, speed of performance is important.
- At each stage the customer is evaluating the system in terms of value for money. Depending on their experience with the system their perception of value for money may increase or decrease.
- Figure 5.2 shows the costs of poor usability to the customer and the supplier. First of all, a cost to the customer is incurred when purchasing the system. Suppose a customer has purchased system Y because it has features they A, B, C, D and E. The system is installed, the users are trained to use it and they go through the 'limited usage' stage using features A, B and C which were taught on the training course. However they are unable to reach the 'full usage' stage, either because they are unable to find out if the system supports D and E, or how to use these features if they do find them - or perhaps they are not even aware that if they were to use features D and E, they would significantly improve their performance

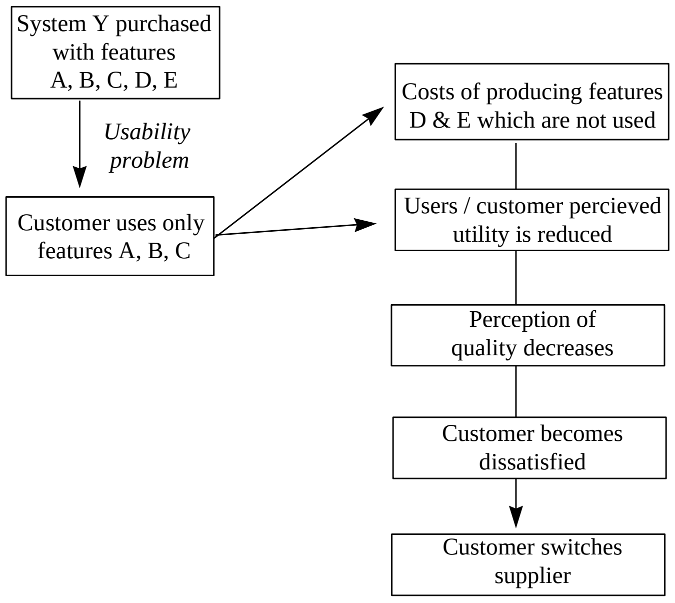

System Y purchased with features A, B, C, D, E
Usability problem
Customer uses only features A, B, C
Costs of producing features D & E which are not used
Users / customer percieved utility is reduced
Perception of quality decreases
Customer becomes dissatisfied
Customer switches supplier

Figure 5.2 – Costs of poor usability to customer & supplier

- Thus the customer has purchased a system with features A, B, C, D and E but the users can only use A, B and C. This is not only a cost to the customer but also to the supplier. The supplier has produced features D and E which are not used and has therefore wasted development time and effort.
- The customer on the other hand perceives that the system cannot perform features D and E and their perception of the usefulness of the system is reduced - it is not the system they thought they purchased. The user becomes dissatisfied with the product and when the 'evaluation' stage of the customer usage cycle is reached, the customer switches to a different supplier.
- Thus usability problems not only incur costs to the users and customers but also to the suppliers of systems. A supplier should seek to ensure that the users have positive usability experiences at each stage in their usage cycle.
- Usability specification is concerned with identifying, for a given situation, what will constitute a positive usability experience. Usability evaluation is concerned with testing a prototype or a system to find out whether it is usable. according to the specification.

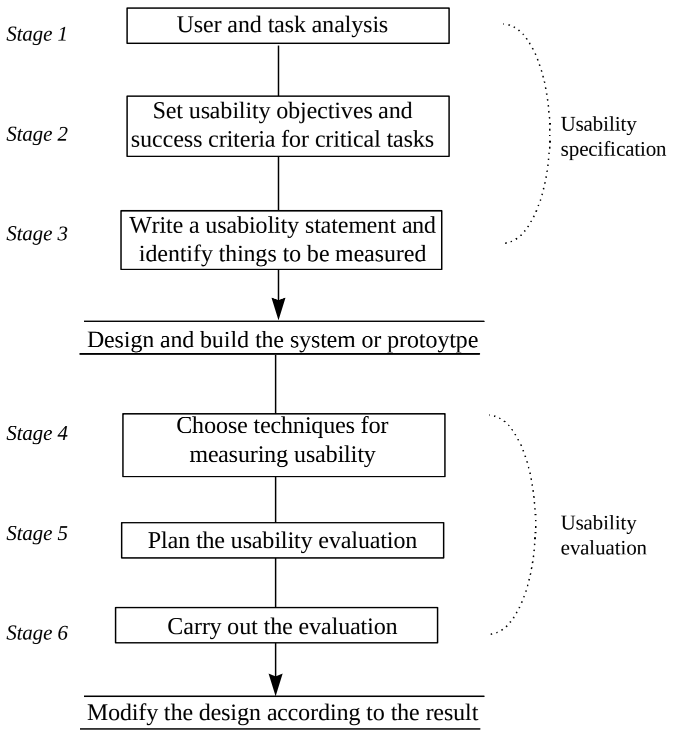

Stage 1
User and task analysis
Stage 2
Set usability objectives and success criteria for critical tasks
Stage 3
Write a usabiolity statement and identify things to be measured
Usability specification
Design and build the system or protoytpe
Stage 4
Choose techniques for measuring usability
Stage 5
Plan the usability evaluation
Stage 6
Carry out the evaluation
Usability evaluation
Modify the design according to the result

Figure 5.3 – The main stages in usability specification and evaluation

- Figure 5.3 shows when usability evaluations can be carried out.

5.3 Usability Objectives

- We can adopt a series of attributes which we will be able to measure:
  1. Learnability
  2. Throughput - ease and efficiency of use
  3. Flexibility
  4. User satisfaction
  5. or any other attributes identified by the team!
- **Learnability**
  - The learnability of a system is:
  - how easy it is for a novice user to learn how to use the system from scratch
  - how soon they achieve a required level of performance.
  - Learnability also implies the maintability of that understanding and performance levels.
  - We might consider it in terms of:- *predictability, familiarity, generalizability and consistency*
  - Measurements might include:
    - The time required to learn the system.
    - The time required to achieve a performance criterion.

    - The difficulties observed in acquiring the necessary skill.
    - User comments, suggestions, preferences
  - The objective indicators things we can measure might be:
    - The frequency of error message.
    - The frequency of use of on-line help.
    - The number of times user needed specific help.

- **Throughput**
  - This means the ease and efficiency of use after the initial learning period.
  - The principles here might be as follows:-
    - *transparency* - how obvious the internal state is to the user.
    - *recoverability* - how easy it is to recover from error
    - *responsiveness* - how quickly the system responds.
    - *conformance* - how far the system supports the user's tasks.
  - Measurements are as follows:
    - time required to perform selected tasks
    - success/failure in completing task
    - frequency of use of various commands or language features
    - time spent looking for information in documentation
    - measures of user problems as for learning
    - user comments, suggestions and preferences
    - how long the system takes to respond.
    - time spent with error

- **Flexibility**
  - This is a measure of how many ways in which the system and the user can interact. The system must accommodate different user types and different modes of work and activities.
    - We can examine the following principles:-
    - *dialogue initiative* - the user chooses the way in which communication occurs and might be able to modify the way in which it operates.
    - *multi-tasking* - the system is able to support more than one user task at a time
    - *task responsibility sharing* - the system might take responsibility for some tasks or
    - share it.
    - *input support* - the user is allowed a range of input values that are readily substituted for each other

- **User Satisfaction**
  - This can be based on user attitudes towards the system. Our aim in designing the
  - system is to promote continued and enhanced use of the system by the user and to
  - ensure that the user has positive feelings.
  - We can measure user attitudes using a questionnaire.
    - eg a 5-point scale in a questionnaire 1 = very bad to 5 = very good
  - We can specify in advance that we aim to achieve a user response of at least 90% users responding with scores of 4 or 5.

5.4 USABILITY SPECIFICATION

- Testing for usability, or usability evaluation can only take place if there are some objectives or targets which are to be tested. The first stage, therefore, is to specify the usability objectives for the system. Usability specification must start with an understanding of the users and their tasks and identification of those tasks which are to be shared between user and computer. It is the shared task for which usability will be an issue, since it is carrying out these tasks that the users is actually using the system

- Usability specification should take place as part of the system design. Usability objectives should be set as part of the requirements for the system so that the designer can design the system and the user interface with these objectives in mind.
- Usability objectives provide key requirements for the design. The software designer needs to be clear what the objectives are before deign begins.
- The usability specification is a statement of usability attributes. It states:
  - How the criteria will be measured.
  - What the criteria for representing the attainment is.
  - Which set or subset of users it applies to.
  - What the pre-conditions for measurement are:
    - eg: after a training period of five months 90% of users should be able to perform the task to a 20% success rate
- Some usability authors have suggested that the usability specification should be expanded to specify performance criteria such as:
  - worst case
  - lowest acceptable level
  - planned case (target level)
  - best case (agreed state-of-the-art level)
  - now level (current system level)

- **Checklist for Measurement Criteria**
  - The time taken to complete task.
  - Percentage of task completed.
  - Percentage of task completed per unit of time.
  - Ratio of success to failure.
  - Time spent with errors.
  - The frequency of use of: help and documentation
  - The amount of time spent in help and documentation
  - The percentage of favourable/unfavourable user comments.
  - The number of repetitions or failed commands.
  - The number of good features recalled by user.
  - The number of commands not used.
- **Usability Metrics**
  - What you have to do:
  - The user performs the task. You measure the performance.
  - Monitor the user when s/he is using the system.
  - Interview.
  - Use questionnaires.
  - Use surveys.

5.5 HEURISTICS

- **Nielsen's heuristics** - Nielsen suggest the use of the following ten heuristics which can be applied by expert and non expert evaluators to a given interface in order to help improve usability.

1. *Simple and natural dialogue* - Nielsen suggests that dialogue should consist of the minimum. It should be natural and logical.
2. *Speak the user's language* - Dialogs should be expressed clearly in words, phrases and concepts that are familiar to the user. They should not be the language of IT.
3. *Minimise the user's memory load* - The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrieved.
4. *Consistency* - Systems should use language and structures in a consistent fashion.
5. *Feedback* - The user should be informed about what is happening. There should be ample feedback.
6. *Clearly marked exits* - Users should be able to exit quickly from parts of the system, especially when they have got there by accident.
7. *Shortcuts* - There should be accelerators to speed up the action of experts. These can be hidden from the novice if need be.
8. *Good error messages* - Messages should be in plain language, indicate the problem and explain how to recover from it.
9. *Prevent errors* - Systems should endeavour to prevent errors in the first place.
10. *Help and documentation* - This should be concise, easy to search, related to the task and be concrete.

- **Shneiderman's Heuristics (8 Golden Rules)** - Shneiderman offers eight heuristics which again can be applied during design or afterwards as a means of checking for usability.
  1. *Strive for consistency* - Consistent sequences of actions should be required in similar circumstances. Identical terminology should be used for prompts, menus, help screens. Exceptions, for example, confirmation of the action of deleting or not echoing passwords, should be limited.
  2. *Enable frequent users to use shortcuts* - Shorter response times are attractive for frequent users and should be made available. Response times can never be too fast for frequent users though novices like to spend more time examining the screen. In any case, however fast the system goes novices can still determine just how quickly they wish to respond to the system. It is probably best to have systems run as fast as possible.
  3. *Offer informative feedback* - For every operator action there should be feedback. This can be modest for a frequent or minor action. For infrequent or major actions the response should be more substantial.
  4. *Design dialogs to yield closure* - Actions should be organised to have a beginning a middle and an end, like a good short story. The informative feedback at the end enables closure to take place so that the users know they are free to move on to the next stage.
  5. *Offer simple error handling* - The possibilities for error should be designed out of the system wherever possible. The user should be offered simple comprehensible mechanism for handling error. Erroneous commands, that is commands that do not mean anything to the system, should leave the system unchanged.
  6. *Permit easy reversal of actions* - Actions should be reversible. This relieves anxiety since the user knows that errors can be undone. It also encourages exploration because users

know that any action can be easily reversed. Just think how much easier life would be if there was an 'undo' button and you will realise what 'undo' means to new users.

7. *Support internal locus of control* - Experienced operators like to know they are in charge of the system. users should initiate actions rather than being on the receiving end of them.
8. *Reduce short term memory load* - Remember seven plus or minus two. Displays should be simple, sufficient training time should be allowed, multiple page displays should be consolidated.

- **Norman - 7 Principles for Transforming Difficult Tasks into Simple Ones**
  1. *Use both knowledge in the world and knowledge in the head* - The knowledge necessary for a task should be available in the world. It should not need to be remembered. Users should be able to derive system behaviour from interacting with the system.
  2. *Simplify the structure of the task* - Tasks should be simple and should minimise the amount of planning that has to take place.
  3. *Make things visible. Bridge the gulfs of Execution and Evaluation* - Make it obvious what can be done and what the effects of actions actually are. Actions should match intentions. System states should be obvious. If the user does something big the system should do something big.
  4. *Get the mappings right* - Exploit natural mappings. Make sure the user can work out relationships between intentions and actions, between actions and their effects on the system, between the actual system state and what the system state appears to be either by sight, feel or sound. All too often users develop superstitious beliefs about systems.
  5. *Exploit the power of constraints both natural and artificial* - Use constraints that feel natural to the user so there is only one possible thing to do. Think of putting a disk into a computer. It will go in one way only or trying a key in a lock; pick the wrong key and the door won't open! The nice thing about natural constraints is that quite often operators won't think of a failure as such. If you choose the wrong key, you re-select and probably, most of the time, it doesn't worry you unduly - in other words you don't think of it as error or failure.
  6. *Design for error* - Assume that error will take place and plan for it. Make it easy to reverse operations. Design explorable systems that allow users to wander about the system without fear of reprisal. If a user is scared of making mistakes that are irreversible they won't explore. Design forcing functions which make the user do the right thing.
  7. *When all else fails, standardise* - Standardise layout, design, mappings, functions, displays, and outcomes. Norman is really suggesting that where there isn't a solution you choose the best option and make it the same. This is really the philosophy behind the style guides and it is sound thinking. Notice that Norman has a contingency plan in principle 7.

- **The Seven Stages of Action and Design**
  - Norman's seven questions amount to something like this:
    1. How easy is it to work out the function of the device?
    2. How easy is it to tell what actions are possible with the device and at that moment?

3. How easy is it to determine the mappings from the intention to the physical movement required in order to achieve the goal?
4. How easy is it to perform the action?
5. How easy is it to tell if the system is now in the desired state?
6. How easy is it to determine the mapping from the system state to the interpretation?
7. How easy is it to tell what state the system is in?

5.6 PROBLEM AREAS IN EVALUATION

- **Common sense** - It is tempting to assume that because something seems obvious it must be correct. However, the results of research often contradict what common sense might tell us. Research has found that workmen in noisy surroundings heard speech better if they wore ear plugs. That is by no means an obvious conclusion. The early generation of chairs provided by ergonomists were later found to be unsuitable since workers could not adopt the positions which common sense told ergonomists they would. It is very easy to make assumptions about user behaviour but you should try not to assume anything and be open minded.
- **Testing it on yourself** - This is another dangerous trap for the unwary designer since the individual HCI practitioner can hardly be expected to be typical. The problem is that the designer as a designer has skills which the average person does not have and the design process will involve adding to those skills. Even if the designer starts off as a naive user of a computer system, the task of design will involve having to learn about the system and the innocence she started the task with will soon disappear.
- **So who's the engine driver?** - It is easy to be told that this is what the worker does. Beware. The boss does not always know how a task is performed. The boss does know how s/he wants the task done, or how it ought to be done, or how it used to be done. For a real understanding of what actually occurs you have to speak to the people performing the tasks. Ignore them at your peril!
- **We've always done it like this!** - It is very important to look at the task and to decide what the best way to perform the task might be. You need to take into consideration the advantages of positive transfer as compared to the problems of learning a new system. If you are asked to design a new system then there are two contradictory errors which might occur:
  1. You keep things as they are because it has always be done in this way.
  2. You change it because it's a new system.
- **Technical Ease in conflict with ease of use** - It is easy to assume a solution because the technicalities pose a solution which is easiest to implement. It is important that human factors come first and that any sacrifice of ease of use for the benefit of implementation is done consciously.
- **I'll evaluate it later** - Evaluation needs to come early. Design decisions can be difficult to reverse if evaluation appears late. It should be seen as an on-going process. Design, - evaluate - design.
- **Evaluation carried out on the wrong people** - The evaluation of the system must be done on a representative group. Many of the design problems we now face have been caused by testing of systems on people who know too much about the system in the first place or are themselves designers.

- **When and what you should evaluate** - Ideally, any testing you carry out should be in circumstances identical/very close to the real conditions the system will be operating in. If it is possible for the system to be tested where it is to be used, then obviously you should do that. As a design passes through the stages and finally becomes a finished product, so the cost of rectifying errors in the design increases. It is therefore of paramount importance to test at the early stages. You want to ensure that the final testing confirms earlier design and brings to light only the most minor of problems.
- **Systems Analysis Phase** - Where possible, you should analyse work which has been done in similar fields. You can get feedback from users of previously designed systems. You can produce scenarios and consider feasibility.
- **System design phase** - Parts of the system can be simulated and tested during the design stage. The prototypes need not be working systems, they can be mock ups, parts of systems, paper systems and so on. These are relatively cheap to produce so the advantages of different types of design can be measured against each other at this stage. Although it is tempting to get on with the production of the system, it is much easier to make changes at this stage rather than later. If a modular approach is taken to design, the cost of making changes is again reduced.
- **Pre-production phase** - When the prototype design is complete, evaluation can take place on a large scale and can concentrate on the details of the system. There are many ways of measuring user performance with a system and these can be used at this stage in order to evaluate the different aspects of the design.

5.7 Evaluation Techniques

- Basically, you can adopt two quite different approaches, or perhaps a mixture of the two. The first method is the **analytic evaluation method** and consists of a formal pencil and paper evaluation of tasks and goal - for example, the GOMS methodology suggested by Card , Moran and Newell. The second method is the **empirical method** and this consists of an analysis of user performance in relation to the proposed system. It might consist of tasks to be performed with the system, observation, questionnaires, and interviews.
- **Experiments** - There are two basic approaches to experimentation on the users of a new system. You can test the new system's performance in relation to an existing system - *comparative experiments* or the new system can be tested in isolation - *absolute experiments*.
- **Hypothesis formation** - A specific hypothesis is formed. For example, "users who receive feedback about their performance will do better than those who do not."
- **Independent and dependent variables** - The factor that we think will affect user ability to perform better is feedback. This is called the independent variable. It is the variable which we set up to see if it affects performance or behaviour. The dependent variable is how well the user performs. However, because of the difficulty of isolating the factor that has affected performance we need to make sure that other factors were not involved. We do this by:
- **Control groups** - We select two groups of users and give one group feedback and none to the other. The group we give feedback to is called the experimental group and the other is the control group.

- **Confounding variables** - Try to match the two groups so that they are as alike as possible.
- **Repeated measures designs** - We ensure that differences do not occur by using the same group. We might give feedback at one time and not at another and then see if there is a difference. We compare the same person's performances. If we compare different people's performances this is called independent measures design.
  - Choosing appropriate groups of people to test is not easy. You could take people at random but this might not necessarily give you the sample you really want. For example, the group of people you are designing the software for might be of a particular age or experience. You might decide that you would prefer to eliminate certain categories of user. Matching groups can also be a problem.
  - You need to consider the conditions under which the testing is to take place. It has to be identical for all participants unless this difference has specifically been accounted for. There are a number of things we can measure in the evaluation of a system but these will be dealt with in a later lecture.
- **Questionnaires** - The advantage of a questionnaire is that once produced it can provide a vast body of information. However, the production of a good questionnaire is time consuming and if it is to be done correctly, requires several stages. Secondly, questionnaires are a good source of attitude measurement but they are not objective. They can be either *interviewer administered* or *self administered*.
- **Interviewer administered questionnaires** - This implies the existence and availability of interviewers trained in the technique. It requires a considerable amount of time to carry out but it has the advantage that:
  1. Gathering of data is controlled by the interviewer.
  2. If there are difficulties over understanding, the interviewer can clarify.
- **Self administered questionnaires** - These require fewer person hours to deliver but there are problems:
  1. There is poor control over returns.
  2. Questions cannot be explained by the interviewer so they have to be carefully written.
  3. There is a high likelihood of bias because of the small number of responses.
- Problems in framing questions are:
  1. *Use of loaded terms* - It is important not to bias the reply by using emotionally significant terms.
  2. *Suggesting the desired response*. For example: "Have you stopped beating your dog?" is difficult to answer without implying some fault. Questions need to be in a neutral tone and should not be phrased in such a way that the person is likely to answer yes. If you use the "You do...don't you" structure then this is likely to lead to answers consisting of yes or no.
  3. *Embarrassment of subject* - this can arise where the subject feels, for example, that the answer implies some inadequacy.
  4. *Lack of precision in questions leading to imprecise answers*. Avoid questions that give answers like 'seldom', 'frequently', 'often'. You don't know what these mean they are too subjective!
- Recommendations
  1. Closed questions are better than open questions. There can always be a 'don't know' reply if necessary.
  2. Questions need to be structured carefully, and progress from the general to the detailed. The first questions should be neutral and easily answered. More sensitive

questions should be left until later in the questionnaire. It is important to order questions carefully and thus to avoid:

- Other methods for the collection of data about the system - Any of the following methods can be used to collect appropriate information about either existing or projected systems: interviews with users, observation, activity sampling, activity logging
- **Interviews with users** - We will return to the idea of interviews at a later date. Any of the evaluation methods could be used as a method of first gathering information about an existing system. It is important that the design team gets the confidence of the end users and involves them in the design process. They should not be passive providers of information but active participants in the design process. that way they will get a system they want and need rather than a system that we think they want and need.
- **Observation** - It is useful to watch the performer of a task do the task. Particularly if she can be persuaded to ignore you and perhaps to vocalise what is being done. Some people are happy to talk to themselves whereas others find it very difficult indeed. It is important not to disturb the way in which an individual works on a task nor must you impose any alien conditions or tools. Make sure that she is happy and that she is doing what it is she really does do. It is easy to be subjected to a special show.

Activity sampling and activity logging - The tasks are sampled or logged, or both. This can be done by either a member of the design team or the performer of the task herself. When done by the latter it can interfere with the way in which a task is performed so be warned.