NATIONAL OPEN UNIVERSITY OF NIGERIA

CIT 811

User Interface Design and Erogonomics

Module 2

CIT 811 User Interface Design and Ergonomics Module 2

Course Developer/Writer
Dr. A. S. Sodiya, University of Agriculture, Abeokuta

Course Coordinator
A. A. Afolorunso, National Open University of Nigeria

Programme Leader
Prof. MoniOluwa Olaniyi

Credits of cover-photo: Henry Ude, National Open University of Nigeria

National Open University of Nigeria - University Village, Plot 91, Cadastral Zone, Nnamdi Azikiwe Expressway, Jabi, Abuja-Nigeria

www.nou.edu.ng centralinfo@nou.edu.ng
oer.nou.edu.ng oerunit@noun.edu.ng OER repository
Published in 2013, 2021 by the National Open University of Nigeria

How to re-use and attribute this content

Unit 1 Human Capabilities in User Interface Design

1.0 Introduction

This unit describes Human capabilities in user interface design. The concept of Human processor model is introduced alongside perception, motor skills, colour, attention and errors.

2.0 Objectives

At the end of this unit, you should be able to:

- explain the human processor model
- describe various terms like perception, motor skills, e.tc
- explain the concepts of attention and errors.

3.0 Main Content

3.1 Human Processor Model

The human processor model is a cognitive modeling method used to calculate how long it takes to perform a certain task. Other cognitive modeling methods include parallel design, GOMS (Goals, Operator, Methods, and Selection), etc. Cognitive modeling methods are one way to evaluate the usability of a product. This method uses experimental times to calculate cognitive and motor processing time. The value of the human processor model is that it allows a system designer to predict the performance with respect to time it takes a person to complete a task without performing experiments. Other modeling methods include inspection methods, inquiry methods, prototyping methods, and testing methods. The Human Processor Model is shown in figure 1.1.

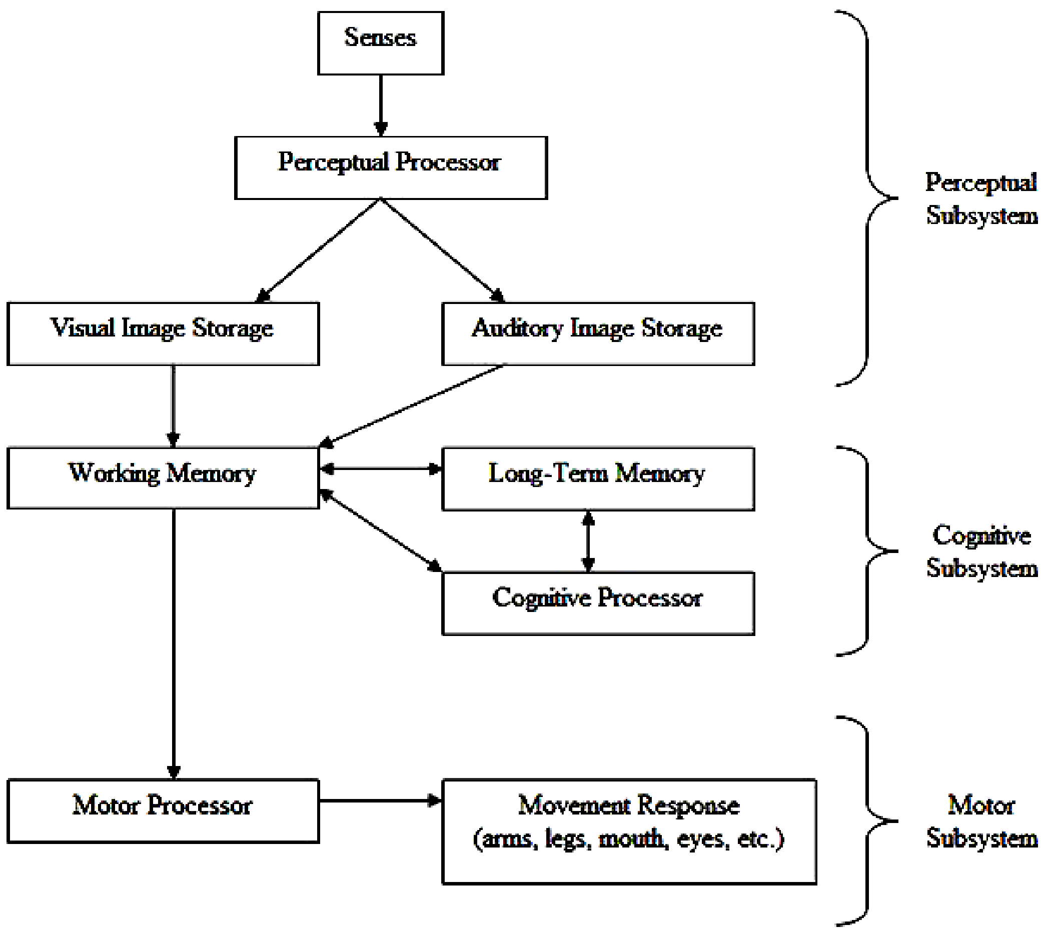


Fig. 1.1: Human Processor Model

Input from the eyes and ears are first stored in the **short-term sensory store (**Visual and image and Auditory image storage). As a computer hardware analogy, this memory is like a frame buffer, storing a single frame of perception.

The **perceptual processor** takes the stored sensory input and attempts to recognise *symbols* in it: letters, words, phonemes, icons. It is aided in this recognition by the **long-term memory**, which stores the symbols you know how to recognise.

The **cognitive processor** takes the symbols recognised by the perceptual processor and makes comparisons and decisions. It might also store and fetch symbols in **working memory** (which you might think of as RAM, although it's pretty small). The cognitive processor does most of the work that we think of as "thinking".

The **motor processor** receives an action from the cognitive processor and instructs the muscles to execute it. There's an implicit **feedback** loop here: the effect of the action (either on the position of your body or on the state of the world) can be observed by your senses, and used to correct the motion in a continuous process.

The **Visual Image Store (VIS)** is basically an image frame from the eyes. It isn't encoded as pixels, but as physical features of the image, such as curvature, length, edges. It retains physical features like intensity that may be discarded in higher-level memories (like the working memory). We measure its size in letters because psych studies have used letters as a convenient stimulus for measuring the properties of the VIS; this doesn't mean that letters are represented symbolically in the VIS. The VIS memory is fleeting, decaying in a few hundred milliseconds.

The **auditory image store** is a buffer for physical sound. Its size is much smaller than the VIS (in terms of letters), but lasts longer – seconds, rather than tenths of a second. Both of these stores are preattentional; that is, they don't need the spotlight of attention to focus on them in order to be collected and stored. Attention can be focused on the visual or auditory stimulus after the fact. That accounts for phenomena like "What did you say? Oh yeah."

Finally, there is a component called **senses**, which corresponds to **attention**. This might be thought of like a thread of control in a computer system. Note that this model is not meant to reflect the anatomy of your nervous system. For example, there probably is not a single area in your brain corresponding to the perceptual processor. Nevertheless, it is a useful abstraction.

It has been shown that the human processor model uses the cognitive, perceptual, and motor processors along with the visual image, working memory, and long term memory storages. Figure 1.1 shows the representation of the model. Each processor has a cycle time and each memory has a decay time. These values are also included in Table 1.1.

By following the connections in figure 1.1, along with the associated cycle or decay times, the time it takes a user to perform a certain task can be calculated. The calculations depend on the ability to break down every step of a task into the basic process level. The more detailed the analysis, the more accurate the model will be to predict human performance. The method for determining processes can be broken down into the following steps:

- write out main steps based on: a working prototype, simulation, step by step walkthrough of all steps
- clearly identify the specific task and method to accomplish that task
- for each final step identify sub-levels down to a basic process (in the diagram or chart below)
- convert into pseudo- code (writing out methods for each step)
- list all assumptions (will be helpful as multiple iterations are completed)
- determine time of each operation (based on the table below)
- determine if operation times should be adjusted (slower for elderly, disability, unfamiliarity, etc.)
- sum up execution times
- iterate as needed and check with prototyping if possible.

Studies into this field were initially done by Card, S. K., Moran T.P., & Newell, A. Current studies in the field include work to distinguish process times in older adults by Jastrembski & Charness (2007). Table 1.1 shows some of the results from the work.

Table 1.1: Processor Actions'Times

Parameter	Mean	Range
Eye movement time	230 ms	70-700 ms
Decay half-life of visual image storage	200 ms	90-1000 ms
Visual Capacity	17 letters	7-17 letters
Decay half-life of auditory storage	1500 ms	90-3500 ms
Auditory Capacity	5 letters	4.4-6.2 letters
Perceptual processor cycle time	100 ms	50-200 ms
Cognitive processor cycle time	70 ms	25-170 ms
Motor processor cycle time	70 ms	30-100 ms
Effective working memory capacity	7 chunks	5-9 chunks
Pure working memory capacity	3 chunks	2.5-4.2 chunks
Decay half-life of working memory	7 sec	5-226 sec
Decay half-life of 1 chunk working memory	73 sec	73-226 sec
Decay half-life of 3 chunks working memory	7 sec	5-34 sec

3.1.1 Potential Uses

Once complete, the calculations can then be used to determine the probability of a user remembering an item that may have been encountered in the process. The following formula can be used to find the probability:

$P = e^{-K*t}$

where K is the decay constant for the respective memory in question (working or long term) and

t is the amount of time elapsed (with units corresponding to that of K).

The probability could then be used to determine whether or not a user would be likely to recall an important piece of information they were presented with while doing an activity.

It is important to deduce beforehand whether the user would be able to repeat the vital information throughout time *t*, as this has a negative impact on the working memory if they cannot. For example, if a user is reading lines of text and is presented with an important phone number in that text, he/she may not be able to repeat the number if they have to continue to read. This would cause the user's working memory's decay time to be smaller, thus reducing their probability of recall.

Example: Find the probability that a user will recall an item of information stored in effective pure working memory. The user has seen the item for a total of twos.

Solution:

$P = e^{-K*t}$

K=7s as given in Table 1 and t = 2s as it was given in the question

So,

$P = e^{-(7*2)}$

$P = e^{-14} = 0.00000083$

This shows that the probability is almost 0.

Note: Chunk is defined as unit of perception or memory

3.2 Perception

In psychology and the cognitive sciences, **perception** is the process of attaining awareness or understanding of sensory information. It is a task far more complex than was imagined in the 1950s and 1960s, when it was predicted that building perceiving machines would take about a decade, a goal which is still very far from fruition. The word comes from the Latin words *perceptio, percipio,* and means "receiving, collecting, action of taking possession, apprehension with the mind or senses.

Perception is one of the oldest fields in psychology. What one perceives is a result of interplays between past experiences, including one's culture, and the interpretation of the perceived. If the percept does not have support in any of these perceptual bases it is unlikely to rise above perceptual threshold.

Two types of consciousness are considered regarding perception: ***phenomenal*** (any occurrence that is observable and physical) and ***psychological***. The difference everybody can demonstrate to himself or herself is by the simple opening and closing of his or her eyes: phenomenal consciousness is thought, on average, to be predominately absent without sight. Through the full or rich sensations present in sight, nothing by comparison is present while the eyes are closed. Using this precept, it is understood that, in the vast majority of cases, logical solutions are reached through simple human sensation.

Passive perception (conceived by René Descartes) can be surmised as the following sequence of events: surrounding → input (senses) → processing (brain) → output (re-action). Although still supported by mainstream philosophers, psychologists and neurologists, this theory is nowadays losing momentum. The theory of active perception has emerged from extensive research of sensory illusions, most notably the works of Richard L. Gregory. This theory, which is increasingly gaining experimental support, can be surmised as dynamic relationship between "description" (in the brain) ↔ senses ↔ surrounding, all of which holds true to the linear concept of experience.

In the case of visual perception, some people can actually see the percept shift in their mind's eye. Others, who are not picture thinkers, may not necessarily perceive the 'shape-shifting' as their world changes. The 'esemplastic' nature has been shown by experiment: an ambiguous image has multiple interpretations on the perceptual level. The question, "Is the glass half empty or half full?" serves to demonstrate the way an object can be perceived in different ways.

Just as one object can give rise to multiple percepts, so an object may fail to give rise to any percept at all: if the percept has no grounding in a person's experience, the person may literally not perceive it.

The processes of perception routinely alter what humans see. When people view something with a preconceived concept about it, they tend to take those concepts and see them whether or not they are there. This problem stems from the fact that humans are unable to understand new information, without the inherent bias of their previous knowledge. A person's knowledge creates his or her reality as much as the truth, because the human mind can only contemplate that to which it has been exposed. When objects are viewed without understanding, the mind will try to reach for something that it already recognises, in order to process what it is viewing. That which most closely relates to the unfamiliar from our past experiences, makes up what we see when we look at things that we do not comprehend.

In interface design, the extent to which a user interface is appreciated depends largely on the perception of users. This is why a major stage in user interface design is to understand user. Psychologists are always involved in this stage.

3.3 Motor Skills

A **motor skill** is a learned series of movements that combine to produce a smooth, efficient action.

Gross motor skills: include lifting one's head, rolling over, sitting up, balancing, crawling, and walking. Gross motor development usually follows a pattern. Generally large muscles develop before smaller ones, thus, gross motor development is the foundation for developing skills in other areas (such as fine motor skills). Development also generally moves from top to bottom. The first thing a baby usually learns to control is its eyes.

Fine motor skills: include the ability to manipulate small objects, transfer objects from hand to hand, and various hand-eye coordination tasks. Fine motor skills may involve the use of very precise motor movement in order to achieve an especially delicate task. Some examples of fine motor skills are using the pincer grasp (thumb and forefinger) to pick up small objects, cutting, colouring, writing, or threading beads. Fine motor development refers to the development of skills involving the smaller muscle groups.

Ambidexterity: is a specialised skill in which there is no dominance between body symmetries, so tasks requiring fine motor skills can be performed with the left or right extremities. The most common example of ambidexterity is the ability to write with the left or right hand, rather than one dominant side.

The motor skills by users significantly affect the ability of users to be able to use the system well and do their work better (Ergonomics). This shows that it is important to identify or measure the motor skills of users before the real design starts.

Fatigue or weariness may lead to temporary short-term deterioration of fine motor skills (observed as visible shaking), serious nervous disorders may result in a loss of both gross and fine motor skills due to the hampering of muscular control. A defect in muscle is also a symptom of motor skill dysfunction. A user interface must be designed to eliminate completely or reduce significantly all the mentioned defects.

3.4 Colour

Colour corresponding in humans to the categories called *red*, *yellow*, *blue* and others. Colour derives from the spectrum of light (distribution of light energy versus wavelength) interacting in the eye with the spectral sensitivities of the light receptors. Colour categories and physical specifications of colour are also associated with objects, materials, light sources, etc., based on their physical properties such as light absorption, reflection, or emission spectra. Colours can be identified by their unique Red, Green and Blue (RGB) and Hue, Saturation and Value (HSV) values.

Typically, only features of the composition of light that are detectable by humans (wavelength spectrum from 380 nm to 740 nm, roughly) are included, thereby objectively relating the psychological phenomenon of colour to its physical specification. Because perception of colour stems from the varying sensitivity of different types of cone cells in the retina to different parts of the spectrum, colours may be defined and quantified by the degree to which they stimulate these cells. These physical or physiological quantifications of colour, however, do not fully explain the psychophysical perception of colour appearance.

The science of colour is sometimes called ***chromatics***. It includes the perception of colour by the human eye and brain, the origin of colour in materials, colour theory in art, and the physics of electromagnetic radiation in the visible range (that is, what we commonly refer to simply as *light*).

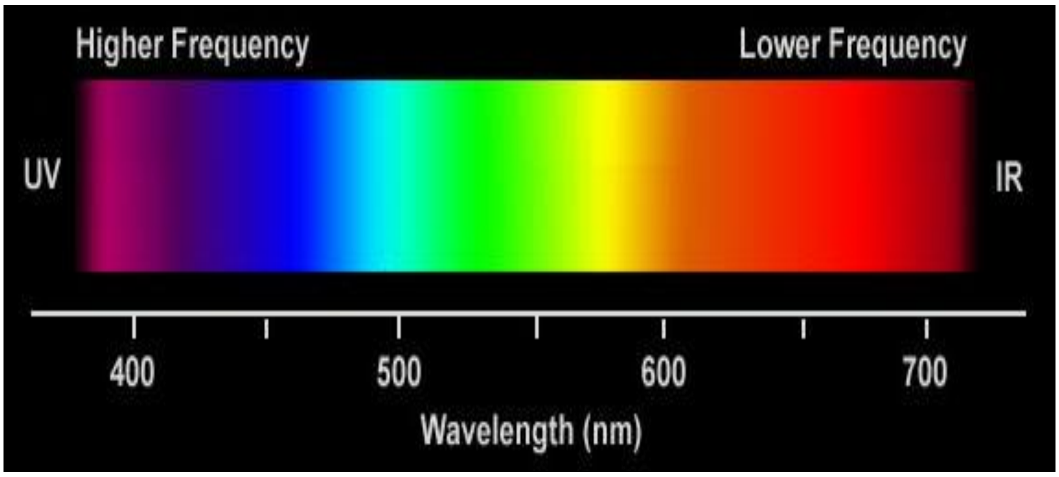


Figure 1.2: Visible Spectrum

The ability of the human eye to distinguish colours is based upon the varying sensitivity of different cells in the retina to light of different wavelengths. The retina contains three types of colour receptor cells, or cones. One type, relatively distinct from the other two, is most responsive to light that we perceive as violet, with wavelengths around 420 nm. (Cones of this type are sometimes called *short-wavelength cones*, *S cones*, or, misleadingly, *blue cones*.) The other two types are closely related genetically and chemically. One of them (sometimes called *long-wavelength cones*, *L cones*, or, misleadingly, *red cones*) is most sensitive to light we perceive as yellowish-green, with wavelengths around 564 nm; the other type (sometimes called *middle-wavelength cones*, *M cones*, or, misleadingly, *green cones*) is most sensitive to light perceived as green, with wavelengths around 534 nm.

3.4.1 Facts about Colour

- Colour can be a powerful tool to improve user interfaces, but inappropriate use can severely reduce human performance
- Colour can be very helpful in gaining the attention of users
- In designing user interfaces, pay attention to how to combine colours and human perception
- Consider people with colour deficiency in your colour combination. The colour limitations such as colour blindness, near/far sighted and focusing speed should be considered
- As we age, sensitivity to blue is even more reduced and perceive a lower level of brightness, the implication is that you do not rely on blue for text or small objects
- Red objects appear closer than blue objects.

3.5 Attention

Attention is the cognitive process of selectively concentrating on one aspect of the environment while ignoring other things. Examples include listening carefully to what someone is saying while ignoring other conversations in a room (the cocktail party effect) or listening to a cell phone conversation while driving a car. Attention is one of the most intensely studied topics within psychology and cognitive neuroscience.

It is the taking possession by the mind, in clear and vivid form, of one out of what seem several simultaneously possible objects or trains of thought. Focalisation, concentration, and consciousness are of its attributes. It implies withdrawal from some things in order to deal effectively with others, and is a condition which has a real opposite in the confused, dazed, scatterbrained state.

Our ability to divide our attention among multiple tasks appears to depend on two things. First is the **structure** of the tasks that are trying to share our attention. Tasks with different characteristics are easier to share; tasks with similar characteristics tend to interfere. Important dimensions for task interference seem to be the **modality** of the task's input (visual or auditory), its **encoding** (e.g., spatial/graphical/sound encoding, vs. words), and the mental **components** required to perform it. For example, reading two things at the same time is much harder than reading and listening, because reading and listening use two different modalities.

The second key influence on multitasking performance is the difficulty of the task. Carrying on a conversation while driving a car is fairly effortless as long as the road is familiar and free of obstacles; when the driver must deal with traffic or navigation, conversation tends to slow down or even stop.

A good user interface should be able to drive the attention of users. The colour combinations, ease of use, performance, etc are some attributes that drive users' full attention. The inability to pay attention often leads to boredom and hence, low-productivity.

3.6 Errors

The word ***error*** has different meanings and usages relative to how it is conceptually applied. The concrete meaning of the Latin word error is "wandering" or "straying". To the contrary of an illusion, an error or a mistake can sometimes be dispelled through knowledge (knowing that one is looking at a mirage and not at real water doesn't make the mirage disappear). However, some errors can occur even when individuals have the required knowledge to perform a task correctly.

An 'error' is a deviation from accuracy or correctness. A 'mistake' is an error caused by a fault: the fault being misjudgment, carelessness, or forgetfulness. For example, if a user unintentionally clicks a wrong button, that is a mistake. This might take several minutes to correct.

User interfaces should be designed in order to prevent intentional and unintentional errors from users. Users should be well guided so that their job could be performed efficiently.

4.0 Conclusion

This unit has introduced you to the Human processor model. The perception concept along with motor skills, colour and attention in Human processor model were also introduced. Error concept in Human processor model was also discussed.

5.0 Summary

What you have learnt in this unit concerns:
The **Human Processor Model** which is a cognitive modeling method used to calculate how long it takes to perform a certain task. The various uses were also discussed.
Perception which is the process of attaining awareness or understanding of sensory information. Passive and visual perception was also discussed.
Motor skill which is a learned series of movements that combine to produce a smooth and efficient action. Gross, fine and Ambidexterity motor skills were discussed.

- The colour concept which can be identified by their unique RGB and HSV values. Various facts about colour were also discussed.
- Error which is a deviation from accuracy or correctness in Human processor model.

6.0 Self-Assessment Exercise

1. Explain briefly the human processor model.
2. Write a short note on Ambidexterity.

7.0 References/Further Reading

Card, S.K., Moran, T. P., & Newell, A. (1986). The Model Human Processor: An Engineering Model of Human Performance. In: K. R. Boff, L. Kaufman, & J. P. Thomas (Eds.). *Handbook of Perception and Human Performance*. Vol. 2: *Cognitive Processes and Performance*, pp. 1–35.
Hirakawa, K. & Parks, T.W. (2005). "Chromatic Adaptation and White-Balance Problem". *IEEE ICIP*. doi:10.1109/ICIP.2005.1530559.

Jastrzembski Tiffany & Charness Neil (2007). The Model Human Processor and the Older Adult: Parameter Estimation and Validation within a Mobile Phone Task. *Journal of Experimental Psychology: Applied***.** Volume 13, Number 4, 2007, pp 224-248.

Knudsen, E. I. (2007). "Fundamental Components of Attention". *Annual Review of Neuroscience* **30** (1): 57–78. doi:10.1146/annurev.neuro. 30.051606.094256. PMID 17417935.

Lui, Yili, Feyen, Robert, & Tsimhoni, Omer (2006). Queueing Network-Model Human Processor (QN-MHP): A Computational Architecture for Multitask Performance in Human-Machine Systems. **ACM Transactions on Computer-Human Interaction.** Volume 13, Number 1, March 2006, pp 37-70.

Morrell, Jessica Page (2006). *Between the Lines: Master the Subtle Elements of Fiction Writing*. Cincinnati, OH: Writer's Digest Books. ISBN 1582973938.

Pattyn, N., Neyt, X., Henderickx, D., & Soetens, E.(2008). Psychophysiological Investigation of Vigilance Decrement: Boredom or Cognitive Fatigue? *Physiology & Behaviour*, 93, 369-378.

Pinel, J. P. (2008). *Biopsychology* (7th ed.). Boston: Pearson. p.357.

Robles-De-La-Torre G. (2006). The Importance of the Sense of Touch in Virtual and Real Environments. *IEEE Multimedia* 13(3), Special issue on Haptic User Interfaces for Multimedia Systems, pp. 24-30.

Wright, R.D. & Ward, L.M. (2008). *Orienting of Attention*. Oxford University Press

Unit 2 Understanding Users and Task Analysis

1.0 Introduction

The general understanding of users and task analysis will be introduced to you in this unit. Using of task in designs and creating of the initial design will also be discussed in this unit.

2.0 Objectives

At the end of this unit, you should be able to:
- describe how to understand users
- explain task analysis
- explain the use of tasks in design
- describe the creation of initial design.

3.0 Main Content

3.1 Understanding Users

To get a good interface you have to figure out who is going to use it and to do what. You may think your idea for a new system is so wonderful that everyone will want it. But history suggests you may be wrong. Even systems that turned out to be useful in unexpected ways, like the spreadsheet, started out by being useful in some expected ways.

You may not have needed selling on this point. "Everybody" knows you have to do some kind of requirements analysis. Yes, but based on what, and in what form? Our advice is to insist that your requirements be grounded in information about real, individual people and real tasks that they really want to perform. Get soft about this and the illusions start to creep in and before you know it you've got another system that everybody wants except people you can actually find.

3.1.1Getting in Touch with Users

The first step is to find some real people who would be potential users of what you are going to build. If you cannot find any you need to worry a lot. If you can't find them now, where will they come from? When it's time to buy? When you have found some, get them to spend some time with you discussing what they do and how your system might fit in. Are they too busy to do this? Then they'll probably be too busy to care about your system after it exists. Do you think the idea is a real winner, and they will care if you explain it to them? Then buy their time in some way. Find people in your target group who are technology nuts and who'll talk with you because you can show them technology. Or go to a professional meeting and offer a unique T-shirt to people who'll talk with you (yes, there are people whose time is too expensive for you to buy for money who will work with you for a shirt or a coffee mug).

3.2 Task Analysis

Task analysis is the analysis of how a task is accomplished, including a detailed description of both manual and mental activities, task and element durations, task frequency, task allocation, task complexity, environmental conditions, necessary clothing and equipment, and any other unique factors involved in or required for one or more people to perform a given task. Task analysis emerged from research in applied behaviour analysis and still has considerable research in that area.

The term "task" is often used interchangeably with activity or process. Task analysis often results in a hierarchical representation of what steps it takes to perform a task for which there is a goal and for which there is some lowest-level "action" that is performed. Task analysis is often performed by human factors professionals.

Task analysis may be of manual tasks, such as bricklaying, and be analyzed as time and motion studies using concepts from industrial engineering. Cognitive task analysis is applied to modern work environments such as supervisory control where little physical works occurs, but the tasks are more related to situation assessment, decision making, and response planning and execution.

Task analysis is also used in user interface design. Information from the task analysis is used in the interface design. This is necessary in order to make system to capture the overall users' requirements.

3.2.1Task Analysis in User Interface Design

The stages involved in task analysis are described as follows:-

Identify Users
The users of a system must be identified. In the case that the population of users is large, a reasonable and representative sample of users should be identified.

Learning about Users' Tasks
The interface designers should interact with the users in order study their tasks. The major interests here are:
- what kind of tasks are they performing
- identify what the users want to do (minor and major tasks) and how they want to do it
- how do users want the work to be done?
- identify how users' tasks can be performed better
- state the examples of concrete tasks perform by users
- who will do what
- they extent of work to be done
- the level of interaction required by the tasks.

Come Up with a Representative Task Description
After establishing a good understanding of the users and their tasks, a more traditional design process might abstract away from these facts and produce a general specification of the system and its user interface. The task-centered design process takes a more concrete approach. The designer should identify several representative tasks that the system will be

used to accomplish. These should be tasks that users have actually described to the designers. The tasks can initially be referenced in a few words, but because they are real tasks, they can later be expanded to any level of detail needed to answer design questions or analyse a proposed interface. Here are a few examples of how to come up with representative task description:
for a word processor: "transcribe a memo and send it to a mailing list"
for a spreadsheet: "produce a salary budget for next year"
for a communications program: "login to the office via modem"
for an industrial control system: "hand over control to next shift"
The tasks selected should provide reasonably complete coverage of the functionality of the system, and the designer may want to make a checklist of functions and compare those to the tasks to ensure that coverage has been achieved. There should also be a mixture of simple and complex tasks. Simple tasks, such as "check the spelling of 'occasional'," will be useful for early design considerations, but many interface problems will only be revealed through complex tasks that represent extended real- world interactions. Producing an effective set of tasks will be a real test of the designer's understanding of the users and their work.

Evaluate
Finally, it is necessary to determine if users' tasks have been adequately captured and well spelt out. After adequate consultations with the users, the next step is to write out descriptions of all the tasks and circulate them to the users. We need to include the queries for more information where we felt the original discussion had left some details out. You will then need to get corrections, clarifications, and suggestions back which are incorporated into the written descriptions.

3.3 Using the Tasks in Design

We then rough out an interface design and produce a SCENARIO for each of the sample tasks. A scenario spells out what a user would have to do and what he or she would see step-by-step in performing a task using a given system. The key distinction between a scenario and a task is that a scenario is design-specific, in that it shows how a task would be performed if you adopt a particular design, while the task itself is design-independent: it is something the user wants to do regardless of what design is chosen. Developing the scenarios forced us to get specific about our design, and it forced us to consider how the various features of the system would work together to accomplish real work. We could settle arguments about different ways of doing things in the interface by seeing how they played out for our example tasks.

Handling design arguments is a key issue, and having specific tasks to work with really helps. Interface design is full of issues that look as if they could be settled in the abstract but really can't. Unfortunately, designers, who often prefer to look at questions in the abstract, waste huge amounts of time on pointless arguments as a result.

For example, if our interface users select graphical objects from a palette and place them on the screen, they do this by clicking on an object in the palette and then click where they want to put it. Now, if they want to place another object of the same kind should they be made to click again on the palette or can they just click on a new location? You can't settle the matter by arguing about it on general grounds.

You can settle it by looking at the CONTEXT in which this operation actually occurs. If the user wants to adjust the position of an object after placing it, and you decide that clicking again somewhere places a new object, and if it's legal to pile objects up in the same place, then you have trouble. How will you select an object for purposes of adjustment if a click means "put another object down"? On the other hand, if your tasks don't require much adjustment, but do require repeated placement of the same kind of object, you're pushed the other way. Our tasks seemed to us to require adjustment more than repeated placement, so we went the first way.

This example brings up an important point about using the example tasks. It's important to remember that they are ONLY EXAMPLES. Often, as in this case, a decision requires you to look beyond the specific examples you have and make a judgement about what will be common and what will be uncommon. You can't do this just by taking an inventory of the specific examples you chose. You can't defend a crummy design by saying that it handles all the examples, any more than you can defend a crummy design by saying it meets any other kind of specification.

Let us create a scenario with the use of STORYBOARDS, which are sequences of sketches showing what the screen would show, and what actions the user would take, at key points in each task. We then showed these to the users, stepping them through the tasks. Here we saw a big gain from the use of the sample tasks. They allowed us to tell the users what they really wanted to know about our proposed design, which was what it would be like to use it to do real work. A traditional design description, showing all the screens, menus, and so forth, out of the context of a real task, is pretty meaningless to users, and so they can't provide any useful reaction to it.

The sample task of using storyboards is somehow complex. What if you leave something out? And won't your design be distorted by the examples you happen to choose? And how do you know the design will work for anything OTHER than your examples?" There is a risk with any spec technique that you will leave something out. In choosing your sample tasks you do whatever you would do in any other method to be sure the important requirements are reflected. As noted above, you treat the sample tasks as examples. Using them does not relieve you of the responsibility of thinking about how other tasks would be handled. But it's better to be sure that your design can do a good job on at least some real tasks, and that it has a good chance of working on other tasks, because you've tried to design for generality, than to trust exclusively in your ability to design for generality. It's the same as that point about users: if a system is supposed to be good for EVERYBODY you'd better be sure it's good for SOMEBODY.

3.4 Creating the Initial Design

The foundation of good interface design is INTELLIGENT BORROWING. That is, you should be building your design on other people's good work rather than coming up with your own design ideas. Also, borrowing will allow designers to start their design on a best note and does not eliminate creativity. Borrowing is important for three distinct reasons. First, given the level of quality of the best user interfaces today, it's unlikely that ideas you come up with will be as good as the best ideas you could borrow. Second, there's a good chance, if you borrow from the right sources, that many of your users will already understand interface features that you borrow, whereas they'd have to invest in learning about features you invent. Finally, borrowing can save you tremendous effort in design and implementation and often in maintenance as well.

3.4.1 Working within Existing Interface Frameworks

The first borrowing you should do is to work within one of the existing user interface frameworks, such as Macintosh, Motif or Windows. The choice may have already been made for you: in in-house development your users may have PCs and already be using Windows, or in commercial development it may be obvious that the market you are trying to reach (you've already found out a lot about who's in the market, if you're following our advice) is UNIX-based. If you want to address several platforms and environments you should adopt a framework like XVT that has multi-environment support.

The advantages of working in an existing framework are overwhelming, and you should think more than twice about participating in a project where you won't be using one. It's obvious that if users are already familiar with Windows there will be big gains for them if you go along. But there are also big advantages to you, as mentioned earlier.

You will get a STYLE GUIDE that describes the various interface features of the framework, such as menus, buttons, standard editable fields and the like. The style guide will also provide at least a little advice on how to map the interface requirements of your application onto these features, though the guides aren't an adequate source on this. This information saves you a tremendous amount of work: you can, and people in the old days often did, waste huge amounts of time designing scroll bars or ways to nest menus. Nowadays these things have been done for you, and better than you could do them yourself.

You also get SOFTWARE TOOLS for implementing your design. Not only does the framework have an agreed design for menus and buttons, but it also will have code that implements these things for you. We will discuss the used of software tool for interface implementation in module 3.

3.4.2 Copying Interaction Techniques from Other Systems

Another kind of borrowing is copying specific interaction techniques from existing systems. If the style guides were good enough you might not have to do this, but the fact is the only way to get an adequate feel for how various interface features should be used, and how different kinds of information should be handled in an interface, is to look at what other applications are doing. The success of the Macintosh in developing a consistent interface style early in its life was based on the early release of a few programs whose interfaces served as models of good practice. An analogous consensus for the IBM PC doesn't really exist even today, but as it forms it is forming around prominent Windows applications like Excel or Word.

It follows from the need to borrow from other applications that you can't be a good designer without becoming familiar with leading applications. You have to seek them out, use them, and analyse them.

The key to "intelligent" borrowing, as contrasted with borrowing pure and simple, is knowing WHY things are done the way they are. If you know why an application used a tool palette rather than a menu of functions, then you have a chance of figuring out whether you want to have a palette or a menu. If you don't know why, you don't know whether the same or a different choice makes sense for you.

Bill Atkinson's MacPaint program was one of the standard- setting early Macintosh programs, and it used a tool palette, a box on the side of the window containing icons. The icons on the palette stand for various functions like "enter text", "move view window", "draw a rectangle", and the like. Why was this good design choice, rather than just listing these functions in a pull-down menu? In fact, some similar functions are listed in a pulldown menu called "goodies". So should you have a palette for what you are doing or not?

Here are some of the considerations:
Operations on menus usually do not permit or require graphical specification of parameters, though they can require responding to queries presented in a dialog box. So an operation like "draw a rectangle", in which you would click on the corners of the intended rectangle, would be odd as a menu item.

- A palette selection actually enters a MODE, a special state in which things happen differently, and keeps you there. This doesn't happen when you make a menu selection. For example, if you select the tool for drawing rectangles from a palette, your mouse clicks get interpreted as corners of rectangle until you get out of rectangle drawing mode. If you selected "draw a rectangle" from a menu, assuming the designer hadn't been worried about the point above, you'd expect to be able to draw just one rectangle, and you'd have to go back to the menu to draw another one.
- So a tool palette is appropriate when it's common to want to do a lot of one kind of thing rather than switching back and forth between different things.
- Modes are generally considered bad. An example which influenced a lot of thinking was the arrangement of input and command modes in early text editors, some of which are still around. In one of these editors what you typed would be interpreted either as text to be included in your document, if you were in input mode, or as a command, if you were in command mode. There were two big problems. First, if you forgot what mode you were in you were in trouble. Something you intended as text, if typed in command mode, could cause the system to do something dreadful like erase your file. Second, even if you remembered what mode you were in, you had the bother of changing modes when you needed to switch between entering text and entering commands.
- But modes controlled by a tool palette are considered OK because:
- there is a change in the cursor to indicate what mode you are in, so it's harder to forget;
- you only get into a mode by choosing a tool, so you know you've done it;
- It’s easy to get out of a mode by choosing another tool.
- In a tool palette, tools are designated by icons; that is little pictures, whereas in menus the choices are indicated by text. There are two sub-issues here. First, for some operations, like drawing a rectangle, it's easy to come up with an easily interpretable and memorable icon, and for others it's not. So sometimes icons will be as good as or better than text, and sometimes not. Second, icons are squarish while text items are long and thin. This means icons PACK differently on the screen: you can have a bunch of icons close together for easy viewing and picking, while text items on a menu form a long column which can be hard to view and pick from.

So, this tells you that you should use a tool palette in your application if you have operations that are often repeated consecutively, and you can think of good icons for them, and they require mouse interaction after the selection of the operation to specify fully what the operation does.

Depending on the style guide you are using, you may or may not find a good, full discussion of matters like this. One of the places where experience will pay off the most for you, and where talking with more experienced designers will be most helpful, is working out this kind of rationale for the use of various interface features in different situations.

3.4.3 When you need to Invent

At some point in most projects, you will probably feel that you've done all the copying that you can, and that you have got design problems that really call for new solutions. Here are some things to do:

- Think again about copying. Have you really beaten the bushes enough for precedents? Make another try at locating a system that does the kind of thing you need. Ask more people for leads and ideas.
- Make sure the new feature is really important. Innovation is risky and expensive. It's just not worth it for a small refinement of your design. The new feature has to be central.
- Be careful and concrete in specifying the requirements for the innovation, that is, the context in which it must work. Rough out some alternatives. Analyse them, and be sure of what you are doing.

4.0 Conclusion

In this unit, you have been introduced to the concept of understanding users. Task analysis and how it applies in user interface design was also discussed. Using task in design and creating initial design were also discussed.

5.0 Summary

In this unit, you have learnt the following:
- understanding users which involves figuring out who is going to use the program and for what it is been used.
- task analysis which includes identifying users, learning about users' tasks, coming up with a representative task description and evaluation.
- application of tasks which includes the production of a scenario that spells out what a user would have to do and what he or she would see step-by-step in performing a task using a given system.
- creating an initial design which is better achieved due to several reasons by borrowing i.e. building your design on other people's good work rather than coming up with your own design ideas.

6.0 Self-Assessment Exercise

1. Explain Task Analysis.
2. Explain the advantages and disadvantages of borrowing in creating initial designs.

7.0 References/Further Reading

Card, S.K., Moran, T. P., & Newell, A. (1986). The Model Human Processor: An Engineering Model of Human Performance. In: K. R. Boff, L. Kaufman, & J. P. Thomas (Eds.). *Handbook of Perception and Human Performance*. Vol. 2: *Cognitive Processes and Performance*. pp 1–35.

Hirakawa, K. & Parks, T.W. (2005). "Chromatic Adaptation and White-Balance Problem". *IEEE ICIP*. doi:10.1109/ICIP.2005.1530559.

Jastrzembski Tiffany & Charness Neil (2007). The Model Human Processor and the Older Adult: Parameter Estimation and Validation within a Mobile Phone Task. *Journal of Experimental Psychology: Applied***.** Volume 13, Number 4, 2007, pp 224-248.

Knudsen, E. I. (2007). "Fundamental Components of Attention". *Annual Review of Neuroscience* **30** (1): 57–78. doi:10.1146/annurev.neuro.30.051606.094256. PMID 17417935.

Lui, Yili, Feyen, Robert, & Tsimhoni, Omer (2006). Queueing Network-Model Human Processor (QN-MHP): A Computational Architecture for Multitask Performance in Human-Machine Systems. **ACM Transactions on Computer-Human Interaction.** Volume 13, Number 1, March 2006, pp 37-70.

Morrell, Jessica Page (2006). *Between the Lines: Master the Subtle Elements of Fiction Writing*. Cincinnati, OH: Writer's Digest Books. ISBN 1582973938.

Pattyn, N., Neyt, X., Henderickx, D., & Soetens, E.(2008). Psychophysiological Investigation of Vigilance Decrement: Boredom or Cognitive Fatigue? *Physiology & Behaviour*, 93, 369-378.

Pinel, J. P. (2008). *Biopsychology* (7th ed.). Boston: Pearson. p.357.

Robles-De-La-Torre G. (2006). The Importance of the Sense of Touch in Virtual and Real Environments. *IEEE Multimedia* 13(3), Special issue on Haptic User Interfaces for Multimedia Systems, pp. 24-30.

Wright, R.D. & Ward, L.M. (2008). *Orienting of Attention*. Oxford University Press.

UNIT 3 User- Centred Design (Ucd)

1.0 Introduction

Reading through this unit, you will be introduced to User Centred Design (UCD), its purpose, the major considerations and the various approaches in UCD. The concept of participatory design will also be introduced.

2.0 Objectives

At the end of this unit, you should be able to:

- explain the term User Centred Design
- describe the various purpose of UCD
- discuss the major considerations and various approaches in UCD
- describe the concept of participatory design.

3.0 Main Content

3.1 Introduction to User-Centred Design

User-centred design (UCD) is a design philosophy and a process in which the needs, wants, and limitations of the end user of an interface or document are given extensive attention at each stage of the design process. User-centered design can be characterised as a multi-stage problem solving process that not only requires designers to analyse and foresee how users are likely to use an interface, but also to test the validity of their assumptions with regards to user behaviour in real world tests with actual users. Such testing is necessary as it is often very difficult for the designers of an interface to understand intuitively what a first-time user of their design experiences, and what each user's learning curve may look like.

The main difference between UCD and other interface design philosophies is that user-centered design tries to optimise the user interface around how people can, want, or need to work, rather than forcing the users to change how they work to accommodate the software developers approach.

3.2 Purpose of User-Centred Design

User-centered design answers questions about users and their tasks and goals then use the findings to make decisions about development and design. UCD seeks to answer the following questions:

- Who are the users of the system?
- What are the users' tasks and goals?
- What are the users' experience levels with the system?
- What functions do the users need from the system?
- What information might the users need, and in what form do they need it?
- How do users think the system should work?

3.3 Major Considerations of UCD

Visibility
Visibility helps the user construct a mental model of the document. Models help the user predict the effect(s) of their actions while using the document. Important elements (such as those that aid navigation) should be emphatic. Users should be able to tell from a glance what they can and cannot do with the system.

Accessibility
Users should be able to find information quickly and easily throughout the system, whether long or short. Users should be offered various ways to find information (such navigational elements, search functions, table of contents, clearly labeled sections, page numbers, colour coding, etc). Navigational elements should be consistent with the genre of the system. 'Chunking' is a useful strategy that involves breaking information into small pieces that can be organised into some type meaningful order or hierarchy. The ability to skim the system allows users to find their piece of information by scanning rather than reading.

Legibility
Text should be easy to read. Through analysis of the rhetorical situation, the designer should be able to determine a useful font style. Ornamental fonts and text in all capital letters are hard to read, but italics and bolding can be helpful when used correctly. Large or small body text is also hard to read. (Screen size of 10-12 pixel sans-serif nand 12-16 pixel serif is recommended.) High figure-ground contrast between text and background increases legibility. Dark text against a light background is most legible.

Language
Depending on the rhetorical situation certain types of language are needed. Short sentences are helpful, as well as short, well written texts used in explanations and similar bulk-text situations. Unless the situation calls for it don't use jargon or technical terms. Many writers will choose to use active voice, verbs (instead of noun strings or nominals), and simple sentence structure.

3.3.1 Rhetorical Situation

A User Centered Design is focused around the rhetorical situation. The rhetorical situation shapes the design of an information medium. There are three elements to consider in a rhetorical situation: Audience, Purpose, and Context.

Audience
The audience is the people who will be using the document. The designer must consider their age, geographical location, ethnicity, gender, education, etc.

Purpose
The purpose is how the document will be used, and what the audience will be trying to accomplish while using the document. The purpose usually includes purchasing a product, selling ideas, performing a task, instruction, and all types of persuasion.

Context
The context is the circumstances surrounding the situation. The context often answers the question: What situation has prompted the need for this document? Context also includes any social or cultural issues that may surround the situation.

3.4 UCD Models and Approaches

Models of a user centered design process help software designers to fulfill the goal of a product engineered for their users. In these models, user requirements are considered right from the beginning and included into the whole product cycle. Their major characteristics are the active participation of real users, as well as an iteration of design solutions.

- Cooperative design: involving designers and users on an equal footing. This is the Scandinavian tradition of design of computer mediated information systems and it has been evolving since 1970.
- Participatory design (PD): a North American term for the same concept, inspired by Cooperative Design, focusing on the participation of users. Since 1990, there has been a bi-annual Participatory Design Conference.
- Contextual design: “customer centred design” in the actual context, including some ideas from Participatory design.

In the next section, we will discuss participatory design as an example of UCD.

All these approaches follow the ISO standard Human-centered design processes for interactive systems (ISO 13407 Model, 1999).

3.5 User-Centred System Development

The user-centred development process follows the standard software development model (Like waterfall model). User-centred interface designers build rapid prototypes and test them with the users to validate or rebut the idea.

There are six major steps in user-centred development are:

Design Research Techniques
Using design research techniques (observations, interviews, questionnaires, and related activities) will allow better understanding of the users and what their interest are. Investigate users and their environment in order to learn more about them and thus be better able to design for them.

Research Analysis and Concept Generation
Drawing on a combination of user research, technological possibilities, and business opportunities, designers create concepts for new software, products, services, or systems. This process may involve multiple rounds of brainstorming, discussion, and refinement.

To help designers realise user requirements, they may use tools such as personas or user profiles that are reflective of their targeted user group. From the users, and the patterns of behaviour observed in the research, designers create scenarios (or user stories) or storyboards, which imagine a future work flow that the users will go through using the product or service.

After thorough analysis using various tools and models, designers create a high level summary spanning across all levels of user requirements. This includes a vision statement regarding the current and future goals of a project.

Alternative Design and Evaluation
Once a clear view of the problem space exists, designers will develop alternative solutions with crude prototypes to help convey concepts and ideas. Proposed solutions are evaluated

and perhaps even merged. The end result should be a design that solves as many of the user requirements as possible.

Some tools that may be used for this process are wire-framing and flow diagrams. The features and functionality of a product or service are often outlined in a document known as a wireframe ("schematics" is an alternate term). Wireframes are a page-by-page or screen-by-screen detail of the system, which include notes ("annotations") as to how the system will operate. Flow diagrams outline the logic and steps of the system or an individual feature.

Prototyping and Usability Testing
User-centred interface designers use a variety of prototyping techniques to test aspects of design ideas. These can be roughly divided into three classes: those that test the **role** of an artefact, those that test its **look and feel** and those that test its **implementation**. Sometimes, these are called **experience prototypes** to emphasise their interactive nature. Prototypes can be physical or digital, high- or low-fidelity.

Implementation
There is also the need to ensure that what was designed is implemented correctly. Standard tools and methodologies can be used to ensure efficient implementation.

System Testing
Before the system is put into practical use, standard usability testing should be carried out. It is necessary to ensure that all forms of errors and bugs are removed from the system but conducting effective system testing.

3.6 Participatory Design

Participatory design is an approach to design that attempts to actively involve the end users in the design process to help ensure that the product designed meets their needs and is usable. It is also used in urban design, architecture, landscape architecture and planning as a way of creating environments that are more responsive and appropriate to their inhabitants and users cultural, emotional, spiritual and practical needs. It is important to understand that this approach is focused on process and is not a design style. For some, this approach has a political dimension of user empowerment and democratisation. For others, it is seen as a way of abrogating design responsibility and innovation by designers.

In participatory design end-users (putative, potential or future) are invited to cooperate with researchers and developers during an innovation process. Potentially, they participate during several stages of an innovation process: they participate during the initial exploration and problem definition both to help define the problem and to focus ideas for solution, and during development, they help evaluate proposed solutions.

Participatory design can be seen as a move of end-users into the world of researchers and developers, whereas empathic design can be seen as a move of researchers and developers into the world of end-users.

3.6.1 Participatory Design and User Interface Design

As mentioned earlier, is a concept of user-centered design and requires involving end users in the design process. Previous researches have shown that user-centered approach to interface design will enhance productivity. Users' interests and active involvements are the main focus of participatory design. Consequently, participatory approach is a key design technique of user interface designer.

3.7 Interaction Design

Interaction Design **(**IxD**)** is the discipline of defining the behaviour of products and systems that a user can interact with. The practice typically centers on complex technology systems such as software, mobile devices, and other electronic devices. However, it can also apply to other types of products and services, and even organisations themselves. Interaction design defines the behaviour (the "interaction") of an artifact or system in response to its users. Certain basic principles of cognitive psychology provide grounding for interaction design. These include mental models, mapping, interface metaphors, and affordances. Many of these are laid out in Donald Norman's influential book *The Psychology of Everyday Things.* Academic research in Human Computer Interaction (HCI) includes methods for describing and testing the usability of interacting with an interface, such as cognitive dimensions and the cognitive walkthrough.

Interaction designers are typically informed through iterative cycles of user research. They design with an emphasis on user goals and experience, and evaluate designs in terms of usability and affective influence.

4.0 Conclusion

In this unit, you have been introduced to the User Centered Design (UCD), its purpose, the major considerations and the various approaches in UCD. The concept of participatory design was also introduced.

5.0 Summary

You have learnt the following in this unit:

- introduction to **User-centred design (UCD)** which is a design philosophy and a process in which the needs, wants, and limitations of the end user of an interface or document are given extensive attention at each stage of the design process
- the purpose of UCD which answers the questions about users and their tasks and goals
- the purpose of UCD which includes visibility, accessibility, legibility, e.t.c.
- UCD Models which are user centered design process that helps software designers to fulfill the goal of a product engineered for their users
- the concepts of participatory and interactive design.

6.0 Self-Assessment Exercise

1. Describe the UCD Model.
2. Write a short note on participatory design and user interface design.

7.0 References/Further Reading

Alan Cooper, Robert M. Reimann & David Cronin (2007). *About Face 3: The Essentials of Interaction Design* (3rd ed.). Wiley. ISBN 0-4700-8411-1.

Helen Sharp, Yvonne Rogers, & Jenny Preece (2007). *Interaction Design - Beyond Human-Computer Interaction*. (2nd ed.). John Wiley & Sons. pp. 181-217.

Lui, Yili, Feyen, Robert, & Tsimhoni, Omer (2006). Queueing Network-Model Human Processor(QN-MHP): A Computational Architecture for Multitask Performance in Human-Machine Systems. *ACM Transactions on Computer-Human Interaction.* Volume 13, Number 1, March, pp. 37-70.

www.wikipedia.org

Unit 4 Usability

1.0 Introduction

Having read through the course guide, you will be introduced to usability, usability concepts and guidelines, usability considerations, ISO standard for usability and various usability methodologies.

2.0 Objectives

At the end of this unit, you should be able to:

- explain usability
- identify the various concepts and guidelines of usability
- discuss the ISO standard for usability
- state the various usability methodologies.

3.0 Main Content

3.1 Introduction to Usability

Usability is a term used to denote the ease with which people can employ a particular tool or other human-made object in order to achieve a particular goal. Usability is a qualitative attribute that assesses how easy user interfaces are to use. The word "usability" also refers to methods for improving ease-of-use during the design process. Usability can also refer to the methods of measuring usability and the study of the principles behind an object's perceived efficiency or elegance.

In human-computer interaction and computer science, usability usually refers to the elegance and clarity with which the interaction with a computer program or a web site is designed. The term is also used often in the context of products like consumer electronics, or in the areas of communication, and knowledge transfer objects (such as a cookbook, a document or online help). It can also refer to the efficient design of mechanical objects such as a door handle or a hammer.

The primary notion of usability is that an object designed with a generalised users' psychology and physiology in mind is, for example:

- More efficient to use—it takes less time to accomplish a particular task
- Easier to learn—operation can be learned by observing the object
- More satisfying to use

Complex computer systems are finding their way into everyday life, and at the same time the market is becoming saturated with competing brands. This has led to usability becoming more popular and widely recognised in recent years as companies see the benefits of researching and developing their products with user-oriented instead of technology-oriented methods. By understanding and researching the interaction between product and user, the *usability expert* can also provide insight that is unattainable by traditional company-oriented market research. For example, after observing and interviewing users, the usability expert may identify needed functionality or design flaws that were not anticipated. A method called "contextual inquiry" does this in the naturally occurring context of the users own environment.

In the user-centered design paradigm, the product is designed with its intended users in mind at all times. In the user-driven or participatory design paradigm, some of the users become actual or de facto members of the design team.

The term *user- friendly* is often used as a synonym for *usable*, though it may also refer to accessibility. Usability is also used to describe the quality of user experience across websites, software, products and environments.

There is no consensus about the relation of the terms ergonomics (or human factors) and usability. Some think of usability as the software specialisation of the larger topic of ergonomics. Others view these topics as tangential, with ergonomics focusing on physiological matters (e.g., turning a door handle) and usability focusing on psychological matters (e.g., recognising that a door can be opened by turning its handle).

Usability is also very important in website development. Studies of user behaviour on the Web find a low tolerance for difficult designs or slow sites. People do not want to wait. They do not also want to learn how to use a home page. There is no such thing as a training class or a manual for a Web site. People have to be able to grasp the functioning of the site immediately after scanning the home page for a few seconds at most. Otherwise, most casual users will simply leave the site and continue browsing or shopping somewhere else.

3.2 Usability Concepts and Guidelines

The major concepts of usability are:

Efficiency: Once users have learned the design, how quickly can they perform tasks?

Learnability: How easy is it for users to accomplish basic tasks the first time they encounter the design?

Memorability: When users return to the design after a period of not using it, how easily can they're establish proficiency?

Errors: How many errors do users make, how severe are these errors, and how easily can they recover from the errors?

Satisfaction: How pleasant is it to use the design?
Usability is often associated with the functionalities of the product, in addition to being solely a characteristic of the user interface (cf. framework of system acceptability, also below, which separates *usefulness* into *utility* and *usability*). For example, in the context of mainstream consumer products, an automobile lacking a reverse gear could be considered *unusable* according to the former view, and *lacking in utility* according to the latter view.

When evaluating user interfaces for usability, the definition can be as simple as "the perception of a target user of the effectiveness (fit for purpose) and efficiency (work or time required to use) of the Interface". Each component may be measured subjectively against criteria e.g. Principles of User Interface Design, to provide a metric, often expressed as a percentage.

It is important to distinguish between **usability testing** and **usability engineering**. **Usability testing** is the measurement of ease of use of a product or piece of software. In contrast, **Usability Engineering (UE)** is the research and design process that ensures a product with good usability.

Usability is an example of a non-functional requirement. As with other non-functional requirements, usability cannot be directly measured but must be quantified by means of indirect measures or attributes such as, for example, the number of reported problems with ease-of-use of a system.

The key principle for maximising usability is to employ iterative design, which progressively refines the design through evaluation from the early stages of design. The evaluation steps enable the designers and developers to incorporate user and client feedback until the system reaches an acceptable level of usability.

The preferred method for ensuring usability is to test actual users on a working system. Although, there are many methods for studying usability, the most basic and useful is user testing, which has three components:

- get some representative users
- ask the users to perform representative tasks with the design
- observe what the users do, where they succeed, and where they have difficulties with the user interface.

It is important to test users individually and let them solve any problems on their own. If you help them or direct their attention to any particular part of the screen, you will bias the test. Rather than running a big, expensive study, it's better to run many small tests and revise the design between each one so you can fix the usability flaws as you identify them. Iterative design is the best way to increase the quality of user experience. The more versions and interface ideas you test with users, the better.

Usability plays a role in each stage of the design process. The resulting need for multiple studies is one reason to make individual studies fast and cheap, and to perform usability testing early in the design process.

During user interface design stage, the following are the major usability guidelines:

- before starting the new design, test the old design to identify the good parts that you should keep or emphasise, and the bad parts that give users trouble
- test competitors' designs to get data on a range of alternative designs
- conduct a field study to see how users behave in their natural habitat.
- make paper prototypes of one or more new design ideas and test them. The less time you invest in these design ideas the better, because you'll need to change them all based on the test results
- refine the design ideas that test best through multiple iterations, gradually moving from low-fidelity prototyping to high-fidelity representations that run on the computer. Test each iteration
- inspect the design relative to established usability guidelines, whether from your own earlier studies or published research
- once you decide on and implement the final design, test it again. Subtle usability problems always creep in during implementation.

Do not defer user testing until you have a fully implemented design. If you do, it will be impossible to fix the vast majority of the critical usability problems that the test uncovers.

Many of these problems are likely to be structural, and fixing them would require major re-architecting. The only way to a high-quality user experience is to start user testing early in the design process and to keep testing every step of the way.

3.3 Usability Considerations

Usability includes considerations such as:

- Who are the users, what do they know, and what can they learn?
- What do users want or need to do?
- What is the general background of the users?
- What is the context in which the user is working?
- What has to be left to the machine?

Answers to these can be obtained by conducting user and task analysis at the start of the project.

Other considerations are:

- Can users easily accomplish their intended tasks? For example, can users accomplish intended tasks at their intended speed?
- How much training do users need?
- What documentation or other supporting materials are available to help the user? Can users find the solutions they seek in these materials?
- What and how many errors do users make when interacting with the product?
- Can the user recover from errors? What do users have to do to recover from errors? Does the product help users recover from errors? For example, does software present comprehensible, informative, non-threatening error messages?
- Are there provisions for meeting the special needs of users with disabilities? (accessibility)
- Are there substantial differences between the cognitive approaches of various users that will affect the design or can a one size fits all approach be used?

Examples of ways to find answers to these and other questions are: user-focused requirements analysis, building user profiles, and usability testing.

Discoverability
Even if software is usable as per the above considerations, it may still be difficult to *learn* to use. Other questions that must be asked are:

- Is the user ever expected to do something that is not obvious? (e.g. are important features only accessible by right-clicking on a menu header, on a text box, or on an unusual GUI element?)
- Are there hints and tips and shortcuts that appear as the user is using the software?
- Should there be instructions in the manual that actually belong as contextual tips shown in the program?
- Is the user at a disadvantage for not knowing certain keyboard shortcuts? A user has the right to know all major and minor keyboard shortcuts and features of an application.
- Is the learning curve (of hints and tips) skewed towards point-and-click users rather than keyboard users?
- Are there any "hidden" or undocumented keyboard shortcuts that would better be revealed in a "Keyboard Shortcuts" Help-Menu item? A strategy to prevent this

"undocumented feature disconnect" is to automatically generate a list of keyboard shortcuts from their definitions in the code.

3.4 Usability Design Methodologies

Any system that is designed for people should be easy to use, easy to learn, and useful for the users. The usability design methodologies are similar to user-centered approach that was explained since both are satisfying the needs and requirements of users. Therefore, when designing for usability, the three principles of design are: early focus on users and tasks, user-centred design and testing.

Early Focus on Users and Tasks
The design team should be user driven and direct contact with potential users is recommended. Several evaluation methods including personas, cognitive modeling, inspection, inquiry, prototyping, and testing methods may be used to gain an understanding of the potential users.

Usability considerations such as who the users are and their experience with similar systems must be examined. As part of understanding users, this knowledge must "be played against the tasks that the users will be expected to perform. This includes the analysis of what tasks the users will perform, which are most important, and what decisions the users will make while using your system. Designers must understand how cognitive and emotional characteristics of users will relate to a proposed system.

One way to stress the importance of these issues in the designers' minds is to use personas, which are made-up representative users. See below for further discussion of personas. Another more expensive but more insightful way to go about it, is to have a panel of potential users work closely with the design team starting in the early formation stages. This has been fully discussed in unit 2, of this module.

User-Centred Design
This was discussed in unit 3 of this module. Ultimately, user-centred design works towards meeting the interest and goals of users such as making the system user friendly, easy to use, easy to operate, simple, etc.

Testing
This includes testing the system for both learnability and usability. (See evaluation methods in module 4). It is important in this stage to use quantitative usability specifications such as time and errors to complete tasks and number of users to test, as well as examine performance and attitudes of the users testing the system. Finally, "reviewing or demonstrating" a system before the user tests it can result in misleading results. More testing and evaluation methods are described in module 4.

4.0 Conclusion

In this unit, you have been introduced to usability, usability concepts and guidelines, usability considerations, ISO standard for usability and various usability methodologies.

5.0 Summary

In this unit, you have learnt the following:

- introduction to **usability** which is a term used to denote the ease with which people can employ a particular tool or other human-made object in order to achieve a particular goal
- usability concepts and guidelines which includes efficiency, learnability, memorability e.t.c
- usability considerations like who the users are, what the user needs, e.t.c
- usability methodologies like early focus on users and tasks, iterative design and testing.

6.0 Self Assessment Exercise

1. Explain usability.
2. Discuss any two usability methodologies.

7.0 References/Further Reading

Knudsen, E. I. (2007). "Fundamental Components of Attention". *Annual Review of Neuroscience* **30** (1): 57–78. doi:10.1146/annurev.neuro. 30.051606.094256. PMID 17417935.

Kuniavsky, M. (2003). *Observing the User Experience: A Practitioner's Guide to User Research*, San Francisco, CA: Morgan Kaufmann.

McKeown, Celine (2008). *Office ergonomics: practical applications*. Boca Raton, FL: Taylor & Francis Group, LLC.

Pattyn, N., Neyt, X., Henderickx, D., & Soetens, E. (2008). Psychophysiological Investigation of Vigilance Decrement: Boredom or Cognitive Fatigue? *Physiology & Behaviour,* 93, 369-378.

Wickens, C. D. *et al.* (2004). *An Introduction to Human Factors Engineering* (2nd ed.).Pearson Education, Inc., Upper Saddle River, NJ: Prentice Hall.

Wright, R.D. & Ward, L.M. (2008). *Orienting of Attention*. Oxford University Press

Unit 5 Interaction Styles and Graphic Design Principles

1.0 Introduction

This unit will introduce you to interaction styles and graphic design principles.

2.0 Objectives

At the end of this unit, you should be able to:
- explain different interaction styles
- identify the advantages and disadvantages of different interaction styles
- explain graphic design principles.

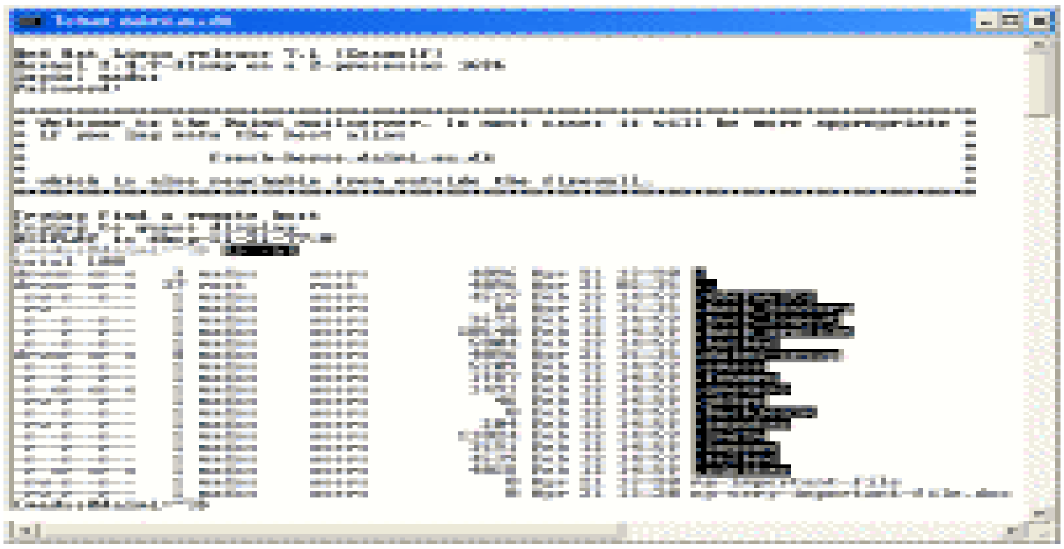
3.0 Main Content

3.1 Interaction Styles

The concept of **Interaction Style** refers to all the ways the user can communicate or otherwise interact with the computer system. The concept is known in HCI and User Interface Design and generally has its roots in the computer systems. These concepts do, however, retain some of their descriptive powers outside the computer medium. For example, you can talk about menu selection (defined below) in mobile phones.

The most common types of interaction styles mentioned are command language, form filling, menu selection, and direct manipulation.

Command Language (or Command Entry)
Command language is the earliest form of interaction style and is still being used, though mainly on Linux/Unix operating systems. These "Command prompts" are used by (usually) expert users who type in commands and possibly some parameters that will affect the way the command is executed. The following screen dump shows a command prompt - in this case, the user has logged on to a (mail) server and can use the server's functions by typing in commands.



```
[illegible]
```

Fig. 5.1: Command Prompt from UNIX Operating System

The command "ls- al" has just been executed *('ls' stands for 'list' and the parameters '-al' specify that the list command should display a detailed list of files).*

Command language places a considerable cognitive burden on the user in that the interaction style relies on recall as opposed to recognition memory. Commands as well as their many parameterised options have to be learned by heart and the user is given no help in this task of retrieving command names from memory. This task is not made easier by the fact that many commands (like the 'ls' command in the above example) are abbreviated in order to minimise the number of necessary keystrokes when typing commands. The learnability of command languages is generally very poor.

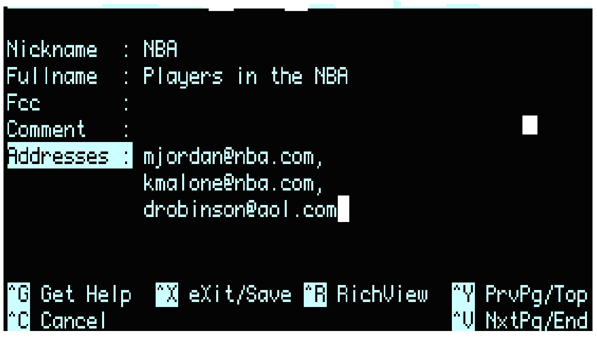
Advantages and disadvantages of Command Language

Advantages
- Flexible
- Appeals to expert users
- Supports creation of user-defined "scripts" or macros
- Is suitable for interacting with networked computers even with low bandwidth.

Disadvantages
- Retention of commands is generally very poor
- Learnability of commands is very poor
- Error rates are high
- Error messages and assistance are difficult to provide because of the diversity of possibilities plus the complexity of mapping from tasks to interface concepts and syntax
- Not suitable for non-expert users.

Form Fillin

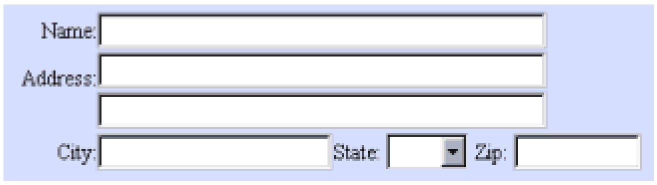
The form fillin interaction style (also called "fill in the blanks") was aimed at a different set of users, especially, non-experts users. When form filling interfaces first appeared, the whole interface style was form-based, unlike much of today's software that mix forms with other interaction styles. Back then, the screen was designed as a form in which data could be entered in the pre-defined form fields. The TAB-key was (and still is) used to switch between the fields and ENTER to submit the form. Thus, there was originally no need for a pointing device such as a mouse and the separation of data in fields allowed for validation of the input. Form filling interfaces were (and still is) especially useful for routine, clerical work or for tasks that require a great deal of data entry. Some examples of form fillin are shown below.



```
Nickname  : NBA
Fullname  : Players in the NBA
Fcc       :
Comment   :
Addresses : mjordan@nba.com,
            kmalone@nba.com,
            drobinson@aol.com

^G Get Help  ^X eXit/Save ^R RichView  ^Y PrvPg/Top
^C Cancel                              ^V NxtPg/End
```

***Fig. 5.2**: Classic Form Fillin via a Terminal*



Name:					
Address:					
City:		State:		Zip:	

Fig. 5.3: **More Modern-day Form Fillin**

(Source: Petrie, 2006)

Even today, a lot of computer programs like video rental software, financial systems, pay roll systems etc. are still purely forms-based.

Please note that "form fillin" is not an abbreviation of "form filling". Instead, it should be read "form fill-*in*".

Advantages and Disadvantages of Form Fillin

Advantages

- Simplifies data entry
- Shortens learning in that the fields are predefined and need only be 'recognised'
- *Guides* the user via the predefined rules.

Disadvantages

- Consumes muchscreen space
- Usually sets the scene for rigid formalisation of the business processes.

Menu selection

A menu is a set of options displayed on the screen where the selection and execution of one (or more) of the options results in a state change of the system. Using a system based on menu-selection, the user selects a command from a predefined selection of commands arranged in menus and observes the effect. If the labels on the menus/commands are understandable (and grouped well) users can accomplish their tasks with negligible learning or memorisation as finding a command/menu item is a recognition as opposed to recall memory task (see recall versus recognition). To save screen space, menu items are often clustered in pull-down or pop-up menus. Some examples of menu selection are shown figures 8 and 9.

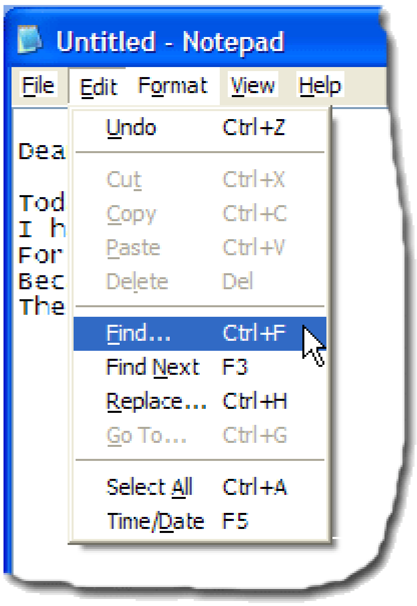


Fig. 5.4: Contemporary Menu Selection

(Notepad by Microsoft)

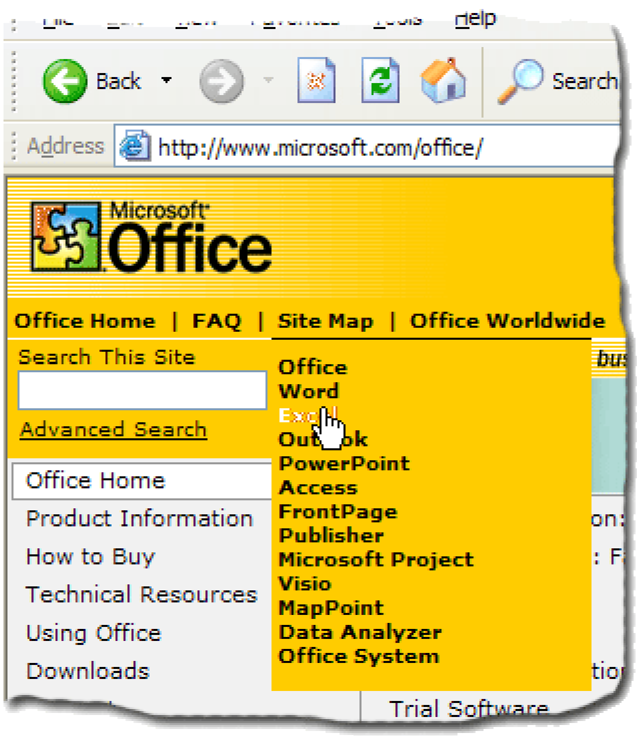


Fig. 5.5: Menu Selection in the Form of a Webpage (microsoft.com)

Webpage in general can be said to be based on menu selection.

Advantages of Menu Selection

- Ideal for novice or intermittent users
- Can appeal to expert users if display and selection mechanisms are rapid and if appropriate "shortcuts" are implemented

- Affords exploration (users can "look around" in the menus for the appropriate command, unlike having to remember the name of a command *and* its spelling when using command language)
- Structures decision making
- Allows easy support of error handling as the user's input does not have to be parsed (as with command language).
  Disadvantages of Menu Selection
- Too many menus may lead to information overload or complexity of discouraging proportions
- May be slow for frequent users
- May not be suited for small graphic displays.

Direct Manipulation
Direct manipulation captures the idea of "direct manipulation of the object of interest", which means that objects of interest are represented as distinguishable objects in the UI and are manipulated in a direct fashion. Direct manipulation is a central theme in interface design.

Direct manipulation systems have the following characteristics:

- Visibility of the object of interest
- Rapid, reversible, incremental actions
- Replacement of complex command language syntax by direct manipulation of the object of interest.

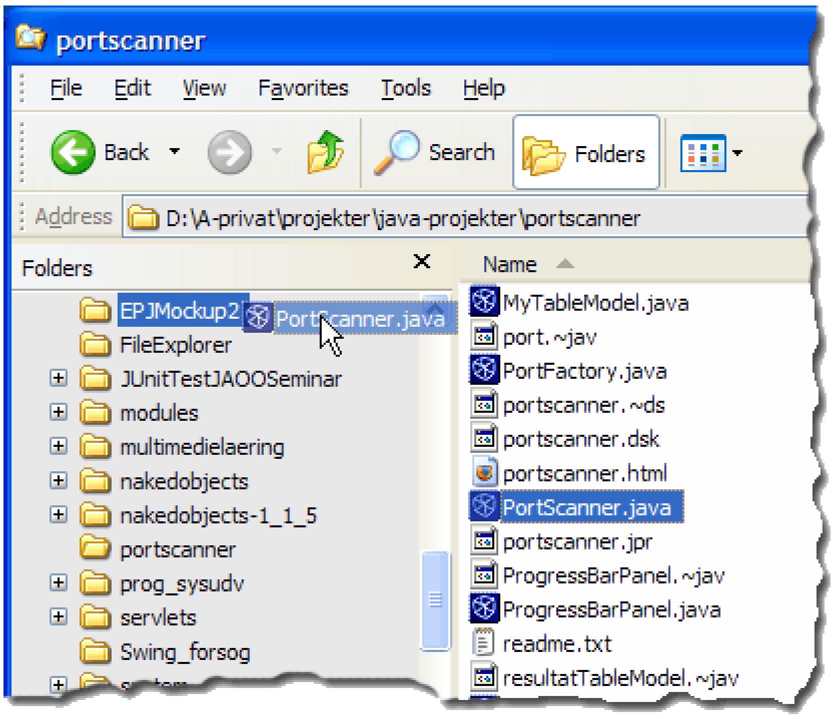


Fig. 5.6: The Text Book Example of Direct Manipulation, the Windows File Explorer, where files are dragged and dropped

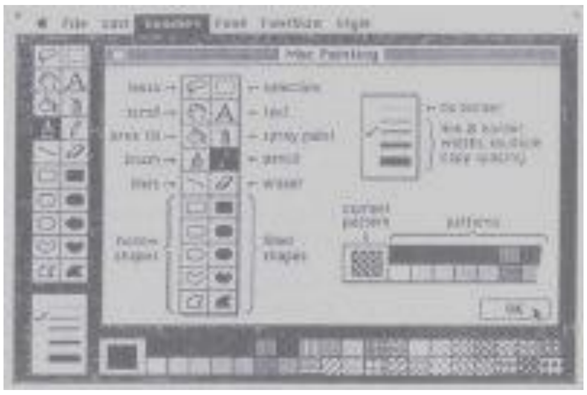


Fig. 5.7: **The Screen of One of the Earliest Commercially Available Direct Manipulation Interfaces Called MacPaint**

Advantages of Direct Manipulation

- Visually presents task concepts
- Easy to learn
- Errors can be avoided more easily
- Encourages exploration
- High subjective satisfaction
- Recognition memory (as opposed to cued or free recall memory).

Disadvantages

- May be more difficult to programme
- Not suitable for small graphic displays
- Spatial and visual representation is not always preferable
- Compact notations may better suit expert users.

3.2 Graphic Design Principles

The graphic design of an interface involves decisions about issues such as

- Where to put things on the screen?
- What size and font to use?
- What colours will work best?

For these questions as for other, more substantive design issues and intelligent borrowing should be one's first approach. But that often leaves one with a lot of decisions still to be made. Here are a few principles of graphic design that will not only make your interface more attractive, but will also make it more usable. Each principle is accompanied by a description of WHY it is important. So, one will be able to consider the tradeoffs when there is a conflict between two principles or between a design principle and a borrowed technique.

The Clustering Principle

Organise the screen into visually separate blocks of similar controls, preferably with a title for each block.

"Controls," as we use the word here, include menus, dialog boxes, on-screen buttons, and any other graphic element that allows the user to interact with the computer. Modern WIMP (Windows-Icons-Menus-Pointer) systems are a natural expression of the Clustering Principle. Similar commands should be on the same menu, which places them in close proximity visually and gives them a single title. Large numbers of commands related to a given area of functionality may also show up in a dialog box, again a visually defined block.

But the same principle should apply if you one is designing a special control screen with many buttons or displays visible, perhaps a touch-screen interface. The buttons for a given function should be grouped together, then visually delineated by colour, or a bounding box, or surrounding space ("white space"). The principle should also be applied within WIMP systems when one is designing a dialog box: If there is a subgroup of related functions, put them together in the box.

There are two important reasons for the clustering principle. First, it helps users search for the command they need. If one is looking for the "Print setup" menu, it's easier to find if it's in a box or menu with the label "Printer" then if it's one of hundreds of command buttons randomly distributed on the top of the screen. Second, grouping commands together helps the user acquire a conceptual organisation for facts about the program. It is useful to know, for example, that Bold, Italic, and Outline are all one kind of font modification, while Times Roman, Palatino, and Courier are another kind. (That distinction, common to most PC-based word processors, doesn't hold for many typesetting systems, where users have to acquire a different conceptual organisation.)

The Visibility Reflects Usefulness Principle
Make frequently used controls obvious, visible, and easy to access; conversely, hide or shrink controls that are used less often.

This is a principle that WIMP systems implement with dialog boxes and, in many recent systems, with "toolbars" of icons for frequently used functions. The reasoning behind this principle is that users can quickly search a small set of controls, and if that set contains the most frequently used items, they'll be able to find and use those controls quickly. A more extended search, through dialog boxes, for example, is justified for controls that are used infrequently.

The Intelligent Consistency Principle
Use similar screens for similar functions. This is similar to intelligent borrowing, but in this case you're borrowing from one part of your design and applying it to another part. The reasoning should be obvious: Once users learn where the controls are on one screen (the "Help" button, for example), they should be able to apply that knowledge to other screens within the same system. This approach allows one to make a concentrated effort to design just a few attractive, workable screens, and then modify those slightly for use in other parts of the application.

However, be careful to use consistency in a meaningful way. Screens should not look alike if they actually do significantly different things. A critical error warning in a real-time system should produce a display that's very different from a help screen or an information message.

The Colour as a Supplement Principle
Do not rely on colour to carry information; use it sparingly to emphasise information provided through other means. Colour is much easier to misuse than to use well. Different colours mean different things to different people, and that relationship varies greatly from culture to culture. Red, for example, means danger in the U.S., death in Egypt, and life in India. An additional problem is that some users can't distinguish colours: about seven percent of all adults have some form of colour vision deficiency.

A good principle for most interfaces is to design them in black and white, make sure they are workable, then add minimal colour to the final design. Colour is certainly useful when a warning or informational message needs to stand out, but be sure to provide additional cues to users who cannot perceive the colour change.

Unless one is an experienced graphic designer, minimal colour is also the best design principle for producing an attractive interface. One should stick with grays for most of the system, with a small amount of bright colour in a logo or a label field to distinguish your product. Remember that many users can and frequently do revise the colour of their windows, highlighting, and other system parameters. Build a product that will work with that user input, not one that fights it.

The Reduced Clutter Principle
Do not put "too much" on the screen.

This loosely defined principle is a good checkpoint to confirm that your design reflects the other principles listed above. If only the most highly used controls are visible, and if controls are grouped into a small number of visual clusters, and if you've used minimal colour, then the screen should be graphically attractive.

This is also a good principle to apply for issues that we have not dealt with specifically. For type size and font, for example: the Reduced Clutter Principle would suggest that one or two type styles are sufficient. Do not try to distinguish each menu by its own font, or work with a large range of sizes. Users typically will not notice the distinction, but they will notice the clutter.

4.0 Conclusion

In this unit, you were introduced to different interaction styles and graphic design principles.

5.0 Summary

The following are the summary of what were discussed in this unit:
- introduction to **interaction styles** which refers to all the ways the user can communicate or otherwise interact with the computer system
- the advantages and disadvantages of various styles like command language and form fillin
- graphics design principles like the clustering principle, visibility reflects usefulness principle, e.t.c.

6.0 Self-Assessment Exercise

1. Explain any two interaction styles.
2. Write a short note on any of the graphic design principle.

7.0 References/Further Reading

Crandall, B., Klein, G., & Hoffman, R. (2006). *Working Minds: A Practitioner's Guide to Cognitive Task Analysis*. MIT Press.

Hackos, JoAnn T. & Redish, Janice C. (1998). *User and Task Analysis for Interface Design*. Wiley.

Petrie, J. (2006). "Mixed-Fidelity Prototyping of User Interfaces." M.Sc. Thesis.