NATIONAL OPEN UNIVERSITY OF NIGERIA

SCHOOL OF SCIENCE AND TECHNOLOGY

COURSE CODE:CIT 811

COURSE TITLE:USER INTERFACE DESIGN AND ERGONOMICS

COURSE GUIDE

CIT 811
USER INTERFACE DESIGN AND ERGONOMICS

Course Developer DR. A. S. SODIYA

Course Editor

Course Co-ordinator Afolorunso, A. A.
National Open University of Nigeria
Lagos.

NATIONAL OPEN UNIVERSITY OF NIGERIA

National Open University of Nigeria
Headquarters
14/16 Ahmadu Bello Way
Victoria Island
Lagos

Abuja Annex
245 Samuel Adesujo Ademulegun Street
Central Business District
Opposite Arewa Suites
Abuja

e-mail: centralinfo@nou.edu.ng
URL: www.nou.edu.ng

First Printed 2009

ISBN

Printed by ……..
For
National Open University of Nigeria

TABLE OF CONTENTS **PAGE**

Introduction

CIT 811 – User Interface Design and Ergonomics is a three [3] credit unit course of twenty one units. It discusses the introduction, design, implementation and evaluation of user interface. The course material was clearly designed to enhance the understanding of students. The aim of this course is to equip you with the basic skills of studying and understanding the concept of user interface design. The techniques involved will be explicitely covered while evaluation procedures will also be discussed. By the end of the course, you should be able to confidently study, analyse and design a standard User Interface.

This Course Guide gives you a brief overview of the course content, course duration, and course materials.

What you will learn in this course

The main purpose of this course is to provide the necessary tools analyzing and designing user interface systems. It makes available the steps and tools that will enable you to make proper and accurate decision about necessary design technique whenever the need arises. This, we intend to achieve through the following:

Course Aims

i. To introduce the concepts user interface design and egonomics;
ii. To describe the various techniques for designing user interface;
iii. To describe the practical implementation of user interface; and
iv. To discuss the techniques for user interface evaluation.

Course Objectives

Certain objectives have been set out to ensure that the course achieves its aims. Apart from the course objectives, every unit of this course has set objectives. In the course of the study, you will need to confirm at the end of each unit if you have met the objectives set at the beginning of each unit. By the end of this course, you should be able to:

i. the essentials of good interface design techniques

iii. describe in general term, the concept of user interface design;

ii. describe design a standard user Interface;.

iv. analyse an existing user interface design.

- explain the various types of challenges that could be encountered in designing a good user interface.
- describe the the various process of implementing designs;
- describe how to evaluate a given design

Working Through This Course

In order to have a thorough understanding of the course units, you will need to read and understand the contents, practise the steps by designing a standard user interface model.

This course is designed to cover approximately sixteen weeks, and it will require your devoted attention. You should do the exercises in the Tutor-Marked Assignments and submit to your tutors.

Course Materials

These include:
1. Course Guide
2. Study Units
3. Recommended Texts
4. A file for your assignments and for records to monitor your progress.

Study Units

There are twenty one study units in this course:

Module1

UNIT 1 – FOUNDATION OF USER INTERFACE DESIGN

UNIT 2 – DESIGNING GOOD USER INTERFACES

UNIT 3 – GRAPHICAL USER INTERFACE (GUI)

UNIT 4 – HUMAN-COMPUTER INTERACTION

UNIT 5 – ERGONOMICS

Module 2

UNIT 1 – HUMAN CAPABILITIES IN USER INTERFACE DESIGN

UNIT 2 – UNDERSTANDING USERS AND TASK ANALYSIS

UNIT 3 - USER-CENTERED DESIGN

UNIT 4 – INTERACTIVE DESIGN

UNIT 5 – USABILITY

UNIT 6 – INTERACTION STYLES AND GRAPHIC DESIGN PRINCIPLES

Module 3

UNIT 1	-	PROTOTYPING
UNIT 2	-	PROTOTYPING/IMPLEMENTATION METHODS AND TOOLKITS
UNIT 3	-	INPUT AND OUTPUT MODELS
UNIT 4	-	MODEL VIEW-CONTROLLER (MVC)
UNIT 5	-	LAYOUTS AND CONSTRAINTS

Module 4

UNIT 1	-	TECHNIQUES FOR EVALUATING AND MEASURING INTERFACE USABILITY
UNIT 2	-	EVALUATING USER INTERFACE WITHOUT THE USER
UNIT 3	-	EVALUATING USER INTERFACE WITH THE USER
UNIT 4	-	OTHER EVALUATION METHODOLOGIES
UNIT 5	-	USABILITY TESTING

Make use of the course materials, do the exercises to enhance your learning.

Textbooks and References

- www.wikipedia.org
- Holm, Ivar (2006). *Ideas and Beliefs in Architecture and Industrial design: How attitudes, orientations, and underlying assumptions shape the built environment*. Oslo School of Architecture and Design. ISBN8254701741.
- Wickens, C.D et al. (2004). *An Introduction to Human Factors Engineering* (2nd Ed), Pearson Education, Inc., Upper Saddle River, NJ : Prentice Hall.
- Dumas, J.S. and Redish, J.C. (1999). *A Practical Guide to Usability Testing* (revised ed.), Bristol, U.K.: Intellect Books.
- 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.
- Nielsen, J. and Molich, R. "Heuristic evaluation of user interfaces." Proc. CHI'90 Conference on Human Factors in Computer Systems. New York: ACM, 1990, pp. 249-256.
- Nielsen, J. "Finding usability problems through heuristic evaluation." Proc. CHI'92 Conference on Human Factors in Computer Systems. New York: ACM, 1992, pp. 373-380.
- Molich, R. and Nielsen, J. "Improving a human-computer dialogue: What designers know about traditional interface design." Communications of the ACM, 33 (March 1990), pp. 338-342.
- Nielsen, J. "Usability Engineering." San Diego, CA: Academic Press, 1992.
- Wharton, C., Rieman, J., Lewis, C., and Polson, P. "The cognitive walkthrough: A practitioner's guide." InJ.NielsenandR.L.Mack(Eds.),UsabilityInspection Methods, NewYork:JohnWiley&Sons(1994).
- Nielsen, J. and Molich, R. "Heuristic evaluation of user interfaces." Proc. CHI'90 Conference on Human Factors in Computer Systems. New York: ACM, 1990, pp. 249-256.
- Nielsen, J. "Finding usability problems through heuristic evaluation." Proc. CHI'92 Conference on Human Factors in Computer Systems. New York: ACM, 1992, pp. 373-380.
- Molich, R. and Nielsen, J. "Improving a human-computer dialogue: What designers know about traditional interface design." Communications of the ACM, 33 (March 1990), pp. 338-342.
- Nielsen, J. "Usability Engineering." San Diego, CA: Academic Press, 1992.

Assignments File

Tutor-Marked Assignment is a supervised assignment. The assignments take a certain percentage of your total score in this course. The Tutor-Marked Assignments will be assessed by your tutor within a specified period. The examination at the end of this course will aim at determining the level of mastery of the subject matter. This course includes sisteen Tutor-Marked Assignments and each must be done and submitted accordingly. Your best scores however, will be recorded for you. Be sure to send these assignments to your tutor before the deadline to avoid loss of marks.

Assessment

There are two aspects to the assessment of the course. First are the tutor marked assignments; second, is a written examination.

In tackling the assignments, you are expected to apply information and knowledge acquired during this course. The assignments must be submitted to your tutor for formal assessment in accordance with the deadlines stated in the Assignment File. The work you submit to your tutor for assessment will count for 30% of your total course mark.

At the end of the course, you will need to sit for a final three-hour examination. This will also count for 70% of your total course mark.

Tutor Marked Assignments (TMAS)

There are sisteen tutor marked assignments in this course. You need to submit all the assignments. The total marks for the best four (6) assignments will be 30% of your total course mark.

Assignment questions for the units in this course are contained in the Assignment File. You should be able to complete your assignments from the information and materials contained in your set textbooks, reading and study units. However, you may wish to use other references to broaden your viewpoint and provide a deeper understanding of the subject.

When you have completed each assignment send it to your tutor. Make sure that each assignment reaches your tutor on or before the deadline given. If, however,

you cannot complete your work on time, contact your tutor before the assignment is done to discuss the possibility of an extension.

Examination and Grading

The final examination for the course will carry 70% percentage of the total marks available for this course. The examination will cover every aspect of the course, so you are advised to revise all your corrected assignments before the examination.

This course endows you with the status of a teacher and that of a learner. This means that you teach yourself and that you learn, as your learning capabilities would allow. It also means that you are in a better position to determine and to ascertain the what, the how, and the when.
The course units are similarly designed with the introduction following the table of contents, then a set of objectives and then the dialogue and so on.

The objectives guide you as you go through the units to ascertain your knowledge of the required terms and expressions.

Course Marking Scheme

This table shows how the actual course marking is broken down.

Assessment	**Marks**
Assignment 1- 16	Sixteen assignments, best six count for 30% of course marks
Final Examination	70% of overall course marks
Total	100% of course marks

Table 1: Course Marking Scheme

Course Overview

Unit	**Title of Work**	**Weeks Activity**	**Assessment (End of Unit)**
	Course Guide		
	Module 1		
1	FOUNDATION OF USER INTERFACE	Week 1	Assignment 1

	DESIGN and DESIGNING GOOD USER INTERFACES		
2	GRAPHICAL USER INTERFACE (GUI)	Week 2	Assignment 2
3	HUMAN-COMPUTER INTERACTION	Week 3	Assignment 3
4	ERGONOMICS	Week 4	Assignment 4
	Module 2		
1	HUMAN CAPABILITIES IN USER INTERFACE DESIGN	Week 5	Assignment 5
2	UNDERSTANDING USERS AND TASK ANALYSIS	Week 6	Assignment 6
3	USER-CENTERED DESIGN and INTERACTIVE DESIGN	Week 7	Assignment 7
4	USABILITY and INTERACTION STYLES AND GRAPHIC DESIGN PRINCIPLES	Week 8	Assignment 8
	Module 3		
1	PROTOTYPING and PROTOTYPING AND IMPLEMENTATION METHODS AND TOOLKITS	Week 9	Assignment 9
2	INPUT AND OUTPUT MODELS	Week 10	Assignment 10
3	MODEL VIEW-CONTROLLER (MVC)	Week 11	Assignment 11
4	LAYOUTS AND CONSTRAINTS	Week 12	Assignment12
	Module 4		
1	TECHNIQUES FOR EVALUATING AND MEASURING INTERFACE USABILITY	Week 13	Assignment 13
2	EVALUATING USER INTERFACE WITHOUT THE USER and EVALUATING USER INTERFACE WITH THE USER	Week 14	Assignment 14
3	OTHER EVALUATION METHODOLOGIES	Week 15	Assignment 15
4	USABILITY TESTING	Week 16	Assignment 16
	Revision	Week 17	
	Examination	Week 18	
Total		18 weeks	

How to get the best from this course

In distance learning the study units replace the university lecturer. This is one of the great advantages of distance learning; you can read and work through specially designed study materials at your own pace, and at a time and place that suit you best. Think of it as reading the lecture instead of listening to a lecturer. In the same way that a lecturer might set you some reading to do, the study units tell you when to read your set books or other material. Just as a lecturer might give you an in-class exercise, your study units provide exercises for you to do at appropriate points.

Each of the study units follows a common format. The first item is an introduction to the subject matter of the unit and how a particular unit is integrated with the other units and the course as a whole. Next is a set of learning objectives. These objectives enable you know what you should be able to do by the time you have completed the unit. You should use these objectives to guide your study. When you have finished the units you must go back and check whether you have achieved the objectives. If you make a habit of doing this you will significantly improve your chances of passing the course.

Remember that your tutor's job is to assist you. When you need help, don't hesitate to call and ask your tutor to provide it.

- Read this *Course Guide* thoroughly.

- Organize a study schedule. Refer to the 'Course Overview' for more details. Note the time you are expected to spend on each unit and how the assignments relate to the units. Whatever method you chose to use, you should decide on it and write in your own dates for working on each unit.

- Once you have created your own study schedule, do everything you can to stick to it. The major reason that students fail is that they lag behind in their course work.

- Turn to *Unit 1* and read the introduction and the objectives for the unit.

- Assemble the study materials. Information about what you need for a unit is given in the 'Overview' at the beginning of each unit. You will almost always

need both the study unit you are working on and one of your set of books on your desk at the same time.

- Work through the unit. The content of the unit itself has been arranged to provide a sequence for you to follow. As you work through the unit you will be instructed to read sections from your set books or other articles. Use the unit to guide your reading.

- Review the objectives for each study unit to confirm that you have achieved them. If you feel unsure about any of the objectives, review the study material or consult your tutor.

- When you are confident that you have achieved a unit's objectives, you can then start on the next unit. Proceed unit by unit through the course and try to pace your study so that you keep yourself on schedule.

- When you have submitted an assignment to your tutor for marking, do not wait for its return before starting on the next unit. Keep to your schedule. When the assignment is returned, pay particular attention to your tutor's comments on the tutor-marked assignment form. Consult your tutor as soon as possible if you have any questions or problems.

- After completing the last unit, review the course and prepare yourself for the final examination. Check that you have achieved the unit objectives (listed at the beginning of each unit) and the course objectives (listed in this *Course Guide*).

Tutors and Tutorials

There are 15 hours of tutorials provided in support of this course. You will be notified of the dates, times and location of these tutorials, together with the name and phone number of your tutor, as soon as you are allocated a tutorial group.

Your tutor will mark and comment on your assignments, keep a close watch on your progress and on any difficulties you might encounter and provide assistance to you during the course. You must mail or submit your tutor-marked assignments to your tutor well before the due date (at least two working days are required). They will be marked by your tutor and returned to you as soon as possible.

Do not hesitate to contact your tutor by telephone, or e-mail if you need help. The following might be circumstances in which you would find help necessary. Contact your tutor if:

- you do not understand any part of the study units or the assigned readings,
- you have a question or problem with an assignment, with your tutor's comments on an assignment or with the grading of an assignment.

You should try your best to attend the tutorials. This is the only chance to have face to face contact with your tutor and to ask questions which are answered instantly. You can raise any problem encountered in the course of your study. To gain the maximum benefit from course tutorials, prepare a question list before attending them. You will learn a lot from participating in discussions actively.

Summary

User Interface Design and Ergonomics deals with analysis, design, implementation and evaluation of user interface design. The aim of this course is to equip you with the basic skills of studying and understanding the concept behind user interface design. By the end of the course, you will be able to confidently study, analyse and design a standard User Interface. The content of the course material was planned and written to ensure that you acquire the proper knowledge and skills for the appropriate situations. Real-life situations have been created to enable you identify with and create some of your own. The essence is to get you to acquire the necessary knowledge and competence, and by equipping you with the necessary tools, we hope to have achieved that.

I wish you success with the course and hope that you will find it both interesting and useful.

CIT 811: USER INTERFACE DESIGN AND ERGONOMICS (3 UNITS)

TABLE OF CONTENT PAGES

UNIT 1 – PROTOTYPING

UNIT 2 - PROTOTYPING AND IMPLEMENTATION METHODS AND TOOLKITS

UNIT 3 - INPUT AND OUTPUT MODELS

UNIT 4 - MODEL VIEW-CONTROLLER (MVC)

UNIT 5 - LAYOUTS AND CONSTRAINTS

MODEL 4 – USER INTERFACE EVALUATION

UNIT 1 - TECHNIQUES FOR EVALUATING AND MEASURING INTERFACE USABILITY

UNIT 2 - EVALUATING USER INTERFACE WITHOUT THE USER

UNIT 3 - EVALUATING USER INTERFACE WITH THE USER

UNIT 4 - OTHER EVALUATION METHODOLOGIES

UNIT 5 - USABILITY TESTING PROCEDURE

UNIT 1 FUNDAMENTALS OF USER INTERFACE DESIGN

TableofContents

1.0 INTRODUCTION

Having read through the course guide, you will have a general understanding of what this unit is about and how it fits into the course as a whole. This unit describes the general fundamentals of user interface design.

2.0 OBJECTIVES

By the end this unit, you should be able to:

- Explain the term user interface design
- Highlight the significance of user interface
- Explain the history behind user interfaces
- Describe the modalities and modes of user interface

3.0 MAIN CONTENT

3.1 USER INTERFACE

The **user interface** (also known as Human Machine Interface (HMI) or Man-Machine Interface (MMI) is the aggregate of means by which people—the *users*—interact with the *system*—a particular machine, device, computer program or other complex tool.

User Interface is the point at which a user or a user department or organization interacts with a computer system. The part of an interactive computer program that sends messages to and receives instructions from a terminal user.

User Interface is the point at which a user or a user department or organization interacts with a computer system. The part of an interactive computer program that sends messages to and receives instructions from a terminal user.

In computer science and human-computer interaction, the *user interface (of a computer program)* refers to the graphical, textual and auditory information the program presents to the user, and the control sequences (such as keystrokes with the computer keyboard, movements of the computer mouse, and selections with the touchscreen) the user employs to control the program.

User interface design or **user interface engineering** is the design of computers, appliances, machines, mobile communication devices, software applications, and websites with the focus on the user's experience and interaction. The goal of user interface design is to make the user's interaction as simple and efficient as possible, in terms of accomplishing user goals—what is often called user-centered design. Good user interface design facilitates finishing the task at hand without drawing unnecessary attention to itself. Graphic design may be utilized to apply a theme or style to the interface without compromising its usability. The design process must balance technical functionality and visual elements (e.g., mental model) to create a system that is not only operational but also usable and adaptable to changing user needs.

Interface design is involved in a wide range of projects from computer systems, to cars, to commercial planes; all of these projects involve much of the same basic human interaction yet also require some unique skills and knowledge. As a result, designers tend to specialize in certain types of projects and have skills centered around their expertise, whether that be software design, user research, web design, or industrial design

3.2 SIGNIFICANCE OF USER INTERFACE

To work with a system, users have to be able to control the system and assess the state of the system. For example, when driving an automobile, the driver uses the steering wheel to control the direction of the vehicle, and the accelerator pedal, brake pedal and gearstick to control the speed of the vehicle. The driver perceives the position of the vehicle by looking through the windscreen and exact speed of the vehicle by reading the speedometer. The *user interface of the automobile* is on the whole composed of the instruments the driver can use to accomplish the tasks of driving and maintaining the automobile.

The term *user interface* is often used in the context of computer systems and electronic devices. The user interface of a mechanical system, a vehicle or an industrial installation is sometimes referred to as the **Human-Machine Interface** (**HMI**). HMI is a modification of the original term MMI (man-machine interface). In practice, the abbreviation MMI is still frequently used although some may claim that MMI stands for something different now. Another abbreviation is HCI, but is more commonly used for human-computer *interaction* than human-computer *interface*. Other terms used are Operator Interface Console (OIC) and Operator Interface Terminal (OIT).

3.3 TYPES OF USER INTERFACES

Currently (as of 2009) the following types of user interface are the most common:

i. **Graphical user interfaces** (GUI) accept input via devices such as computer keyboard and mouse and provide articulated graphical output on the computer monitor. There are at least two different principles widely used in GUI design: Object-oriented user interfaces (OOUIs) and application oriented interfaces.

ii. **Web-based user interfaces** or **web user interfaces** (WUI) accept input and provide output by generating web pages which are transmitted via the Internet and viewed by the user using a web browser program. Newer implementations utilize Java, AJAX, Adobe Flex, Microsoft .NET, or similar technologies to provide real-time control in a separate program, eliminating the need to refresh a traditional HTML based web browser. Administrative web interfaces for web-servers, servers and networked computers are often called Control panels.

User interfaces that are common in various fields outside desktop computing:

iii**.** **Command line interfaces**, where the user provides the input by typing a command string with the computer keyboard and the system provides output by printing text on the computer monitor. Used for system administration tasks etc.

iv. **Tactile interfaces** supplement or replace other forms of output with haptic feedback methods. This is also used in computerized simulators etc.

v. **Touch user interface** are graphical user interfaces using a touchscreen display as a combined input and output device. Used in many types of point of sale, industrial processes and machines, self-service machines etc.

Other types of user interfaces:

vi. **Attentive user interfaces** manage the user attention deciding when to interrupt the user, the kind of warnings, and the level of detail of the messages presented to the user.

vii. **Batch interfaces** are non-interactive user interfaces, where the user specifies all the details of the *batch job* in advance to batch processing, and receives the output when all the processing is done. The computer does not prompt for further input after the processing has started.

viii. **Conversational Interface Agents** attempt to personify the computer interface in the form of an animated person, robot, or other character (such as Microsoft's Clippy the paperclip), and present interactions in a conversational form.

ix. **Crossing-based interfaces** are graphical user interfaces in which the primary task consists in crossing boundaries instead of pointing.

x. **Gesture interface** are graphical user interfaces which accept input in a form of hand gestures, or mouse gestures sketched with a computer mouse or a stylus.

xi. **Intelligent user interfaces** are human-machine interfaces that aim to improve the efficiency, effectiveness, and naturalness of human-machine interaction by representing, reasoning, and acting on models of the user, domain, task, discourse, and media (e.g., graphics, natural language, gesture).

xii. **Motion tracking interfaces** monitor the user's body motions and translate them into commands, currently being developed by Apple.

xiii. **Multi-screen interfaces**, employ multiple displays to provide a more flexible interaction. This is often employed in computer game interaction in both the commercial arcades and more recently the handheld markets.

xiv. **Non-command user interfaces**, which observe the user to infer his/her needs and intentions, without requiring that he/she formulate explicit commands.

xv. **Object-oriented user interface (OOUI)** :- The following are examples of OOUI:-

- **Reflexive user interfaces** where the users control and redefine the entire system via the user interface alone, for instance to change its command verbs. Typically this is only possible with very rich graphic user interfaces.
- **Tangible user interfaces**, which place a greater emphasis on touch and physical environment or its element.
- **Text user interfaces** are user interfaces which output text, but accept other form of input in addition to or in place of typed command strings.
- **Voice user interfaces**, which accept input and provide output by generating voice prompts. The user input is made by pressing keys or buttons, or responding verbally to the interface.
- **Natural-Language interfaces** - Used for search engines and on webpages. User types in a question and waits for a response.
- **Zero-Input interfaces** get inputs from a set of sensors instead of querying the user with input dialogs.
- **Zooming user interfaces** are graphical user interfaces in which information objects are represented at different levels of scale and detail, and where the user can change the scale of the viewed area in order to show more detail

3.4 HISTORY OF USER INTERFACES

The history of user interfaces can be divided into the following phases according to the dominant type of user interface:
Batch interface, 1945-1968
Command-line user interface, 1969 to present
Graphical user interface, 1981 to present

3.5 USER INTERFACE MODALITIES AND MODES

A **modality** is a path of communication employed by the user interface to carry input and output. Examples of modalities:

Input — allowing the users to manipulate a system. For example the computer keyboard allows the user to enter typed text, digitizing tablet allows the user to create free-form drawing

Output — allowing the system to indicate the effects of the users' manipulation. For example the computer monitor allows the system to display text and graphics (*vision modality*), loudspeaker allows the system to produce sound (*auditory modality*)

The user interface may employ several redundant input modalities and output modalities, allowing the user to choose which ones to use for interaction.

A **mode** is a distinct method of operation within a computer program, in which the same input can produce different perceived results depending of the state of the computer program. Heavy use of modes often reduces the usability of a user interface, as the user must expend effort to remember current mode states, and switch between mode states as necessary.

4.0 CONCLUSION

In this unit, you have been introduced to the fundamental concepts user interface. You have also learnt the history and significance of user interface design.

5.0 SUMMARY

You must have learnt the following in this unit:-

- The introduction of user interface which is the aggregate of means by which users interact with a particular machine, device, computer program or any other complex tool.

- The study of the various types of user interface design which includes graphical user interfaces, web-based user interfaces, command line interfaces e.t.c

- The history of user interfaces which can be divided into batch interface, command-line user interface and graphical user interface.

- The modality of a user interface which is a path of communication employed by the user interface to carry input and output.

6.0 TUTOR MARKED ASSIGNMENT

a. Explain the Microsoft's Clippy the paperclip.

b. Write a short note on the Command-line user interface.

7.0 FURTHER READING AND OTHER RESOURCES

www.en.wikipedia.org

MODULE 1

UNIT 2 DESIGNING GOOD USER INTERFACES

TableofContents

1.0 INTRODUCTION

Having read through the course guide, you will have a general understanding of what this unit is about and how it fits into the course as a whole. This unit describes the essentials of designing good interface designs.

2.0 OBJECTIVES

By the end this unit, you should be able to:

- Explain the essentials of good interface design
- Identify the necessary tips needed for designing a good interface
- Have good understanding of various users

3.0 MAIN CONTENT

3.1 ESSENTIALS OF INTERFACE DESIGN

There are three pillars to an application's success:

- Features
- Function
- Face

Features refers to what the application will do for the user. Features are the requirements for the software.
Function refers to how well the software operates. Bug-free software will function perfectly.
Face refers to how the application presents itself to the user - the application's "user interface."

Features, function and face can be restated as questions:
Does the software meet the user's requirements? (Features)
Does the software operate as intended? (Function)

Is the software easy to use? (Face)

The goal of user interface design is to put a happy face on the application. Stated in more concrete terms, a successful user interface will require *Zero Training* and will be *friendly not foreboding*.

Zero Training

The goal of Zero Training could be considered as a sub-goal of friendly not foreboding. However, training costs are a major impediment to the usage of software making Zero Training an important goal by itself.

There are two types of training involved in software design: software training and job training. Software training assumes the user knows how to do the job at hand and only needs to learn how to use the software to do the job. Job training teaches the user how to do the job - which can include more than how to use the application to do the job. The goal of Zero Training relates to *zero software training*. Job training can be integrated with software training, but results in a much more ambitious project.

Friendly not Foreboding

Almost everything you do to implement the goal of Zero Training will further the goal of being friendly not foreboding. However, some techniques for reducing training may slow up experienced users. For example, you could pop-up new user messages whenever the user lands in a particular field. Seeing the same message will get old after awhile; the experienced user should be able to dispense with the messages.

Being friendly is an attitude and encompasses more than what is necessary for the Zero Training goal. Applications do have an attitude. For example, consider the following sets of application messages:

"Database does not exist" "I could not find database "CorpInfo". If you are sure this name is correct, CorpInfo could be unavailable due to maintenance or LAN problems. You should contact the help desk to see when CorpInfo will again be available."

"SQL access error 123" "I could not save information to the database. You can try to save again to see if the error clears. If you call the help desk concerning this problem, tell them you have a "SQL access error 123".

"out of hunk"[1] "I have run out of memory (RAM). This typically happens do to a bug in the program which causes it to lose memory over time. Save your game if possible. To free the memory you will need to reset the computer (turn it off and then on)."

The attitude of the first message is "you did something that caused me a problem" while the attitude of the second message is "I have a problem. Could you give me a hand?"

3.1.1 DESIGNING A GOOD USER INTERFACE

Designing a good user interface is an iterative process. First, you design and implement a user interface using appropriate techniques. Then you evaluate the design. The results of the evaluation feed the next design and implementation. You stop the process when you have met your design goals or you run out of time and/or money.

Note that if you have different user communities (or the same user with different jobs), you may need different user interfaces, customizable user interfaces or both. For example, Microsoft Word provides four user interfaces: normal, outline, page layout and master. In addition, Microsoft Word provides a host of customization features for the keyboard, menu and toolbars.

While design is important, the real key to creating a good user interface is in your evaluation techniques. Obviously, you should use your own user interface. If you can't use it, how can anyone else? Next, get feed back from your alpha testers, then from you beta testers.

The best evaluations are done by watching over the shoulder of the user. The key here is watching. If you are telling the user what to do, you will never find out if your interface is easy to use. Let the user figure it out by himself. If the user has to ask you what to do or pauses to figure out what to do next, you may need to work on your interface. If the user grimaces, find out why. Learn from the experience. Some of my most innovative designs were shot down when the users could not figure them out.

You will need both new and experienced users for testing your interface. The new users will help you determine if you meet the Zero Training goal. The experienced users will let you know if your methods for meeting the Zero Training goal interfere with getting work done once the user has learned the software.

3.1.2 TIPS FOR DESIGNINIG GOOD USER INTERFACE

1. **Consistency**. The most important thing that you can possibly do is make sure that your user interface works consistently. If you can double-click on items in one list and have something happen then you should be able to double-click on items in any other list and have the same sort of thing happen. Put your buttons in consistent places on all of your windows, use the same wording in labels and messages, and use a consistent colour scheme throughout. Consistency in your user interface allows your users to build an accurate mental model of the way that it works, and accurate mental models lead to lower training and support costs.

2. **Set standards and stick to them**. The only way that you'll be able to ensure consistency within your application is to set design standards and then stick to them. The best approach is to adopt an industry standard and then fill any missing guidelines that are specific to your needs. Industry standards, such as the ones set by IBM (1993) and Microsoft (1995), will often define 95%-99% of what you need. By adopting industry

standards you not only take advantage of the work of others you also increase the chance that your application will look and feel like other applications that your users purchase or have built. User interface design standards should be set during the Define Infrastructure Stage.

3. **Explain the rules**. Your users need to know how to work with the application that you built for them. When an application works consistently it means you only have to explain the rules once. This is a lot easier than explaining in detail exactly how to use each and every feature in an application step by step.

4. **Support both novices and experts**. Although a library-catalog metaphor might be appropriate for casual users of a library system, library patrons, it probably is not all that effective for expert users, librarians. Librarians are highly trained people who are able to use complex search systems to find information in a library, therefore you should consider building a set of search screens to support their unique needs.

5. **Navigation between screens is important**. If it is difficult to get from one screen to another then your users will quickly become frustrated and give up. When the flow between screens matches the flow of the work that the user is trying to accomplish, then your application will make sense to your users. Because different users work in different ways, your system will need to be flexible enough to support their various approaches. Interface-flow diagrams can be used during the Model Stage to model the flow between screens.

6. **Navigation within a screen is important**. In Western societies people read left to right and top to bottom. Because people are used to this should you design screens that are also organized left to right and top to bottom. You want to organize navigation between widgets on your screen in a manner that users will find familiar to them.

7. **Word your messages and labels appropriately**. The text that you display on your screens is a primary source of information for your users. If your text is worded poorly then your interface will be perceived poorly by your users. Using full words and sentences, as opposed to abbreviations and codes makes your text easier to understand. Your messages should be worded positively, imply that the user is in control, and provide insight into how to use the application properly. For example, which message do you find more appealing "You have input the wrong information" or "An account number should be 8 digits in length."? Furthermore, your messages should be worded consistently and displayed in a consistent place on the screen. Although the messages "The person's first name must be input." and "An account number should be input." are separately worded well, together they are inconsistent. In light of the first message, a better wording of the second message would be "The account number must be input" to make the two messages consistent.

8. **Understand your widgets**. You should use the right widget for the right task, helping to increase the consistency in your application and probably making it easier to build the application in the first place. The only way that you can learn how to use

widgets properly is to read and understand the user-interface standards and guidelines that your organization has adopted.

9. **Look at other applications with a grain of salt**. Unless you know that another application follows the user-interface standards and guidelines of your organization, you must not assume that the application is doing things right. Although it is always a good idea to look at the work of others to get ideas, until you know how to distinguish between good user-interface design and bad user-interface design you have to be careful. Too many developers make the mistake of imitating the user interface of another application that was poorly designed.

10. **Use colour appropriately**. Colour should be used sparingly in your applications, and if you do use it you must also use a secondary indicator. The problem is that some of your users may be color blind – if you are using color to highlight something on a screen then you need to do something else to make it stand out if you want these people to notice it, such as display a symbol beside it. You also want to use colors in your application consistently so that you have a common look and feel throughout your application. Also, colour generally does not port well between platform – what looks good on one system often looks poor on another system. We have all been to presentations where the presenter said "it looks good on my machine at home."

11. **Follow the contrast rule**. If you are going to use colour in your application you need to ensure that your screens are still readable. The best way to do this is to follow the contrast rule: Use dark text on light backgrounds and light text on dark backgrounds. It is very easy to read blue text on a white background but very difficult to read blue text on a red background. The problem is that there is not enough contrast between blue and red to make it easy to read, whereas there is a lot of contrast between blue and white.

12. **Use fonts appropriately** – Old English fonts might look good on the covers of William Shakespeare's plays, but they are really hard to read on a screen. Use fonts that are easy to read, such as serif fonts, Times Roman. Furthermore, use your fonts consistently and sparingly. A screen using two or three fonts effectively looks a lot better than a screen that uses five or six. Never forget that you are using a different font every time you change the size, style (bold, italics, underlining, ...), typeface, or colour.

13. **Gray things out, do not remove them**. You often find that at certain times it is not applicable to give your users access to all the functionality of an application. You need to select an object before you can delete it, so to reinforce your mental model the application should do something with the Delete button and/or menu item. Should the button be removed or grayed out? Gray it out, never remove it. By graying things out when they shouldn't be used people can start building an accurate mental model as to how your application works. If you simply remove a widget or menu item instead of graying it out then it is much more difficult for your users to build an accurate mental model because they only know what is currently available to them, and not what is not available. The old adage that out of sight is out of mind is directly applicable here.

14. **Use non destructive default buttons**. It is quite common to define a default button on every screen, the button that gets invoked if the user presses the Return/Enter key. The problem is that sometimes people will accidentally hit the Enter/Return key when they do not mean to, consequently invoking the default button. Your default button shouldn't be something that is potentially destructive, such as delete or save (perhaps your user really did not want to save the object at that moment).

15. **Alignment of fields** . When a screen has more than one editing field you want to organize the fields in a way that is both visually appealing and efficient. As shown in Figure 1 I have always found that the best way to do so is to left-justify edit fields, or in other words make the left-hand side of each edit field line up in a straight line, one over the other. The corresponding labels should be right justified and placed immediately beside the field. This is a clean and efficient way to organize the fields on a screen.

16. **Justify data appropriately**. For columns of data it is common practice to right justify integers, decimal align floating point numbers, and left justify strings.

17. **Do not create busy screens**. Crowded screens are difficult to understand and hence are difficult to use. Experimental results show that the overall density of the screen should not exceed 40%, whereas local density within groupings shouldn't exceed 62%.

18. **Group things on the screen effectively**. Items that are logically connected should be grouped together on the screen to communicate that they are connected, whereas items that have nothing to do with each other should be separated. You can use whitespace between collections of items to group them and/or you can put boxes around them to accomplish the same thing.

19. **Open windows in the center of the action**. When your user double-clicks on an object to display its edit/detail screen then his or her attention is on that spot. Therefore it makes sense to open the window in that spot, not somewhere else.

20. **Pop-up menus should not be the only source of functionality**. Your users cannot learn how to use your application if you hide major functionality from them. One of the most frustrating practices of developers is to misuse pop-up, also called context-sensitive, menus. Typically there is a way to use the mouse on your computer to display a hidden pop-up menu that provides access to functionality that is specific to the area of the screen that you are currently working in.

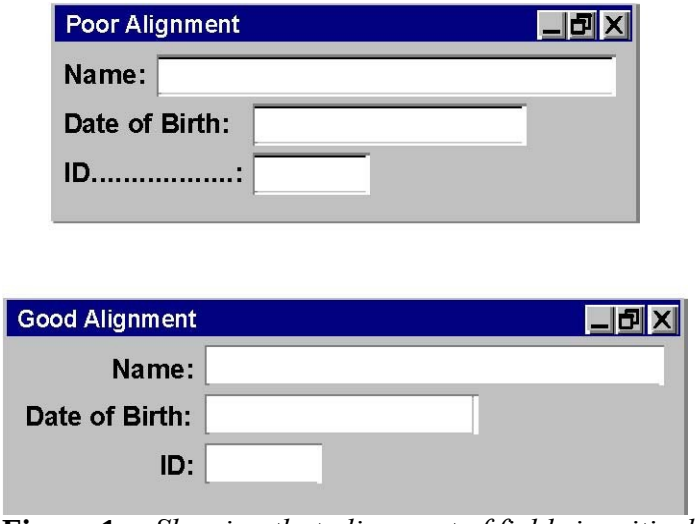

Figure 1:- *Showing that alignment of fields is critical*

3.2 UNDERSTANDING USERS

You must understand the user to be able to put a happy face on your application. You should understand the user's job, how the software fits in with that job and how the user goes about getting the job done. You need to approach the design of software from the user's viewpoint not from an abstract requirements document. Specifically, you should understand what the user will be doing with the application. If you can think like a user, you can create a much better user interface.
Here are some basic principles to remember about users:

a. Your software is like a hammer - the user doesn't really care how well crafted it is, the user just wants nails put in the wall. Users just want to do their job (or play their game). They don't care about you or your software. Your software is just an expedient tool to take the user where the user wants to go.

b. Given a selection of hammers to buy at the hardware store, the user will select the one which will be most fun to use. Of course, this varies by user - some will want the plastic handle, some the wood, some the green, etc. When evaluating your software, users are often swayed by looks, not function. Thus, steps taken to make the product look good (nice icons, pictures, good colour scheme, fields aligned, etc.) will often favourably enhance evaluations of your software.

c. It had better drive nails. The user will not really know if your software is adequate to the job until the user has used the software to do actual work. From an interface perspective, the software should not look like it can do more than it can.

d. Some users will try to use a hammer to drive a screw. If your software is good, some user somewhere will try to use the software for some purpose for which you never intended it to be used. Obviously, you can not design a user interface to deal with uses you can not foresee. There is no single rigid model of the right way to use the software, so build in flexibility.

e. Users will not read an instruction manual for a hammer. They won't read one for your software either, unless they really have too. Users find reading instruction manuals almost as pleasurable as dental work.

f. A user reading the instruction manual for a hammer is in serious trouble. When you create your help system (and printed manual), remember that the user will only resort to those materials if the user is in trouble. The user will want a problem solved as fast and as easily as possible.

g. Hammers don't complain. You should try to eliminate error messages and any error messages your program needs should have the right attitude.

4.0 CONCLUSION

In this unit, you have been introduced to the essentials of good interface design. You have also learnt the necessary tips needed for designing a good interface and the need for understanding various users.

5.0 SUMMARY

What you have learnt in this unit are:-

- Introduction to essentials of interface design with emphasis on the features, functions and the face of the software.
- Designing a good user interface which has been described as an iterative process. You design, implement, evaluate and redesign until all removable errors have been taken care of.
- The tips necessary for designing a good designing user interface which includes consistency, setting standards and sticking to them, supporting of both novices and experts, e.t.c.
- Understanding the user's job, how the software fits in with that job and how the user goes about getting the job done.

Exercises

1. Describe the essentials of interface design.
2. Write a short note on any three tips necessary for designing a good user interface.

6.0 TUTOR MARKED ASSIGNMENT

a. How do you ensure that the interface support both novices and expert?
b. Write a short note on the design of a user interface for a user with hearing disability.

7.0 FURTHER READING AND OTHER RESOURCES

www.wikipedia.org
http://www.linfo.org/gui.html

MODULE 1

UNIT 3 GRAPHICAL USER INTERFACE

TableofContents

1.0 INTRODUCTION

This unit describes the general concept of Graphical user interface (GUI) and also the history and elements of graphical user interface. The concept of three dimensional (3D) graphical user interfaces is also introduced.

2.0 OBJECTIVES

By the end this unit, you should be able to:

- Describe a graphical user interface
- Explain the history behind graphical user interface
- Highlight the elements of a graphical user interface
- Describe the three-dimensional user interfaces

3.0 MAIN CONTENT

3.1 INTRODUCTION TO GRAPHICAL USER INTERFACE

Graphical user interfaces, also known as GUIs, offer a consistent visual language to represent information stored in computers. This makes it easier for people with little computer skills to work with and use computer software. This article explains the most common elements of the visual language interfaces.

A **graphical user interface** is a type of user interface which allows people to interact with electronic devices such as computers; hand-held devices such as MP3 Players, Portable Media Players or Gaming devices; household appliances and office equipment with images rather than text commands. A *GUI* offers graphical icons, and visual indicators, as opposed to text-based interfaces, typed command labels or text navigation to fully represent the information and actions available to a user. The actions are usually performed through direct manipulation of the graphical elements.
The term *GUI* is historically restricted to the scope of two-dimensional display screens with display resolutions capable of describing generic information, in the tradition of the computer science research at Palo Alto Research Center (PARC). The term *GUI* earlier might have been applicable to other high-resolution types of interfaces that are non-generic, such as videogames, or not restricted to flat screens, like volumetric displays.

3.2 HISTORY OF GRAPHICAL USER INTERFACE

Precursor to GUI

The precursor to GUIs was invented by researchers at the Stanford Research Institute, led by Douglas Engelbart. They developed the use of text-based hyperlinks manipulated with a mouse for the On-Line System. The concept of hyperlinks was further refined and extended to graphics by researchers at Xerox PARC, who went beyond text-based hyperlinks and used a GUI as the primary interface for the Xerox Alto computer. Most modern general-purpose GUIs are derived from this system. As a result, some people call this class of interface a PARC User Interface (PUI) (note that PUI is also an acronym for perceptual user interface).
Ivan Sutherland developed a pointer-based system called the Sketchpad in 1963. It used a light-pen to guide the creation and manipulation of objects in engineering drawings.

PARC User Interface

The PARC User Interface consisted of graphical elements such as windows, menus, radio buttons, check boxes and icons. The PARC User Interface employs a pointing device in addition to a keyboard. These aspects can be emphasized by using the alternative acronym WIMP, which stands for **W**indows, **I**cons, **M**enus and **P**ointing device.

Evolution

Following PARC the first GUI-centric computer operating model was the Xerox 8010 Star Information System in 1981[4] followed by the Apple Lisa (which presented concept of menu bar as well as window controls), in 1982 and the Atari ST and Commodore Amiga in 1985.
The GUIs familiar to most people today are Microsoft Windows, Finder Interface (Mac OS X), and the X Window System interfaces. Apple, IBM and Microsoft used many of Xerox's ideas to develop products, and IBMs Common User Access specifications formed the basis of the user interface found in Microsoft Windows, IBM OS/2

Presentation Manager, and the Unix Motif toolkit and window manager. These ideas evolved to create the interface found in current versions of Microsoft Windows, as well as in Mac OS X and various desktop environments for Unix-like operating systems, such as Linux. Thus most current GUIs have largely common idioms.

Post-WIMP interfaces

Smaller mobile devices such as PDAs and smart phones typically use the WIMP elements with different unifying metaphors, due to constraints in space and available input devices. Applications for which WIMP is not well suited may use newer interaction techniques, collectively named as post-WIMP user interfaces.[5]
Some touch-screen-based operating systems such as Apple's iPhone OS currently use post-WIMP styles of interaction. The i Phone's use of more than one finger in contact with the screen allows actions such as pinching and rotating, which are not supported by a single pointer and mouse.
A class of GUIs sometimes referred to as post-WIMP include 3D compositing window manager such as Compiz, Desktop Window Manager, and LG3D. Some post-WIMP interfaces may be better suited for applications which model immersive 3D environments, such as Google Earth.

3.3 ELEMENTS OF GRAPHICAL USER INTERFACES

A GUI uses a combination of technologies and devices to provide a platform the user can interact with, for the tasks of gathering and producing information.
A series of elements conforming a visual language have evolved to represent information stored in computers. This makes it easier for people with little computer skills to work with and use computer software. The most common combination of such elements in GUIs is the WIMP paradigm, especially in personalcomputers.
User interfaces use visual conventions to represent the generic information shown. Some conventions are used to build the structure of the static elements on which the user can interact, and define the appearance of the interface.

The key elements of GUI are divided into two categories viz Structural and Interactive elements.

3.3.1 Structural elements

User interfaces use visual conventions to represent the generic information shown. Some conventions are used to build the structure of the static elements on which the user can interact, and define the appearance of the interface.

a. Window

A window is an area on the screen that displays information, with its contents being displayed independently from the rest of the screen. An example of a window is what appears on the screen when the "My Documents" icon is clicked in the Windows

Operating System. It is easy for a user to manipulate a window: it can be opened and closed by clicking on an icon or application, and it can be moved to any area by dragging it (that is, by clicking in a certain area of the window – usually the title bar along the tops – and keeping the pointing device's button pressed, then moving the pointing device). A window can be placed in front or behind another window, its size can be adjusted, and scrollbars can be used to navigate the sections within it. Multiple windows can also be open at one time, in which case each window can display a different application or file – this is very useful when working in a multitasking environment. The system memory is the only limitation to the amount of windows that can be open at once. There are also many types of specialized windows.

A **Container Window** a window that is opened while invoking the icon of a mass storage device, or directory or folder and which is presenting an ordered list of other icons that could be again some other directories, or data files or maybe even executable programs. All modern container windows could present their content on screen either acting as browser windows or text windows. Their behaviour can automatically change according to the choices of the single users and their preferred approach to the graphical user interface.

A **browser window** allows the user to move forward and backwards through a sequence of documents or web pages. Web browsers are an example of these types of windows.

Text terminal windows are designed for embedding interaction with text user interfaces within the overall graphical interface. MS-DOS and UNIX consoles are examples of these types of windows.

A **child window** opens automatically or as a result of a user activity in a parent window. Pop-up windows on the Internet can be child windows.

A **message window**, or **dialog box**, is a type of child window. These are usually small and basic windows that are opened by a program to display information to the user and/or get information from the user. They usually have a button that must be pushed before the program can be resumed.

b. Menus

Menus allow the user to execute commands by selecting from a list of choices. Options are selected with a mouse or other pointing device within a GUI. A keyboard may also be used. Menus are convenient because they show what commands are available within the software. This limits the amount of documentation the user reads to understand the software.

A **menu bar** is displayed horizontally across the top of the screen and/or along the tops of some or all windows. A pull-down menu is commonly associated with this menu type. When a user clicks on a menu option the pull-down menu will appear.

A **menu** has a visible title within the menu bar. Its contents are only revealed when the user selects it with a pointer. The user is then able to select the items within the pull-down menu. When the user clicks elsewhere the content of the menu will disappear.

A **context menu** is invisible until the user performs a specific mouse action, like pressing the right mouse button. When the software-specific mouse action occurs the menu will appear under the cursor.

Menu extras are individual items within or at the side of a menu.

c. Icons

An icon is a small picture that represents objects such as a file, program, web page, or command. They are a quick way to execute commands, open documents, and run programs. Icons are also very useful when searching for an object in a browser list, because in many operating systems all documents using the same extension will have the same icon.

d. Controls (or Widgets)

Interface element that a computer user interacts with, and is also known as a **control** or Widget.

Window

A paper-like rectangle that represents a "window" into a document, form, or design area.

Pointer (or mouse cursor)

The spot where the mouse "cursor" is currently referencing.

Text box

A box in which to enter text or numbers.

Button

An equivalent to a push-button as found on mechanical or electronic instruments.

Hyperlink

Text with some kind of indicator (usually underlining and/or color) that indicates that clicking it will take one to another screen or page.

Drop-down list

A list of items from which to select. The list normally only displays items when a special button or indicator is clicked.

Check box

A box which indicates an "on" or "off" state via a check-mark or an "×".

Radio button

A button, similar to a check-box, except that only one item in a group can be selected. Its name comes from the mechanical push-button group on a car radio receiver. Selecting a new item from the group's buttons also deselects the previously selected button.

Data grid

A spreadsheet-like grid that allows numbers or text to be entered in rows and columns.

d. Tabs

A tab is typically a rectangular small box which usually contains a text label or graphical icon associated with a view pane. When activated the view pane, or window, displays widgets associated with that tab; groups of tabs allow the user to switch quickly between different widgets. This is used in the web browsers Firefox, Internet Explorer, Konqueror, Opera, and Safari. With these browsers, you can have multiple web pages open at once in one window, and quickly navigate between them by clicking on the tabs associated with the pages. Tabs are usually placed in groups at the top of a window, but may also be grouped on the side or bottom of a window.

3.3.2 **Interaction elements**

Some common idioms for interaction have evolved in the visual language used in GUIs. Interaction elements are interface objects that represent the state of an ongoing operation or transformation, either as visual remainders of the user intent (such as the pointer), or as affordances showing places where the user may interact.

a. Cursor

A cursor is an indicator used to show the position on a computer monitor or other display device that will respond to input from a text input or pointing device.

b. Pointer

One of the most common components of a GUI on the personal computer is a pointer: a graphical image on a screen that indicates the location of a pointing device, and can be used to select and move objects or commands on the screen. A pointer commonly appears as an angled arrow, but it can vary within different programs or operating systems.

Example of this can be found within text-processing applications, which uses an I-beam pointer that is shaped like a capital I, or in web browsers which often indicate that the pointer is over a hyperlink by turning the pointer in the shape of a gloved hand with outstretched index finger.

The use of a pointer is employed when the input method, or pointing device, is a device that can move fluidly across a screen and select or highlight objects on the screen. Pointer trails can be used to enhance its visibility during movement. In GUIs where the input method relies on hard keys, such as the five-way key on many mobile phones, there is no pointer employed, and instead the GUI relies on a clear focus state.

c. Selection

A selection is a list of items on which user operations will take place. The user typically adds items to the list manually, although the computer may create a selection automatically.

d. Adjustment handle

A handle is an indicator of a starting point for a drag and drop operation. Usually the pointer shape changes when placed on the handle, showing an icon that represents the supported drag operation.

Exercise 1:- Identify and study these elements within Window operating system

3.4 Three-dimensional user interfaces

For typical computer displays, *three-dimensional* is a misnomer—their displays are two-dimensional. Three-dimensional images are projected on them in two dimensions. Since this technique has been in use for many years, the recent use of the term three-dimensional must be considered a declaration by equipment marketers that the speed of three dimension to two dimension projection is adequate to use in standard GUIs.

Motivation

Three-dimensional GUIs are quite common in science fiction literature and movies, such as in *Jurassic Park*, which features Silicon Graphics' three-dimensional file manager, "File system navigator", an actual file manager that never got much widespread use as the user interface for a Unix computer. In fiction, three-dimensional user interfaces are often immersible environments like William Gibson's Cyberspace or Neal Stephenson's Metaverse.

Three-dimensional graphics are currently mostly used in computer games, art and computer-aided design (CAD). There have been several attempts at making three-dimensional desktop environments like Sun's Project Looking Glass or SphereXP from Sphere Inc. A three-dimensional computing environment could possibly be used for

collaborative work. For example, scientists could study three-dimensional models of molecules in a virtual reality environment, or engineers could work on assembling a three-dimensional model of an airplane. This is a goal of the Croquet project and Project LookingGlass.

Technologies

The use of three-dimensional graphics has become increasingly common in mainstream operating systems, from creating attractive interfaces—eye candy— to functional purposes only possible using three dimensions. For example, user switching is represented by rotating a cube whose faces are each user's workspace, and window management is represented in the form or via a Rolodex-style flipping mechanism in Windows Vista (see Windows Flip 3D). In both cases, the operating system transforms windows on-the-fly while continuing to update the content of those windows.

Interfaces for the X Window System have also implemented advanced three-dimensional user interfaces through compositing window managers such as Beryl, Compiz and KWin using the AIGLX or XGL architectures, allowing for the usage of OpenGL to animate the user's interactions with the desktop.
Another branch in the three-dimensional desktop environment is the three-dimensional GUIs that take the desktop metaphor a step further, like the BumpTop, where a user can manipulate documents and windows as if they were "real world" documents, with realistic movement and physics.

The Zooming User Interface (ZUI) is a related technology that promises to deliver the representation benefits of 3D environments without their usability drawbacks of orientation problems and hidden objects. It is a logical advancement on the GUI, blending some three-dimensional movement with two-dimensional or "2.5D" vector objects.

4.0 CONCLUSION

In this unit, you have been introduced to graphical user interface (GUI). The history of graphical user interface was also discussed. The element of a graphical user interface was also explained. You were also introduced to three-dimensional user interfaces.

5.0 SUMMARY

What you have learnt in this unit concern:

- Introduction to graphical user interface which is a type of user interface which allows people to interact with electronic devices such as computers, hand-held devices, household appliances and office equipment with images rather than text commands.
- The history of graphical user interface, precursor to GUI, PARC user interface and the evolution of other graphical user interface

- The elements of graphical user interface which are divided into two categories that includes structural and interactive elements.

Exercises

1. What do you understand by Graphical User Interface?

2. Explain the structural and iterative elements of graphical user interface.

6.0 TUTOR MARKED ASSIGNMENT

Explain the PARC graphical user interface

7.0 FURTHER READING AND OTHER RESOURCES

www.wikipedia.com
http://www.linfo.org/gui.html.

MODULE 1

UNIT 4 HUMAN COMPUTER INTERACTION

TableofContents

1.0 INTRODUCTION

In this course guide, you will be introduced to Human Computer Interaction (HCI) and its differences with other related fields. The goals and future development of Human Computer Interaction and the general concept of Human Computer Interface will be introduced.

2.0 OBJECTIVES

By the end this unit, you should be able to:

- Explain the term Human Computer Interaction
- Identify the various goals of Human Computer Interaction
- Differentiate Human Computer Interaction from other related fields
- Describe the future development of HCI and explain the Human computer Interface.

3.0 MAIN CONTENT

3.1 INTRODUCTION TO HUMAN COMPUTER INTERACTION Human–

computer interaction (**HCI**) is the study of interaction between people (users) and computers. It is often regarded as the intersection of computer science, behavioral sciences, design and several other fields of study. Interaction between users and computers occurs at the user interface (or simply *interface*), which includes both software and hardware, for example, general-purpose computer peripherals and large-scale mechanical systems, such as aircraft and power plants. The Association for Computing

Machinery defines human-computer interaction as "a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them.

Because human-computer interaction studies a human and a machine in conjunction, it draws from supporting knowledge on both the machine and the human side. On the machine side, techniques in computer graphics, operating systems, programming languages, and development environments are relevant. On the human side, communication theory, graphic and industrial design disciplines, linguistics, social sciences, cognitive psychology, and human performance are relevant. Engineering and design methods are also relevant. Due to the multidisciplinary nature of HCI, people with different backgrounds contribute to its success. HCI is also sometimes referred to as **man–machine interaction (MMI)** or **computer–human interaction (CHI)**.

3.2 GOALS OF HCI

A basic goal of HCI is to improve the interactions between users and computers by making computers more usable and receptive to the user's needs. Specifically, HCI is concerned with:

- methodologies and processes for designing interfaces (i.e., given a task and a class of users, design the best possible interface within given constraints, optimizing for a desired property such as learning ability or efficiency of use)
- methods for implementing interfaces (e.g. software toolkits and libraries; efficient algorithms)
- techniques for evaluating and comparing interfaces
- developing new interfaces and interaction techniques
- developing descriptive and predictive models and theories of interaction

A long term goal of HCI is to design systems that minimize the barrier between the human's cognitive model of what they want to accomplish and the computer's understanding of the user's task.

Professional practitioners in HCI are usually designers concerned with the practical application of design methodologies to real-world problems. Their work often revolves around designing graphical user interfaces and web interfaces. Researchers in HCI are interested in developing new design methodologies, experimenting with new hardware devices, prototyping new software systems, exploring new paradigms for interaction, and developing models and theories of interaction.

3.3 DIFFERNCES WITH RELATED FIELDS

HCI focuses on user interface design mainly for users of computer system and effective interaction between computers and users (human). User Interface Design is concerned with the users of devices such as computers, appliances, machines, mobile communication devices, software applications, and websites. In HCI, efficient user interface is critical.

HCI differs from human factors in that there the focus is more on users working specifically with computers, rather than other kinds of machines or designed artifacts. There is also a focus in HCI on how to implement the computer software and hardware mechanisms to support human-computer interaction. Thus, human factors is a broader term; HCI could be described as the human factors of computers - although some experts try to differentiate these areas.

According to some experts, HCI also differs from ergonomics in that there is less of a focus on repetitive work-oriented tasks and procedures, and much less emphasis on physical stress and the physical form or industrial design of the user interface, such as keyboards and mice. However, this does not take a full account of ergonomics, the oldest areas of which were mentioned above, but which more recently has gained a much broader focus (equivalent to human factors). Cognitive ergonomics, for example, is a part of ergonomics, of which *software ergonomics* (an older term, essentially the same as HCI) is a part.

Three areas of study have substantial overlap with HCI even as the focus of inquiry shifts. In the study of Personal Information Management (PIM), human interactions with the computer are placed in a larger informational context - people may work with many forms of information, some computer-based, many not (e.g., whiteboards, notebooks, sticky notes, refrigerator magnets) in order to understand and effect desired changes in their world. In Computer Supported Cooperative Work (CSCW), emphasis is placed on the use of computing systems in support of the collaborative work of a group of people. The principles of Human Interaction Management (HIM) extend the scope of CSCW to an organizational level and can be implemented without use of computer systems.

3.4 FUTURE DEVELOPMENT OF HCI

The means by which humans interact with computers continues to evolve rapidly. Human–computer interaction is affected by the forces shaping the nature of future computing. These forces include:

Decreasing hardware costs leading to larger memories and faster systems
Miniaturization of hardware leading to portability
Reduction in power requirements leading to portability

New display technologies leading to the packaging of computational devices in new forms
Specialized hardware leading to new functions
Increased development of network communication and distributed computing
Increasingly widespread use of computers, especially by people who are outside of the computing profession
Increasing innovation in input techniques (i.e., voice, gesture, pen), combined with lowering cost, leading to rapid computerization by people previously left out of the "computer revolution."
Wider social concerns leading to improved access to computers by currently disadvantaged groups

The future for HCI is expected to include the following characteristics:

Ubiquitous communication. Computers will communicate through high speed local networks, nationally over wide-area networks, and portably via infrared, ultrasonic, cellular, and other technologies. Data and computational services will be portably accessible from many if not most locations to which a user travels.

High functionality systems. Systems will have large numbers of functions associated with them. There will be so many systems that most users, technical or non-technical, will not have time to learn them in the traditional way (e.g., through thick manuals).

Mass availability of computer graphics. Computer graphics capabilities such as image processing, graphics transformations, rendering, and interactive animation will become widespread as inexpensive chips become available for inclusion in general workstations.

Mixed media. Systems will handle images, voice, sounds, video, text, formatted data. These will be exchangeable over communication links among users. The separate worlds of consumer electronics (e.g., stereo sets, VCRs, televisions) and computers will partially merge. Computer and print worlds will continue to cross assimilate each other.

High-bandwidth interaction. The rate at which humans and machines interact will increase substantially due to the changes in speed, computer graphics, new media, and new input/output devices. This will lead to some qualitatively different interfaces, such as virtual reality or computational video.

Large and thin displays. New display technologies will finally mature enabling very large displays and also displays that are thin, light weight, and have low power consumption. This will have large effects on portability and will enable the development of paper-like, pen-based computer interaction systems very different in feel from desktop workstations of the present.

Embedded computation. Computation will pass beyond desktop computers into every object for which uses can be found. The environment will be alive with little computations from computerized cooking appliances to lighting and plumbing fixtures to

window blinds to automobile braking systems to greeting cards. To some extent, this development is already taking place. The difference in the future is the addition of networked communications that will allow many of these embedded computations to coordinate with each other and with the user. Human interfaces to these embedded devices will in many cases be very different from those appropriate to workstations.

Augmented reality. A common staple of science fiction, augmented reality refers to the notion of layering relevant information into our vision of the world. Existing projects show real-time statistics to users performing difficult tasks, such as manufacturing. Future work might include augmenting our social interactions by providing additional information about those we converse with.

Group interfaces. Interfaces to allow groups of people to coordinate will be common (e.g., for meetings, for engineering projects, for authoring joint documents). These will have major impacts on the nature of organizations and on the division of labor. Models of the group design process will be embedded in systems and will cause increased rationalization of design.

User Tailorability. Ordinary users will routinely tailor applications to their own use and will use this power to invent new applications based on their understanding of their own domains. Users, with their deeper knowledge of their own knowledge domains, will increasingly be important sources of new applications at the expense of generic systems programmers (with systems expertise but low domain expertise).

Information Utilities. Public information utilities (such as home banking and shopping) and specialized industry services (e.g., weather for pilots) will continue to proliferate. The rate of proliferation will accelerate with the introduction of high-bandwidth interaction and the improvement in quality of interfaces.

3.5 HUMAN–COMPUTER INTERFACE

The human–computer interface can be described as the point of communication between the human user and the computer. The flow of information between the human and computer is defined as the loop of interaction. The loop of interaction has several aspects to it including:

Task Environment: The conditions and goals set upon the user.
Machine Environment: The environment that the computer is connected to, i.e a laptop in a college student's dorm room.
Areas of the Interface: Non-overlapping areas involve processes of the human and computer not pertaining to their interaction. Meanwhile, the overlapping areas only concern themselves with the processes pertaining to their interaction.
Input Flow: Begins in the task environment as the user has some task that requires using their computer.
Output: The flow of information that originates in the machine environment.

Feedback: Loops through the interface that evaluate, moderate, and confirm processes as they pass from the human through the interface to the computer and back.

4.0 CONCLUSION

In this unit you have been introduced to the concepts of Human Computer Interaction. You have also been introduced to the various goals of HCI and also the difference between HCI and other related fields. The future of HCI was also discussed. The concept of the Human computer interface was also introduced.

5.0 SUMMARY

You have learnt the following:-

- Introduction to Human–computer interaction (HCI) which is the study of interaction between people (users) and computers. HCI is also sometimes referred to as man–machine interaction (MMI) or computer–human interaction (CHI).
- The goal of HCI which is basically to improve the interactions between users and computers by making computers more usable and receptive to the user's needs.
- The difference between HCI and other related fields like Graphical user interface, ergonomics e.t.c.
- The future development in HCI like Ubiquitous communication, High functionality systems, Mass availability of computer graphics e.t.c.
- The human–computer interface which can be described as the point of communication between the human user and the computer.

Exercises

1. What do you understand by Human computer Interaction?

2. Describe the Human computer interface

6.0 TUTOR MARKED ASSIGNMENT

Discuss briefly the future development in HCI.

7.0 FURTHER READING AND OTHER RESOURCES

More discussion of the differences between these terms can be found in the ACM SIGCHI Curricula for Human-Computer Interaction

Green, Paul (2008). Iterative Design. Lecture presented in Industrial and Operations Engineering 436 (Human Factors in Computer Systems, University of Michigan, Ann Arbor, MI, February 4, 2008.

Wickens, Christopher D., John D. Lee, Yili Liu, and Sallie E. Gordon Becker. An Introduction to Human Factors Engineering. Second ed. Upper Saddle River, NJ: Pearson Prentice Hall, 2004. 185–193.

Brown, C. Marlin. Human-Computer Interface Design Guidelines. Intellect Books, 1998.

UNIT 5 ERGONOMICS

TableofContents

1.0 INTRODUCTION

Having read through the course guide, you will have a general understanding of Ergonomics. This unit describes in great detail the various aspects of ergonomics alongside the history behind it. The efficiency and benefits of ergonomics will also be discussed. Different fields of ergonomics will also be highlighted.

2.0 OBJECTIVES

By the end this unit, you should be able to:

- Explain the term Ergonomics
- Identify the various aspects of Ergonomics.
- Have good knowledge of the history of Ergonomics.
- Describe efficiency and ergonomics
- Identify the various benefits of ergonomics
- Highlight the various fields of ergonomics

3.0 MAIN CONTENT

3.1 INTRODUCTION TO ERGONOMICS

Ergonomics derives from two Greek words: ergon, meaning work, and nomoi, meaning natural laws, to create a word that means the science of work and a person's relationship to that work.

The International Ergonomics Association has adopted this technical definition: ergonomics (or human factors) is the scientific discipline concerned with the understanding of interactions among humans and other elements of a system, and the profession that applies theory, principles, data and methods to design in order to optimize human well-being and overall system performance.

Ergonomics is the science of making things comfy. It also makes things efficient. However for simplicity, ergonomics makes things comfortable and efficient.
At its simplest definition ergonomics literally means the science of work. So ergonomists, i.e. the practitioners of ergonomics, study work, how work is done and how to work better. It is the attempt to make work better that ergonomics becomes so useful. And that is also where making things comfortable and efficient comes into play.

Ergonomics is commonly thought of in terms of products. But it can be equally useful in the design of services or processes. It is used in design in many complex ways. However, what you, or the user, is most concerned with is, "How can I use the product or service, will it meet my needs, and will I like using it?" Ergonomics helps define how it is used, how it meets you needs, and most importantly if you like it. It makes things comfy and efficient.

Ergonomics is concerned with the 'fit' between people and their work. It takes account of the worker's capabilities and limitations in seeking to ensure that tasks, equipment, information and the environment suit each worker.

To assess the fit between a person and their work, ergonomists consider the job being done and the demands on the worker; the equipment used (its size, shape, and how appropriate it is for the task), and the information used (how it is presented, accessed, and changed). Ergonomics draws on many disciplines in its study of humans and their environments, including anthropometry, biomechanics, mechanical engineering, industrial engineering, industrial design, kinesiology, physiology and psychology.

Typically, an ergonomist will have a BA or BS in Psychology, Industrial/Mechanical Engineering or Health Sciences, and usually an MA, MS or PhD in a related discipline. Many universities offer Master of Science degrees in Ergonomics, while some offer Master of Ergonomics or Master of Human Factors degrees. In the 2000s, occupational therapists have been moving into the field of ergonomics and the field has been heralded as one of the top ten emerging practice areas.

3.2 **FIVE ASPECTS OF ERGONOMICS**

There are five aspects of ergonomics: safety, comfort, ease of use, productivity/performance, and aesthetics. Based on these aspects of ergonomics, examples are given of how products or systems could benefit from redesign based on ergonomic principles.

Safety – This has to do with the ability to use a device or work with a device without short or long term damage to parts of the body. For example in Medicine bottles: The print on them could be larger so that a sick person who may have bad vision (due to sinuses, etc.) can more easily see the dosages and label. Ergonomics could design the print style, colour and size for optimal viewing.

Comfort – Comfort in the human-machine interface is usually noticed first. Physical comfort in how an item feels is pleasing to the user. If you do not like to touch it you won't. If you do not touch it you will not operate it. If you do not operate it, then it is useless. For example, in Alarm clock display: Some displays are harshly bright, drawing one's eye to the light when surroundings are dark. Ergonomic principles could re-design this based on contrast principles.

Ease of use – This has to do with the ability to use a device with no stress. For example in Street Signs: In a strange area, many times it is difficult to spot street signs. This could be addressed with the principles of visual detection in ergonomics.

Productivity/performance – For example in HD TV: The sound on HD TV is much lower than regular TV. So when you switch from HD to regular, the volume increases dramatically. Ergonomics recognizes that this difference in decibel level creates a difference in loudness and hurts human ears and this could be solved by evening out the decibel levels.

Aesthetics - the look and feel of the object, the user experience.

3.3 HISTORY OF ERGONOMICS

The foundations of the science of ergonomics appear to have been laid within the context of the culture of Ancient Greece. A good deal of evidence indicates that Hellenic civilization in the 5th century BCE used ergonomic principles in the design of their tools, jobs, and workplaces.

The term ergonomics is derived from the Greek words *ergon* [work] and *nomos* [natural laws] and first entered the modern lexicon when Wojciech Jastrzębowski used the word in his 1857 article *Rys ergonomji czyli nauki o pracy, opartej na prawdach poczerpniętych z Nauki Przyrody* (The Outline of Ergonomics, i.e. Science of Work, Based on the Truths Taken from the Natural Science).

Later, in the 19th century, Frederick Winslow Taylor pioneered the "Scientific Management" method, which proposed a way to find the optimum method for carrying out a given task. Taylor found that he could, for example, triple the amount of coal that workers were shoveling by incrementally reducing the size and weight of coal shovels until the fastest shoveling rate was reached. Frank and Lillian Gilbreth expanded Taylor's methods in the early 1900s to develop "Time and Motion Studies". They aimed to improve efficiency by eliminating unnecessary steps and actions. By applying this

approach, the Gilbreths reduced the number of motions in bricklaying from 18 to 4.5, allowing bricklayers to increase their productivity from 120 to 350 bricks per hour.

World War II marked the development of new and complex machines and weaponry, and these made new demands on operators' cognition. The decision-making, attention, situational awareness and hand-eye coordination of the machine's operator became key in the success or failure of a task. It was observed that fully functional aircraft, flown by the best-trained pilots, still crashed. In 1943, Alphonse Chapanis, a lieutenant in the U.S. Army, showed that this so-called "pilot error" could be greatly reduced when more logical and differentiable controls replaced confusing designs in airplane cockpits.

In the decades since the war, ergonomics has continued to flourish and diversify. The Space Age created new human factors issues such as weightlessness and extreme g-forces. How far could environments in space be tolerated, and what effects would they have on the mind and body? The dawn of the Information Age has resulted in the new ergonomics field of human-computer interaction (HCI). Likewise, the growing demand for and competition among consumer goods and electronics has resulted in more companies including human factors in product design.

At home, work, or play new problems and questions must be resolved constantly. People come in all different shapes and sizes, and with different capabilities and limitations in strength, speed, judgment, and skills. All of these factors need to be considered in the design function. To solve design problems, physiology and psychology must be included with a engineering approach.

3.4 ERGONOMICS IN WORKPLACE

Figure 2:- *Description of workplace environment*

Fundamentals for the Flexible Workplace Variability and compatibility with desk components, that flex from individual work activities to team settings. Workstations provide supportive ergonomics for task-intensive environments.

Outside of the discipline itself, the term 'ergonomics' is generally used to refer to physical ergonomics as it relates to the workplace (as in for example ergonomic chairs and keyboards). Ergonomics in the workplace has to do largely with the safety of employees, both long and short-term. Ergonomics can help reduce costs by improving safety. This

would decrease the money paid out in workers' compensation. For **example**, over five million workers sustain overextension injuries per year. Through ergonomics, workplaces can be designed so that workers do not have to overextend themselves and the manufacturing industry could save billions in workers' compensation.

Workplaces may either take the reactive or proactive approach when applying ergonomics practices. Reactive ergonomics is when something needs to be fixed, and corrective action is taken. Proactive ergonomics is the process of seeking areas that could be improved and fixing the issues before they become a large problem. Problems may be fixed through equipment design, task design, or environmental design. Equipment design changes the actual, physical devices used by people. Task design changes what people do with the equipment. Environmental design changes the environment in which people work, but not the physical equipment they use.

3.5 EFFICIENCY AND ERGONOMICS

Efficiency is quite simply making something easier to do. Several forms of efficiency are:-

- Reducing the strength required makes a process more physically efficient.
- Reducing the number of steps in a task makes it quicker (i.e. efficient) to complete.
- Reducing the number of parts makes repairs more efficient.
- Reducing the amount of training needed, i.e. making it more intuitive, gives you a larger number of people who are qualified to perform the task. Imagine how in-efficient trash disposal would be if your teenage child wasn't capable of taking out the garbage. What? They're not? Have you tried an ergonomic trash bag?

Efficiency can be found almost everywhere. If something is easier to do you are more likely to do it. If you do it more, then it is more useful. Again, utility is the only true measure of the quality of a design.

And if you willingly do something more often you have a greater chance of liking it. If you like doing it you will be more comfortable doing it.
So the next time you hear the term ergonomics you will know what it means to you. And I hope that is a comforting thought.

Ergonomics can help you in many ways. Among other things, it can benefit your life, health, productivity and accuracy. One of the best benefits of ergonomics is saving time. We never seem to have enough of it as it is, so why not try to get a little more out of your day?

Ergonomics is about making things more efficient. By increasing the efficiency of a tool or a task, you tend to shorten the length of time it takes to accomplish your goal.

3.6 BENEFITS OF ERGONOMICS

The three main benefits of ergonomics are:-

a. ***Slim Down the Task***

Have you ever wondered why some things are so convoluted, cumbersome and chaotic? And they take forever to complete. And most of what you do does not aid the outcome. For example, think back to the last time you got hired for a job, bought a house or car, or did something else that required a ton of paperwork. How many different forms did you write the same information on? That was not very ergonomic.

You can almost always make a task a little leaner. But first you have to understand the task. For that we use a task analysis.

Pick any mundane task you typically do at least once a week. Write out a task analysis for it. Don't worry about wasting your time doing this. You will make it up with the time savings you create.

Once you have all the steps written out, you need to take a good look at them and identify areas that you can "ergonomize":

Repetition – Look for steps that are repeated and see if they are all necessary.
Order – See if you can re-order the steps to optimize your effort.
Synergy – Can you combine things or somehow get more bang for your buck?
Value Added – Look at every step and make sure it adds value to the outcome. If it doesn't, cut it.
Necessity – Make sure the quantity of the step is needed. Do you really need to brush your teeth with 57 strokes, or will 32 do?

b. **Simplify the Task**

You can also save time by simplifying the task. This is less about reducing the number of steps, but making those steps easier to perform. The less training and/or skill that required for a task, the quicker the pace at which it tends to get finished.
This is a great ergonomic tip, especially when the task requires more than one person. If you are trying to get your kids to pick up their toys before they go to bed, you can save a lot of time by making it easier. That is what the toy chest is for. Instead of having different places for different things, they can just throw everything in one place.

c. Increase Body Mechanism

Ergonomic can increase your body mechanics. A good ergonomic tool acts as extension of your body enhancing capabilities. Some tools make you more effective and faster at completing a task. (Try cutting a log without an axe and see how long it takes you.)

3.7 FIELDS OF ERGONOMICS

a. Engineering psychology

Engineering psychology is an interdisciplinary part of Ergonomics and studies the relationships of people to machines, with the intent of improving such relationships. This may involve redesigning equipment, changing the way people use machines, or changing the location in which the work takes place. Often, the work of an engineering psychologist is described as making the relationship more "user-friendly."
Engineering Psychology is an applied field of psychology concerned with psychological factors in the design and use of equipment. Human factors is broader than engineering psychology, which is focused specifically on designing systems that accommodate the information-processing capabilities of the brain.

b. Macroergonomics

Macroergonomics is an approach to ergonomics that emphasizes a broad system view of design, examining organizational environments, culture, history, and work goals. It deals with the physical design of tools and the environment. It is the study of the society/technology interface and their consequences for relationships, processes, and institutions. It also deals with the optimization of the designs of organizational and work systems through the consideration of personnel, technological, and environmental variables and their interactions. The goal of macroergonomics is a completely efficient work system at both the macro- and micro-ergonomic level which results in improved productivity, and employee satisfaction, health, safety, and commitment. It analyzes the whole system, finds how each element should be placed in the system, and considers all aspects for a fully efficient system. A misplaced element in the system can lead to total failure.

c. Seating Ergonomics

The best way to reduce pressure in the back is to be in a standing position. However, there are times when you need to sit. When sitting, the main part of the body weight is transferred to the seat. Some weight is also transferred to the floor, back rest, and armrests. Where the weight is transferred is the key to a good seat design. When the proper areas are not supported, sitting in a seat all day can put unwanted pressure on the back causing pain.

The lumbar (bottom five vertebrate in the spine) needs to be supported to decrease disc pressure. Providing both a seat back that inclines backwards and has a lumbar support is critical to prevent excessive low back pressures. The combination which minimizes pressure on the lower back is having a backrest inclination of 120 degrees and a lumbar support of 5 cm. The 120 degrees inclination means the angle between the seat and the backrest should be 120 degrees. The lumbar support of 5 cm means the chair backrest supports the lumbar by sticking out 5 cm in the lower back area.

Another key to reducing lumbar disc pressure is the use of armrests. They help by putting the force of your body not entirely on the seat and back rest, but putting some of this pressure on the armrests. Armrest needs to be adjustable in height to assure shoulders are not overstressed.

4.0 CONCLUSION

In this unit, you have been introduced to the fundamental concepts of Ergonomics. You have also learnt the different aspect of ergonomics and also the history of ergonomics. Ergonomics in work place was also discussed alongside achieving efficiency in ergonomics. The various benefits of ergonomics were also discussed. The various fields of ergonomics was also briefly explained.

5.0 SUMMARY

You must have learnt the following in this unit:-

Introduction to Ergonomics which is derived from two Greek words: ergon, meaning work, and nomoi, meaning natural laws, to create a word that means the science of work and a person's relationship to that work.

Highlighting of the various aspect of Ergonomics like safety, comfort, ease of use e.t.c.

The history of ergonomics whose foundations appears to have been laid within the context of the culture of Ancient Greece.

The discussion of Ergonomics in work place and achieving efficiency in ergonomics.

Explanation of the various benefits of Ergonomics which was discussed in greater detail.

The discussion of various fields of ergonomics like Engineering psychology, Macroergonomics, Seating Ergonomics.

Exercises

1. What do you understand by Ergonomics?
2. Highlight the various benefits of ergonomics.

6.0 TUTOR MARKED ASSIGNMENT

Discuss briefly any two fields of ergonomics

7.0 FURTHER READING AND OTHER RESOURCES

www.wikipedia.com

Berkeley Lab. *Integrated Safety Management: Ergonomics*. Website. Retrieved 9 July 2008.

Berkeley lab. *Today at Berkeley lab: Ergonomic Tips for Computer Users*. Retrieved 8 January 2009.

Wickens and Hollands (200). *Engineering Psychology and Human Performance*.

Brookhuis, K., Hedge, A., Hendrick, H., Salas, E., and Stanton, N. (2005). Handbook of Human Factors and Ergonomics Models. Florida: CRC Press.

MODULE 2 – USER INTERFACE DESIGN TECHNIQUES

UNIT 1 – HUMAN CAPABILITIES IN USER INTERFACE DESIGN

UNIT 2 – UNDERSTANDING USERS AND TASK ANALYSIS

UNIT 3 - USER-CENTERED DESIGN

UNIT 4 – INTERACTIVE DESIGN

UNIT 5 – USABILITY

UNIT 6 – INTERACTION STYLES AND GRAPHIC DESIGN PRINCIPLES

MODULE 2

UNIT 1 HUMAN CAPABILITIES IN USER INTERFACE DESIGN

TableofContents

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

By the end 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

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, and KLM (human-computer interaction). 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 3.

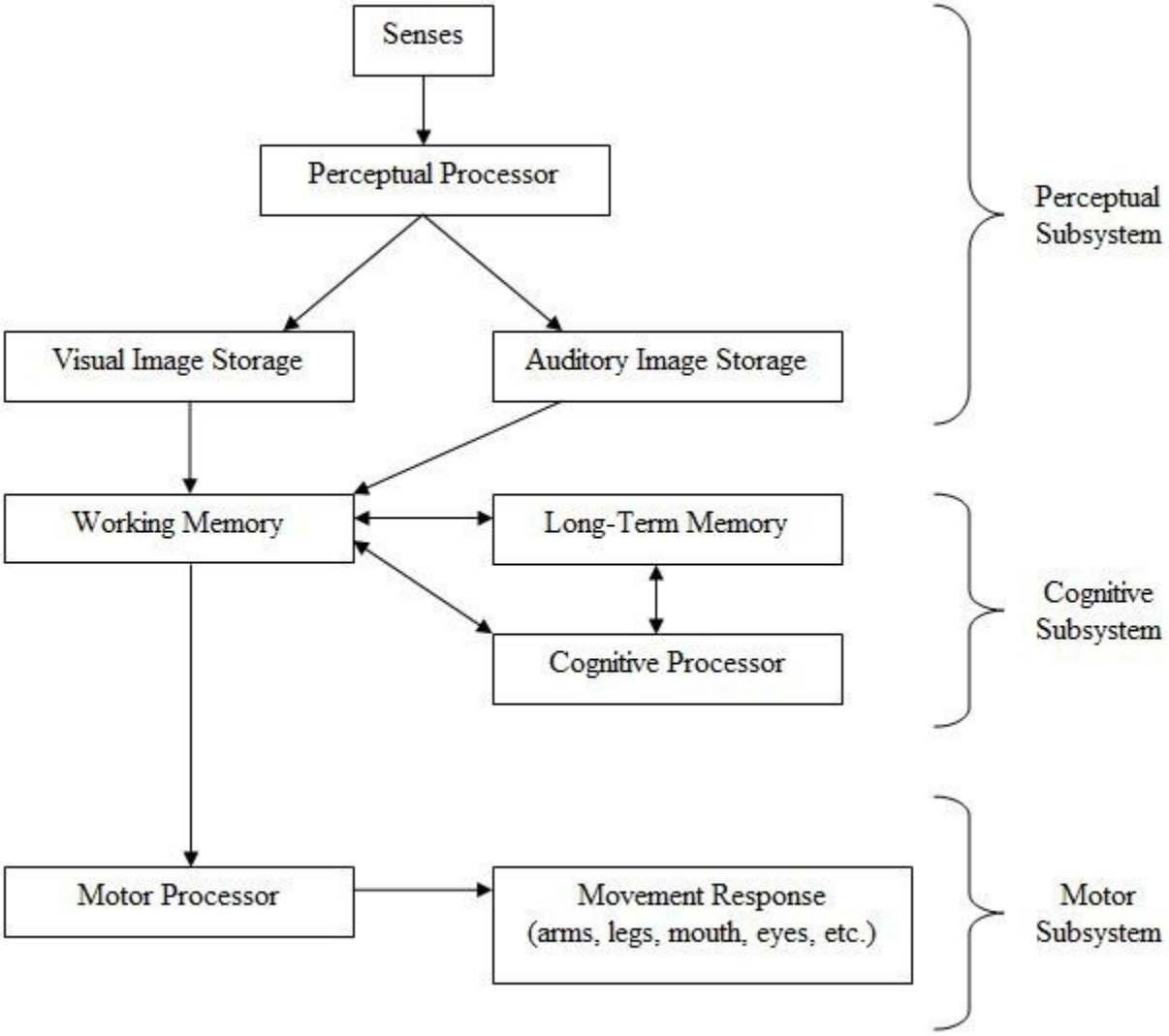

Figure 3:- ***Human Processor Model***

Input from the eyes and ears are first stored in the **short-term sensory store**. 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 recognize *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 recognize.

The **cognitive processor** takes the symbols recognized 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** 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 corresponding to your **attention**, which might be thought of like a thread of control in a computer system.
Note that this model isn't meant to reflect the anatomy of your nervous system. There probably isn't a single area in your brain corresponding to the perceptual processor, for example. But it's a useful abstraction nevertheless.

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 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.
By following the connections in figure 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 walk-through 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 Tiffany Jastrembski and Neil Charness (2007). Table 1 shows some of the results from the work

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

Table 1:- ***Processor actions' times***

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, they 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.

3.2 PERCEPTION

In philosophy, 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 considerable regarding perception: phenomenal (any occurrence that is observable and physical) and psychological. The difference everybody can demonstrate to him- 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 recognizes, 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, coloring, writing, or threading beads. Fine motor development refers to the development of skills involving the smaller muscle groups.

Ambidexterity is a specialized 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 design 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. Color derives from the spectrum of light (distribution of light energy versus wavelength) interacting in the eye with the spectral sensitivities of the light receptors. Color categories and physical specifications of color are also associated with objects, materials, light sources, etc., based on their physical properties such as light absorption, reflection, or emission spectra. Colors can be identified by their unique RGB and 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 color to its physical specification. Because perception of color stems from the varying sensitivity of different types of cone cells in the retina to different parts of the spectrum, colors may be defined and quantified by the degree to which they stimulate these cells. These physical or physiological quantifications of color, however, do not fully explain the psychophysical perception of color appearance.

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

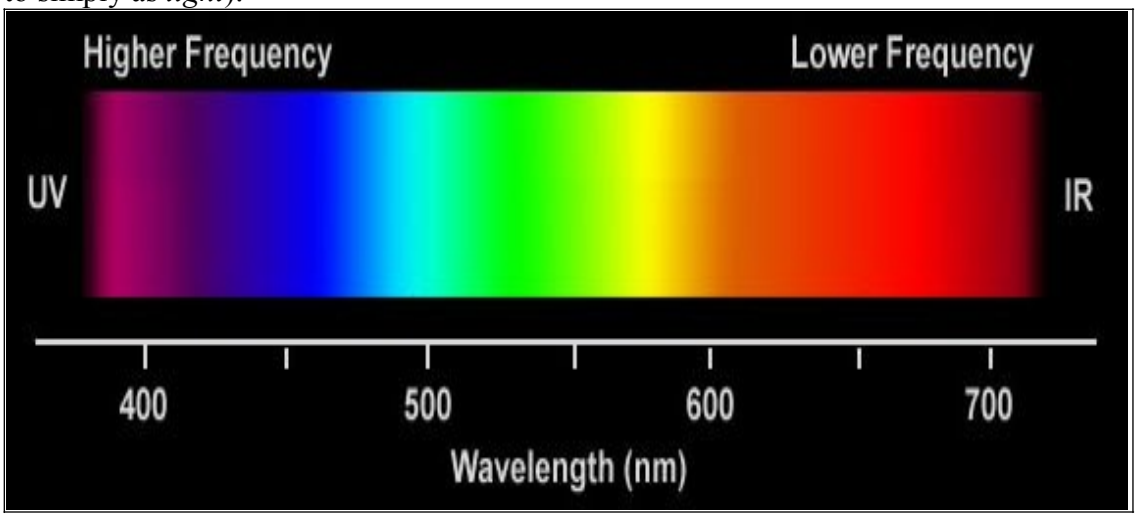

Figure 4 - ***Visible Spectrum***

The ability of the human eye to distinguish colors is based upon the varying sensitivity of different cells in the retina to light of different wavelengths. The retina contains three types of color 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 color 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. Focalization, 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 was 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 of the model were also discussed.

- **Perception** which is the process of attaining awareness or understanding of sensory information. Passive and visual perception were 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 TUTOR MARKED ASSIGNMENT

a. Explain briefly the Human processor model.

b. Write a short note on Ambidexterity.

7.0 FURTHER READING AND OTHER RESOURCES

Lui, Yili; Feyen, Robert; and Tsimhoni, Omer. *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, pages 37-70.

Card, S.K; Moran, T. P; and Newell, A. *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, 1986, pages 1–35.

Jastrzembski, Tiffany; and Charness, Neil. *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, pages 224-248.
http://en.wikipedia.org/wiki

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

Robles-De-La-Torre G. 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 (2006).

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

Wright, R.D. & Ward, L.M. (2008). Orienting of Attention. Oxford University Press
Pinel, J. P. (2008). *Biopsychology* (7th ed.). Boston: Pearson. (p. 357)

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

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

MODULE 2

UNIT 2 UNDERSTANDING USERS AND TASK ANALYSIS

TableofContents

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

By the end 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.

3.1.1 GETTING 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 can not 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 behavior 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.

3.2.1 TASK ANALYSIS IN USER INTERFACE DESIGN

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

a. 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.

b. 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

c. 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 analyze a proposed interface. Here are a few examples:

- 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"

Again, these should be real tasks that users have faced, and the design team should collect the materials needed to do them: a copy of the tape on which the memo is dictated, a list of salaries for the current year and factors to be considered in their revision, etc.

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 more 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.

d. 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 produced 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's 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, in 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 clicking 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 spec.

We represented our scenarios with 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. Our scenarios let users see what the design would really give them.

"This sample task idea seems crazy. 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. 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'll 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 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 analyze 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 a 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'll probably feel that you've done all the copying that you can, and that you've 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. Analyze 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

What you have learnt in this unit concerns

- Understanding users which involves figuring out who is going to use the program and for what it is been used for.

- 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 TUTOR MARKED ASSIGNMENT

a. Explain Task Analysis.
b. Explain the advantages and disadvantages of borrowing in creating Initial designs.

7.0 FURTHER READING AND OTHER RESOURCES

Lui, Yili; Feyen, Robert; and Tsimhoni, Omer. *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, pages 37-70.

Card, S.K; Moran, T. P; and Newell, A. *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, 1986, pages 1–35.

Jastrzembski, Tiffany; and Charness, Neil. *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, pages 224-248.
http://en.wikipedia.org/wiki

Morrell, Jessica Page (2006). *Between the Lines: Master the Subtle Elements of Fiction Writing*. Cincinnati, OH: Writer's Digest Books.
Robles-De-La-Torre G. 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 (2006).

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

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

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

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

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

MODULE 2

UNIT 3 USER CENTERED DESIGN (UCD)

TableofContents

1.0 INTRODUCTION

Reading through this unit, you will be introduced to User Centered 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

By the end this unit, you should be able to:

- Explain the term User Centered Design
- Describe the various purpose of UCD.
- Highlights the major considerations and various approaches in UCD.
- Describe concept of participatory design.

3.0 MAIN CONTENT

3.1 INTRODUCTION TO USER-CENTERED DESIGN

User-centered 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 characterized as a multi-stage problem solving process that not only requires designers to analyze 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 chief difference from other interface design philosophies is that user-centered design tries to optimize 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-CENTERED DESIGN

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

3.3 MAJOR CONSIDERATIONS OF UCD

a. 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 document.

b. Accessibility

Users should be able to find information quickly and easily throughout the document, whether it be 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, color coding, etc). Navigational elements should be consistent with the genre of the document. 'Chunking' is a useful strategy that involves breaking information into small pieces that can be organized into some type meaningful order or hierarchy. The ability to skim the document allows users to find their piece of information by scanning rather than reading. bold and italic words are often used.

c. 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.

d. 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 IT artifacts and it has been evolving since 1970.
- Participatory design (PD): a North Ameri:an 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 centered 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.

3.5 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.

3.5.1 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.

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-centered 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.

6.0 TUTOR MARKED ASSIGNMENT

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

7.0 FURTHER READING AND OTHER RESOURCES

Lui, Yili; Feyen, Robert; and Tsimhoni, Omer. *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, pages 37-70.
www.wikipedia.org

MODULE 2

UNIT 4 INTERACTION DESIGN

TableofContents

1.0 INTRODUCTION

This course guide will introduce you to Interaction design. User-Centered Interaction design will also be discussed. The history, relationship, methodologies and various aspects of User-Centered Interaction Design will also be explained. Interaction design domains will also be discussed.

2.0 OBJECTIVES

By the end this unit, you should be able to:

- Explain interaction Design and user-Centered Interaction Design.
- Discuss the history, relationship and methodologies of Interaction Design.
- Have good knowledge of the various aspect of User-Centered Interaction Design.
- Describe Interaction design domains.

3.0 MAIN CONTENT

3.1 INTERACTION DESIGN

Interaction Design **(**IxD**)** is the discipline of defining the behavior 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

organizations themselves. Interaction design defines the behavior (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.

3.2 USER-CENTERED INTERACTION DESIGN

As technologies are often overly complex for their intended target audience, interaction design aims to minimize the learning curve and to increase accuracy and efficiency of a task without diminishing usefulness. The objective is to reduce frustration and increase user productivity and satisfaction.

Interaction design attempts to improve the usability and experience of the product, by first researching and understanding certain users' needs and then designing to meet and exceed them. (Figuring out who needs to use it, and how those people would like to use it.)

Only by involving users who will use a product or system on a regular basis will designers be able to properly tailor and maximize usability. Involving real users, designers gain the ability to better understand user goals and experiences. (see also: User-centered design) There are also positive side effects which include enhanced system capability awareness and user ownership. It is important that the user be aware of system capabilities from an early stage so that expectations regarding functionality are both realistic and properly understood. Also, users who have been active participants in a product's development are more likely to feel a sense of ownership, thus increasing overall satisfaction.

3.3 HISTORY OF INTERACTION DESIGN

The term *interaction design* was first proposed by Bill Moggridge and Bill Verplank in the late 1980s. To Verplank, it was an adaptation of the computer science term *user interface design* to the industrial design profession.[3] To Moggridge, it was an improvement over *soft-face*, which he had coined in 1984 to refer to the application of industrial design to products containing software (Moggridge 2006).
In 1989, Gillian Crampton-Smith established an interaction design MA at the Royal College of Art in London (originally entitled "computer-related design" and now known as "design interactions"). In 2001, she helped found the Interaction Design Institute Ivrea, a small institute in Northern Italy dedicated solely to interaction design; the institute

moved to Milan in October 2005 and merged courses with Domus Academy. Today, some of the people originally involved with IDII have now set up a new institute in Copenhagen, called the Copenhagen Institute of Interaction Design or CIID. Today, interaction design is taught in many schools worldwide.

3.4 RELATIONSHIP WITH USER INTERFACE DESIGN

Interaction Design is often associated with the design of system interfaces in a variety of media (see also: Interface design, Experience design) but concentrates on the aspects of the interface that define and present its behavior over time, with a focus on developing the system to respond to the user's experience and not the other way around. The system interface can be thought of as the artifact (whether visual or other sensory) that represents an offering's designed interactions. Interactive voice response (Telephone User Interface) is an example of interaction design without graphical user interface as a media.

Interactivity, however, is not limited to technological systems. People have been interacting with each other as long as humans have been a species. Therefore, interaction design can be applied to the development of all solutions (or offerings), such as services and events. Those who design these offerings have, typically, performed interaction design inherently without naming it as such.

3.5 INTERACTIVE DESIGN METHODOLOGIES

Interaction designers often follow similar processes to create a solution (not *the* solution) to a known interface design problem. Designers build rapid prototypes and test them with the users to validate or rebut the idea.
There are six major steps in interaction design. These are:-

a. Design research

Using design research techniques (observations, interviews, questionnaires, and related activities) designers investigate users and their environment in order to learn more about them and thus be better able to design for them.

b. 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 realize user requirements, they may use tools such as personas or user profiles that are reflective of their targeted user group. From these personae, and the patterns of behavior observed in the research, designers create scenarios (or user stories) or storyboards, which imagine a future work flow 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.

c. Alternative design and evaluation

Once 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 wireframing 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.

d. Prototyping and usability testing

Interaction 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 artifact, those that test its **look and feel** and those that test its **implementation**. Sometimes, these are called **experience prototypes** to emphasize their interactive nature. Prototypes can be physical or digital, high- or low-fidelity.

e. Implementation

Interaction designers need to be involved during the development of the product or service to ensure that what was designed is implemented correctly. Often, changes need to be made during the building process, and interaction designers should be involved with any of the on-the-fly modifications to the design.

f. System testing

Once the system is built, often another round of testing, for both usability and errors ("bug catching") is performed. Ideally, the designer will be involved here as well, to make any modifications to the system that are required.

3.6 ASPECTS OF INTERACTION DESIGN

Social interaction design

Social interaction design (SxD) is emerging because many of our computing devices have become networked and have begun to integrate communication capabilities. Phones, digital assistants and the myriad connected devices from computers to games facilitate talk and social interaction. Social interaction design accounts for interactions among

users as well as between users and their devices. The dynamics of interpersonal communication, speech and writing, the pragmatics of talk and interaction--these now become critical factors in the use of social technologies. And they are factors described less by an approach steeped in the rational choice approach taken by cognitive science than that by sociology, psychology, and anthropology.

Affective interaction design

Throughout the process of interaction design, designers must be aware of key aspects in their designs that influence emotional responses in target users. The need for products to convey positive emotions and avoid negative ones is critical to product success.[1] These aspects include positive, negative, motivational, learning, creative, social and persuasive influences to name a few. One method that can help convey such aspects is the use of expressive interfaces. In software, for example, the use of dynamic icons, animations and sound can help communicate a state of operation, creating a sense of interactivity and feedback. Interface aspects such as fonts, color pallet, and graphical layouts can also influence an interface's perceived effectiveness. Studies have shown that affective aspects can affect a user's perception of usability.[1]
Emotional and pleasure theories exist to explain peoples responses to the use of interactive products. These includes Don Norman's emotional design model, Patrick Jordan's pleasure model, and McCarthy and Wright's Technology as Experience framework.

3.7 INTERACTION DESIGN DOMAINS

Interaction designers work in many areas, including software interfaces, (business) information systems, internet, physical products, environments, services, and systems which may combine many of these. Each area requires its own skills and approaches, but there are aspects of interaction design common to all.
Interaction designers often work in interdisciplinary teams as their work requires expertise in many different domains, including graphic design, programming, psychology, user testing, product design, etc (see below for more related disciplines). Thus, they need to understand enough of these fields to work effectively with specialists.

4.0 CONCLUSION

In this unit, you have been introduced to the principles of interaction design. User-Centered Interaction design was also discussed. The history, relationship, methodologies and various aspects of User-Centered Interaction Design was also explained. An interaction design domain was also discussed.

5.0 SUMMARY

The summary of what have been discussed in this course are:-

- Interaction design which is the discipline of defining the behavior of products and systems that a user can interact with.
- User-Centered Interaction Design which attempts to improve the usability and experience of a product, by first researching and understanding certain users' needs and then designing to meet and exceed them by involving the real users.
- The history, relationship, methodologies and various aspects of Interaction Design which was discussed in greater detail.
- Interaction design domains which are usually in interdisciplinary teams because most work requires expertise in many different domains.

6.0 TUTOR MARKED ASSIGNMENT

a. Explain the User-Centered Interaction Design how it can be used to improve user interface.

b. Discuss Interaction design domains.

7.0 FURTHER READING AND OTHER RESOURCES

Helen Sharp, Yvonne Rogers, & Jenny Preece, Interaction Design - beyond human-computer interaction, 2007 (2nd Edition ed., pp. 181-217). John Wiley & Sons.

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

MODULE 2

UNIT 5 - USABILITY

TableofContents

1.0 INTRODUCTION

Having read through the course guide, you will be introduction to Usability, Usability Concepts and Guidelines, Usability Considerations, ISO Standard for Usability and various Usability Methodologies.

2.0 OBJECTIVES

By the end this unit, you should be able to:

Explain usability
Identify the various concepts and guidelines of usability
Have good understanding of the ISO Standard for usability.
Highlight 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 generalized 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 recognized 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 specialization 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., recognizing that a door can be opened by turning its handle).
Usability is also very important in website development. Studies of user behavior 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 maximizing 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 emphasize, 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 hard 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 ISO STANDARDS FOR USABILITY

ISO/TR 16982:2002 "Ergonomics of human-system interaction -- Usability methods supporting human-centered design". This standard provides information on human-centred usability methods which can be used for design and evaluation. It details the advantages, disadvantages and other factors relevant to using each usability method.

It explains the implications of the stage of the life cycle and the individual project characteristics for the selection of usability methods and provides examples of usability methods in context.

The main users of ISO/TR 16982:2002 will be project managers. It therefore addresses technical human factors and ergonomics issues only to the extent necessary to allow managers to understand their relevance and importance in the design process as a whole.

The guidance in ISO/TR 16982:2002 can be tailored for specific design situations by using the lists of issues characterizing the context of use of the product to be delivered. Selection of appropriate usability methods should also take account of the relevant life-cycle process.

ISO/TR 16982:2002 is restricted to methods that are widely used by usability specialists and project managers.

ISO/TR 16982:2002 does not specify the details of how to implement or carry out the usability methods described.

ISO 9241

ISO 9241 is a multi-part standard covering a number of aspects for people working with computers. Although originally titled Ergonomic requirements for office work with visual display terminals (VDTs) it is being retitled to the more generic Ergonomics of Human System Interaction by ISO. As part of this change, ISO is renumbering the standard so that it can include many more topics. The first part to be renumbered was part 10 (now renumbered to part 110).
Part 1 is a general introduction to the rest of the standard. Part 2 addresses task design for working with computer systems. Parts 3–9 deal with physical characteristics of computer equipment. Parts 110 and parts 11–19 deal with usability aspects of software, including Part 110 (a general set of usability heuristics for the design of different types of dialogue) and Part 11 (general guidance on the specification and measurement of usability).

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. Therefore, when designing for usability, the three principles of design, are: early focus on users and tasks, Iterative Design and Testing.

a. 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.

b. Iterative Design

Iterative Design is a design methodology based on a cyclic process of prototyping, testing, analyzing, and refining a product or process. Based on the results of testing the most recent iteration of a design, changes and refinements are made. This process is intended to ultimately improve the quality and functionality of a design. In iterative design, interaction with the designed system is used as a form of research for informing and evolving a project, as successive versions, or iterations of a design are implemented. The key requirements for Iterative Design are: identification of required changes, an ability to make changes, and a willingness to make changes. When a problem is encountered, there is no set method to determine the correct solution. Rather, there are empirical methods that can be used during system development or after the system is delivered, usually a more inopportune time. Ultimately, iterative design works towards meeting goals such as making the system user friendly, easy to use, easy to operate, simple, etc.

c. 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

You should have understood the following in this unit:-

- 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 TUTOR MARKED ASSIGNMENT

a. Explain Usability.

b. Discuss any two usability methodologies.

7.0 FURTHER READING AND OTHER RESOURCES

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

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.

Wright, R.D. & Ward, L.M. (2008). Orienting of Attention. Oxford University Press
Pinel, J. P. (2008). *Biopsychology* (7th ed.). Boston: Pearson. (p. 357)

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

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

MODULE 2

UNIT 6 - INTERACTION STYLES AND GRAPHIC DESIGN PRINCIPLES

TableofContents

1.0 INTRODUCTION

This unit will introduce you to Interaction Styles and Graphic Design Principles.

2.0 OBJECTIVES

By the end this unit, you should be able to:

Explain different Interaction Styles
Identify their advantages and disadvantages of different interaction styles.
Explain Graphic Design Principles

3.0 MAIN CONTENT

3.1 INTERACTION STYLES

The concept of **Interaction Styles** 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 syatems. 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.

a. 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.

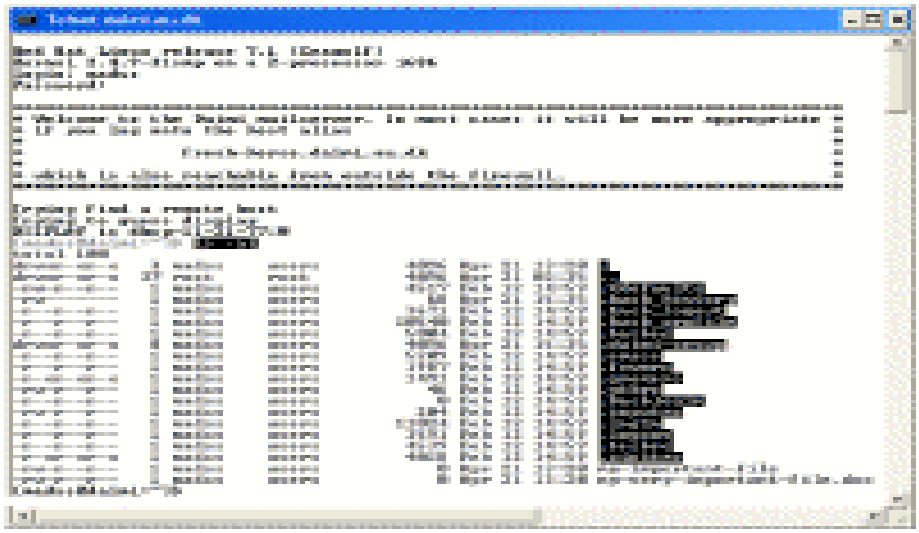

Figure 5: Command prompt. The command "ls- al" has just been executed ('ls' stands for 'list' and the parameters '-al' specify that the list command should display a detailed list of files).

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 minimize the number of necessary keystrokes when typing commands. The learnability of command languages is generally very poor.

Advantages of Command Language

- 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 hard 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.

b. **Form fillin**

The form filling interaction style (also called "fill in the blanks") was aimed at a different set of users than command language, namely non-experts users. When form filling interfaces first appeared, the whole interface 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 filling are shown below.

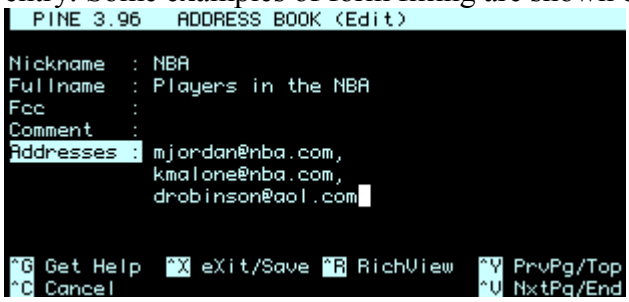

Figure 6: Classic Form fillin via a terminal

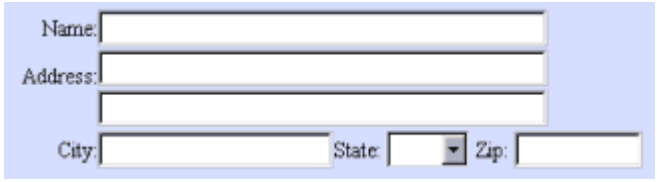

Figure 7:- More modern-day form fillin,

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

Advantages of Form Fillin

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 screen space.

Usually sets the scene for rigid formalisation of the business processes.

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

c. **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. 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 is shown below.

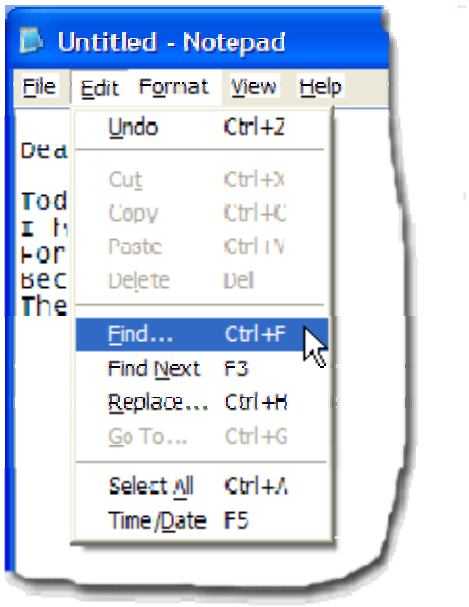

Figure 8: Contemporary menu selection (Notepad by Microsoft Cooperation)

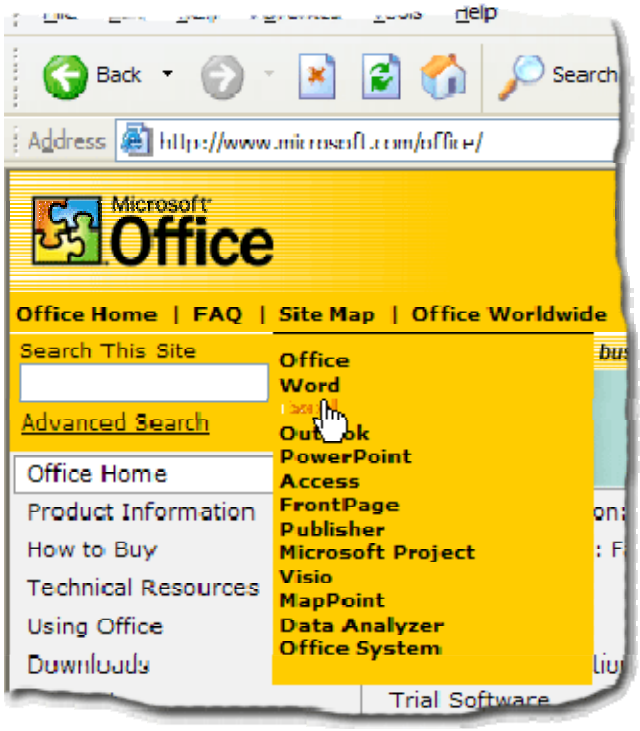

Figure 9: Menu selection in the form of a webpage (microsoft.com). Webpage in general can be said to be based on menu selection.

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

- 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.

d. **Direct manipulation**

Direct manipulation is a central theme in interface design and is treated in a separate encyclopedia entry (see this). Below, Direct manipulation is only briefly described.

The term direct manipulation was introduced by Ben Shneiderman in his keynote address at the NYU Symposium on User Interfaces and more explicitly in Shneiderman (1983) to describe a certain 'direct' software interaction style that can be traced back to Sutherlands sketchpad. 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 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.

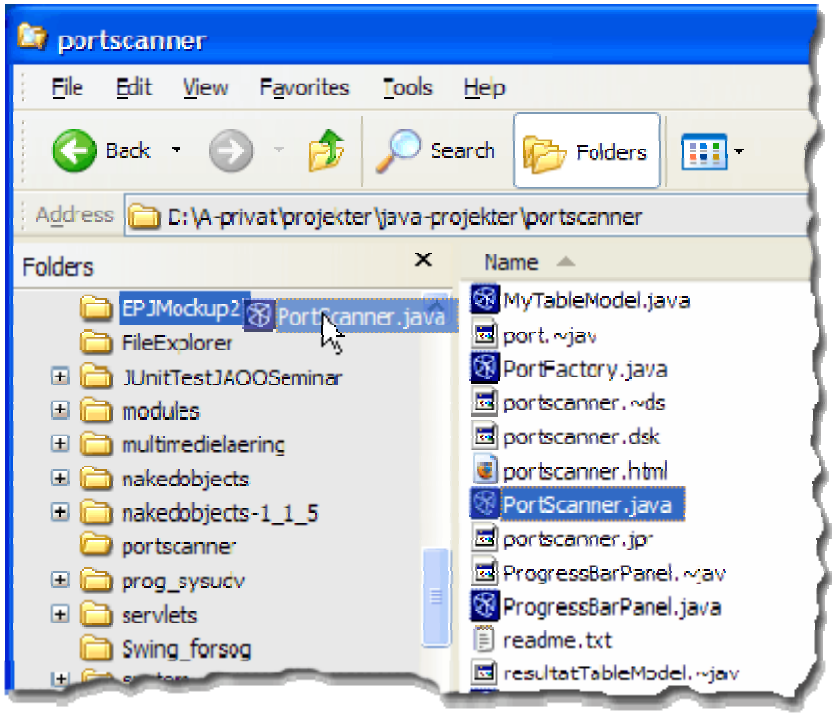

Figure 10: The text-book example of Direct Manipulation, the Windows File Explorer, where files are dragged and dropped.

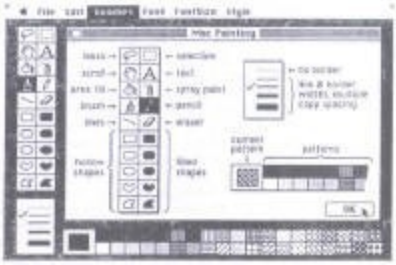

Figure 11: The screen of one of the earliest commercially available direct manipulation interfaces called MacPaint.

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.
- Metaphors can be misleading since the "the essence of metaphor is understanding and experiencing one kind of thing in terms of another", which, by definition, makes a metaphor different from what it represents or points to.
- 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 if type to use, and what colors will work best. For these questions as for other, more substantive design issues, intelligent borrowing should be your first approach. But that often leaves you 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 you will be able to consider the tradeoffs when there is a conflict between two principles or between a design principle and a borrowed technique.

a. **The Clustering Principle:** Organize 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 are 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 color, or a bounding box, or surrounding space ("white space"). The principle should also be applied within WIMP systems when you design 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 you're 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 organization for facts about the program. It's 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 organization.)

b. **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.

c. **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 lets you make a concentrated effort to design just a few attractive, workable screens, then modify those slightly for use in other parts of the application.
Be careful to use consistency in a meaningful way, however. Screens shouldn't 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 informational message.

d. **The Color As a Supplement Principle:** Don't rely on color to carry information; use it sparingly to emphasize information provided through other means.
Color is much easier to misuse than to use well. Different colors 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 colors: about 7 percent of all adults have some form of color vision deficiency.

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

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

e. **The Reduced Clutter Principle:** Don't 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 color, then the screen should be graphically attractive.

This is also a good principle to apply for issues that we haven't dealt with specifically. Type size and font, for example: the Reduced Clutter Principle would suggest that one or two type styles are sufficient. Don't try to distinguish each menu by its own font, or work with a large range of sizes. Users typically won't 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, The Visibility Reflects Usefulness Principle, and e.t.c.

6.0 TUTOR MARKED ASSIGNMENT

a. Explain any two Interaction Styles.
b. Write a short note on any of the Graphic Design Principle.

7.0 FURTHER READING AND OTHER RESOURCES

Crandall, B., Klein, G., and Hoffman, R. (2006). *Working minds: A practitioner's guide to cognitive task analysis*. MIT Press.

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

MODEL 3 – USER INTERFACE IMPLEMENTATION

UNIT 1 – PROTOTYPING

UNIT 2 - IMPLEMENTATION METHODS AND TOOLKITS

UNIT 3 - INPUT AND OUTPUT MODELS

UNIT 4 - MODEL VIEW-CONTROLLER (MVC)

UNIT 5 - LAYOUTS AND CONSTRAINTS

UNIT 1 PROTOTYPING

TableofContents

1.0 INTRODUCTION

This unit will give you a general understanding of prototyping. User interface prototyping will be discussed in details as well as the various tips and techniques of prototyping.

2.0 OBJECTIVES

By the end this unit, you should be able to:

- Explain Prototyping in detail.
- Describe the User interface Prototyping Process.
- Highlight various tips and techniques in Prototyping.
- Describe Interface-Flow Diagram.

3.0 MAIN CONTENT

3.1 PROTOTYPING

A prototype is a concrete, but partial implementation of a system design. Prototypes are used extensively in design and construction work; to demonstrate a concept (e.g. a prototype car), in early design, later design and as specification. It may be made of paper and cardboard or it may use a sophisticated software package.
Prototyping involves creating mock-ups representing the user interface of the final design. Prototypes serve as a common language with users, software engineers, and other stakeholders, offering a way for designers to explore design ideas and elicit feedback from users prior to committing to designs. Since prototyping helps flesh out requirements, prototypes may be used as a specification for developers. Prototyping is important in arriving at a well-designed user interface, and from many users' perspective the user interface is the software. Prototyping is very important for ensuring the most usable, accurate, and attractive design is found for the final product.

3.2 USER INTERFACE PROTOTYPING PROCESS

Prototyping is an iterative analysis technique in which users are actively involved in the mocking-up of screens and reports. The four major stages of prototyping are:-

a. **Determine the needs of your users** . The requirements of your users drive the development of your prototype as they define the business objects that your system must support. You can gather these requirements in interviews, in CRC (class responsibility collaborator) modeling sessions, in use-case modeling sessions, and in class diagramming sessions (Ambler 2001; Ambler, 1998a; Ambler, 1998b).

b. **Build the prototype**. Using a prototyping tool or high-level language you develop the screens and reports needed by your users. The best advice during this stage of the process is to not invest a lot of time in making the code "good" because chances are high that you may just scrap your coding efforts anyway after evaluating the prototype.

c. **Evaluate the prototype**. After a version of the prototype is built it needs to be evaluated. The main goal is that you need to verify that the prototype meets the needs of your users. I've always found that you need to address three basic issues during evaluation: What's good about the prototype, what's bad about the prototype, and what's missing from the prototype. After evaluating the prototype you'll find that you'll need to scrap parts, modify parts, and even add brand-new parts.

d. **Determine if you're finished yet**. You want to stop the prototyping process when you find the evaluation process is no longer generating any new requirements, or is generating a small number of not-so-important requirements.

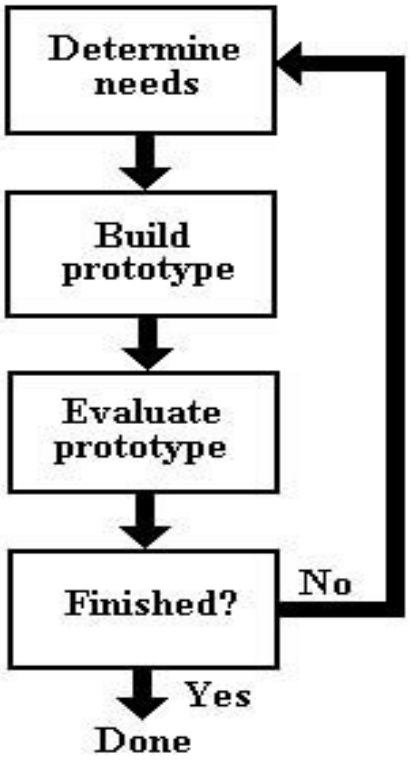

Figure 12: *The iterative steps of prototyping*

3.3 PROTOTYPING TIPS AND TECHNIQUES

I have covered the fundamentals of the prototyping process, so now I want to share with you several tips and techniques that you can use to create truly world-class prototypes.

a. **Look for real-world objects**. Good UIs allow users to work with the real-world objects they are used to. Therefore you should start by looking for these kinds of objects and identify how people interact with them.

b. **Work with the real users**. The best people to get involved in prototyping are the ones who will actually use the application when it's done. These are the people who have the most to gain from a successful implementation, and these are the ones who know their own needs best.

c. **Set a schedule and stick to it**. By putting a schedule in place for when you will get together with your users to evaluate the prototype, you set their expectations and you force yourself to actually get the work done. A win-win situation.

d. **Use a prototyping tool**. Invest the money in a prototyping tool that allows you to put screens together quickly. Because you probably won't want to keep the prototype code that you write, code that's written quickly is rarely worth keeping, you shouldn't be too concerned if your prototyping tool generates a different type of code than what you intend to develop in.

e. **Get the users to work with the prototype**. Just like you want to take a car for a test drive before you buy it your users should be able to take an application for a test drive before it is developed. Furthermore, by working with the prototype hands-on they will quickly be able to determine whether or not the system will meet their needs. A good approach is to ask them to work through some use-case scenarios using the prototype as if it is the real system.

f. **Understand the underlying business**. You need to understand the underlying business before you can develop a prototype that will support it. Perform interviews with key users, read internal documentation of how the business runs, and read documentation about how some of your competitors operate. The more you know about the business the more likely it is that you'll be able to build a prototype that supports it.

g. **There are different levels of prototype**. I like to successively develop three different types of prototypes of a system: A hand-drawn prototype that shows its basic/rough functionality, an electronic prototype that shows the screens but not the data that will be displayed on them, and then finally the screens with data. By starting out simple in the beginning I avoid investing a lot of time in work that will most likely be thrown away. By successively increasing the complexity of the prototype as it gets closer to the final solution, my users get a better and better idea of how the application will actually work, providing the opportunity to provide greater and greater insight into improving it.

h. **Do not spend a lot of time making the code good**. At the beginning of the prototyping process you will throw away a lot of your work as you learn more about the business. Therefore it doesn't make sense to invest a lot of effort in code that you probably aren't going to keep anyway.

3.4 INTERFACDE-FLOW DIAGRAMS

To your users, the user interface is the system. In Figure 3, we see an example of an interface-flow diagram for an order-entry system. The boxes represent user interface objects (screens, reports, or forms) and the arrows represent the possible flow between screens. For example, when you are on the main menu screen you can go to either the customer search screen or to the order-entry screen. Once you are on the order-entry screen you can go to the product search screen or to the customer order list. Interface-flow diagrams allow you to easily gain a high-level overview of the interface for your application.

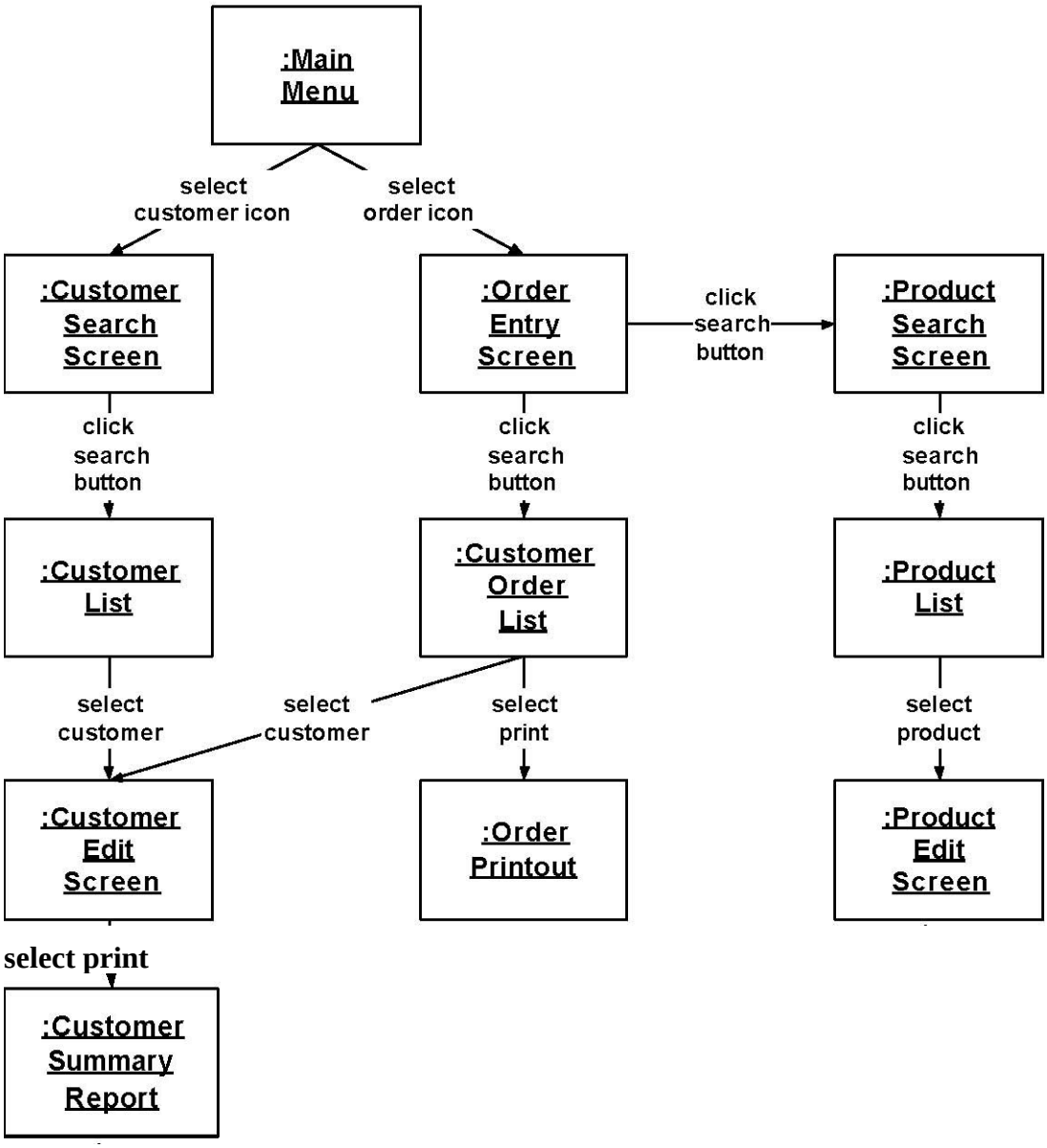

Figure 13: *An interface-flow diagram for an order-entry system*

Since interface-flow diagrams offer a high-level view of the interface of a system, you can quickly gain an understanding of how the system is expected to work. Furthermore, why cannot I get the same sort of list from the point of view of a product? In some cases it might be interesting to find out which orders include a certain product, especially when

the product is back-ordered or no longer available. Also, interface-flow diagrams can be used to determine if the user interface has been design consistently, for example in Figure 3 you see that to create the customer summary report and a printed order that you select the print command. It appears from the diagram that the user interface is consistent, at least with respect to printing.

4.0 CONCLUSION

In this unit, prototyping was discussed. Various tips and techniques of prototyping highlighted while the concept of Interface-Flow Diagram was also discussed.

5.0 SUMMARY

- The requirements of your users drive the development of your prototype.
- During evaluation ask: What's good about the prototype, what's bad about the prototype, and what's missing from the prototype.
- Stop the prototyping process when you find the evaluation process is generating few or no new requirements.
- Look for real-world objects and identify how users work with them.
- Work with the people who will use the application when it's done.
- Set a prototyping schedule and stick to it.
- Use a prototyping tool.
- Get the users to work with the prototype, to take it for a test drive.
- Understand the underlying business.
- Do not invest a lot of time in something that you'll probably throw away.
- Document interface objects once they have stabilized.
- Develop an interface-flow diagram for your prototype.
- For each interface object that makes up a prototype, document
  Its purpose and usage
  An indication of the other interface objects it interacts with
  The purpose and usage of each of its components

6.0 TUTOR MARKED ASSIGNMENT

a. Explain the different prototyping techniques.

b. scribe the Interface-Flow Diagram.

7.0 FURTHER READING AND OTHER RESOURCES

Ambler, S.W. & Constantine, L.L. (2000a). *The Unified Process Inception Phase*. Gilroy, CA: CMP Books. http://www.ambysoft.com/inceptionPhase.html.

Ambler, S.W. & Constantine, L.L. (2000b). *The Unified Process Elaboration Phase*. Gilroy, CA: CMP

Ambler, S.W. & Constantine, L.L. (2000c). *The Unified Process Construction Phase*. Gilroy, CA: CMP

Ambler, S.W. (2001). *The Object Primer 2nd Edition: The Application Developer's Guide to Object Orientation*. New York: Cambridge University Press.

UNIT 2 PROTOTYPING METHODS AND TOOLS

TableofContents

1.0 INTRODUCTION

You will be introduced to various prototyping methods and tools in this unit. Fidelity of prototypes will be discussed with emphasis on Low-Fidelity, Medium-Fidelity and High-Fidelity Prototypes. User Interface modes and modalities are also mentioned.

2.0 OBJECTIVES

By the end this unit, you should be able to:

- Explain various prototype terms
- Describe the user interface modes and modalities.
- Explain the term widget.

3.0 MAIN CONTENT

3.1 FIDELITY OF PROTOTYPES

This refers to how accurately the prototypes resemble the final design in terms of visual appearance, interaction style, and level of detail.

The three main fidelity types are:-

- Low-Fidelity prototypes
- Medium-Fidelity prototypes
- High-Fidelity prototype

3.2 LOW-FIDELITY PROTOTYPES

Low-fidelity prototypes, also know as Lo-fi prototypes, depict rough conceptual interface design ideas. Low-fidelity prototypes consist of little details of the actual interface. They are traditionally paper-based prototypes, making them quick, easy, and low-cost to create and modify. Low-fidelity prototypes are sketches of static screens, presented either separately or in a series to tell a specific story, which is called storyboarding. These prototypes convey the general look and feel of the user interface as well as basic functionality. Low-fidelity prototypes are particularly well suited for understanding screen layout issues but not for navigation and interaction issues. The purpose of low-fidelity prototypes is to try out alternative design ideas while seeking frequent feedback from users and other stakeholders. Low-fidelity prototypes are best used early in the design process when trying to understand basic user requirements and expectations.

3.2.1 TECHNIQUES USED IN LOW-FIDELITY PROTOTYPES

a. SKETCHING: Sketching is one of the most common techniques used in creating low-fidelity prototypes. It is a natural and low effort technique that allows for abstract ideas to be rapidly translated from a designer's conception onto a more permanent medium. Sketching is beneficial to the design process because it encourages thinking and, ultimately, creativity. Sketches are also important to design because they are intentionally vague and informal, which allows for details to be later worked out without hindering the creative flow of the moment. This technique also encourages contributions from users and other stakeholders since it is in a visual form that everyone is familiar with.

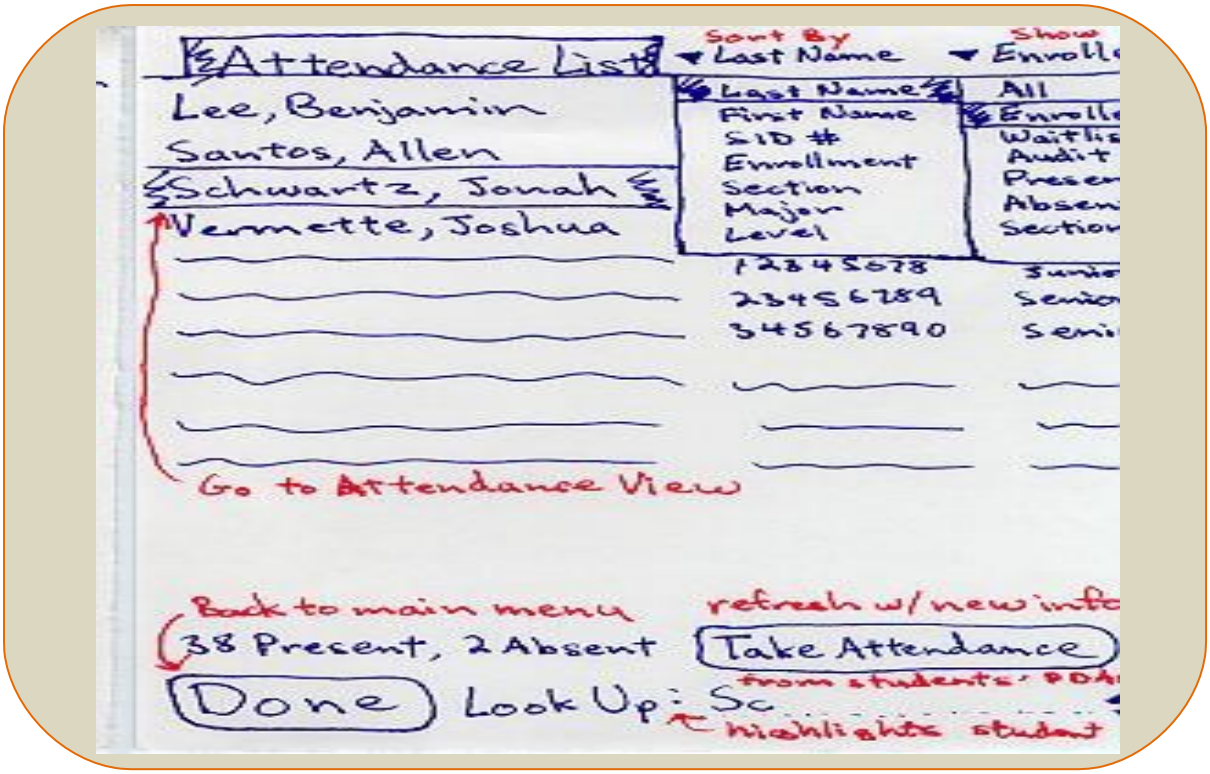

Figure 14:- *Example of Sketching I*

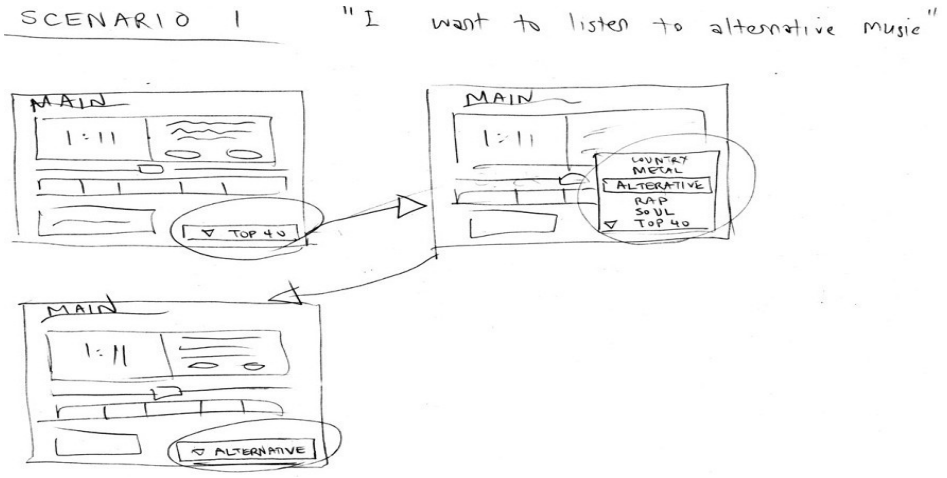

Figure 14:- *Example of Sketching II*

b. **STORYBOARDS:** This Lo-fi technique is similar to that of Sketching but differ in the sense that it is mainly obtained from films and animation. Storyboards give a script of important events by leaving out the details and concentrating on the important interactions.

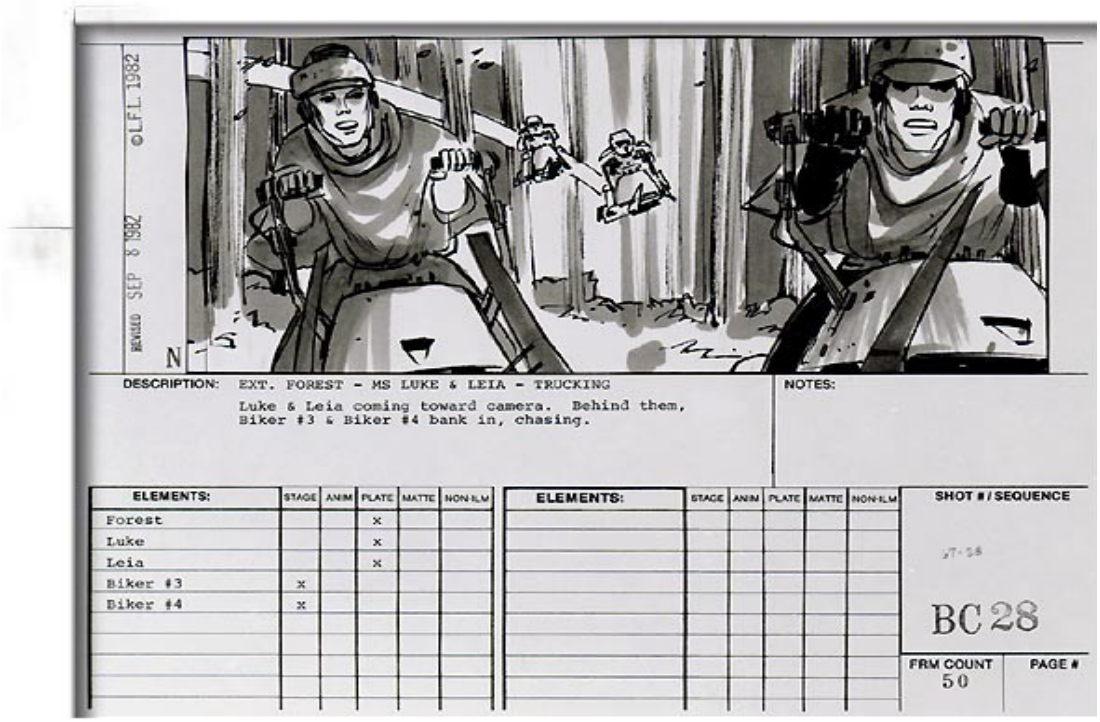

Figure 15:- *An example of Storyboard I*

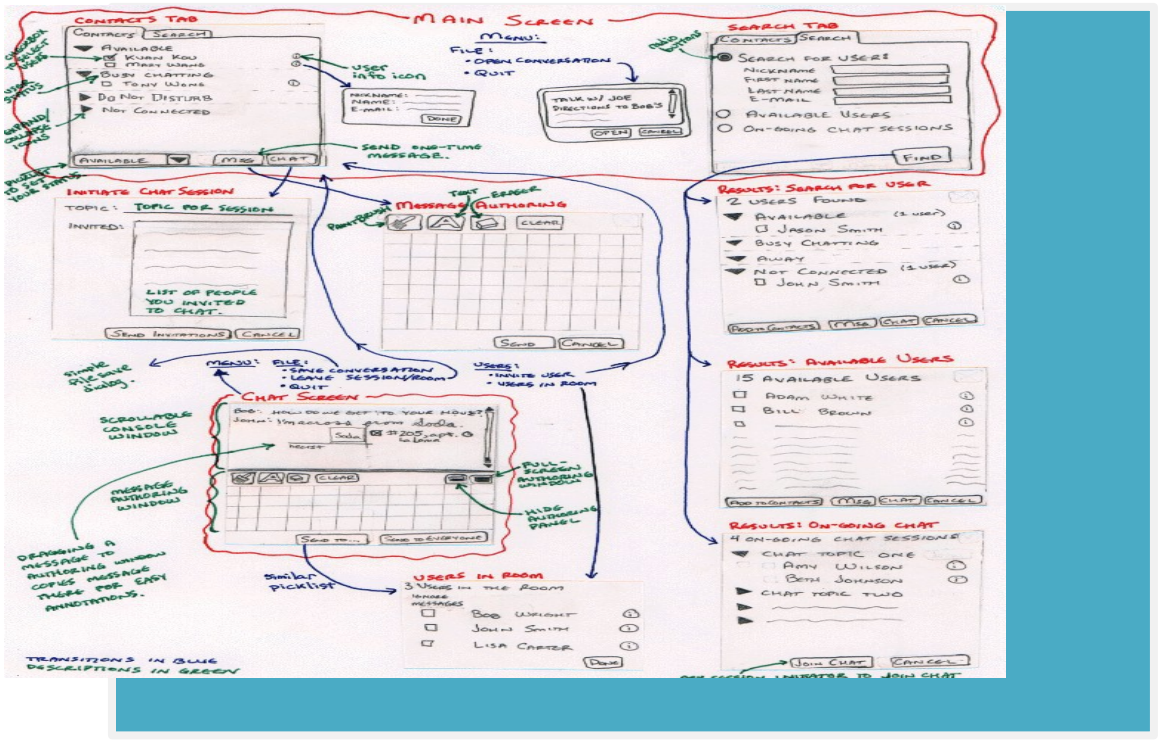

Figure 16:- *An example of Storyboard II*

c. **PICTIVE:** The other commonly used low-fidelity prototyping technique is the PICTIVE technique. PICTIVE also called Plastic Interface for Collaborative Technology Initiatives through Video Exploration, is a technique for creating and modifying prototypes in real-time with users. The PICTIVE technique involves using standard office supplies such as sticky notes, labels, and plastic overlays as well as paper copies of pre-built interface components such as buttons, icons, and menus. Materials are transformed through cutting and labeling to represent desired interface elements. Multiple layers of these elements are attached to the paper prototype, as needed, to demonstrate interaction to the users. PICTIVE is a flexible technique that encourages active user participation. As the name suggests, PICTIVE prototyping sessions with the users may be videotaped and later analyzed by designers to see how the prototypes evolved and how users responded to different designs.

3.2.2 ADVANTAGES OF LOW-FIDELITY PROTOTYPES

- Takes only a few hours
- No expensive equipment needed.
- Fast iterations
- It can test multiple alternatives and the number of iterations is tied to final quality
- Almost all iterations can be faked. Example of this is gotten from the film ”Wizard of OZ” and “The man behind the curtain”
- It can evolve into implementation.

- Much more important for features that are hard to implement such as speech and handwriting recognition.

3.3 MEDIUM-FIDELITY PROTOTYPES

Medium-fidelity prototypes lie on the continuum between low- and high-fidelity prototypes, thus sharing some of the advantages and disadvantages of the other two fidelities. Medium-fidelity prototypes are refined versions of the low-fidelity prototypes and are created on computer. They are best used after low-fidelity prototyping once only a small number of alternative designs remain under consideration and require further refinement. They resemble the end product more than low-fidelity prototypes and require less effort than high-fidelity prototypes. Medium-fidelity prototypes are commonly created using multimedia design tools, interface builders, or certain scripting languages.

3.4 HIGH-FIDELITY PROTOTYPES

High-fidelity prototypes also known as Hi-fi prototypes allow users to interact with the prototypes as though they are the end product. High-fidelity prototypes strongly represent the final product in terms of visual appearance, interactivity, and navigation. As well, high-fidelity prototypes usually have some level of functionality implemented and may link to some sample data. High-fidelity prototypes are computer-based prototypes that are often developed using interface builders or scripting languages to speed up the process. High-fidelity prototypes are particularly useful for performing user evaluations as well as for serving as a specification for developers and as a tool for marketing and stakeholder buy-in. On the negative side, these prototypes are time-consuming and costly to develop. As such, high-fidelity prototypes are best used near the end of the design phase once the user interface requirements have been fully understood and a single design has been agreed upon.

3.5 USER INTERFACE PROTOTYPING TOOLS

Today, the majority of applications are developed using some type of user interface prototyping tool such as an **interface builder** or **a multimedia design tool**. However, there are several limitations in using these types of tools that hinder the design process. In attempts to overcome some of these limitations, researchers in the academic community have been investigating informal prototyping tools such as **SILK, DENIM**, and **Freeform**. While useful, these tools are not without drawbacks.

a. Interface Builders

Interface builders are tools for creating and laying out user interfaces by allowing interface components to be dragged and placed into position on the desired window or dialog box. Some commercial examples include Microsoft Visual Basic, Java's NetBeansTM, Borland DelphiTM, and MetrowerksTM CodeWarriorTM. While interface

builders are primarily intended for final product implementation, they are useful for medium- and high-fidelity prototyping.

Interface builders are commonly used for the following reasons:

They visually represent visual concepts such as layout,
They speed up implementation by auto-generating certain code,
They are generally easy to use even for non-programmers.

On the other hand, interface builders are restrictive in terms of what designs designers can build and the order in which designers have to build it. Also, interface builders require significant time and effort to create a prototype. Thus they are not suitable for early stages of prototyping when many alternate and ill-defined design concepts need to be explored.

b. **Multimedia Design Tools**

Multimedia design tools are often used in designing user interfaces, not because they are particularly well suited for software interfaces, but rather because of the lack of better prototyping-specific tools. Multimedia tools are useful in creating and demonstrating storyboards in medium-fidelity prototyping. Specifically, multimedia tools allow for creation of images that can represent user interface screens and components. They also allow for playing out transitions from one screen to the next that can convey the general picture of a user navigating through the interface screens. On the negative side, the interactivity supported by multimedia design tools is very limited, usually to only basic mouse clicks, and so is support for creating functionality and tying in data. Examples of commonly used commercial multimedia design tools are Macromedia Director and Flash. Apple HyperCard is another commercial tool that has been widely used in the past. There have also been several multimedia tools to come out of the user interface research community including DEMAIS and Anecdote.

c. **SILK**

SILK was one of the first tools to support informal sketching of user interfaces. The purpose of SILK is to preserve the benefits of paper-based sketching while taking advantage of computerized design tools' features. The main features of SILK include support for stylus-based freehand sketching, annotation layers that allow for notes to be created and associated with specific user interface elements, and a run mode to show screen transitions. Also, SILK attempts to provide support for transitioning to higher-fidelity prototyping through automatic interface component recognition and transformation to real components; however, this feature is not suitable for low-fidelity prototyping and restricts the designer to existing toolkit components and interactions.

d. **DENIM**

DENIM, an extension of SILK, is a tool aimed at supporting the early stages of web design through informal sketching. While DENIM is intended for website design, many of the features and concepts are applicable to the design of most graphical user interfaces.

DENIM provides the following design features:

These features make DENIM appropriate for low- fidelity and medium-fidelity prototyping. On the other hand, DENIM provides little support for transitioning to high-fidelity prototyping. As shown in the figure below.

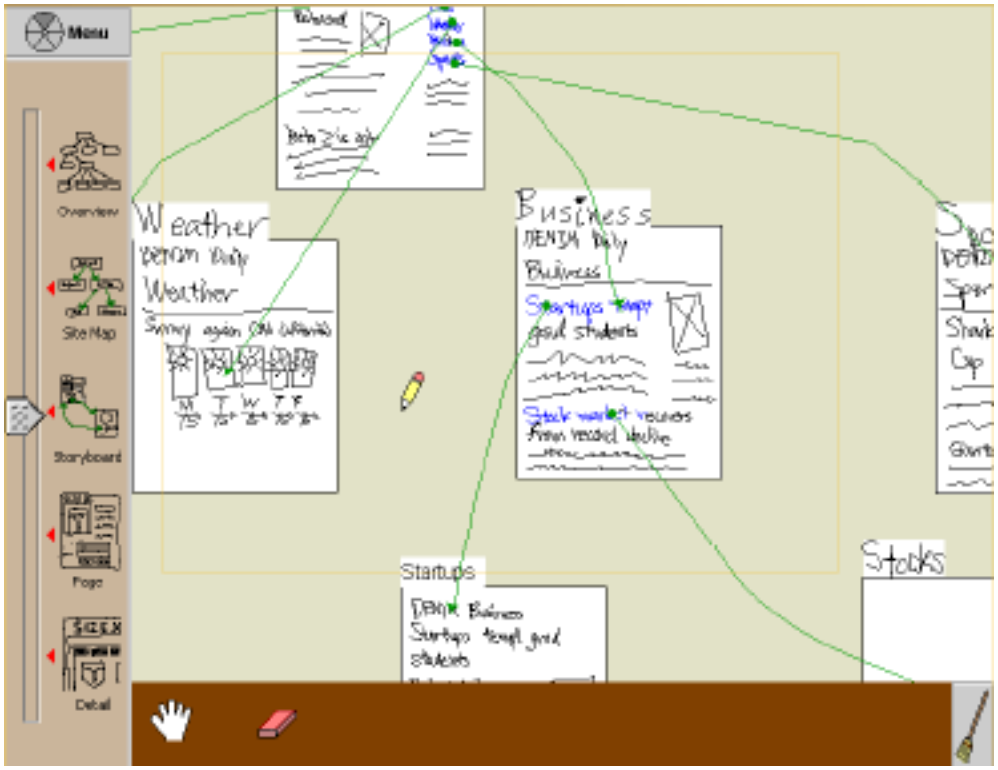

Figure 17: *Showing an example of DENIM prototyping screen*

e. Freeform

Freeform is another tool that supports sketching of user interfaces. It also aims at support high-fidelity prototyping. Specifically, Freeform is a Visual Basic add-in that translates the recognized sketched interface components and text into VB code. However, this feature restricts the interface designer to simple forms-based interfaces that use standard Visual Basic components. Also, the interfaces generated from sketches are not very visually appealing. Freeform is intended for use on large displays; however, there are no

unique features in Freeform that make it better suited for use on large displays versus traditional desktop displays as shown in the figure below:

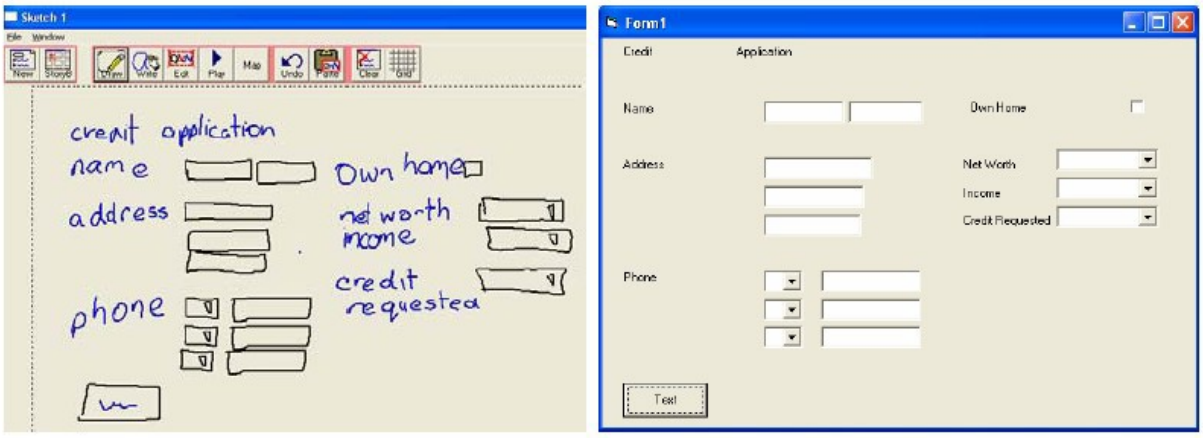

Figure 18: *Showing an example of Freeform prototyping screen*

3.5.1 ANALYSIS OF PROTOTYPING TOOLS

Each type of user interface prototyping tool discussed above offers some unique features or advantages to designers. However, most tools are designed for very specific purposes and have very little or no support for transitioning between each of the different fidelities. None of the tools support mixing of all three fidelities within a single prototype. Also, none of the tools are specifically designed for collaborative use on a large display; however, simply using tools such as DENIM on a large display may offer some benefits over use on a traditional desktop display.

3.6 WIDGETS

Widgets are reusable user interface components. As stated in Module 1, they are also called controls, interactors, gizmos, or gadgets. Some examples of widgets have been given in Unit 5 of Module 1. **Widgets** are user interface toolkits. Widgets are a success story for user interface software, and for object-oriented programming in general. Many GUI applications derive substantial reuse from widgets in a toolkit.

ADVANTAGES OF USING WIDGETS

Reuse of development efforts
Coding, testing, debugging, maintenance
Iteration and evaluation

DISADVANTAGES

Constrain designer's thinking

Encourage menu & forms style, rather than richer direct manipulation style
May be used inappropriately

Widget reuse is beneficial in two ways, actually. First are the conventional software engineering benefits of reusing code, like shorter development time and greater reliability. A widget encapsulates a lot of effort that somebody else has already put in.
Second are usability benefits. Widget reuse increases consistency among the applications on a platform. It also (potentially) represents *usability* effort that its designers have put into it. A scrollbar's affordances and behavior have been carefully designed, and hopefully evaluated. By reusing the scrollbar widget, you don't have to do that work yourself.

One problem with widgets is that they constrain your thinking. If you try to design an interface using a GUI builder – with a palette limited to standard widgets – you may produce a clunkier, more complex interface than you would if you sat down with paper and pencil and allowed yourself to think freely. A related problem is that most widget sets consist mostly of form-style widgets: text fields, labels, checkboxes – which leads a designer to think in terms of menu/form style interfaces. There are few widgets that support direct visual representations of application objects, because those representations are so application-dependent. So if you think too much in terms of widgets, you may miss the possibilities of direct manipulation.

Finally, widgets can be abused, applied to UI problems for which they are not suited. An example is when a scrollbar is used for selection, rather than scrolling.

Widgets generally combine a view and a controller into a single tightly-coupled object. For the widget's model, however, there are two common approaches. One is to fuse the model into the widget as well, making it a little MVC complex. With this embedded model approach, application data must be copied into the widget to initialize it. When the user interacts with the widget, the user's changes or selections must be copied back out.
The other alternative is to leave the model separate from the widget, with a well-defined interface that the application can implement. Embedded models are usually easier for the developer to understand and use for simple interfaces, but suffer from serious scaling problems. For example, suppose you want to use a table widget to show the contents of a database. If the table widget had an embedded model, you would have to fetch the entire database and load it into the table widget, which may be prohibitively slow and memory-intensive. Furthermore, most of this is wasted work, since the user can only see a few rows of the table at a time. With a well-designed linked model, the table widget will only request as much of the data as it needs to display.
The linked model idea is also called **data binding**.

Generally, every user interface consists of the following:-
Components (View hierarchy)
Stroke drawing
Pixel model

Input handling
Widgets

Every modern GUI toolkit provides these pieces in some form. Microsoft Windows, for example, has widgets (e.g., buttons, menus, text boxes), a view hierarchy (consisting of *windows* and *child windows*), a stroke drawing package (GDI), pixel representations (called bitmaps), and input handling (messages sent to a *window procedure*).

User interface toolkits are often built on top of other toolkits, sometimes for portability or compatibility across platforms, and sometimes to add more powerful features, like a richer stroke drawing model or different widgets.
X Windows demonstrates this layering technique. The view hierarchy, stroke drawing, and input handling are provided by a low-level toolkit called XLib. But XLib does not provide widgets, so several toolkits are layered on top of XLib to add that functionality: Athena widgets and Motif, among others. More recent X-based toolkits (GTK+ and Qt) not only add widgets to XLib, but also hide XLib's view hierarchy, stroke drawing, and input handling with newer, more powerful models, although these models are implemented internally by calls to XLib.

Here is what the layering looks like for some common Java user interface toolkits. AWT (Abstract Window Toolkit, usually pronounced like "ought") was the first Java toolkit. Although its widget set is rarely used today, AWT continues to provide drawing and input handling to more recent Java toolkits. Swing is the second-generation Java toolkit, which appeared in the Java API starting in Java 1.2.

Swing adds a new view hierarchy (JComponent) derived from AWT's view hierarchy (Component and Container). It also replaces AWT's widget set with new widgets that use the new view hierarchy. subArctic was a research toolkit developed at Georgia Tech. Like Swing, subArctic relies on AWT for drawing and input handling, but provides its own widgets and views.

Not shown in the picture is SWT, IBM's Standard Widget Toolkit. (Usually pronounced "swit".

Confusingly, the W in SWT means something different from the W in AWT.) Like AWT, SWT is implemented directly on top of the native toolkits. It provides different interfaces for widgets, views, drawing, and input handling.

Cross-platform toolkits face a special issue: should the native widgets of each platform be reused by the toolkit? One reason to do so is to preserve consistency with other applications on the same platform, so that applications written for the cross-platform toolkit look and feel like native applications. This is what we've been calling external consistency.

Another problem is that native widgets may not exist for all the widgets the cross-platform toolkit wants to provide. AWT throws up its hands at this problem, providing only the widgets that occur on every platform AWT runs on: e.g., buttons, menus, list boxes, text boxes.

One reason NOT to reuse the native widgets is so that the application looks and behaves consistently with itself across platforms – a variant of internal consistency, if you consider all the instantiations of an application on various platforms as being part of the same system. Cross-platform consistency makes it easier to deliver a well-designed, usable application on all platforms – easier to write documentation and training materials, for example. Java Swing provides this by reimplementing the widget set using its default ("Metal") look and feel. This essentially creates a Java "platform", independent of and distinct from the native platform.

3.6.1 **PICCOLO TOOLKIT**

Piccolo is a novel UI toolkit developed at University of Maryland. Piccolo is specially designed for building **zoomable** interfaces, which use smooth animated panning and zooming around a large space. We can look at Piccolo in terms of the various aspects we have discussed in this unit.

Layering: First, Piccolo is a layered toolkit: it runs on top of Java Swing. It also runs on top of .NET, making it a cross-platform toolkit. Piccolo ignores the platform widgets entirely, making no attempt to reimplement or reuse them. (An earlier version of Piccolo, called Jazz, could reuse Swing widgets.)

Components: Piccolo has a view hierarchy consisting of PNode objects. The hierarchy is not merely a tree, but in fact a graph: you can install camera objects in the hierarchy which act as viewports to other parts of the hierarchy, so a component may be seen in more than one place on the screen. Another distinction between Piccolo and other toolkits is that every component has an arbitrary transform relative to its parent's coordinate system – not just translation (which all toolkits provide), but also rotation and scaling. Furthermore, in Piccolo, parents do not clip their children by default. If you want this behavior, you have to request it by inserting a special clipping object (a component) into the hierarchy. As a result, components in Piccolo have two bounding boxes – the bounding box of the node itself (getBounds()), and the bounding box of the node's entire subtree (getFullBounds()).

Strokes: Piccolo uses the Swing Graphics package, augmented with a little information such as the camera and transformations in use.

Pixels: Piccolo uses Swing images for direct pixel representations.
Input: Piccolo has the usual mouse and keyboard input (encapsulated in a single event-handling interface called BasicInput), plus generic controllers for common operations

like dragging, panning, and zooming. By default, panning and zooming is attached to any camera you create: dragging with the left mouse button moves the camera view around, and dragging with the right mouse button zooms in and out.

Widgets: the widget set for Piccolo is fairly small by comparison with toolkits like Swing and .NET, probably because Piccolo is a research project with limited resources. It's worth noting, however, that Piccolo provides reusable components for shapes (e.g. lines, rectangles, ellipses, etc), which in other toolkits would require revering to the stroke model.

3.7 THE WIZARD OF OZ PROTOTYPE

The Wizard of Oz is a 1939 American musical-fantasy film mainly directed by Victor Fleming and based on the 1900 children's novel *The Wonderful Wizard of Oz* by L. Frank Baum.

A **Wizard of Oz prototype** uses a human in the backend, but the frontend is an actual computer system instead of a paper mockup. The term Wizard of Oz comes from the movie of the same name, in which the wizard was a man hiding behind a curtain, controlling a massive and impressive display.
In a Wizard of Oz prototype, the "wizard" is usually but not always hidden from the user. Wizard of Oz prototypes are often used to simulate future technology that is not available yet, particularly artificial intelligence. A famous example was the listening typewriter. This study sought to compare the effectiveness and acceptability of isolated-word speech recognition, which was the state of the art in the early 80's, with continuous speech recognition, which wasn't possible yet. The interface was a speech-operated text editor. Users looked at a screen and dictated into a microphone, which was connected to a typist (the wizard) in another room. Using a keyboard, the wizard operated the editor showing on the user's screen.

The wizard's skill was critical in this experiment. She could type 80 wpm, she practiced with the simulation for several weeks (with some iterative design on the simulator to improve her interface), and she was careful to type *exactly* what the user said, even exclamations and parenthetical comments or asides. The computer helped make her responses a more accurate simulation of computer speech recognition. It looked up every word she typed in a fixed dictionary, and any words that were not present were replaced with X's, to simulate misrecognition. Furthermore, in order to simulate the computer's ignorance of context, homophones were replaced with the most common spelling, so "done" replaced "dun", and "in" replaced "inn". The result was an extremely effective illusion. Most users were surprised when told (midway through the experiment) that a human was listening to them and doing the typing.
Thinking and acting mechanically is harder for a wizard than it is for a paper prototype simulator, because the tasks for which Wizard of Oz testing is used tend to be more "intelligent". It helps if the wizard is personally familiar with the capabilities of similar

interfaces, so that a realistic simulation can be provided. It also helps if the wizard's interface can intentionally dumb down the responses, as was done in the Gould study.
A key challenge in designing a Wizard of Oz prototype is that you actually have two interfaces to worry about: the user's interface, which is presumably the one you're testing, and the wizard's.

4.0 CONCLUSION

In this unit, you have been introduced to various prototyping methods and tools. Fidelity of prototypes like Low-Fidelity, Medium-Fidelity and High-Fidelity Prototypes was also discussed in detail. User Interface modes and modalities was also introduced along with a discussion on widgets.

5.0 SUMMARY

- The fidelity of prototypes refers to how accurately the prototypes resemble the final design in terms of visual appearance, interaction style, and level of detail.

- Low-fidelity prototypes, also know as Lo-fi prototypes, depict rough conceptual interface design ideas. Low-fidelity prototypes consist of little details of the actual interface.

- High-fidelity prototypes also known as Hi-fi prototypes allow users to interact with the prototypes as though they are the end product. High-fidelity prototypes strongly represent the final product in terms of visual appearance, interactivity, and navigation.

- User interface prototyping tools include **interface builder** or **multimedia design tools**. However, there are several limitations in using these types of tools because they hinder the design process.

- Interface builders are tools for creating and laying out user interfaces by allowing interface components to be dragged and placed into position on the desired window or dialog box.

- Multimedia design tools are often used in designing user interfaces, not because they are particularly well suited for software interfaces, but rather because of the lack of better prototyping-specific tools.

- Widgets are reusable user interface components. They are also called controls, interactors, gizmos, or gadgets.

6.0 TUTOR MARKED ASSIGNMENT.

a. List and explain the different types of Information Systems.
b. Write a short note on the areas of application of IS

7.0 FURTHER READING AND OTHER RESOURCES

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

Ambler, S.W. (2001). *The Object Primer 2nd Edition: The Application Developer's Guide to Object Orientation*. New York: Cambridge University Press.

Piccolo home page: http://www.cs.umd.edu/hcil/piccolo/

UNIT 3 INPUT AND OUTPUT MODELS

TableofContents

1.0 INTRODUCTION

This unit looks at mechanics of implementing user interfaces, by looking at **Input** and **Output model** in detail. Different kinds of both input and output events will also be examined in greater detail.

2.0 OBJECTIVES

By the end this unit, you should be able to:

- Explain the Input model
- Explain the Output model

3.0 MAIN CONTENT

3.1 INPUT MODEL

Virtually all GUI toolkits use event handling for input. Why? Recall, when you first learned to program, you probably wrote user interfaces that printed a prompt and then waited for the user to enter a response. After the user gave their answer, you produced another prompt and waited for another response. Command-line interfaces (e.g. the Unix shell) and menu-driven interfaces (e.g., Pine) have interfaces that behave this way. In this user interface style, the system has complete control over the dialogue – the order in which inputs and outputs will occur.

Interactive graphical user interfaces can not be written this way – at least, not if they care about giving the user control and freedom. One of the biggest advantages of GUIs is that a user can click anywhere on the window, invoking any command that's available at the moment, interacting with any view that's visible. In a GUI, the balance of power in the dialogue swings strongly over to the user's side.

As a result, GUI programs can not be written in a synchronous, prompt-response style. A component can't simply take over the entire input channel to wait for the user to interact with it, because the user's next input may be directed to some other component on the screen instead. So GUI programs are designed to handle input asynchronously, receiving it as events.

3.1.1 KINDS OF INPUT EVENTS

There are two major categories of input events: raw and translated.
A raw event comes right from the device driver. Mouse movements, mouse button down and up, and keyboard key down and up are the raw events seen in almost every capable GUI system. A toolkit that does not provide separate events for down and up is poorly designed, and makes it difficult or impossible to implement input effects like drag-and-drop or video game controls. For many GUI components, the raw events are too low-level, and must be translated into higher-level events. For example, a mouse button press and release is translated into a mouse click event (assuming the mouse didn't move much between press and release – if it did, these events would be translated into a drag rather than a click). Key down and up events are translated into character typed events, which take modifiers into account to produce an ASCII character rather than a keyboard key.

If you hold a key down, multiple character typed events may be generated by an autorepeat mechanism. Mouse movements and clicks also translate into keyboard focus changes. When a mouse movement causes the mouse to enter or leave a component's bounding box, entry and exit events are generated, so that the component can give feedback – e.g., visually highlighting a button, or changing the mouse cursor to a text I-bar or a pointing finger.

3.1.2 PROPERTIES OF AN INPUT EVENT

Input events have some or all of these properties. On most systems, all events include the modifier key state, since some mouse gestures are modified by Shift, Control, and Alt. Some systems include the mouse position and button state on all events; some put it only on mouse-related events.
The timestamp indicates when the input was received, so that the system can time features like autorepeat and double-clicking. It is essential that the timestamp be a property of the event, rather than just read from the clock when the event is handled. Events are stored in a queue, and an event may languish in the queue for an uncertain interval until the application actually handles it.

User input tends to be bursty – many seconds may go by while the user is thinking, followed by a flurry of events. The event queue provides a buffer between the user and the application, so that the application doesn't have to keep up with each event in a burst.

Recall that perceptual fusion means that the system has 100 milliseconds in which to respond.

Edge events (button down and up events) are all kept in the queue unchanged. But multiple events that describe a continuing state – in particular, mouse movements – may be **coalesced** into a single event with the latest known state. Most of the time, this is the right thing to do. For example, if you're dragging a big object across the screen, and the application can't repaint the object fast enough to keep up with your mouse, you don't want the mouse movements to accumulate in the queue, because then the object will lag behind the mouse pointer, diligently (and foolishly) following the same path your mouse did.

Sometimes, however, coalescing hurts. If you're sketching a freehand stroke with the mouse, and some of the mouse movements are coalesced, then the stroke may have straight segments at places where there should be a smooth curve. If application delays are bursty, then coalescing may hurt even if your application can usually keep up with the mouse.

The event loop reads events from the queue and dispatches them to the appropriate components in the view hierarchy. On some systems (notably Microsoft Windows), the event loop also includes a call to a function that translates raw events into higher-level ones. On most systems, however, translation happens when the raw event is added to the queue, not when it is removed.

Every GUI program has an event loop in it somewhere. Some toolkits require the application programmer to write this loop (e.g., Win32); other toolkits have it built-in (e.g., Java Swing). Unfortunately, Java's event loop is written as essentially an infinite loop, so the event loop thread never cleanly exits. As a result, the normal clean way to end a Java program – waiting until all the threads are finished – doesn't work for GUI programs. The only way to end a Java GUI program is System.exit(). This despite the fact that Java best practices say *not* to use System.exit(), because it doesn't guarantee to garbage collect and run finalizers.

Swing lets you configure your application's main JFrame with EXIT_ON_CLOSE behavior, but this is just a shortcut for calling System.exit().

Event dispatch chooses a component to receive the event. Key events are sent to the component with the keyboard focus, and mouse events are generally sent to the component under the mouse. An exception is **mouse capture**, which allows any component to grab all mouse events (essentially a mouse analogue for keyboard focus). Mouse capture is done automatically by Java when you hold down the mouse button to drag the mouse. Other UI toolkits give the programmer direct access to mouse capture – in the Windows API, for example, you'll find a SetMouseCapture function.
If the target component declines to handle the event, the event propagates up the view hierarchy until some component handles it. If an event bubbles up to the top without being handled, it is ignored.

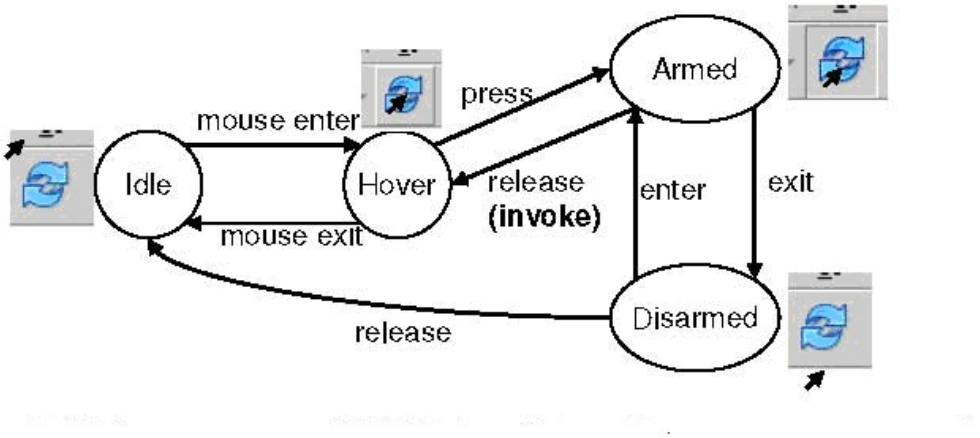

Figure 19: *The model for designing a controller (From MIT Open Courseware)*

Now let's look at how components that handle input are typically structured. A controller in a direct manipulation interface is a **finite state machine**. Here's an example of the state machine for a push button's controller. **Idle** is the normal state of the button when the user isn't directing any input at it. The button enters the **Hover** state when the mouse enters it. It might display some feedback to reinforce that it affords clickability. If the mouse button is then pressed, the button enters the **Armed** state, to indicate that it's being pushed down. The user can cancel the button press by moving the mouse away from it, which goes into the **Disarmed** state. Or the user can release the mouse button while still inside the component, which invokes the button's action and returns to the **Hover** state. Transitions between states occur when a certain input event arrives, or sometimes when a timer times out. Each state may need different feedback displayed by the view. Changes to the model or the view occur on transitions, not states: e.g., a push button is actually invoked by the release of the mouse button.

An alternative approach to handling low-level input events is the **interactor** model, introduced by the Garnet and Amulet research toolkits from CMU. Interactors are generic, reusable controllers, which encapsulate a finite state machine for a common task. They're mainly useful for the component model, in which the graphic output is represented by objects that the interactors can manipulate.

3.2 OUTPUT MODEL

There are basically three ways to represent the output of a graphical user interface.

Components is the same as the view hierarchy we discussed last week. Parts of the display are represented by view objects arranged in a spatial hierarchy, with automatic redraw propagating down the hierarchy. There have been many names for this idea over the years; the GUI community has not managed to settle on a single preferred term.

Strokes draws output by making calls to high-level drawing primitives, like drawLine, rawRectangle, drawArc, and drawText.

Pixels regards the screen as an array of pixels and deals with the pixels directly.
All three output models appear in virtually every modern GUI application. The component model always appears at the very top level, for windows, and often for components within the windows as ell. At some point, we reach the leaves of the view hierarchy, and the leaf views draw themselves ith stroke calls. A graphics package then converts those strokes into pixels displayed on the screen.

For performance reasons, a component may short-circuit the stroke package and draw pixels on the screen directly. On Windows, for example, video players do this using the DirectX interface to have direct control over a particular screen rectangle.

What model do each of the following representations use? HTML (component); Postscript laser printer (stroke input, pixel output); plotter (stroke input and output); PDF (stroke); LCD panel (pixel).

Since every application uses all three models, the design question becomes: at which points in your application do you want to step down into a lower-level output model? Here's an example. Suppose you want to build a view that displays a graph of nodes and edges.

One approach would represent each node and edge in the graph by a component. Each node in turn might have two components, a rectangle and a label. Eventually, you'll get down to primitive components available in your GUI toolkit. Most GUI toolkits provide a label component; most don't provide a primitive circle component. One notable exception is Amulet, which has component equivalents for all the common drawing primitives. This would be a **pure component model**, at least from your application's point of view – stroke output and pixel output would still happen, but inside primitive components that you took from the library.

Alternatively, the top-level window might have *no* subcomponents. Instead, it would draw the entire graph by a sequence of stroke calls: drawRectangle for the node outlines, drawText for the labels, drawLine for the edges. This would be a **pure stroke model**.
Finally, your graph view might bypass stroke drawing and set pixels in the window directly. The text labels might be assembled by copying character images to the screen.

This **pure pixel model** is rarely used nowadays, because it's the most work for the programmer, but it used to be the only way to program graphics.

Hybrid models for the graph view are certainly possible, in which some parts of the output use one model, and others use another model. The graph view might use components for nodes, but draw the edges itself as strokes. It might draw all the lines itself, but use label components for the text.

3.2.1 ISSUES IN CHOOSING OUTPUT MODEL

Layout: Components remember where they were put, and draw themselves there. They also support automatic layout. With stroke or pixel models, you have to figure out (at drawing time) where each piece goes, and put it there.

Input: Components participate in event dispatch and propagation, and the system automatically does **hit-testing** (determining whether the mouse is over the component when an event occurs) for components, but not for strokes. If a graph node is a component, then it can receive its own click and drag events. If you stroked the node instead, then you have to write code to determine which node was clicked or dragged.

Redraw: An automatic redraw algorithm means that components redraw themselves automatically when they have to. Furthermore, the redraw algorithm is efficient: it only redraws components whose extents intersect the damaged region. The stroke or pixel model would have to do this test by hand. In practice, most stroked components don't bother, simply redrawing everything whenever some part of the view needs to be redrawn.

Drawing order: It's easy for a parent to draw before (underneath) or after (on top of) all of its children. But it's not easy to interleave parent drawing with child drawing. So if you're using a hybrid model, with some parts of your view represented as components and others as strokes, then the components and strokes generally fall in two separate layers, and you can't have any complicated z-ordering relationships between strokes and components.

Heavyweight objects: Every component must be an object (and even an object with no fields costs about 20 bytes in Java). As we've seen, the view hierarchy is overloaded not just with drawing functions but also with event dispatch, automatic redraw, and automatic layout, so that further bulks up the class. The flyweight pattern used by InterView's Glyphs can reduce this cost somewhat. But views derived from large amounts of data – say, a 100,000-node graph – generally can't use a component model.

Device dependence: The stroke model is largely device independent. In fact, it's useful not just for displaying to screens, but also to printers, which have dramatically different resolution. The pixel model, on the other hand, is extremely device dependent. A

directly-mapped pixel image will not look the same on a screen with a different resolution.

Drawing is a top-down process: starting from the root of the component tree, each component draws itself, then draws each of its children recursively. The process is optimized by passing a **clipping region** to each component, indicating the area of the screen that needs to be drawn. Children that do not intersect the clipping region are simply skipped, not drawn. In the example above, nodes B and C would not need to be drawn. When a component partially intersects the clipping region, it must be drawn – but any strokes or pixels it draws when the clipping region is in effect will be masked against the clip region, so that only pixels falling inside the region actually make it onto the screen.

For the root component, the clipping region might be the entire screen. As drawing descends the component tree, however, the clipping region is intersected with each component's bounding box. So the clipping region for a component deep in the tree is the intersection of the bounding boxes of its ancestors.

For high performance, the clipping region is normally rectangular, using component **bounding boxes** rather than the components' actual shape. But it doesn't have to be that way. A clipping region can be an arbitrary shape on the screen. This can be very useful for visual effects: e.g., setting a string of text as your clipping region, and then painting an image through it like a stencil. Postscript was the first stroke model to allow this kind of nonrectangular clip region. Now many graphics toolkits support nonrectangular clip regions. For example, on Microsoft Windows and X Windows, you can create nonrectangular windows, which clip their children into a nonrectangular region.

When a component needs to change its appearance, it doesn't repaint itself directly. It *can't*, because the drawing process has to occur top-down through the component hierarchy: the component's ancestors and older siblings need to have a chance to paint themselves underneath it. (So, in Java, even though a component can call its paint() method directly, you shouldn't do it!)

Instead, the component asks the graphics system to repaint it at some time in the future. This request includes a **damaged region**, which is the part of the screen that needs to be repainted. Often, this is just the entire bounding box of the component; but complex components might figure out which part of the screen corresponds to the part of the model that changed, so that only that part is damaged.

The repaint request is then **queued** for later. Multiple pending repaint requests from different components are consolidated into a single damaged region, which is often represented just as a rectangle – the bounding box of all the damaged regions requested by individual components. That means that undamaged screen area is being considered damaged, but there is a tradeoff between the complexity of the damaged region representation and the cost of repainting.

Eventually – usually after the system has handled all the input events (mouse and keyboard) waiting on the queue --the repaint request is finally satisfied, by setting the clipping region to the damaged region and redrawing the component tree from the root.

There is an unfortunate side-effect of the automatic damage/redraw algorithm. If we draw a component tree directly to the screen, then moving a component can make the screen appear to flash – objects flickering while they move, and nearby objects flickering as well.
When an object moves, it needs to be erased from its original position and drawn in its new position. The erasure is done by redrawing all the objects in the view hierarchy that intersect this damaged region. If the drawing is done directly on the screen, this means that all the objects in the damaged region temporarily *disappear*, before being redrawn. Depending on how screen refreshes are timed with respect to the drawing, and how long it takes to draw a complicated object or multiple layers of the hierarchy, these partial redraws may be briefly visible on the monitor, causing a perceptible flicker.

Double-buffering solves this flickering problem. An identical copy of the screen contents is kept in a memory buffer. (In practice, this may be only the part of the screen belonging to some subtree of the view hierarchy that cares about double-buffering.) This memory buffer is used as the drawing surface for the automatic damage/redraw algorithm. After drawing is complete, the damaged region is just copied to screen as a block of pixels. Double-buffering reduces flickering for two reasons: first, because the pixel copy is generally faster than redrawing the view hierarchy, so there's less chance that a screen refresh will catch it half-done; and second, because unmoving objects that happen to be caught, as innocent victims, in the damaged region are never erased from the screen, only from the memory buffer.

It is a waste for every individual view to double-buffer itself. If any of your ancestors is double-buffered, then you'll derive the benefit of it. So double-buffering is usually applied to top-level windows.

Why is it called double-buffering? Because it used to be implemented by two interchangeable buffers in video memory. While one buffer was showing, you'd draw the next frame of animation into the other buffer. Then you'd just tell the video hardware to switch which buffer it was showing, a very fast operation that required no copying and was done during the CRT's vertical refresh interval so it produced no flicker at all.

Every stroke model has some notion of a **drawing surface**. The screen is only one place where drawing might go. Another common drawing surface is a memory buffer, which is an array of pixels just like the screen. Unlike the screen, however, a memory buffer can have arbitrary dimensions. The ability to draw to a memory buffer is essential for double-buffering. Another target is a printer driver, which forwards the drawing instructions on to a printer. Although most printers have a pixel model internally (when the ink actually hits the paper), the driver often uses a stroke model to communicate with the printer, for compact transmission. Postscript, for example, is a stroke model.

Most stroke models also include some kind of a **graphics context**, an object that bundles up drawing parameters like colour, line properties (width, end cap, join style), fill properties (pattern), and font. The stroke model may also provide a current **coordinate system**, which can be translated, scaled, and rotated around the drawing surface. We've already discussed the **clipping region**, which acts like a stencil for the drawing. Finally, a stroke model must provide a set of **drawing primitives**, function calls that actually produce graphical output.

Many systems combine all these responsibilities into a single object. Java's Graphics object is a good example of this approach. In other toolkits, the drawing surface and graphics context are independent objects that are passed along with drawing calls.
When state like graphics context, coordinate system, and clipping region are embedded in the drawing surface, the surface must provide some way to save and restore the context. A key reason for this is so that parent views can pass the drawing surface down to a child's draw method without fear that the child will change the graphics context. In Java, for example, the context can be saved by Graphics.create(), which makes a copy of the Graphics object. Notice that this only duplicates the graphics context; it doesn't duplicate the drawing surface, which is still the same.

3.2.2 HINTS FOR DUBUGGING OUTPUT

Wrong place: what's the origin of the coordinate system? What's the scale? Where is the component located in its parent?

Wrong size: if a component has 0 width and 0 height, it will be completely invisible no matter what it tries to draw– everything will be clipped. 0 width and 0 height is the default for all components in Swing – you have to use automatic layout or manual setting to make it a more reasonable size. Check whether the component (and its ancestors) have nonzero sizes.

Wrong color: is the drawing using the same color as the background? Is it using 100% alpha?

Wrong z-order: is something else drawing on top?

4.0 CONCLUSION

In this unit, you have been introduced to **Input** and **Output models** in detail. Different kinds of both input and output events have also be examined in greater detail.

5.0 SUMMARY

- Input models are GUI programs that are designed to handle input asynchronously, receiving it as events.
- There are two major categories of input events: raw and translated. A raw event comes right from the device driver while a raw event changed into an higher event is called translated.
- Output model represented using components, strokes or pixels.
- Issues in choosing output model includes layout, input redraw e.t.c.

6.0 TUTOR MARKED ASSIGNMENT.

a. Describe the Input and Output model.

b. Highlight various issues considered in choosing output models.

7.0 FURTHER READING AND OTHER RESOURCES

Boodhoo, Jean-Paul (August 2006). "Design Patterns: Model View Presenter". http://msdn.microsoft.com/en-us/magazine/cc188690.aspx. Retrieved on 2009-07-07.

World Wide Web Consortium (December 9, 2008). "The Forms Working Group". http://www.w3.org/MarkUp/Forms/. Retrieved on 2009-07-07.

UNIT 4 MODEL VIEW-CONTROLLER (MVC)

TableofContents

1.0 INTRODUCTION

The concept of Model View-Controller (MVC) is introduced in this unit. We will also discuss the history of MVC, pattern description, MVC implementation framework. Implementing MVC as GUI frameworks is also explained.

2.0 OBJECTIVES

By the end this unit, you should be able to:

- Explain the term model view-controller (MVC)
- Have good knowledge of the history of MVC
- Explain pattern description and MVC implementation framework
- Describe how to implement MVC as GUI frameworks

3.0 MAIN CONTENT

3.1 MODEL VIEW-CONTROLLER

Model–view–controller (MVC) is an architectural pattern used in software engineering. The pattern isolates business logic from input and presentation, permitting independent development, testing and maintenance of each.

An MVC application is a collection of model/view/controller triplets (a central dispatcher is often used to delegate controller actions to a view-specific controller). Each model is associated with one or more views (projections) suitable for presentation (not necessarily visual presentation). When a model changes its state, it notifies its associated views so they can refresh. The controller is responsible for initiating change requests and providing any necessary data inputs to the model.

MVC is frequently and needlessly convoluted by tying it directly to a graphical user interface. That a controller is often driven indirectly from a GUI is incidental. Likewise, rendering views graphically is an application of MVC, not part of the pattern definition. A business-to-business interface can leverage an MVC architecture equally well.

3.2 HISTORY

MVC was first described in 1979 by Trygve Reenskaug, then working on Smalltalk at Xerox PARC. The original implementation is described in depth in the influential paper "Applications Programming in Smalltalk-80: How to use Model–View–Controller".
There have been several derivatives of MVC. For example, Model View Presenter is used with the .NET Framework, and the XForms standard uses a "model-view-controller-connector architecture". However, standard MVC remains popular.

3.3 PATTERN DESCRIPTION

Model–view–controller is both an architectural pattern and a design pattern, depending on where it is used.

3.3.1 AS AN ARCHITECTURAL PATTERN

It is common to split an application into separate layers that can be analyzed, and sometimes implemented, separately.

- MVC is often seen in web applications, where the view is the actual HTML or XHTML page, and the controller is the code that gathers dynamic data and generates the content within the HTML or XHTML. Finally, the model is represented by the actual content, which is often stored in a database or in XML nodes, and the business rules that transform that content based on user actions.
- Though MVC comes in different flavors, control flow is generally as follows:
- The user interacts with the user interface in some way (for example, presses a mouse button).
- The controller handles the input event from the user interface, often via a registered handler or callback.
- The controller notifies the model of the user action, possibly resulting in a change in the model's state. (For example, the controller updates the user's shopping cart.)
- A view uses the model indirectly to generate an appropriate user interface (for example, the view lists the shopping cart's contents). The view gets its own data from the model. The model and controller have no direct knowledge of the view.
- The user interface waits for further user interactions, which restarts the cycle.
- Some implementations such as the W3C XForms also use the concept of a dependency graph to automate the updating of views when data in the model changes.
- By decoupling models and views, MVC helps to reduce the complexity in architectural design and to increase flexibility and reuse of code.

3.3.2 AS A DESIGN PATTERN

- MVC encompasses more of the architecture of an application than is typical for a design pattern. When considered as a design pattern, MVC is semantically similar to the Observer pattern.
- Model
- Is the domain-specific representation of the data on which the application operates. Domain logic adds meaning to raw data (for example, calculating whether today is the user's birthday, or the totals, taxes, and shipping charges for shopping cart items).
- Many applications use a persistent storage mechanism (such as a database) to store data. MVC does not specifically mention the data access layer because it is understood to be underneath or encapsulated by the model.
- View
- Renders the model into a form suitable for interaction, typically a user interface element. Multiple views can exist for a single model for different purposes.
- Controller
- Processes and responds to events (typically user actions) and may indirectly invoke changes on the model.

3.4 MVC IMPLEMENTATION FRAMEWORK

3.4.1 GUI FRAMEWORK

a. Java: Java Swing

Java Swing is different from the other frameworks in that it supports two MVC patterns:

Model

Frame level model—Like other frameworks, the design of the real model is usually left to the developer.

Control level model—Swing also supports models on the level of controls (elements of the graphical user interface). Unlike other frameworks, Swing exposes the internal storage of each control as a model.

View

The view is represented by a class that inherits from Component.

Controller

Java Swing doesn't use a single controller. Because its event model is based on interfaces, it is common to create an anonymous action class for each event.] In fact, the real controller is in a separate thread, the Event dispatching thread. It catches and propagates the events to the view and model.

b. Combined frameworks

Java: Java Platform, Enterprise Edition (Java EE)

Simple Version using only Java Servlets and JavaServer Pages from Java EE:

Model
The model is a collection of Java classes that forms a software application intended to store, and optionally moves, data. There is a single front end class that can communicate with any user interface (for example: a console, a graphical user interface, or a web application).

View
The view is represented by JavaServer Page, with data being transported to the page in the HttpServletRequest or HttpSession.

Controller
The controller servlet communicates with the front end of the model and loads the HttpServletRequest or HttpSession with appropriate data, before forwarding the HttpServletRequest and Response to the JSP using a RequestDispatcher.
The Servlet is a Java class, and it communicates and interacts with the model but does not need to generate HTML or XHTML output; the JSPs do not have to communicate with the model because the Servlet provides them with the information—they can concentrate on creating output.

Unlike the other frameworks, Java EE defines a pattern for model objects.

Model
The model is commonly represented by entity beans, although the model can be created by a servlet using a business object framework such as Spring.

View
The view in a Java EE application may be represented by a JavaServer Page, which may be currently implemented using JavaServer Faces Technology (JSF). Alternatively, the code to generate the view may be part of a servlet.

Controller
The controller in a Java EE application may be represented by a servlet, which may be currently implemented using JavaServerFaces (JSF).

c. XForms

XForms is an XML format for the specification of a data processing model for XML data and user interface(s) for the XML data, such as web forms.

Model
XForms stores the Model as XML elements in the browser. They are usually placed in the non-visible `<head>` elements of a web page.

View

The Views are XForms controls for screen elements and can be placed directly in the visible section of web page. They are usually placed in the <body> elements of a web page.
The model and views are bound together using reference or binding statements. These binding statements are used by the XForms dependency graph to ensure that the correct views are updated when data in the model changes. This means that forms developers do not need to be able to understand either the push or pull models of event processing.

Controller
All mouse events are processed by XForms controls and XML events are dispatched.

3.5 Implementations of MVC as GUI frameworks

Smalltalk's MVC implementation inspired many other GUI frameworks, such as the following:

- Cocoa framework and its GUI part AppKit, as a direct descendant of OpenStep, encourage the use of MVC. Interface Builder constructs Views, and connects them to Controllers via Outlets and Actions.
- GNUstep, also based on OpenStep, encourages MVC as well.
- GTK+.
- JFace.
- MFC (called Document/View architecture here).
- Microsoft Composite UI Application Block, part of the Microsoft Enterprise Library.
- Qt since Qt4 release.
- Java Swing.
- Adobe Flex.
- Wavemaker open source, browser-based development tool based on MVC.
- WPF uses a similar Model–view–viewmodel pattern.
- Visual FoxExpress is a Visual FoxPro MVC framework.

Exercise:- Search the internet and find out about the implementation strategies of the mentioned framework.

Some common programming languages and tools that support the implementation of MVC are:-

- .NET
- Actionscript
- ASP
- C++
- ColdFusion
- Flex
- Java
- Informix 4GL
- Lua
- Perl

- PHP
- Python
- Ruby
- Smalltalk
- XML

4.0 CONCLUSION

In this unit you, have been introduced to the model view-controller (MVC). We also discussed in detail the history of MVC, pattern description, MVC implementation framework and how to implement MVC as GUI frameworks.

5.0 SUMMARY

- The **Model–view–controller** (MVC) is an architectural pattern used in software engineering. The pattern isolates business logic from input and presentation, permitting independent development; testing and maintenance of each.requirements of your users drive the development of your prototype.
- MVC was first described in 1979 by Trygve Reenskaug, then working on Smalltalk at Xerox PARC.
- **Model–view–controller** is both an architectural pattern and a design pattern, depending on where it is used.
- GUI framework includes Java Swing, Combined and XForms
- Java Swing is different from the other frameworks in that it supports two MVC patterns: model and controller.
- Combined frameworks uses only Java Servlets and JavaServer Pages from Java EE.
- XForms is an XML format for the specification of a data processing model for XML data and user interface(s) for the XML data, such as web forms.
- Smalltalk's MVC implementation inspired many other GUI frameworks, such as Cocoa framework, GNUstep, GTK+, e.t.c

6.0 TUTOR MARKED ASSIGNMENT.

a. List and explain any two implementation of MVC as GUI framework.

b. Write a short note on Xforms.

7.0 FURTHER READING AND OTHER RESOURCES

Boodhoo, Jean-Paul (August 2006). "Design Patterns: Model View Presenter". http://msdn.microsoft.com/en-us/magazine/cc188690.aspx. Retrieved on 2009-07-07.
World Wide Web Consortium (December 9, 2008). "The Forms Working Group". http://www.w3.org/MarkUp/Forms/. Retrieved on 2009-07-07. "JavascriptMVCLearning Center". http://www.javascriptmvc.com.

UNIT 5 LAYOUTS AND CONSTRAINTS

TableofContents

1.0 INTRODUCTION

Understanding layouts and constraints will be the main goal of this unit. Layout managers and hints for layouts are discussed. Various types of constraints are also explained.

2.0 OBJECTIVES

By the end this unit, you should be able to:

- Explain User Interface
- Highlight the significance of user interface and identify the various types of user interfaces.
- Have good knowledge of the history of user interfaces
- Describe modes and modalities of User Interfaces

3.0 MAIN CONTENT

3.1 LAYOUT

Layout is determining the positions and sizes of graphical objects. This can be done manually or automatically. Layout ranges in difficulty and constraints. Some layouts require simple one pass algorithm and some require dynamic programming and other techniques.

There is need to do layout automatically since there is need to change the states and conditions of windows, screens, fonts, widgets, etc. This must be planned for at designed level and must be implemented effectively to enhance the user interface.

3.2 LAYOUT MANAGERS

Layout Managers are software tools for specifying and designing the appropriate layout for a job. They are also called geometry managers and are used to represent a bundle of constraint equations.
Also called geometry managers(Tk, Motif)

Abstract
Respresents a bundle of constraint equations

Local
Involve only the children of one container in the view hierarchy

Layout Propagation Algorithm

- Layout (Container parent, Rectangle parentSize)
- For each child in parent,
- Get child's size request
- Apply layout constraints to fit children into parentSize
- For each child,
- Set child's size and position

3.3 KINDS OF LAYOUT MANAGERS

- Packing
- One dimensional
- Tk: pack
- Java: BorderLayout, FlowLayout, BoxLayout
- Gridding
- Two dimensional
- Tk:grid
- Java: GridLayout, GridBagLayout, TableLayout
- General
- Java: SpringLayout

3.4 HINTS FOR LAYOUT

- Use packing layouts when alignments are 1D
- Borders for top-level
- Nested boxes for internal
- Reserve gridding layouts for 2D alignment
- Unfortunately common when fields have captions!
- TableLayout is easier than GridBag

3.5 CONSTRAINTS

Constraints are relationships expressed by the programmer and automatically maintained by the UI toolkit.

3.5.1 USES

a. Layout:- Constraints are used in Layout to express the relationships between interface items. For example,
field.left = label.right + 10

b. Value propagation:- They are used for an action to be invoked when a value is entered or reached. An example is
deleteAction.enabled = (selection != null)

c. Synchronization of views to models:- Constraints are used to expressed the relationships between models.

d. Interaction:- They are also used to express interaction such as
rect.corner = mouse

3.6 TYPES OF CONSTRAINTS

a. One-Way Constraints

Also called formulas, after spreadsheet
Y= f(x1,x2,x3, …)
Y depends on (points to) x1,x2,x3, …
Algorithms
Data-driven
Reevaluate formulas when a value is changed
Demand-driven
Reevaluate formulas whenever a value is requested
Lazy
When dependent value changes, invalidate all values that depend on it.
When invalid value is requested, recalculate it

b. Variants

Multi-output formulas
(y1,y2, …) = f(x1,x2,x3, …)
Cyclic dependencies
Detect cycles and break them

Constraint hierarchies
Some constraints stronger than others
Side effects
If f has side effects, when do they happen?
Lazy evaluation makes side effects unpredictable
Amulet: eager evaluation

4.0 CONCLUSION

Detailed description of Layouts and constraints were discussedt. Layout managers and hints for layouts were also described.

5.0 SUMMARY

- Layout is determining the positions and sizes of graphical objects. This can be done manually or automatically.
- Layout Managers are software tools for specifying and designing the appropriate layout for a job.
- Layout managers include packing, gridding and general kinds.
- Hints for layout include using packing layouts when alignments are 1D and reserving gridding layouts for 2D alignment.
- Constraints are relationships expressed by the programmer and automatically maintained by the UI toolkit.
- Constraints are of two types: One-Way Constraints and variants.

6.0 TUTOR MARKED ASSIGNMENT

Explain Layouts and constraints.
Describe any two layout manager.

7.0 FURTHER READING AND OTHER RESOURCES

Ambler, S.W. & Constantine, L.L. (2000a). *The Unified Process Inception Phase*. Gilroy, CA: CMP Books. http://www.ambysoft.com/inceptionPhase.html.

Ambler, S.W. & Constantine, L.L. (2000b). *The Unified Process Elaboration Phase*. Gilroy, CA: CMP

Ambler, S.W. & Constantine, L.L. (2000c). *The Unified Process Construction Phase*. Gilroy, CA: CMP

Ambler, S.W. (2001). *The Object Primer 2nd Edition: The Application Developer's Guide to Object Orientation*. New York: Cambridge University Press.

MODEL 4 – USER INTERFACE EVALUATION

UNIT 1 - TECHNIQUES FOR EVALUATING AND MEASURING INTERFACE USABILITY

UNIT 2 - EVALUATING USER INTERFACE WITHOUT THE USER

UNIT 3 - EVALUATING USER INTERFACE WITH THE USER

UNIT 4 - OTHER EVALUATION METHODOLOGIES

UNIT 5 - USABILITY TESTING PROCEDURE

MODULE 4

UNIT 1 TECHNIQUES FOR EVALUATING AND MEASURING INTERFACE USABILITY

TableofContents

1.0 INTRODUCTION

This unit describes usability evaluation methods. Cognitive modeling, Inspection, Inquiry and prototyping methods and Inspection methods are described in detail. Evaluation metrics are also discussed.

2.0 OBJECTIVES

By the end this unit, you should be able to:

- Explain generally various usability evaluation methods.
- Explain the differences between these various methods.
- Describe evaluation metrics.

3.0 MAIN CONTENT

3.1 USABILITY EVALUATION METHODS

There are a variety of methods currently used to evaluate usability. Certain methods make use of data gathered from users, while others rely on usability experts. There are usability evaluation methods that apply to all stages of design and development, from product definition to final design modifications. When choosing a method you must consider the cost, time constraints, and appropriateness of the method. For a brief overview of methods, see Comparison of usability evaluation methods or continue reading below. Usability methods can be further classified into the following subcategories:

3.1.1 COGNITIVE MODELING METHODS

Cognitive modeling involves creating a computational model to estimate how long it takes people to perform a given task. Models are based on psychological principles and experimental studies to determine times for cognitive processing and motor movements. Cognitive models can be used to improve user interfaces or predict problem errors and pitfalls during the design process. A few examples of cognitive models include:

a. Parallel Design

With parallel design, several people create an initial design from the same set of requirements. Each person works independently, and when finished, shares his/her concepts with the group. The design team considers each solution, and each designer uses the best ideas to further improve their own solution. This process helps to generate many different, diverse ideas and ensures that the best ideas from each design are integrated into the final concept. This process can be repeated several times until the team is satisfied with the final concept.

b. GOMS

GOMS is an acronym that stands for Goals, Operator, Methods, and Selection Rules. It is a family of techniques that analyzes the user complexity of interactive systems. Goals are what the user has to accomplish. An operator is an action performed in service of a goal. A method is a sequence of operators that accomplish a goal. Selection rules specify which method should be used to satisfy a given goal, based on the context.

c. Human Processor Model

Sometimes it is useful to break a task down and analyze each individual aspect separately. This allows the tester to locate specific areas for improvement. To do this, it is necessary to understand how the human brain processes information. This has been fully described in Module 1.

d. Keystroke level modelling

Keystroke level modeling is essentially a less comprehensive version of GOMS that makes simplifying assumptions in order to reduce calculation time and complexity. You can read more about Keystroke level model for more information.

3.1.2 INSPECTION METHODS

These usability evaluation methods involve observation of users by an experimenter, or the testing and evaluation of a program by an expert reviewer. They provide more quantitative data as tasks can be timed and recorded.

a. Card Sorting

Card sorting is a way to involve users in grouping information for a website's usability review. Participants in a card sorting session are asked to organize the content from a Web site in a way that makes sense to them. Participants review items from a Web site and then group these items into categories. Card sorting helps to learn how users think about the content and how they would organize the information on the Web site. Card sorting helps to build the structure for a Web site, decide what to put on the home page, and label the home page categories. It also helps to ensure that information is organized on the site in a way that is logical to users.

b. Ethnography

Ethnographic analysis is derived from anthropology. Field observations are taken at a site of a possible user, which track the artifacts of work such as Post-It notes, items on desktop, shortcuts, and items in trash bins. These observations also gather the sequence of work and interruptions that determine the user's typical day.

c. Heuristic Evaluation

Heuristic evaluation is a usability engineering method for finding and assessing usability problems in a user interface design as part of an iterative design process. It involves having a small set of evaluators examining the interface and using recognized usability principles (the "heuristics"). It is the most popular of the usability inspection methods, as it is quick, cheap, and easy. It is fully discussed in Unit 2 of this module.

Usability Inspection

Usability inspection is a review of a system based on a set of guidelines. The review is conducted by a group of experts who are deeply familiar with the concepts of usability in design. The experts focus on a list of areas in design that have been shown to be troublesome for users.

i. Pluralistic Inspection

Pluralistic Inspections are meetings where users, developers, and human factors people meet together to discuss and evaluate step by step of a task scenario. As more people inspect the scenario for problems, the higher the probability to find problems. In addition, the more interaction in the team, the faster the usability issues are resolved.

ii. Consistency Inspection

In consistency inspection, expert designers review products or projects to ensure consistency across multiple products to look if it does things in the same way as their own designs.

iii. Activity Analysis

Activity analysis is a usability method used in preliminary stages of development to get a sense of situation. It involves an investigator observing users as they work in the field. Also referred to as user observation, it is useful for specifying user requirements and studying currently used tasks and subtasks. The data collected is qualitative and useful for defining the problem. It should be used when you wish to frame what is needed, or "What do we want to know?"

3.1.3 INQUIRY METHODS

The following usability evaluation methods involve collecting qualitative data from users. Although the data collected is subjective, it provides valuable information on what the user wants.

a. Task Analysis

Task analysis means learning about users' goals and users' ways of working. Task analysis can also mean figuring out what more specific tasks users must do to meet those goals and what steps they must take to accomplish those tasks. Along with user and task analysis, we often do a third analysis: understanding users' environments (physical, social, cultural, and technological environments).

b. Focus Groups

A focus group is a focused discussion where a moderator leads a group of participants through a set of questions on a particular topic. Although typically used as a marketing tool, Focus Groups are sometimes used to evaluate usability. Used in the product definition stage, a group of 6 to 10 users are gathered to discuss what they desire in a product. An experienced focus group facilitator is hired to guide the discussion to areas of interest for the developers. Focus groups are typically videotaped to help get verbatim quotes, and clips are often used to summarize opinions. The data gathered is not usually quantitative, but can help get an idea of a target group's opinion.

c. Questionnaires/Surveys

Surveys have the advantages of being inexpensive, require no testing equipment, and results reflect the users' opinions. When written carefully and given to actual users who have experience with the product and knowledge of design, surveys provide useful feedback on the strong and weak areas of the usability of a design. This is a very common method and often does not appear to be a survey, but just a warranty card.

3.1.4 PROTOTYPING METHODS

Rapid prototyping is a method used in early stages of development to validate and refine the usability of a system. It can be used to quickly and cheaply evaluate user-interface designs without the need for an expensive working model. This can help remove hesitation to change the design, since it is implemented before any real programming begins. One such method of rapid prototyping is paper prototyping.

3.1.5 TESTING METHODS

These usability evaluation methods involve testing of subjects for the most quantitative data. Usually recorded on video, they provide task completion time and allow for observation of attitude.

a. Remote usability testing

Remote usability testing (also known as unmoderated or asynchronous usability testing) involves the use of a specially modified online survey, allowing the quantification of user testing studies by providing the ability to generate large sample sizes. Additionally, this style of user testing also provides an opportunity to segment feedback by demographic, attitudinal and behavioural type. The tests are carried out in the user's own environment (rather than labs) helping further simulate real-life scenario testing. This approach also provides a vehicle to easily solicit feedback from users in remote areas.

b. Thinking Aloud

The Think aloud protocol is a method of gathering data that is used in both usability and psychology studies. It involves getting a user to verbalize their thought processes as they perform a task or set of tasks. Often an instructor is present to prompt the user into being more vocal as they work. Similar to the Subjects-in-Tandem method, it is useful in pinpointing problems and is relatively simple to set up. Additionally, it can provide insight into the user's attitude, which can not usually be discerned from a survey or questionnaire.

c. Subjects-in-Tandem

Subjects-in-tandem is pairing of subjects in a usability test to gather important information on the ease of use of a product. Subjects tend to think out loud and through their verbalized thoughts designers learn where the problem areas of a design are. Subjects very often provide solutions to the problem areas to make the product easier to use.

3.1.6 OTHER METHODS

Cognitive walkthrough is a method of evaluating the user interaction of a working prototype or final product. It is used to evaluate the system's ease of learning. Cognitive walkthrough is useful to understand the user's thought processes and decision making when interacting with a system, specially for first-time or infrequent users.

a. Benchmarking

Benchmarking creates standardized test materials for a specific type of design. Four key characteristics are considered when establishing a benchmark: time to do the core task, time to fix errors, time to learn applications, and the functionality of the system. Once there is a benchmark, other designs can be compared to it to determine the usability of the system.

b. Meta-Analysis

Meta-Analysis is a statistical procedure to combine results across studies to integrate the findings. This phrase was coined in 1976 as a quantitative literature review. This type of evaluation is very powerful for determining the usability of a device because it combines multiple studies to provide very accurate quantitative support.

c. Persona

Personas are fictitious characters that are created to represent a site or product's different user types and their associated demographics and technographics. Alan Cooper introduced the concept of using personas as a part of interactive design in 1998 in his book *The Inmates Are Running the Asylum*, but had used this concept since as early as 1975.

Personas are a usability evaluation method that can be used at various design stages. The most typical time to create personas is at the beginning of designing so that designers have a tangible idea of who the users of their product will be. Personas are the archetypes that represent actual groups of users and their needs, which can be a general description of person, context, or usage scenario. This technique turns marketing data on target user population into a few physical concepts of users to create empathy among the design team, with the final aim of tailoring a product more closely to how the personas will use it.

To gather the marketing data that personas require, several tools can be used, including online surveys, web analytics, customer feedback forms, and usability tests, and interviews with customer-service representatives.
Cognitive walkthrough is fully discussed in Unit 2 of this Module.

3.2 EVALUATION WITH TESTS AND METRICS

Regardless to how carefully a system is designed, all theories must be tested using usability tests. Usability tests involve typical users using the system (or product) in a realistic environment. Observation of the user’s behavior, emotions, and difficulties while performing different tasks, often identify areas of improvement for the system.

3.2.1 THE USE OF PROTOTYPES

It is often very difficult for designers to conduct usability tests with the exact system being designed. Cost constraints, size, and design constraints usually lead the designer to creating a prototype of the system. Instead of creating the complete final system, the designer may test different sections of the system, thus making several small models of each component of the system. The types of usability prototypes may vary from using paper models, index cards, hand drawn models, or storyboards.

Prototypes are able to be modified quickly, often are faster and easier to create with less time invested by designers and are more apt to change design; although sometimes are not an adequate representation of the whole system, are often not durable and testing results may not be parallel to those of the actual system.

3.2.2 METRICS

While conducting usability tests, designers must use usability metrics to identify what it is they are going to measure, or the usability metrics. These metrics are often variable, and change in conjunction with the scope and goals of the project. The number of subjects being tested can also affect usability metrics, as it is often easier to focus on specific demographics. Qualitative design phases, such as general usability (can the task be accomplished?), and user satisfaction are also typically done with smaller groups of subjects. Using inexpensive prototypes on small user groups provides more detailed information, because of the more interactive atmosphere, and the designer's ability to focus more on the individual user.

As the designs become more complex, the testing must become more formalized. Testing equipment will become more sophisticated and testing metrics become more quantitative. With a more refined prototype, designers often test effectiveness, efficiency, and subjective satisfaction, by asking the user to complete various tasks. These categories are measured by the percent that complete the task, how long it takes to complete the tasks, ratios of success to failure to complete the task, time spent on errors, the number of errors, rating scale of satisfactions, number of times user seems frustrated, etc. Additional observations of the users give designers insight on navigation difficulties, controls, conceptual models, etc. The ultimate goal of analyzing these metrics is to find/create a prototype design that users like and use to successfully perform given tasks.

After conducting usability tests, it is important for a designer to record what was observed, in addition to why such behavior occurred and modify the model according to the results. Often it is quite difficult to distinguish the source of the design errors, and what the user did wrong. However, effective usability tests will not generate a solution to the problems, but provide modified design guidelines for continued testing.

4.0 CONCLUSION

The concept of evaluation of user interface without the users was introduced in this unit. Users walkthrough, action analysis and heuristics analysis concepts was also discussed in greater detail while emphasis on the need for evaluation was also introduced.

.

5.0 SUMMARY

- Cognitive modeling involves creating a computational model to estimate how long it takes people to perform a given task.
- Inspection usability evaluation methods involve observation of users by an experimenter, or the testing and evaluation of a program by an expert reviewer. They provide more quantitative data as tasks can be timed and recorded.
- Inquiry usability evaluation methods involve collecting qualitative data from users. Although the data collected is subjective, it provides valuable information on what the user wants.
- Rapid prototyping is a method used in early stages of development to validate and refine the usability of a system. It can be used to quickly and cheaply evaluate user-interface designs without the need for an expensive working model.
- Testing usability evaluation methods involve testing of subjects for the most quantitative data. Usually recorded on video, they provide task completion time and allow for observation of attitude.
- Cognitive walkthrough is a method of evaluating the user interaction of a working prototype or final product. It is used to evaluate the system's ease of learning.
- Usability metrics is used to identify the features that will be measured.

6.0 TUTOR MARKED ASSIGNMENT

a. Explain Cognitive modeling.
b. Describe GOMS briefly.
c. Identify some metrics and use them to evaluate the main menu interface of WINDOWS OS

7.0 REFERENCES AND FURTHER READING

www.wikipedia.org

Holm, Ivar (2006). *Ideas and Beliefs in Architecture and Industrial design: How attitudes, orientations, and underlying assumptions shape the built environment*. Oslo School of Architecture and Design. ISBN8254701741.

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

Dumas, J.S. and Redish, J.C. (1999). *A Practical Guide to Usability Testing* (revised ed.), Bristol, U.K.: Intellect Books.

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.

MODULE 4

UNIT 2 EVALUATING USER INTERFACE WITHOUT THE USERS

TableofContents

1.0 INTRODUCTION

This unit describes the concept of evaluation of user interface without the users. Users walkthrough, action analysis and heuristics analysis are various concepts that will all be discussed throughout this unit.

2.0 OBJECTIVES

By the end this unit, you should be able to:

Explain concept of evaluation of user interface without the users.
Describe Users walkthrough.
Explain action analysis.
Describe heuristics analysis.

3.0 MAIN CONTENT

3.1 EVALUATING USER INTERFACE WITHOUT THE USERS

Throughout this course material, we have emphasized the importance of bringing users into the interface design process. However, as a designer, you will also need to evaluate the evolving design when no users are present. Users' time is almost never a free or unlimited resource. Most users have their own work to do, and they are able to devote only limited time to your project. When users do take time to look at your design, it should be as free as possible of problems. This is a courtesy to the users, who shouldn't have to waste time on trivial bugs that you could have caught earlier. It also helps build

the users' respect for you as a professional, making it more likely that they will give the design effort serious attention.

A second reason for evaluating a design without users is that a good evaluation can catch problems that an evaluation with only a few users may not reveal. The numbers tell the story here: An interface designed for a popular personal computer might be used by thousands of people, but it may be tested with only a few dozen users before beta release. Every user will have a slightly different set of problems, and the testing will not uncover problems that the few users tested don't have. It also won't uncover problems that users might have after they get more experience. An evaluation without users won't uncover all the problems either. But doing both kinds of evaluation significantly improves the chances of success.

In this unit, we will describe three approaches for evaluating user interface in the absence of users. The first approach is the **cognitive walkthrough**, a task-oriented technique that fits especially well in the context of task-centered design. The second approach is **action analysis**, which allows a designer to predict the time that an expert user would need to perform a task, and which forces the designer to take a detailed look at the interface. The third approach is **heuristic evaluation**, a kind of check-list approach that catches a wide variety of problems but requires several evaluators who have some knowledge of usability problems.

3.1.1 CONGNITIVE WALKTHROUGHS

The cognitive walkthrough is a formalized way of imagining people's thoughts and actions when they use an interface for the first time. Briefly, a walkthrough goes like this: You have a prototype or a detailed design description of the interface, and you know who the users will be. You select one of the tasks that the design is intended to support. Then you try to tell a believable story about each action a user has to take to do the task. To make the story believable you have to motivate each of the user's actions, relying on the user's general knowledge and on the prompts and feedback provided by the interface. If you can't tell a believable story about an action, then you have located a problem with the interface.

You can see from the brief example that the walkthrough can uncover several kinds of problems. It can question assumptions about what the users will be thinking ("why would a user think the machine needs to be switched on?"). It can identify controls that are obvious to the design engineer but may be hidden from the user's point of view ("the user wants to turn the machine on, but can she find the switch?"). It can suggest difficulties with labels and prompts ("the user wants to turn the machine on, but which is the power switch and which way is on?"). And it can note inadequate feedback, which may make the users balk and retrace their steps even after they've done the right thing ("how does the user know it's turned on?").

The walkthrough can also uncover shortcomings in the current specification, that is, not in the interface but in the way it is described. Perhaps the copier design really was

"intended" to have a power-on indicator, but it just didn't get written into the specs. The walkthrough will ensure that the specs are more complete. On occasion the walkthrough will also send the designer back to the users to discuss the task. Is it reasonable to expect the users to turn on the copier before they make a copy? Perhaps it should be on by default, or turn itself on when the "Copy" button is pressed.

Walkthroughs focus most on problems that users will have when they first use an interface, without training. For some systems, such as "walk-up-and-use" banking machines, this is obviously critical. But the same considerations are also important for sophisticated computer programs that users might work with for several years. Users often learn these complex programs incrementally, starting with little or no training, and learning new features as they need them. If each task-oriented group of features can pass muster under the cognitive walkthrough, then the user will be able to progress smoothly from novice behavior to productive expertise.

One other point from the example: Notice that we used some features of the task that were implicitly pulled from a detailed, situated understanding of the task: the user is sitting at a desk, so she can't see the power switch. It would be impossible to include all relevant details like this in a written specification of the task. The most successful walkthroughs will be done by designers who have been working closely with real users, so they can create a mental picture of those users in their actual environments.
Now here are some details on performing walkthroughs and interpreting their results.

a. Who should do a walkthrough, and when?

If you are designing a small piece of the interface on your own, you can do your own, informal, "in your head" walkthroughs to monitor the design as you work. Periodically, as larger parts of the interface begin to coalesce, it's useful to get together with a group of people, including other designers and users, and do a walkthrough for a complete task. One thing to keep in mind is that the walkthrough is really a tool for developing the interface, not for validating it. You should go into a walkthrough expecting to find things that can be improved. Because of this, we recommend that group walkthroughs be done by people who are roughly at the same level in the company hierarchy. The presence of high-level managers can turn the evaluation into a show, where the political questions associated with criticizing someone else's work overshadow the need to improve the design.

b. Who should do a walkthrough, and when?

You need information about four things. (1) You need a description or a prototype of the interface. It doesn't have to be complete, but it should be fairly detailed. Things like exactly what words are in a menu can make a big difference. (2) You need a task description. The task should usually be one of the representative tasks you're using for task-centered design, or some piece of that task. (3) You need a complete, written list of the actions needed to complete the task with the interface. (4) You need an idea of who

the users will be and what kind of experience they'll bring to the job. This is an understanding you should have developed through your task and user analysis.

c. Who should do a walkthrough, and when?

You have defined the task, the users, the interface, and the correct action sequence. You've gathered a group of designers and other interested folk together. Now it's time to actually DO THE WALKTHROUGH.
In doing the walkthrough, you try to tell a story about why the user would select each action in the list of correct actions. And you critique the story to make sure it's believable. We recommend keeping four questions in mind as you critique the story:

- Will users be trying to produce whatever effect the action has?
- Will users see the control (button, menu, switch, etc.) for the action?
- Once users find the control, will they recognize that it produces the effect they want?
- After the action is taken, will users understand the feedback they get, so they can go on to the next action with confidence?

Here are a few examples -- "failure stories" -- of interfaces that illustrate how the four questions apply.

The first question deals with what the user is thinking. Users often are not thinking what designers expect them to think. For example, one portable computer we have used has a slow-speed mode for its processor, to save battery power. Assume the task is to do some compute-intensive spreadsheet work on this machine, and the first action is to toggle the processor to high-speed mode. Will users be trying to do this? Answer: Very possibly not! Users don't expect computers to have slow and fast modes, so unless the machine actually prompts them to set the option, many users may leave the speed set at its default value -- or at whatever value it happened to get stuck in at the computer store.

The second question concerns the users' ability to locate the control -- not to identify it as the right control, but simply to notice that it exists! Is this often a problem? You bet. Attractive physical packages commonly hide "ugly" controls. One of our favorite examples is an office copier that has many of its buttons hidden under a small door, which has to be pressed down so it will pop up and expose the controls. If the task is, for example, to make double-sided copies, then there's no doubt that users with some copier experience will look for the control that selects that function. The copier in question, in fact, has a clearly visible "help" sheet that tells users which button to push. But the buttons are hidden so well that many users have to ask someone who knows the copier where to find them. Other interfaces that take a hit on this question are graphic interfaces that require the user to hold some combination of keys while clicking or dragging with a mouse, and menu systems that force the users to go through several levels to find an option. Many users will never discover what they're after in these systems without some kind of training or help.

The third question involves identifying the control. Even if the users want to do the right thing and the control is plainly visible, will they realize that this is the control they're after? An early version of a popular word processor had a table function. To insert a new table the user selected a menu item named "Create Table." This was a pretty good control name. But to change the format of the table, the user had to select a menu item called "Format Cells." The designers had made an analogy to spreadsheets, but users weren't thinking of spreadsheets -- they were thinking of tables. They often passed right over the correct menu item, expecting to find something called "Format Table." The problem was exacerbated by the existence of another menu item, "Edit Table," which was used to change the table's size.

Notice that the first three questions interact. Users might not want to do the right thing initially, but an obvious and well labeled control could let them know what needs to be done. A word processor, for example, might need to have a spelling dictionary loaded before the spelling checker could run. Most users probably wouldn't think of doing this. But if a user decided to check spelling and started looking through the menus, a "load spelling dictionary" menu item could lead them to take the right action. Better yet, the "check spelling" menu item could bring up a dialog box that asked for the name of the spelling dictionary to load.

The final question asks about the feedback after the action is performed. Generally, even the simplest actions require some kind of feedback, just to show that the system "noticed" the action: a light appears when a copier button is pushed, an icon highlights when clicked in a graphical interface, etc. But at a deeper level, what the user really needs is evidence that whatever they are trying to do (that "goal" that we identified in the first question) has been done, or at least that progress has been made. Here's an example of an interface where that fails. A popular file compression program lets users pack one or more files into a much smaller file on a personal computer. The program presents a dialog box listing the files in the current directory. The user clicks on each file that should be packed into the smaller file, then, after each file, clicks the "Add" button. But there's no change visible after a file has been added. It stays in the list, and it isn't highlighted or grayed. As a result, the user isn't sure that all the files have been added, so he or she may click on some files again -- which causes them to be packed into the smaller file twice, taking up twice the space!

d. What do you do with the results of the walkthrough?

Fix the interface! Many of the fixes will be obvious: make the controls more obvious, use labels that users will recognize (not always as easy as it sounds), provide better feedback. Probably the hardest problem to correct is one where the user doesn't have any reason to think that an action needs to be performed. A really nice solution to this problem is to eliminate the action -- let the system take care of it. If that can't be done, then it may be possible to re-order the task so users will start doing something they know needs doing, and then get prompted for the action in question. The change to the "spelling dictionary" interaction that we described is one example. For the portable computer speed problem, the system might monitor processor load and ask if the user wanted to change to low

speed whenever the average load was low over a 20 minute period, with a similar prompt for high speed.

3.1.2 ACTION ANALYSIS

Action analysis is an evaluation procedure that forces you to take a close look at the sequence of actions a user has to perform to complete a task with an interface. In this unit, we will distinguish between two flavors of action analysis. The first, "formal" action analysis, is often called "keystroke-level analysis" in HCI work. The formal approach is characterized by the extreme detail of the evaluation. The detail is so fine that, in many cases, the analysis can predict the time to complete tasks within a 20 percent margin of error, even before the interface is prototyped. It may also predict how long it will take a user to learn the interface. Unfortunately, formal action analysis is not easy to do.
The second flavor of action analysis is what we call the "back of the envelope" approach. This kind of evaluation will not provide the detailed predictions of task time and interface learnability, but it can reveal large-scale problems that might otherwise get lost in the forest of details that a designer is faced with. As its name implies, the back-of- the-envelope approach does not take a lot of effort.

Action analysis, whether formal or back-of-the-envelope, has two fundamental phases. The first phase is to decide what physical and mental steps a user will perform to complete one or more tasks with the interface. The second phase is to analyze those steps, looking for problems. Problems that the analysis might reveal are that it takes too many steps to perform a simple task, or it takes too long to perform the task, or there is too much to learn about the interface. The analysis might also reveal "holes" in the interface description -- things that the system should be able to do but can not. And it could be useful in writing or checking documentation, which should describe the facts and procedures that the analysis has shown the user needs to know.

a. Formal Action Analysis

The formal approach to action analysis has been used to make accurate predictions of the time it takes a skilled user to complete tasks. To predict task times, the times to perform each small step of the task, physical or mental, are estimated, and those times are totalled. Most steps take only a fraction of a second. A typical step is a keystroke, which is why the formal approach is often called "keystroke- level analysis."

The predictions of times for each small step are found by testing hundreds of individual users, thousands of individual actions, and then calculating average values. These values have been determined for most of the common actions that users perform with computer interfaces. We summarize those values in the tables below. If an interface control is not in the table, it might be possible to extrapolate a reasonable value from similar devices, or user testing might have to be done for the new control.

The procedure for developing the list of individual steps is very much like programming a computer. The basic task is divided into a few subtasks, like subroutines in a computer

program. Then each of those subtasks is broken into smaller subtasks, and so on until the description reaches the level of the fraction-of-a-second operations listed in the table. The end result is a hierarchical description of a task and the action sequence needed to accomplish it.

Table 4:- ***Table: Average times for computer interface actions***
[Based on detailed information in Judith Reitman Olson and Gary M. Olson, "The growth of cognitive modeling in human- computer interaction since GOMS," Human-Computer Interaction, 5 (1990), pp. 221-265. Many values given in this table are averaged and rounded.]

PHYSICAL MOVEMENTS		
Enter one keystroke on a standard keyboard:	.28 second	Ranges from .07 second for highly skilled typists doing transcription, to .2 second for an average 60-wpm typist, to over 1 second for a bad typist. Random sequences, formulas, and commands take longer than plain text.
Use mouse to point at object on screen	1.5 second	May be slightly lower -- but still at least 1 second -- for a small screen and a menu. Increases with larger screens, smaller objects.
Move hand to pointing device or function key	.3 second	Ranges from .21 second for cursor keys to .36 second for a mouse.
VISUAL PERCEPTION		
Respond to a brief light	.1 second	Varies with intensity, from .05 second for a bright light to .2 second for a dim one.
Recognize a 6-letter word	.34 second	
Move eyes to new location on screen (saccade)	.23 second	
MENTAL ACTIONS		
Retrieve a simple item from long-term memory	1.2 second	A typical item might be a command abbreviation ("dir"). Time is roughly halved if the same item needs to be retrieved again immediately.
Learn a single "step" in a procedure	25 seconds	May be less under some circumstances, but most research shows 10 to 15 seconds as a minimum. None of these figures include the time needed to get started in a training situation.
Execute a mental "step"	.075 second	Ranges from .05 to .1 second, depending on what kind of mental step is being performed.
Choose among methods	1.2 second	Ranges from .06 to at least 1.8 seconds, depending on complexity of factors influencing the decision.

A full-blown formal analysis of a complex interface is a daunting task. The example and the size of the time values in the table should give some idea of why this is so. Imagine you want to analyze two designs for a spreadsheet to see which is faster on a given task. The task is to enter some formulas and values, and you expect it to take the skilled user on the order of 10 minutes. To apply the formal action analysis approach you'll have to break the task down into individual actions, most of which take less than a second. That comes out to around 1000 individual actions, just to analyze a single 10-minute task! (There will probably be clusters of actions that get repeated; but the effort is still nontrivial.)

A further problem with formal analysis is that different analysts may come up with different results, depending on how they see the task hierarchy and what actions they predict a user will take in a given situation. (Will the user scan left, then down the spreadsheet? Down then left? Diagonally? The difference might be seconds, swamping other details.) Questions like this may require user testing to settle.

Because it is so difficult, we think that formal action analysis is useful only in special circumstances -- basically, when its high cost can be justified by a very large payoff. One instance where this was the case was the evaluation of a proposed workstation for telephone operators (see the article by Gray et al listed in Credits and Pointers, below). The phone company contracting the action analysis calculated that a savings of a few seconds in a procedure performed by thousands of operators over hundreds of thousands of calls would more than repay the months of effort that went into the evaluation.
Another place formal action analysis can be effective is for segments of the interface that users will access repeatedly as part of many tasks. Some examples of this are choosing from menus, selecting or moving objects in a graphics package, and moving from cell to cell within a spreadsheet. In each of these examples, a savings of a few tenths of a second in an interaction might add up to several minutes during an hour's work. This could justify a detailed analysis of competing designs.

In most cases, however, a few tenths of a second saved in performing an action sequence, and even a few minutes saved in learning it, are trivial compared to the other aspects of the interface that we emphasize in this book. Does the interface (and the system) do what the user needs, fitting smoothly into the rest of the user's work? Can the user figure out how the interface works? Does the interface's combination of controls, prompts, warning messages, and other feedback allow the user to maintain a comfortable "flow" through a task? If the user makes an error, does the system allow graceful recovery? All of these factors are central, not only to productivity but also to the user's perception of the system's quality. A serious failure on any of these points is not going to be countered by shaving a few seconds off the edges of the interface.

b. **Back-of-the-Envelope Action Analysis**

The back-of-the-envelope approach to action analysis foregoes detailed predictions in an effort to get a quick look at the big picture. We think this technique can be very valuable, and it's easy to do. Like the formal analysis, the back-of- the-envelope version has two

phases: list the actions, and then think about them. The difference is that you do not need to spend a lot of time developing a detailed hierarchical breakdown of the task. You just list a fairly "natural" series of actions and work with them.

A process that works well for listing the actions is to imagine you are explaining the process to a typical user. That means you aren't going to say, "take your hand off the keyboard and move it to the mouse," or "scan down the menu to the 'chooser' item." You will probably just say, "select 'chooser' from the apple menu." You should also put in brief descriptions of mental actions, such as "remember your password," or "convert the time on your watch to 24-hour time."

Once you have the actions listed there are several questions you can ask about the interface:

> Can a simple task be done with a simple action sequence?
> Can frequent tasks be done quickly?
> How many facts and steps does the user have to learn?
> Is everything in the documentation?

You can get useful answers to all these questions without going into fraction-of-a-second details. At the action level you'd use in talking to a user, EVERY ACTION TAKES AT LEAST TWO OR THREE SECONDS. Selecting something from a menu with the mouse, entering a file name, deciding whether to save under a new name or the old one, remembering your directory name -- watch over a user's shoulder sometime, or videotape a few users doing random tasks, and you'll see that combined physical and mental time for any one of these actions is a couple of seconds on a good day, three or four or even ten before morning coffee. And you'll have to start measuring in minutes whenever there's any kind of an error or mistake.

By staying at this level of analysis, you're more likely to keep the task itself in mind, along with the user's work environment, instead of getting lost in a detailed comparison of techniques that essentially do the same thing. For example, you can easily use the back-of-the-envelope results to compare your system's proposed performance with the user's ability to do the same task with typewriters, calculators, and file cabinets.

This kind of action analysis is especially useful in deciding whether or not to add features to an interface, or even to a system. Interfaces have a tendency to accumulate features and controls like a magnet accumulates paperclips in a desk drawer. Something that starts out as a simple, task-oriented action sequence can very quickly become a veritable symphony of menus, dialog boxes, and keystrokes to navigate through the options that various people thought the system should offer. Often these options are intended as "time savers," but the user ends up spending an inordinate amount of time just deciding which time saver to use and which to ignore. (One message you should take away from the table of action times is that it takes real time to decide between two ways of doing something.) A few quick calculations can give you ammunition for convincing other members of a development team which features should or should not be added. Of course, marketing

arguments to the contrary may prevail: it often seems that features sell a program, whether or not they're productive. But it's also true that popular programs sometimes become so complicated that newer, simpler programs move in and take over the low end of the market. The newcomers may even eventually displace the high-functionality leaders. (An example of this on a grand scale is the effect of personal computers on the mainframe market.)

3.1.3 **HEURISTIC ANALYSIS**

Heuristics, also called guidelines, are general principles or rules of thumb that can guide design decisions. As soon as it became obvious that bad interfaces were a problem, people started proposing heuristics for interface design, ranging from short lists of very general platitudes ("be informative") to a list of over a thousand very detailed guidelines dealing with specific items such as menus, command names, and error messages. None of these efforts has been strikingly successful in improving the design process, although they're usually effective for critiquing favorite bad examples of someone else's design. When the short lists are used during the design process, however, a lot of problems get missed; and the long lists are usually too unwieldy to apply. In addition, all heuristics require that an analyst have a fair amount of user interface knowledge to translate the general principles into the specifics of the current situation.

Recently, Jacob Nielsen and Rolf Molich have made a real breakthrough in the use of heuristics. Nielsen and Molich have developed a short list of general heuristics, and more importantly, they've developed and tested a procedure for using them to evaluate a design. We give the details of that procedure below, but first we want to say something about why heuristics, which are not necessarily a task-oriented evaluation technique, can be an important part of task- centered design.

The other two evaluation methods described in this unit, the cognitive walkthrough and action analysis, are task- oriented. That is, they evaluate an interface as applied to a specific task that a user would be doing with the interface. User testing, discussed in chapter 6, is also task oriented. Task-oriented evaluations have some real advantages. They focus on interface problems that occur during work the user would actually be doing, and they give some idea of the importance of the problems in the context of the job. Many of the problems they reveal would only be visible as part of the sequence of actions needed to complete the task. But task-oriented evaluations also have some shortcomings. The first shortcoming is coverage: There's almost never time to evaluate every task a user would perform, so some action sequences and often some controls aren't evaluated. The second shortcoming is in identifying cross-task interactions. Each task is evaluated standing alone, so task-oriented evaluations won't reveal problems such as command names or dialog-box layouts that are done one way in one task, another way in another.

Task-free evaluation methods are important for catching problems that task-oriented methods miss. Both approaches should be used as the interface develops. Now, here's how the heuristic analysis approach works.

Nielsen and Molich used their own experience to identify nine general heuristics (see table, below), which, as they noted, are implicit or explicit in almost all the lists of guidelines that have been suggested for HCI. Then they developed a procedure for applying their heuristics. The procedure is based on the observation that no single evaluator will find every problem with an interface, and different evaluators will often find different problems. So the procedure for heuristic analysis is this: Have several evaluators use the nine heuristics to identify problems with the interface, analyzing either a prototype or a paper description of the design. Each evaluator should do the analysis alone. Then combine the problems identified by the individual evaluators into a single list. Combining the individual results might be done by a single usability expert, but it's often useful to do this as a group activity.

Nielsen and Molich's Nine Heuristics

- **Simple and natural dialog** - Simple means no irrelevant or rarely used information. Natural means an order that matches the task.
- **Speak the user's language** - Use words and concepts from the user's world. Don't use system-specific engineering terms.
- **Minimize user memory load** - Don't make the user remember things from one action to the next. Leave information on the screen until it's not needed.
- **Be consistent** - Users should be able to learn an action sequence in one part of the system and apply it again to get similar results in other places.
- **Provide feedback** - Let users know what effect their actions have on the system.
- **Provide clearly marked exits** - If users get into part of the system that doesn't interest them, they should always be able to get out quickly without damaging anything.
- **Provide shortcuts** - Shortcuts can help experienced users avoid lengthy dialogs and informational messages that they don't need.
- **Good error messages** - Good error messages let the user know what the problem is and how to correct it.
- **Prevent errors** - Whenever you write an error message you should also ask, can this error be avoided?

The procedure works. Nielsen and Molich have shown that the combined list of interface problems includes many more problems than any single evaluator would identify, and with just a few evaluators it includes most of the major problems with the interface. Major problems, here, are problems that confuse users or cause them to make errors. The list will also include less critical problems that only slow or inconvenience the user.

How many evaluators are needed to make the analysis work? That depends on how knowledgeable the evaluators are. If the evaluators are experienced interface experts, then 3 to 5 evaluators can catch all of the "heuristically identifiable" major problems, and they can catch 75 percent of the total heuristically identifiable problems. (We'll explain what "heuristically identifiable" means in a moment.) These experts might be people who've worked in interface design and evaluation for several years, or who have several years of graduate training in the area. For evaluators who are also specialists in the

specific domain of the interface (for example, graphic interfaces, or voice interfaces, or automated teller interfaces), the same results can probably be achieved with 2 to 3 evaluators. On the other hand, if the evaluators have no interface training or expertise, it might take as many as 15 of them to find 75 percent of the problems; 5 of these novice evaluators might find only 50 percent of the problems.

We need to caution here that when we say "all" or "75 percent" or "50 percent," we're talking only about "heuristically identifiable" problems. That is, problems with the interface that actually violate one of the nine heuristics. What's gained by combining several evaluators' results is an increased assurance that if a problem can be identified with the heuristics, then it will be. But there may still be problems that the heuristics themselves miss. Those problems might show up with some other evaluation method, such as user testing or a more task-oriented analysis.

Also, all the numbers are averages of past results, not promises. Your results will vary with the interface and with the evaluators. But even with these caveats, the take-home message is still very positive: Individual heuristic evaluations of an interface, performed by 3 to 5 people with some expertise in interface design, will locate a significant number of the major problems.

To give you a better idea of how Nielsen and Molich's nine heuristics apply, one of the authors has done a heuristic evaluation of the Macintosh background printing controls.

4.0 CONCLUSION

The concept of evaluation of user interface without the users was introduced in this unit. Users walkthrough, action analysis and heuristics analysis concepts was also discussed in greater detail while emphasis on the need for evaluation was also introduced.

5.0 SUMMARY

- The cognitive walkthrough is a formalized way of imagining people's thoughts and actions when they use an interface for the first time.
- Action analysis is an evaluation procedure that forces you to take a close look at the sequence of actions a user has to perform to complete a task with an interface. In this unit, we will distinguish between two flavors of action analysis.
- The formal approach to action analysis has been used to make accurate predictions of the time it takes a skilled user to complete tasks.
- Heuristics, also called guidelines, are general principles or rules of thumb that can guide design decisions.

6.0 TUTOR MARKED ASSIGNMENT

a. Explain the term heuristics.

b. Explain the Nielsen and Molichs heuristics.

7.0 REFERENCES AND FURTHER READING

Nielsen, J. and Molich, R. "Heuristic evaluation of user interfaces." Proc. CHI'90 Conference on Human Factors in Computer Systems. New York: ACM, 1990, pp. 249-256.

Nielsen, J. "Finding usability problems through heuristic evaluation." Proc. CHI'92 Conference on Human Factors in Computer Systems. New York: ACM, 1992, pp. 373-380.

Molich, R. and Nielsen, J. "Improving a human-computer dialogue: What designers know about traditional interface design." Communications of the ACM, 33 (March 1990), pp. 338-342.

Nielsen, J. "Usability Engineering." San Diego, CA: Academic Press, 1992.

Wharton, C., Rieman, J., Lewis, C., and Polson, P. "The cognitive walkthrough: A practitioner's guide." In J. Nielsen and R.L. Mack (Eds.), Usability Inspection Methods, New York:JohnWiley&Sons(1994).

MODULE 4

UNIT 3 EVALUATING THE DESIGN WITH THE USERS

TableofContents

1.0 INTRODUCTION

Having read through the course guide, you will be introduced to evaluating designs with users. Choosing the users to test, getting the users to know what to do, providing the necessary systems and data needed for the test are the various necessary processes in this evaluation and they will be discussed in detail.

2.0 OBJECTIVES

By the end this unit, you should be able to:

- Explain how to choose users.
- Highlight the significance of the user's presence.
- Explain the necessary process in evaluating with users.

3.0 MAIN CONTENT

3.1 EVALUATING WITH THE USERS

You can not really tell how well or bad your interface is going to be without getting people to use it. So as your design matures, but before the whole system gets set in concrete, you need to do some user testing. This means having real people try to do things with the system and observing what happens. To do this you need people, some tasks for them to perform, and some version of the system for them to work with. Let us consider these necessities in order.

3.1.1 CHOOSING USERS TO TEST

The point of testing is to anticipate what will happen when real users start using your system. So the best test users will be people who are representative of the people you expect to have as users. If the real users are supposed to be doctors, get doctors as test

users. If you do not, you can be badly misled about crucial things like the right vocabulary to use for the actions your system supports. Yes, we know it isn't easy to get doctors, as we noted when we talked about getting input from users early in design. But that doesn't mean it isn't important to do. And, as we asked before, if you can't get any doctors to be test users, why do you think you will get them as real users?

If it is hard to find really appropriate test users you may want to do some testing with people who represent some approximation to what you really want, like medical students instead of doctors, say, or maybe even premeds, or college- educated adults. This may help you get out some of the big problems (the ones you overlooked in your cognitive walkthrough because you knew too much about your design and assumed some things were obvious that aren't). But you have to be careful not to let the reactions and comments of people who aren't really the users you are targeting drive your design. Do as much testing with the right kind of test users as you can.

3.1.2 GETTING THE USERS TO KNOW WHAT TO DO

In your test, you will be giving the test users some things to try to do, and you will be keeping track of whether they can do them. Just as good test users should be typical of real users, so test tasks should reflect what you think real tasks are going to be like. If you have been following our advice you already have some suitable tasks: the tasks you developed early on to drive your task-centered design.

You may find you have to modify these tasks somewhat for use in testing. They may take too long, or they may assume particular background knowledge that a random test user will not have. So you may want to simplify them. But be careful in doing this! Try to avoid any changes that make the tasks easier or that bend the tasks in the direction of what your design supports best.

If you base your test tasks on the tasks you developed for task-centered design, you'll avoid a common problem: choosing test tasks that are too fragmented. Traditional requirements lists naturally give rise to suites of test tasks that test the various requirements separately.

3.1.3 PROVIDING A SYSTEM FOR TEST USERS TO USE

The key to testing early in the development process, when it is still possible to make changes to the design without incurring big costs, is using mockups in the test. These are versions of the system that do not implement the whole design, either in what the interface looks like or what the system does, but do show some of the key features to users. Mockups blur into PROTOTYPES, with the distinction that a mockup is rougher and cheaper and a prototype is more finished and more expensive.

The simplest mockups are just pictures of screens as they would appear at various stages of a user interaction. These can be drawn on paper or they can be, with a bit more work, created on the computer using a tool like HyperCard for the Mac or a similar system for Windows. A test is done by showing users the first screen and asking them what they

would do to accomplish a task you have assigned. They describe their action, and you make the next screen appear, either by rummaging around in a pile of pictures on paper and holding up the right one, or by getting the computer to show the appropriate next screen.

This crude procedure can get you a lot of useful feedback from users. Can they understand what's on the screens, or are they baffled? Is the sequence of screens well-suited to the task, as you thought it would be when you did your cognitive walkthrough, or did you miss something?

To make a simple mockup like this you have to decide what screens you are going to provide. Start by drawing the screens users would see if they did everything the best way. Then decide whether you also want to "support" some alternative paths, and how much you want to investigate error paths. Usually it won't be practical for you to provide a screen for every possible user action, right or wrong, but you will have reasonable coverage of the main lines you expect users to follow.

During testing, if users stay on the lines you expected, you just show them the screens they would see. What if they deviate, and make a move that leads to a screen you don't have? First, you record what they wanted to do: that is valuable data about a discrepancy between what you expected and what they want to do, which is why you are doing the test. Then you can tell them what they would see, and let them try to continue, or you can tell them to make another choice. You won't see as much as you would if you had the complete system for them to work with, but you will see whether the main lines of your design are sound.

Some systems have to interact too closely with the user to be well approximated by a simple mockup. For example a drawing program has to respond to lots of little user actions, and while you might get information from a simple mockup about whether users can figure out some aspects of the interface, like how to select a drawing tool from a palette of icons, you won't be able to test how well drawing itself is going to work. You need to make more of the system work to test what you want to test.

The thing to do here is to get the drawing functionality up early so you can do a more realistic test. You would not wait for the system to be completed, because you want test results early. So you would aim for a prototype that has the drawing functionality in place but does not have other aspects of the system finished off.
In some cases, you can avoid implementing stuff early by faking the implementation. This is the WIZARD OF OZ method: you get a person to emulate unimplemented functions and generate the feedback users should see. John Gould at IBM did this very effectively to test design alternatives for a speech transcription system for which the speech recognition component was not yet ready. He built a prototype system in which test users' speech was piped to a fast typist, and the typist's output was routed back to the test users' screen. This idea can be adapted to many situations in which the system you are testing needs to respond to unpredictable user input, though not to interactions as dynamic as drawing.

If you are led to develop more and more elaborate approximations to the real system for testing purposes you need to think about controlling costs. Simple mockups are cheap, but prototypes that really work, or even Wizard of Oz setups, take substantial implementation effort.

Some of this effort can be saved if the prototype turns out to be just part of the real system. As we will discuss further when we talk about implementation, this is often possible. A system like Visual Basic or Hypercard allows an interface to be mocked up with minimal functionality but then hooked up to functional modules as they become available. So don't plan for throwaway prototypes: try instead to use an implementation scheme that allows early versions of the real interface to be used in testing.

3.1.4 DECIDING WHAT DATA TO COLLECT

Now that we have people, tasks, and a system, we have to figure out what information to gather. It is useful to distinguish PROCESS DATA from BOTTOM-LINE data. Process data are observations of what the test users are doing and thinking as they work through the tasks. These observations tell us what is happening step-by-step, and, we hope, suggests WHY it is happening. Bottom-line data give us a summary of WHAT happened: how long did users take, were they successful, how many errors did they make.
It may seem that bottom-line data are what you want. If you think of testing as telling you how good your interface is, it seems that how long users are taking on the tasks, and how successful they are, is just what you want to know.

We argue that process data are actually the data to focus on first. There's a role for bottom-line data, as we discuss in connection with Usability Measurement below. But as a designer you will mostly be concerned with process data. To see why, consider the following not-so-hypothetical comparison.

Suppose you have designed an interface for a situation in which you figure users should be able to complete a particular test task in about a half-hour. You do a test in which you focus on bottom-line data. You find that none of your test users was able to get the job done in less than an hour. You know you are in trouble, but what are you going to do about it? Now suppose instead you got detailed records of what the users actually did. You see that their whole approach to the task was mistaken, because they didn't use the frammis reduction operations presented on the third screen. Now you know where your redesign effort needs to go.

We can extend this example to make a further point about the information you need as a designer. You know people weren't using frammis reduction, but do you know why? It could be that they understood perfectly well the importance of frammis reduction, but they didn't understand the screen on which these operations were presented. Or it could be that the frammis reduction screen was crystal clear but they didn't think frammis reduction was relevant.

Depending on what you decide here, you either need to fix up the frammis reduction screen, because it isn't clear, or you have a problem somewhere else. But you can't decide just from knowing that people didn't use frammis reduction. To get the why information you really want, you need to know what users are thinking, not just what they are doing. That's the focus of the thinking-aloud method, the first testing technique we'll discuss.

3.1.5 CHOOSING USERS TO TEST

The basic idea of thinking aloud is very simple. You ask your users to perform a test task, but you also ask them to talk to you while they work on it. Ask them to tell you what they are thinking: what they are trying to do, questions that arise as they work, things they read. You can make a recording of their comments or you can just take notes. You'll do this in such a way that you can tell what they were doing and where their comments fit into the sequence.

You'll find the comments are a rich lode of information. In the frammis reduction case, with just a little luck, you might get one of two kinds of comments: "I know I want to do frammis reduction now, but I don't see anyway to do it from here. I'll try another approach," or "Why is it telling me about frammis reduction here? That's not what I'm trying to do." So you find out something about WHY frammis reduction wasn't getting done, and whether the frammis reduction screen is the locus of the problem.
You can use the thinking-aloud method with a prototype or a rough mock-up, for a single task or a suite of tasks. The method is simple, but there are some points about it that repay some thought. Here are some suggestions on various aspects of the procedure. This material is adapted from Lewis, C. "Using the thinking-aloud method in cognitive interface design," IBM Research Report RC 9265, Yorktown Heights, NY, 1982.

a. Instructions

The basic instructions can be very simple: "Tell me what you are thinking about as you work." People can respond easily to this, especially if you suggest a few categories of thoughts as examples: things they find confusing, decisions they are making, and the like. There are some other points you should add. Tell the user that you are not interested in their secret thoughts but only in what they are thinking about their task. Make clear that it is the system, not the user, that is being tested, so that if they have trouble it's the system's problem, not theirs. You will also want to explain what kind of recording you will make, and how test users' privacy will be protected.

b. The Role of the Observer

Even if you do not need to be available to operate a mockup, you should plan to stay with the user during the test. You'll do two things: prompt the user to keep up the flow of comments, and provide help when necessary. But you'll need to work out a policy for prompting and helping that avoids distorting the results you get.

It's very easy to shape the comments users will give you, and what they do in the test, by asking questions and making suggestions. If someone has missed the significance of some interface feature a word from you may focus their attention right on it. Also, research shows that people will make up an answer to any question you ask, whether or not they have any basis for the answer. You are better off, therefore, collecting the comments people offer spontaneously than prodding them to tell you about things you are interested in.

On helping, keep in mind that a very little help can make a huge difference in a test, and you can seriously mislead yourself about how well your interface works by just dropping in a few suggestions here and there. Try to work out in advance when you will permit yourself to help. One criterion is: help only when you won't get any more useful information if you don't, because the test user will quit or cannot possibly continue the task. If you do help, be sure to record when you helped and what you said.
A consequence of this policy is that you have to explain to your test users that you want them to tell you the questions that arise as they work, but that you won't answer them. This seems odd at first but becomes natural after a bit.

c. Recording

There are plain and fancy approaches here. It is quite practical to record observations only by taking notes on a pad of paper: you write down in order what the user does and says, in summary form. But you'll find that it takes some experience to do this fast enough to keep up in real time, and that you won't be able to do it for the first few test users you see on a given system and task. This is just because you need a general idea of where things are going to be able to keep up. A step up in technology is to make a video record of what is happening on the screen, with a lapel mike on the user to pick up the comments. A further step is to instrument the system to pick up a trace of user actions, and arrange for this record to be synchronized in some way with an audio record of the comments. The advantage of this approach is that it gives you a machine readable record of user actions that can be easier to summarize and access than video.
A good approach to start with is to combine a video record with written notes. You may find that you are able to dispense with the video, or you may find that you really want a fancier record. You can adapt your approach accordingly. But if you do not have a video setup, do not let that keep you from trying the method.

d. Summarizing the Data

The point of the test is to get information that can guide the design. To do this you will want to make a list of all difficulties users encountered. Include references back to the original data so you can look at the specifics if questions arise. Also try to judge why each difficulty occurred, if the data permit a guess about that.

e. Using the Results

Now you want to consider what changes you need to make to your design based on data from the tests. Look at your data from two points of view. First, what do the data tell you about how you THOUGHT the interface would work? Are the results consistent with your cognitive walkthrough or are they telling you that you are missing something? For example, did test users take the approaches you expected, or were they working a different way? Try to update your analysis of the tasks and how the system should support them based on what you see in the data. Then use this improved analysis to rethink your design to make it better support what users are doing.

Second, look at all of the errors and difficulties you saw. For each one make a judgement of how important it is and how difficult it would be to fix. Factors to consider in judging importance are the costs of the problem to users (in time, aggravation, and possible wrong results) and what proportion of users you can expect to have similar trouble. Difficulty of fixes will depend on how sweeping the changes required by the fix are: changing the wording of a prompt will be easy, changing the organization of options in a menu structure will be a bit harder, and so on. Now decide to fix all the important problems, and all the easy ones.

3.1.6 MEASURING BOTTOM-LINE USABILITY

There are some situations in which bottom-line numbers are useful. You may have a definite requirement that people must be able to complete a task in a certain amount of time, or you may want to compare two design alternatives on the basis of how quickly people can work or how many errors they commit. The basic idea in these cases is that you will have people perform test tasks, you will measure how long they take and you will count their errors.

Your first thought may be to combine this with a thinking- aloud test: in addition to collecting comments you'd collect these other data as well. Unfortunately this doesn't work as well as one would wish. The thinking-aloud process can affect how quickly and accurately people work. It's pretty easy to see how thinking-aloud could slow people down, but it has also been shown that sometimes it can speed people up, apparently by making them think more carefully about what they are doing, and hence helping them choose better ways to do things. So if you are serious about finding out how long people will take to do things with your design, or how many problems they will encounter, you really need to do a separate test.
Getting the bottom-line numbers won't be too hard. You can use a stopwatch for timings, or you can instrument your system to record when people start and stop work. Counting errors, and gauging success on tasks, is a bit trickier, because you have to decide what an error is and what counts as successful completion of a task. But you won't have much trouble here either as long as you understand that you can't come up with perfect criteria for these things and use your common sense.

a. Analyzing the Bottom-Line Numbers

When you've got your numbers you'll run into some hard problems. The trouble is that the numbers you get from different test users will be different. How do you combine these numbers to get a reliable picture of what's happening?

Suppose users need to be able to perform some task with your system in 30 minutes or less. You run six test users and get the following times:

20 min
15 min
40 min
90 min
10 min
5 min

Are these results encouraging or not? If you take the average of these numbers you get 30 minutes, which looks fine. If you take the MEDIAN, that is, the middle score, you get something between 15 and 20 minutes, which look even better. Can you be confident that the typical user will meet your 30-minute target?

The answer is no. The numbers you have are so variable, that is, they differ so much among themselves, that you really can't tell much about what will be "typical" times in the long run. Statistical analysis, which is the method for extracting facts from a background of variation, indicates that the "typical" times for this system might very well be anywhere from about 5 minutes to about 55 minutes. Note that this is a range for the "typical" value, not the range of possible times for individual users. That is, it is perfectly plausible given the test data that if we measured lots and lots of users the average time might be as low as 5 min, which would be wonderful, but it could also be as high as 55 minutes, which is terrible.

There are two things contributing to our uncertainty in interpreting these test results. One is the small number of test users. It's pretty intuitive that the more test users we measure the better an estimate we can make of typical times. Second, as already mentioned, these test results are very variable: there are some small numbers but also some big numbers in the group. If all six measurements had come in right at (say) 25 minutes, we could be pretty confident that our typical times would be right around there. As things are, we have to worry that if we look at more users we might get a lot more 90-minute times, or a lot more 5-minute times.

It is the job of statistical analysis to juggle these factors -- the number of people we test and how variable or consistent the results are -- and give us an estimate of what we can conclude from the data. This is a big topic, and we won't try to do more than give you some basic methods and a little intuition here.

Here is a cookbook procedure for getting an idea of the range of typical values that are consistent with your test data.

Add up the numbers. Call this result "sum of x". In our example this is 180.

Divide by the n, the number of numbers. The quotient is the average, or mean, of the measurements. In our example this is 30.

Add up the squares of the numbers. Call this result "sum of squares" In our example this is 10450.

Square the sum of x and divide by n. Call this "foo". In our example this is 5400.

Subtract foo from sum of squares and divide by n-1. In our example this is 1010.

Take the square root. The result is the "standard deviation" of the sample. It is a measure of how variable the numbers are. In our example this is 31.78, or about 32. 7. Divide the standard deviation by the square root of n.

This is the "standard error of the mean" and is a measure of how much variation you can expect in the typical value. In our example this is 12.97, or about 13.

I

t is plausible that the typical value is as small as the mean minus two times the standard error of the mean, or as large as the mean plus two times the standard error of the mean. In our example this range is from 30-(2*13) to 30+(2*13), or about 5 to 55. (The "*" stands for multiplication.)

What does "plausible" mean here? It means that if the real typical value is outside this range, you were very unlucky in getting the sample that you did. More specifically, if the true typical value were outside this range you would only expect to get a sample like the one you got 5 percent of the time or less.

Experience shows that usability test data are quite variable, which means that you need a lot of it to get good estimates of typical values. If you pore over the above procedure enough you may see that if you run four times as many test users you can narrow your range of estimates by a factor of two: the breadth of the range of estimates depends on the square root of the number of test users. That means a lot of test users to get a narrow range, if your data are as variable as they often are.

What this means is that you can anticipate trouble if you are trying to manage your project using these test data. Do the test results show we are on target, or do we need to pour on more resources? It's hard to say. One approach is to get people to agree to try to manage on the basis of the numbers in the sample themselves, without trying to use statistics to figure out how uncertain they are. This is a kind of blind compromise: on the average the typical value is equally likely to be bigger than the mean of your sample, or smaller. But if the stakes are high, and you really need to know where you stand, you'll need to do a lot of testing. You'll also want to do an analysis that takes into account the cost to you of being wrong about the typical value, by how much, so you can decide how big a test is really reasonable.

b. Comparing Two Design Alternatives

If you are using bottom-line measurements to compare two design alternatives, the same considerations apply as for a single design, and then some. Your ability to draw a firm conclusion will depend on how variable your numbers are, as well as how many test users

you use. But then you need some way to compare the numbers you get for one design with the numbers from the others.

The simplest approach to use is called a BETWEEN-GROUPS EXPERIMENT. You use two groups of test users, one of which uses version A of the system and the other version B. What you want to know is whether the typical value for version A is likely to differ from the typical value for version B, and by how much. Here's a cookbook procedure for this.

Using parts of the cookbook method above, compute the means for the two groups separately. Also compute their standard deviations. Call the results ma, mb, sa, sb. You'll also need to have na and nb, the number of test users in each group (usually you'll try to make these the same, but they don't have to be.)

Combine sa and sb to get an estimate of how variable the whole scene is, by computing

s = sqrt((na*(sa**2) + nb*(sb**2)) / (na + nb - 2))
("*" represents multiplication; "sa**2" means "sa squared").
Compute a combined standard error:

se = s * sqrt(1/na + 1/nb)

Your range of typical values for the difference between version A and version B is now:
ma - mb plus-or-minus 2*se

Another approach you might consider is a WITHIN-GROUPS EXPERIMENT. Here you use only one group of test users and you get each of them to use both versions of the system. This brings with it some headaches. You obviously can't use the same tasks for the two versions, since doing a task the second time would be different from doing it the first time, and you have to worry about who uses which system first, because there might be some advantage or disadvantage in being the system someone tries first. There are ways around these problems, but they aren't simple. They work best for very simple tasks about which there are not much to learn. You might want to use this approach if you were comparing two low-level interaction techniques, for example. You can learn more about the within-groups approach from any standard text on experimental psychology.

3.1.7 DETAILS OF SETTING UP A USABILITY STUDY

The description of user testing given up to this point should be all the background you need during the early phases of a task-centered design project. When you are actually ready to evaluate a version of the design with users, you will have to consider some of the finer details of setting up and running the tests. This section, which you may want to skip on the first reading of the chapter, will help with many of those details.

a. Choosing the Order of Test Tasks

Usually you want test users to do more than one task. This means they have to do them in some order. Should everyone do them in the same order, or should you scramble them, or what? Our advice is to choose one sensible order, starting with some simpler things and working up to some more complex ones, and stay with that. This means that the tasks that come later will have the benefit of practice on the earlier one, or some penalty from test users getting tired, so you can't compare the difficulty of tasks using the results of a test like this. But usually that is not what you are trying to do.

b. Training Test Users

Should test users hit your system cold or should you teach them about it first? The answer to this depends on what you are trying to find out, and the way your system will be used in real life. If real users will hit your system cold, as is often the case, you should have them do this in the test. If you really believe users will be pre-trained, then train them before your test. If possible, use the real training materials to do this: you may as well learn something about how well or poorly it works as part of your study.

c. The Pilot Study

You should always do a pilot study as part of any usability test. A pilot study is like a dress rehearsal for a play: you run through everything you plan to do in the real study and see what needs to be fixed. Do this twice, once with colleagues, to get out the biggest bugs, and then with some real test users. You'll find out whether your policies on giving assistance are really workable and whether your instructions are adequately clear. A pilot study will help you keep down variability in a bottom-line study, but it will avoid trouble in a thinking-aloud study too. Don't try to do without!

d. What If Someone Does not Complete a Task?

If you are collecting bottom-line numbers, one problem you will very probably encounter is that not everybody completes their assigned task or tasks within the available time, or without help from you. What do you do? There is no complete remedy for the problem. A reasonable approach is to assign some very large time, and some very large number of errors, as the "results" for these people. Then take the results of your analysis with an even bigger grain of salt than usual.

e. Keeping Variability Down

As we've seen, your ability to make good estimates based on bottom-line test results depends on the results not being too variable. There are things you can do to help, though these may also make your test less realistic and hence a less good guide to what will happen with your design in real life. Differences among test users is one source of variable results: if test users differ a lot in how much they know about the task or about the system you can expect their time and error scores to be quite different. You can try to

recruit test users with more similar background, and you can try to brief test users to bring them close to some common level of preparation for their tasks.

Differences in procedure, how you actually conduct the test, will also add to variability. If you help some test users more than others, for example, you are asking for trouble. This reinforces the need to make careful plans about what kind of assistance you will provide. Finally, if people don't understand what they are doing your variability will increase. Make your instructions to test users and your task descriptions as clear as you can.

f. Debriefing Test Users

It has been stressed that it is unwise to ask specific questions during a thinking aloud test, and during a bottom-line study. But what about asking questions in a debriefing session after test users have finished their tasks? There's no reason not to do this, but do not expect too much. People often don't remember very much about problems they have faced, even after a short time. Clayton remembers vividly watching a test user battle with a text processing system for hours, and then asking afterwards what problems they had encountered. "That wasn't too bad, I don't remember any particular problems," was the answer. He interviewed a real user of a system who had come within one day of quitting a good job because of failure to master a new system; they were unable to remember any specific problem they had had. Part of what is happening appears to be that if you work through a problem and eventually solve it, even with considerable difficulty, you remember the solution but not the problem.

There is an analogy here to those hidden picture puzzles you see on kids' menus at restaurant: there are pictures of three rabbits hidden in this picture, can you find them? When you first look at the picture you can't see them. After you find them, you can not help seeing them. In somewhat the same way, once you figure out how something works it can be hard to see why it was ever confusing.

Something that might help you get more info out of questioning at the end of a test is having the test session on video so you can show the test user the particular part of the task you want to ask about. But even if you do this, don't expect too much: the user may not have any better guess than you have about what they were doing.

Another form of debriefing that is less problematic is asking for comments on specific features of the interface. People may offer suggestions or have reactions, positive or negative, that might not otherwise be reflected in your data. This will work better if you can take the user back through the various screens they've seen during the test.

4.0 CONCLUSION

In this unit, you have been introduced to evaluating designs with users. Choosing the users to test, getting the users to know what to do, providing the necessary systems and

data needed for the test are the various necessary processes evaluation that were all discussed fully.

5.0 SUMMARY

- Evaluating with users involves having real people try to do things with the system and observing what happens.
- The best test users will be people who are representative of the people you expect to have as users.
- The test tasks should reflect what you think real tasks are going to be like.
- Choosing a user also involves handling out instructions, getting an observer, recording, summarizing the data and using the result.
- Setting up usability study includes choosing the order of Test Tasks, training Test Users, the Pilot Study, e.t.c

6.0 TUTOR MARKED ASSIGNMENT

a. Explain Bottom-Line Numbers.
b. Describe a mockup.
c. How would you select test users who are true representation of the users population?

7.0 REFERENCES AND FURTHER READING

Nielsen, J. and Molich, R. "Heuristic evaluation of user interfaces." Proc. CHI'90 Conference on Human Factors in Computer Systems. New York: ACM, 1990, pp. 249-256.

Nielsen, J. "Finding usability problems through heuristic evaluation." Proc. CHI'92 Conference on Human Factors in Computer Systems. New York: ACM, 1992, pp. 373-380.

Molich, R. and Nielsen, J. "Improving a human-computer dialogue: What designers know about traditional interface design." Communications of the ACM, 33 (March 1990), pp. 338-342.

Nielsen, J. "Usability Engineering." San Diego, CA: Academic Press, 1992.

MODULE 4

UNIT 4 OTHER EVALUATION ISSUES

TableofContents

1.0 INTRODUCTION

This unit describes other evaluation issues. Advantages and disadvantages of model based evaluations techniques are discussed. Current issues concerning evaluation methodologies are also mentioned.

2.0 OBJECTIVES

By the end this unit, you should be able to:

- Explain new modeling techniques
- Highlight the advantages and disadvantages of model based evaluations
- Discuss current issues concerning evaluation methodologies

3.0 MAIN CONTENT

3.1 OTHER MODELLING TECHNIQUES

The EPIC (Executive-Process/Interactive Control) system simulates the human perceptual and motor performance system. Epic can interact as a human would with a simulation of a user interface system. EPIC is being used to study users engaged in multiple tasks, such as using a car navigation system while driving. Using EPIC involves writing production rules for using the interface and writing a task environment to simulate the behaviour of the user interface.

A model of information foraging useful in evaluating information seeking in web sites is based on the ACT-R model. The ACT-IF model was developed to use in testing simulated users interacting with designs for web sites and predicts optimal behaviour in large collections of web documents. The information foraging model is being used to

understand the decisions that users of the web make in following various links to satisfy information goals.

3.1.1 ADVANTAGES AND DISAVANTAGES OF MODEL-BASED EVALUATIONS

The use of models to predict user behaviour is less expensive than empirical, user-centered evaluations. Thus many more iterations of the design can be tested. However, a necessary first step is conducting the task-level cognitive task analysis to use in producing model description. This is time consuming but can be used for testing many user interface designs.
Models must be tested for validity. This is accomplished by watching humans perform the tasks and coding their behaviour for comparison with the model. This is time consuming but necessary to determine if the model predicts are accurate.

3.2 CURRENT ISSUES CONCERNING EVALUATION METHODOLOGIES

While the HCI community has come a long way in developing and using methods to evaluate usability, the problem is by no means solved. This chapter has described three basic methods for evaluation but there is not yet agreement in the community about which evaluation is more useful than another. Although a number of studies have been done to compare these methods, the comparison is difficult and flaws have been pointed out in a number of these studies. First, there is the issue of using experimental (user-centered) methods to obtain answers to large questions of usability as opposed to the more narrow questions that are the more traditional use for experimental methods. A second issue is what should be used for the comparison? Should user-centered methods be considered as the ground truth? All usability tests are not created equal. There are certainly flaws in the way tests are design, conducted, and analyzed. . While individual methods have limitations and can be flawed in their implementation, it is certain that performing some evaluation methodology is better than doing nothing. The current best practice is to use a number of different evaluation methodologies to provide rich data on usability.

Evaluation methodologies were, for the most part, developed to evaluate the usability of desk-top systems. The current focus in technology development of mobile and ubiquitous computing presents challenges for current usability evaluation methods. Laboratory evaluations will be hard pressed to simulate use conditions for these applications. Going out into the field to evaluate use places constraints on how early evaluations can be done. Mobile and multi-user systems must be evaluated for privacy and any usability issues entailed in setting up, configuring, and using such policies. The use of such devices in the context of doing other work also has implications for determining the context of use for usability testing. We need to test car navigation systems in the car – not the usability lab.

Technology is being used by more users. The range of users using mobile phones, for example, means that representative users need to be selected from teenagers to grandparents. The accessibility laws in the United States require that federal information is accessible by persons with disabilities. Again, this requires inclusion of more users from the disable population in user-centered evaluations.

Web sites are also of interest in usability evaluation. Again, there is a matter of a broad user population. Design and development cycles in web site development are extremely fast and doing extensive usability evaluation is usually not feasible. Usability practitioners are looking at remote testing methods to more closely replicate context of usage for web site evaluation.

International standards exist for user centered design processes, documentation, and user interfaces. Usability is becoming a requirement for companies in purchasing software as they recognize that unusable software will increase the total cost of ownership.
New usability evaluation methodologies will be developed to meet the demands of our technology-focused society. Researchers and practitioners in usability will need to join forces to meet this challenge.

4.0 CONCLUSION

In this unit, you have been introduced to evaluation issues. Advantages and disadvantages of model based evaluations techniques and current issues concerning evaluation methodologies were discussed.

5.0 SUMMARY

- EPIC (Executive-Process/Interactive Control) system simulates the human perceptual and motor performance system.
- ACT-IF model was developed to use in testing simulated users interacting with designs for web sites and predicts optimal behaviour in large collections of web documents.
- The use of models to predict user behaviour is less expensive than empirical, user-centred evaluations and this is it main advantage.
- The main disadvantage of models is that it is time consuming.

6.0 TUTOR MARKED ASSIGNMENT

What is the significance of EPIC (Executive-Process/Interactive Control) system?

7.0 REFERENCES AND FURTHER READING

Mayhew, D. 1999. *The Usability Engineering Lifecycle*. San Francisco,CA: Morgan Kauffman.

MODULE 4

UNIT 5 USABILITY TESTING PROCEDURE

TableofContents

1.0 INTRODUCTION

After designing and implementing user interface, it is important to determine the acceptability of the user interface using usability testing. The steps involved and the techniques of usability testing are discussed in this unit.

2.0 OBJECTIVES

By the end this unit, you should be able to:

- Understand usability testing
- Understand how to carry out usability testing

3.0 MAIN CONTENT

3.1 INTRODUCTION TO USABILITY TESTING

I have noticed that the term usability testing is often used rather indiscrimately to refer to any technique used to evaluate a product or system. Throughout this unit, the term usability testing is referred to as the process that employs participants who are representative of the target population to evaluate the degree to which a product (User interface) meets specific usability criteria. This inclusion of representative users eliminates labelling as usability testing such techniques as expert evaluations, walk-through, and the like that do not require representative users as part of the process.

Usability testing is a research tool, with its roots in classical experimental methodology. The range of tests one can conduct is considerable, from true classical experiments with large sample sizes and complex test designs, to very informal qualitative studies with only a single participant. Each testing approach has different objectives, as well as different time and resource requirements. The emphasis of this book will be on more informal, less complex tests designed for quick turnaround of results in industrial product development environments.

3.2 PREPARING FOR USABILITY TESTING

For many of those contemplating the implementation of the usability testing program, the disciple has become synonymous with a high-powered, well-appointed, well-equipped, expensive laboratory. For some organizations, the usability lab (and by that I mean physical plant) has become more prominent and more important than the testing process itself. Some organizations, in their zeal to impress customers and competitors alike with their commitment to usability, have created awe-inspiring palaces of high-tech wizardry prior to laying the foundation for an on-going testing program. Not realizing that instituting a program of usability engineering requires a significant shift in the culture of the organization, these organizations have put the proverbial cart before the horse, in their attempts to create instant programs, rather than building programs over time.

This approach to usability testing is rather superficial and short-sighted, and has a high risk of failure. It approaches usability engineering as the latest fad to be embraced rather than as a program that require effort, commitment, and time in order to have lasting effects on the organization and its products. I know of at least two organizations with newly built, sophisticated usability laboratories that unfortunately are now operating as the world's most elaborate storage rooms. (An analogy is a retail store that requires and outfits a new store for business, only to realize that it does not have any interested customers). Having succumbed to the misperception that equates

the laboratory with the process itself, these organizations have discovered only too late that usability testing is much more than a collection of cameras and recorders. Rather, a commitment to usability must be embedded in the very philosophy and underpinning of the organization itself in order to guarantee success.

In that vein, if you have been charged with developing a testing program and have been funded to build an elaborate testing lab as the initial step, resist the temptation to accept the offer. Rather, start small and build the organization from the ground up instead of from the top down.

Regardless of whether you will be initiating a large testing program or simply testing your own product, you need not have elaborate, expensive lab to achieve your goals.

SIX STAGES OF CONDUCTING A USABILITY TEST

3.3.1 DEVELOPING THE TEST PLAN

The test plan is the foundation for the entire test. It addresses the how, when, where, who, why, and what of your usability test. Under the sometimes unrelenting time pressure of project deadline, there could be a tendency to forego writing a detailed test plan. Perhaps, feeling that you have a good idea of what you would like to test in your head, you decide not to bother writing it down. This informal approach is a mistake, and invariably will come back to haunt you. Following are some important reasons why it is necessary to develop a comprehensive test plan, as well as some ways to use it as a communication vehicle among the development team.

It serves as the blueprint for the test. Much as the blueprint for a house describes exactly what you will build, the test plan describes exactly how you will go about testing your product. Just as you would not want your building contractor to "wing it" when building your house, so the exact same logic applies here. The test plan sets the stage for all that will follow. You do not want to have any loose ends just as you are about to test your first participant.

It serve as the main communication vehicle among the main developer, the test monitor, and the rest of the development team. The test plan is the document that all involved member of the development team as well as the management (if it is interested and involved) should review in order to understand how the test will proceed and see whether their particular needs are being met. Use it to get buy-in and feedback from other members to ensure that everyone agrees on what will transpire. Since projects are dynamic and change from day to day and from week to week, you do not want someone to say at the end of the test that his or her particular agenda was not addressed. Especially when your organization is first starting to test, everyone who is directly affected by the test results should review the test plan. This makes good business sense and political sense too.

It describes or implies required resources, both internal and external. Once you delineate exactly what will happen and when, it is a much easier task to foretell what you will need to accomplish your test. Either directly or by implication, the test plan should communicate the resources that are required to complete the test successfully.

It provides a real focal point for the test and a milestone for the product being tested. Without the test plan, details get fuzzy and ambiguous, especially under time pressure. The test plan forces you to approach the job of testing systematically, and it reminds the development team of the impending dates. Having said all that, it is perfectly acceptable and highly probable that the test plan will be developed in stages s you gradually understand more of the test objectives and talk to the people who will be involved. Projects are dynamic and the best laid plans will change as you begin to approach testing. By developing the test plan in stages, you can accommodate changes.

For example, as your time and resource constraints become clearer, your test may become less ambitious and simpler. Or, perhaps you will not be able to acquire as many qualified participants as you thought. Perhaps not all modules or section of the document will ready in time. Perhaps your test objectives are too imprecise and need to be simplified and focused. These are all real-world example that force you to revise the test and the test plan.

A sound approach is to start writing the test plan as soon as you know you will be testing. Then, as the project proceeds, continue to refine it, get feedback, buy-in, and so forth. Of course, there is a limit to flexibility, so you need to set a reasonable deadline prior to the test after which the test plan may not change. Let that date serve as a point at which the product can no longer change until after the test. You may find that the test is the only concrete milestone at that point in time in the development cycle and, as such, serves an important function.

Once you reach the cut-off date, do all that you can to freeze the design of the product you will be have to test. Additional revisions may invalidate the test design you have chosen, the questions you ask, even the way you collect data. If you are pressured to revise the test after the cut-off date, make sure everyone understand the risks involved. The test may be invalidated, and the product may not work properly with changes made so close to the test date.

Remember to keep the end user in mind as you develop the test plan. If you are very close to the project, there is a tendency to forget that you are not testing the product- you are testing its relationship to a human being with certain specific characteristics.

SUGGESTED FORMAT

Test plan formats will vary according to the type of test and the degree of formality required in your organization. However, following are the typical sections to include;

- Purpose
- Problem statement/test objectives
- User profile
- Method (test design)
- Task list
- Test environment/equipment
- Test monitor role
- Evaluation measures (data to be collected)

- Report contents and presentation

3.3.2 SELECTING AND ACQUIRING PARTICIPANTS

The selection and acquisition of participant whose background and abilities are representative of your product‘s intended end user is a crucial element of the testing process. After all, your test result will only be valid if the people you test are typical end users of the product, or as close to that criterion as possible. If you test the “wrong” people, it does not matter how much effort you put into the rest of the test preparation. Your result will be questionable and of limited value.

Selecting participants involves identifying and describing the relevant skills and knowledge of the person(s) who will use your product. This description is known as user profile or user characterization of the target population and should have been developed in the earl stages of the product development. Then, once that has been determined, you must ascertain the most effective way to acquire people from this target population to serve as participants within your constraints of time, money, resources, and so on.

3.3.3 PREPARING THE TEST MATERIALS

One of the more labour-intensive activities required to conduct a usability test is developing the test material that will be used to communicate with the participants, collect the data, and satisfy legal requirements. It is important to develop all important test materials well in advance of the time you will need them. Apart from the obvious benefit of not having to scurry around at the last minute, developing materials early on helps to explicitly structure and organize the test. In fact, if you have difficulty developing one particular type of test material, it can be a sign that there are flaws in your test objectives and test design.

While the specific content of the materials will vary from test to test, the general categories required will hardly vary at all. This chapter contains a list of the most common materials you need to develop a test, as well as examples of the various types of test materials. As you develop them, think of these materials as aids to the testing process. Once they are developed, their natural flow will guide the test for you. Be sure to leave enough time to include the materials in your pilot test. The test materials reviewed in this chapter are as follows:

Screening questionnaire
Orientation script
Background questionnaire
Data collection instruments (data loggers)
Nondisclosure agreement and tape consent form
Pre-test questionnaire
Task scenarios
Prerequisite training materials
Post-test questionnaire
Debriefing topics guide

3.3.4 CONDUCTING THE TEST

Having completed the basic groundwork and preparation for your usability test, you are almost ready to begin testing. While there exit an almost endless variety of sophisticated esoteric tests one might conduct (from a test comprising a single participant and lasting several days to a fully automated test with 200 or more participants), in this chapter I will focus on the guidelines and activities for conducting classic "one-on-one" test. This "typical" test consists of four to ten participants, each of whom is observed and questioned individually by a test monitor seating in the same room. This method will work for any of the four types of tests mentioned: exploratory, assessment, validation, or comparison. The main difference is the types of objectives pursued, that is, more conceptual for an exploratory test, more behaviour oriented for assessment and validation tests. The other major difference is the amount of interaction between participant and test monitor. The earlier exploratory test will have much interaction. The later validation test will have much less interaction, since the objective is measurement against standard.

For "first time" testers, I recommend beginning with an assessment test; it is probably the most straightforward to conduct. At the end of this chapter, I will review several variations and enhancements to the basic testing technique that you can employ as you gain confidence.

In terms of what to test, I would like to raise an issue previously mentioned in chapter 2, because it is crucial. That is, the importance of testing the whole integrated product and not just separate components. Testing a component, such as documentation, separately, without ever testing it with the rest of the product, does nothing to ensure ultimate product usability. Rather it enforces the lack of product integration. In short, you eventually would like to test all components together, with enough lead time to make revisions as required. However, that being said, there is absolutely nothing wrong with testing separate components as that are developed throughout the life cycle, as long as you eventually test them all together.

There is one exception to this rule, if you believe that the only way to begin any kind of testing program within an organization is to test a component separately as your only test, then by all means do so. However, you should explain to management the limited nature of those results.

3.3.5 DEBRIEFING THE PARTICIPANT

Debriefing refers to the interrogation and review with the participant of his or her own actions during the performance portion of a usability test. When first sitting down to organize this book, I was not sure whether to assign debriefing to its own stage of testing or to combine it with the previous stage of conducting the test. After all, one could argue that debriefing is really an extension of the testing process. Participants perform some tasks, and you interrogate that either in phases or after the entire test.

But the more I thought about how much I had learned during the debriefing portions of tests, the more I felt that debriefing warranted its own separate treatment, or stage of testing. More often the not, the debriefing session is the key to understanding how

to fix the problems uncovered during the performance portion of the test. While the performance of the usability test uncovers and exposes the problems, it is often the debriefing session that shed light on why these problems have occurred. Quite often, it is not until the debriefing session that one understands motive, rationale, and very subtle points of confusion. If you think of usability testing as a mystery to be solved, it is not until the debriefing session that all the pieces come together.

3.3.6 **TRANSFORMING DATA INTO FINDINGS AND RECOMMENDATIONS**

Finally, you have completed testing and are now ready to dive in and transform a wealth of data into recommendations for improvement. Typically, the analysis of data falls into two distinct processes with two different deliverables.

The first process is a preliminary analysis and is intended to quickly ascertain the hot spots (i.e., word problems), so that the designers can work on these immediately without having to wait for the final test report. This preliminary analysis takes place as soon as it is feasible after testing has been completed. Its deliverable is either a small written report or a verbal presentation of findings and recommendation.

The second process is a comprehensive analysis, which takes place during a two-to-four-week period after the test. Its deliverables is a final, more exhaustive report. This final report should include all the findings in the preliminary report, updated if necessary, plus all the other analysis and findings that were not previously covered.

A word of caution is in order regarding preliminary findings and recommendations. Developing and reporting preliminary recommendations creates a predicament for the test monitor. Your recommendations must be timely so that members of the development team, such as designers and writers, can begin implementing changes. However, you also need to be thorough, in the sense of not missing anything important. Once preliminary recommendations are circulated for public consumption, they quickly lose their preliminary flavour. Designers will begin to implement changes, and it is difficult to revisit changes at a later time and say, “Oops, I don’t really think we should change that module after all.

You could simply avoid producing preliminary recommendations, but if designers viewed the tests they are sure to act on what they saw prior to your final report, therefore not providing preliminary recommendations is not a satisfactory option. The best compromise is to provide preliminary findings and recommendations, but be cautious and err on the conservative side by providing too little rather than too much. Stick to very obvious problems that do not require further analysis on your part. If you are unsure about a finding or a recommended solution without performing further analysis, simply say so.

4.0 CONCLUSION

In this unit, you have been introduced to the stages involved in carrying out usability testing.

5.0 SUMMARY

The stages involved in usability testing are:-

- Develop the test plan
- Selecting and acquiring participants
- Preparing the test materials
- Conducting the test
- Debriefing the participants
- Transforming data into findings and recommendations

6.0 TUTOR MARKED ASSIGNMENT

a. What are the goals of usability testing

b. Justify whether the usability testing steps described in this unit are adequate.

7.0 REFERENCES AND FURTHER READING

Mayhew, D. 1999. *The Usability Engineering Lifecycle*. San Francisco,CA: Morgan Kauffman.

Rubin, J. (1994). "Handbook of Usability Testing", John Wiley & Sons Inc.