NATIONAL OPEN UNIVERSITY OF NIGERIA

CIT 811

User Interface Design and Erogonomics

Module 4

CIT 811 User Interface Design and Ergonomics Module 4

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

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

Programme Leader
Prof. MoniOluwa Olaniyi

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

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

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

Unit 1 Techniques for Evaluating and Measuring Interface Usability

1.0 Introduction

This unit describes usability evaluation methods. Cognitive modelling, inspection, inquiry and prototyping methods are described.

2.0 Objectives

At the end of this unit, you should be able to:
- explain various usability evaluation methods
- explain the differences between these 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. Usability methods can be further classified into the following subcategories:

3.1.1 Cognitive Modelling 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:

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.

GOMS
GOMS is an acronym that stands for Goals, Operator, Methods, and Selection Rules. It is a family of techniques that analyses 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.

Human Processor Model
Sometimes it is useful to break a task down and analyse 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.

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 that can be timed and recorded.

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 organise 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 organise 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 organised on the site in a way that is logical to users.

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.

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 recognised 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.

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.

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.

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.

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).

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 summarise opinions. The data gathered is not usually quantitative, but can help get an idea of a target group's opinion.

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.

3.1.3 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.

3.1.4 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.

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.

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 verbalise 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.

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 verbalised 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.

3.1.5 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.

Benchmarking
Benchmarking creates standardised 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.

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 techniques to provide very accurate quantitative support.

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 of how carefully a system is designed, all techniques 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 behaviour, 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, are often 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 formalised. 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. The tasks are measured by:

- the percentage of the task completed
- 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 analysing 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 behaviour 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 problem, but provide modified design guidelines for continued testing.

4.0 Conclusion

Usability is now recognised as an important software quality attribute, earning its place among more traditional attributes such as performance and robustness. Indeed, various academic programs focus on usability. Also several usability consultancy companies have emerged, and traditional consultancy and design firms are offering similar services.

5.0 Summary

In this unit, you have learnt the following:

- cognitive modelling involves 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
- inquiry usability evaluation methods involve collecting qualitative data from users. Although the data collected are subjective, they provide 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 Self-Assessment Exercise

1. Explain cognitive modelling.
2. Describe GOMS briefly.
3. Identify some metrics and use them to evaluate the main menu interface of WINDOWS OS.

7.0 References/Further Reading

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

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. ISBN 8254701741.

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.

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

www.wikipedia.org

Unit 2 Evaluating User Interface without the Users

1.0 Introduction

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

2.0 Objectives

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

- Explain The Concept Of Evaluating Of User Interface Without The User
- 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 emphasised 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 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-centred 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 Cognitive Walkthroughs

The cognitive walkthrough is a formalised 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 specifications. 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 behaviour 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.

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 criticising someone else's work overshadow the need to improve the design.

Preparing to do a walkthrough

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-centred 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.

Doing a walkthrough

You have defined the task, the users, the interface, and the correct action sequence. You've gathered a group of designers and other interested stakeholders. 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 recognise 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 favourite 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 realise 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 labelled 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!

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 recognise (not always as easy as it sounds), provide better feedback. Probably the most difficult 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 flavours of action analysis. The first, "formal" action analysis, is often called "keystroke-level analysis" in HCI work. The formal approach is characterised 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 flavour 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 analyse 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.

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 added together. 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 summarise those values in Table 2.1. 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 2.1: Average Times for Computer Interface Actions

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.
Recognise 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.

Source: Olson and Olson(1990)

Many values given in this table are averaged and rounded.

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 analyse 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 analyse 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 emphasise 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 control, prompt, 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.

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 favourite 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 has 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, 1990). They 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- centred 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. Task-oriented evaluations have some real advantages. They focus on interface problems that occur during work 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 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, analysing 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. For example, it might be necessary to interact with users in the north using Hausa.

Minimise 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 three to five 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 two to three 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; five of these novice evaluators might find only 50 percent of the problems.

We need to caution here that when we say "all" or "75 per cent" or "50 per cent," 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 three to five people with some expertise in interface design, will locate a significant number of the major problems.

4.0 Conclusion

The concept of evaluation of user interface without the users was introduced in this unit. User walkthrough, action analysis and heuristics analysis were also discussed.

5.0 Summary

In this unit, you have learnt the following:

- the cognitive walkthrough is a formalised 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 flavours 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 Self-Assessment Exercise

1. What do you understand by Nielsen and Molich's heuristics?
2. What are the advantages of the heuristics?

7.0 References/Further Reading

Molich, R. & Nielsen, J. (1990). "Improving a Human-Computer Dialogue: What Designers know about Traditional Interface Design." *Communications of the ACM*, 33, pp. 338-342.

Nielsen, J. & Molich, R. (1990)"Heuristic Evaluation of User Interfaces." Proc. CHI'90 Conference on Human Factors in Computer Systems. New York: *ACM,* pp. 249-256.

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

Nielsen, J. (1992). "Finding Usability Problems through Heuristic Evaluation." Proc. CHI'92 Conference on Human Factors in Computer Systems. New York: ACM, pp. 373-380.

Olson, J. R. & Olson, G. M. (1990). "The Growth of Cognitive Modeling in Human-Computer Interaction since GOMS," *Human-Computer Interaction*, 5 pp 221-265.

Wharton, C., Rieman, J., Lewis, C., & Polson, P. (1994). "The Cognitive Walkthrough: A Practitioner's Guide." In Nielsen, J. & Mack, R. L. (Eds.), Usability Inspection Methods, New York: John Wiley & Sons.

Unit 3 Evaluating the Design with the Users

1.0 Introduction

Having read through the course guide, you will have been introduced to evaluating designs with the users. The steps involved in evaluating the design with the user and usability testing are discussed in this unit.

2.0 Objectives

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

- explain the steps involved in evaluating the design with the users
- highlight the significance of users presence
- explain usability testing procedure.

3.0 Main Content

3.1 Evaluating with the Users

You cannot really tell how good or bad your interface is going to be without getting people to use it. Before the whole system is finally ready, 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.

Since it is not possible to test with all the users, there is a need to get a few representative samples for the test. This can be achieved by using various sampling techniques such as random sampling, purposive sampling, e.t.c.

If it is difficult 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-centred design.

You may find out that 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-centred 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 is to make changes to the design without incurring big costs, is using mock-ups in the test. The simplest mock-ups 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 mock-up 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 mock-up. For example a drawing program has to respond to lots of little user actions, and while you might get information from a simple mock-up 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.

A good illustration is the work by John Gould at IBM in 1992. He tested design alternatives for a speech transcription system for which the speech recognition component was not yet ready (Nielsen, 1992). 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 mock-ups 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. But as a designer you will mostly be concerned with process data.

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 screen 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 Other Major Activities in Testing with Users

The other major activities in testing with the users are:

Giving 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.

The Role of the Observer
Even if you don't need to be available to operate a mock-up, 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.

Most people won't give you a good flow of comments without being pushed a bit. So say things that encourage them to talk, but that do not direct what they should say. Good

choices are "Tell me what you are thinking" or "Keep talking". Bad choices would be "What do you think those prompts about frammis mean?" or "Why did you do that?"

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.

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 synchronised 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 summarise 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.

Summarising 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.

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 organisation 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

We argue that you usually want process data, not bottom-line data, but 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 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 difficult. 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.

Analysing the Bottom-Line Numbers
When you've got your numbers you'll run into some difficult 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 is happening?

Illustration 1:
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 statistical procedure (Computation of standard deviation and standard error) 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.

Instead of following the steps above, you may simply use formulas for computing standard deviation and standard error.

It 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.

Comparing Two Design Alternatives

If you are using bottom-line measurements to compare two design alternatives, your ability to draw a firm conclusion will depend on how different 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.

Between-Groups Experiment

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 (Say, ma and mb). Also compute their standard deviations (Say, 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)

The difference between versions or your range of typical values for the difference between version A and version B is now:

(ma – mb) +- (2*se)

Within-Groups Experiment

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.

3.2 Usability Testing

3.2.1 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 material, 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 eliminate labelling as usability testing such techniques as expert evaluations, walkthrough, and 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.

3.2.2 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 organisations, the usability lab (and by that I mean physical plant) has become more prominent and more important than the testing process itself. Some organisations, 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 realising that instituting a program of usability engineering requires a significant shift in the culture of the organisation, these organisations have put the proverbial cart before the horse in their attempt 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 requires effort, commitment, and time in order to have lasting effects on the organisation and its products. I know of at least two organisations 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 realise that it does not have any interested customers). Having succumbed to the misperception that equates the laboratory with the process itself, these organisations 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 organisation 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 organisation from the ground up instead of from 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.

3.2.3 Six Stages of Conducting a Usability Test

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 to haunt you.

The following are some important reasons 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 serves 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 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 organisation 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 as 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 be 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 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 have to test. Additional revisions may invalidate the test design you have chosen, the question 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 organisation. 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

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 relevant skills and knowledge of the person(s) who will use your product. This description is known as user profile or user characterisation of the target population and should have been developed in the early 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 constraint of time, money, resources, and so on.

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 benefits of not having to scurry around at the last minute, developing materials early on time helps to explicitly structure and organise 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 material will vary from test to test, the general categories required will hardly vary at all. This unit 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 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

Conducting the Test

Having completed the basic groundwork and preparation for your usability test, you are almost ready to begin testing. While there exists 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). It is necessary to focus on the guidelines and activities for performing classic "one-to-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 type 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 the participant and the test monitor. The earlier explanatory 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.

It is important to test 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 they 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 organisation 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.

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. 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 debriefing warranted its own separate treatment, or stage of testing. More often than 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 test as a mystery to be solved, it is not until the debriefing session that all pieces come together.

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 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 think we should change that module after all.

You could simply avoid producing preliminary recommendations, but designers viewing the test are sure to act on what they see prior to your final report, therefore not providing preliminary recommendation 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 methods of testing the design with users and the stages involved in carrying out usability testing.

5.0 Summary

In This unit, you have learnt the following:

- evaluating with users involves having real users 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 giving out instructions, getting an observer, recording, summarising the data and using the result
- setting up usability study includes choosing the order of the test tasks, training test users, the pilot study, e.t.c.
- The stages involved in usability testing are:
  - develop the test plan
  - selecting and acquiring participants
  - preparing test materials
  - conducting the test
  - debriefing the participants
  - transforming data into findings and recommendations.

6.0 Self-Assessment Exercise

1. Explain bottom-line numbers.
2. Describe a mock-up.
3. How would you select test users who are true representation of the users' population?
4. What are the goals of usability testing?
5. Justify whether the usability testing steps described in this unit are adequate.

7.0 References /Further Reading

Mayhew, D. (1999). *The Usability Engineering Lifecycle*. San Francisco, CA: Morgan Kauffman.

Molich, R. & Nielsen, J.(1990). "Improving a Human-Computer Dialogue: What Designers know about Traditional Interface Design." *Communications of the ACM,* 33, pp. 338-342.

Nielsen, J. (1992). "Finding Usability Problems through Heuristic Evaluation." Proc. CHI'92 Conference on Human Factors in Computer Systems. New York: *ACM*, pp. 373-380.

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

Nielsen, J. & Molich, R. (1990). "Heuristic Evaluation of User Interfaces." Proc. CHI'90 Conference on Human Factors in Computer Systems. New York: *ACM*, pp. 249-256.

Rubin, J. (1994). "Handbook of Usability Testing", John Wiley & Sons Inc.

Unit 4 Other Evaluation Issues

1.0 Introduction

This unit describes other evaluation issues. Advantages and disadvantages of model-based evaluation techniques are discussed. Current issues concerning evaluation methodologies are also mentioned.

2.0 Objectives

At the end of 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 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 Active Control of Thought – Rational (ACT-R) model. The ACT-R 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 Disadvantages of Model-Based Evaluations

The use of models to predict user behaviour is less expensive than empirical, user-centred evaluations. Thus many more iterations of the design can be tested. However, a necessary first step is conducting the cognitive task analysis that is needed 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 what the model will predict is accurate.

3.2 Other Evaluation Techniques

The other similar evaluation techniques are:

Formative Evaluations
Formative evaluations obtain user feedback for early concepts or designs of software products. Formative evaluations are typically more informal in that the goal is to collect information to be used for design as opposed to collecting measures of usability. The primary source of data in formative evaluations is verbal data from the user. Early evaluations may use paper prototypes or initial screen designs. Later evaluations can be done on partial prototypes or prototypes of only one portion of a user interface. When possible, logging software is also used to capture user interaction with the software. Additionally, usability engineers often take notes of critical incidents that occur during the evaluation. The debriefing or post-evaluation interview is an excellent source of information in formative evaluations. Usability engineers can probe in depth to understand sources of confusion in the interface.

Formative evaluations need to be conducted in a fairly rapid pace in order to provide design input when it is needed. As a consequence, the evaluations usually focus on a small portion of the user interface, involve relatively few user-participants, and have less formal reporting mechanisms than summative evaluations. Ideally, software designers and developers can observe the evaluations and discuss results and potential solutions with the usability engineers after the evaluations.

Summative Evaluations
Summative evaluations are more formal evaluations conducted to document the usability characteristics of a software product. These evaluations involve a number of users. The recommendation is five to seven users per cell, where a cell represents a class of end-users. For example, if a product is being design for both home and small business users, then representatives of users of each type must be included in the evaluation. If both adults and teenagers will be using the home product, then representatives from both groups need to be included as evaluation participants. Good experimental design is essential to summative evaluation. The metrics of efficiency, effectiveness, and user satisfaction are typically used and the design of the evaluation must include the measures and collection methodology. Tasks used in the evaluation usually represent core functionality of the software but may also include new or improved functionality. Directions and materials given to the users need to be designed and tested in a pilot evaluation to make sure that they are understandable. Usability evaluation has always tried to make the "context-of-use" as realistic as possible. However, a usability laboratory cannot duplicate actual conditions of use within an office or home. Interruptions and other demands for attention do not, and cannot, occur during usability evaluation conditions. As such these evaluations represent the best case condition. If the software is not usable in the laboratory, it will certainly not be usable in real-world use. However, usability in the laboratory does not guarantee usability in more realistic conditions.

The desired level of usability should be defined early in the usability engineering lifecycle and the actual results from the summative evaluation are compared to this. If good usability engineering practices have been followed, including a number of formative evaluations, it is likely that the desired levels will be achieved. If this is not the case, then a decision must be made as to whether or not to release the software or to redesign and re-evaluate the usability.

Expert-based Evaluations
Expert evaluations of usability are similar to design reviews of software projects and code walkthroughs. Inspection methods include heuristic evaluation, guideline reviews, pluralistic walkthroughs, consistency inspections, standards inspections, cognitive walkthroughs, formal usability inspections, and feature inspections.

3.3 Current Issues Concerning Evaluation Methodologies

While the human computer interaction community has come a long way in developing and using methods to evaluate usability, the problem is by no means solved. Although a numbers 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-centred) 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-centred 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 analysed. 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-centred 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- centred design processes, documentation, and user interfaces. Usability is becoming a requirement for companies in purchasing software as they recognise 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 were mentioned, as well as current issues in evaluation methodologies.

5.0 Summary

In this unit, you have learnt the following:

- the EPIC (Executive-Process/Interactive Control) system simulates the human perceptual and motor performance system
- 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
- other evaluation techniques are formative, summative and expert system
- the use of models to predict user behaviour is less expensive than empirical, user-centred evaluations
- the main disadvantages of model-based methodologies is that it is time consuming.

6.0 Self-Assessment Exercise

What is the significance of EPIC (Executive-Process/Interactive Control) system?

7.0 Reference/Further Reading

Mayhew, D. (1999). *The Usability Engineering Lifecycle*. San Francisco,CA: Morgan Kauffman.