**INTERACTION**

**3.1 Interaction styles include:**

* Command
* Menus
* Form fill
* Questions and Answers
* Direct Manipulation
* GUI

**Command**

Commands provide a way of expressing instructions to the computer directly. Communication between user and computer is purely textual. Commands can take the form of function keys, single characters, short abbreviations, whole words, or combination of the first two. An issue with command interface is the number of keystrokes required to complete a command.

Commands must be remembered, so care must be taken in choosing the commands for the system.

Command-entry remains popular in applications where the users are experts and become very familiar with the commands they can use, for example:

* System administrators, particularly on Unix and Linux systems
* Network administrators – Cisco devices (routers, switches, etc) use a command entry Interface

*Advantages of command interfaces*

* Quick and powerful for experienced users
* Can be used in conjunction with other user interfaces.

*Disadvantages of command interfaces*

* Relies on recall of commands and syntax.
* Difficult to learn.
* Error prone

**Menus**

A list of options is presented to the user and the appropriate decision is selected by typing some code or selecting the option required. Unlike command-driven systems, menus do not require the user to remember the item they want; they only need to recognize it. Menus favour **recognition** over **recall.** This is often described by cognitive psychologists as favouring ‘**knowledge in the world’** over ‘**knowledge in the head’**. The majority of user interfaces are now designed to make use of knowledge in the world.

Menus are used extensively in **GUI** systems such as Windows, and are also widely used to provide navigation systems in **web sites**. Horizontally oriented menus in web pages are often referred to as navigation bars.

A number of menu interface styles

* + flat lists, drop-down, pop-up, contextual, and expanding ones, e.g., scrolling and cascading

***Fitts’ Law***

Fitts' law is a mathematical model that predicts how long it will take to point to a target. It takes into account where you are currently pointing relative to the target; how far the target is and how big the target is.

Essentially, Fitts’ Law states that big targets at close distance are reached faster than small targets at long range. Theoretically, the following principles exist when applying Fitts’ Law to interface designs:

* Things done more often should be assigned a larger button. This seems an intuitive principle, but it needs to be used very carefully, since it harms the consistency of the interface.
* Things done more often should be closer to the average position of the user's cursor.
* The top, bottom, and sides of the screen are infinitely targetable because of the boundary created by the edges of the screen. They should be fully utilized.

There are tradeoffs when applying Fitts' Law to interface designs. Fitts' Law suggests that interface components should be made larger and positioned closer to the average cursor position. These suggestions may act in opposition to other factors that make an efficient interface, such as organization and use of available screen space.

Advantages of menus

* Users do not have to memorize complex commands.
* Structured navigation benefits novices and casual users.
* Can shorten users learning time and effort.

Disadvantages of menus

* Can force users through many levels of menus.
* Users may get lost in menu hierarchies.
* Menu terms and names may not be meaningful to users.

**Question and answer dialogues**

In this kind of interaction questions are asked one at a time and the next question may depend on the previous answer. Question and answer dialogues are often used in tasks where information is elicited from users in a prescribed and limited form, such as a bank ATM. They are now widely used in PC software, in the form of ‘Wizards’ to guide novice users through tasks. Wizards can be very frustrating for experienced users who know what they want to do.

**Form-fill**

The user type the data in specific fields, similar to the fields on a paper fill-in form. Many office and database applications use this style. Form-fill is also widely used in web pages, to obtain feedback or to elicit customer details in transactions.

Some of the issues which should be considered when designing forms include:

* Clarity of layout
* Types of fields – the users should be able to tell what kinds of data are permissible in each field.
* Help text (for the form and for each field)
* Automatic advancement (tabbing from field to field) – this can be very helpful as long as it done in the correct order.
* Cancel option (what does it mean in the situation)
* Corrections – how can the user correct data once entered
* Corresponding paper-form (for example order entry)
* Pre-filled fields – it may save the user time if some fields contain commonly used initial values
* Use of list boxes to select from predetermined list of optional values.
* The spreadsheet is a variation on the form-fill interaction style.

**Direct Manipulation**

Direct manipulation is a style of interaction which features a natural representation of task objects and actions promoting the notion of people performing a task themselves (directly) not through an intermediary like a computer. A direct manipulation system has the **following features**:

• Visibility of the objects of interest

• Rapid, reversible, incremental actions

• Replacement of complex command syntax by direct manipulation of the object of interest

The visual representation usually takes the form of a metaphor related to the actual task being performed. Well-designed direct manipulation systems tend to make the interaction **enjoyable** for the users.

Why are DM interfaces so enjoyable?

* Novices can learn the basic functionality quickly
* Users can immediately see if their actions are furthering their goals
* Users experience less anxiety because the system is comprehensible and because actions are reversible
* Users gain confidence because they fell in control and can predict system responses

Examples

File managers, such as Windows Explorer (drag-and-drop copying)

• Graphics software (drawing tools such as Photoshop and Fireworks)

• Games

**WIMP/GUI interfaces**

* Windows
  + could be scrolled, stretched, overlapped, opened, closed, and moved around the screen using the mouse
* Icons
  + represented applications, objects, commands, and tools that were opened when clicked on
* Menus
  + offering lists of options that could be scrolled through and selected
* Pointing device
  + a mouse controlling the cursor as a point of entry to the windows, menus, and icons on the screen

**User Frustration with interfaces**

When using an interface the user may get frustrated when

* When an application doesn’t work properly or crashes
* When a system doesn’t do what the user wants it to do
* When a user’s expectations are not met
* When a system does not provide sufficient information to enable the user to know what to do
* When error messages pop up that are vague, obtuse or condemning
* When the appearance of an interface is garish, noisy, gimmicky or patronizing
* When a system requires users to carry out too many steps to perform a task, only to discover a mistake was made earlier and they need to start all over again

**3.2 Interface metaphors**

Metaphors convey an abstract concept in a more familiar and accessible form. A metaphor is a figure of speech in which an expression is used to refer to something that it does not literally denote in order to suggest a similarity. Metaphors are widely used to make use of users’ existing knowledge when learning new computer systems.

4 **Interaction Design**

"design is a plan or scheme conceived in the mind and intended for subsequent execution"

The ultimate goal of design is to develop a product that helps its users achieve their goals.  Developing a product must begin with gaining understanding of what is required of it.

***4.1 The process of design***



***Steps***

* requirements
  + what is there and what is wanted …
* analysis
  + ordering and understanding
* design
  + what to do and how to decide
* iteration and prototyping
  + getting it right … and finding what is really needed!
* implementation and deployment
  + making it and getting it out there

***What is Interaction Design?***

In Interaction Design, we investigate the artifact's use and target domain by taking a user-centered approach to development.  The users' concerns direct the development rather than technical concerns.

Design is also about trade-offs and about balancing conflicting requirements.  Generating alternatives is a key principle and one that should be encouraged in interaction.

***Four Basic Activities:***

*1.  Identify needs and establish requirements*  
- Who is the target user?  
- What kind of support will the interactive product provide?

*2.  Develop alternative designs that meet those requirements*  
- suggest ideas to meet the requirements  
- Conceptual design:  produce the conceptual model for the product  
- Physical design:  consider the details of the product (colors, sounds, images, menu design, icons, etc.)  Alternatives are considered at every point.

*3.  Build interactive versions (so that they can be communicated and assessed)*  
- a software version is not required- paper based prototypes are quick and cheap to build  
- through role-playing, users can get a real sense of what it is like to interact with the product

*4.  Evaluate the designs (measure their acceptability)*  
- determine the usability of the product or design.  Criteria are:  how appealing is it?  how well does it match the requirements?  Is the product fit for the purpose?  
- Evaluation results are fed back into further design (FEEDBACK / ITERATIVE DESIGN PROCESS)

***Three Characteristics of Interaction Design:***

**1.  Focus on the USERS**  
- involve users in the interactive design process, provide opportunities for evaluation and user feedback

**2.  Specific usability and user experience goals**  
- identify and clearly document these at the beginning of the project.  They help designers to choose between different alternative designs

**3.  Iteration**  
- allows for designs to be refined.  It is always necessary to revise ideas in light of feedback, several times.  Innovation rarely emerges whole and ready to go.  Iteration is inevitable because designers never get the solution right the first time

**Shneiderman’s Eight Golden Rules**

To improve the usability of an application it is important to have a well designed interface. Shneiderman's "Eight Golden Rules of Interface Design" are a guide to good interaction design.

1. **Strive for consistency.**

Consistent sequences of actions should be required in similar situations; identical terminology should be used in prompts, menus, and help screens; and consistent commands should be employed throughout.

1. **Enable frequent users to use shortcuts.**

As the frequency of use increases, so do the user's desires to reduce the number of interactions and to increase the pace of interaction. Abbreviations, function keys, hidden commands, and macro facilities are very helpful to an expert user.

1. **Offer informative feedback.**

For every operator action, there should be some system feedback. For frequent and minor actions, the response can be modest, while for infrequent and major actions, the response should be more substantial.

1. **Design dialog to yield closure.**

Sequences of actions should be organized into groups with a beginning, middle, and end. The informative feedback at the completion of a group of actions gives the operators the satisfaction of accomplishment.

1. **Offer simple error handling.**

As much as possible, design the system so the user cannot make a serious error. If an error is made, the system should be able to detect the error and offer simple, comprehensible mechanisms for handling the error.

1. **Permit easy reversal of actions.**

This feature relieves anxiety, since the user knows that errors can be undone; it thus encourages exploration of unfamiliar options. The units of reversibility may be a single action, a data entry, or a complete group of actions.

1. **Support internal locus of control.**

Experienced operators strongly desire the sense that they are in charge of the system and that the system responds to their actions. Design the system to make users the initiators of actions rather than the responders.

1. **Reduce short-term memory load.**

The limitation of human information processing in short-term memory requires that displays be kept simple, multiple page displays be consolidated, window-motion frequency be reduced, and sufficient training time be allotted for codes, mnemonics, and sequences of actions.

**Nielsen’s Usability Heuristics**

Nielsen lists ten usability heuristics for user interfaces, and these are commonly used in relation to web interfaces. The heuristics are listed below

1. **Visibility of system status**

The system should always keep users informed about what is going on, through appropriate feedback within reasonable time. Probably the two most important things that users need to know are "Where am I?" and "Where can I go next?"

1. **Match between system and the real world**

The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.

1. **User control and freedom**

Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.

Some "emergency exits" are provided by the browser, for example the ‘Back’ button, but there is still plenty of room on your site to support user control and freedom. A "home" button on every page is a simple way to let users feel in control of your site.

1. **Consistency and standards**

Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions. "Platform conventions" on the web means realizing your site is not an island. Users will be jumping onto (and off of) your site from others, so you need to fit in with the rest of the web to some degree. "Standards" on the web means following HTML and other specifications. Deviations from the standards will be opportunities for unusable features to creep into your site.

1. **Error prevention**

Even better than good error messages is a careful design which prevents a problem from

occurring in the first place.

1. **Recognition rather than recall**

Make objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.

For the web, this heuristic is closely related to system status. If users can recognize where they are by looking at the current page, without having to recall their path from the home page, they are less likely to get lost.

1. **Flexibility and efficiency of use**

Accelerators -- unseen by the novice user -- may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions. Some of the best accelerators are provided by the browser, such as bookmarks.

1. **Aesthetic and minimalist design**

Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.

Make sure your content is written for the web and not just a repackaged brochure. Break information into chunks and use links to connect the relevant chunks so that you can support different uses of your content.

1. **Help users recognize, diagnose, and recover from errors**

Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.

1. **Help and documentation**

Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

[http://www.hostserver150.com/usabilit/tool**s/r\_international.htm**](http://www.hostserver150.com/usabilit/tools/r_international.htm) **CAT**

**4.3 Lifecycle Models:  Showing how the activities are related**

Lifecycle models represent a set of activities and how they are used; management tools; simplified versions of reality  
  
***- some models from software engineering include  waterfall, spiral, RAD, etc***

**4.3.1 Waterfall model(evaluate and compare with software eng approaches)**

The waterfall model is a software life-cycle model, in which development is supposed to proceed linearly through the phases of requirements analysis, design, implementation, testing (validation), integration and maintenance.

There have been a number of criticisms of the standard waterfall model, including

* There is a ‘contract’ relationship between client and developers – the user is not involved beyond the initial phase.
* Problems are not discovered until system testing
* Requirements must be fixed before the system is designed, whereas in reality it is often impossible to understand requirements fully until a fair amount of design has been undertaken.
* System performance cannot be tested until the system is almost coded

***Some models from HCI include:  Star, usability engineering***

**4.3.4 The Star Model**

The star model emphasizes that the design of interactive systems typically does not follow a specific order of steps. Evaluation represents the central phase in the development cycle. Development can start from any point in the star and any stage can be followed by any other stage. Evaluation is always done before moving to a new stage. The requirements, design and product gradually evolve, becoming increasingly well-defined.

The star model can give the user a significant role throughout the project since evaluation is central to the cycle. It is particularly oriented towards the development of interactive systems that will be usable by people. Evaluation can be based on any representation of the system. A variety of representations may be used during the development, including sketches, scenarios, prototypes and formal models.

**The star** 

**4.3.5 The Usability Engineering lifecycle**

Usability engineering (UE) is a systematic approach to making software easier to use for the individuals who actually use it to get their work done. It is an approach to the development of software and systems which involves user participation from the outset and guarantees the usefulness of the product through the use of a **usability specification** and **metrics**. UE thus refers to the entire process of producing usable software products from requirements gathering to installation. UE is based on a User-Centered Design (UCD) approach to analysis and design.

The life cycle

|  |  |
| --- | --- |
| **Stage** | **Information Produced** |
| * **Know the user** | User characteristics, User background |
| * **Know the task** | User’s current task, Task analysis |
| * **User requirements capture** | User requirements |
| * **Setting usability goals** | Usability specification |
| * **Design process** | Design |
| * **Apply guidelines, heuristics** | Feedback for design |
| * **Prototyping** | Prototype for user testing |
| * **Evaluation with users** | Feedback for design |
| * **Redesign and evaluate with users** | Finished product |
| * **Evaluate with users and report** | Feedback on product for future systems |

**4.5 User-centred design**

User centered design is an approach to software development user-centred design focuses specifically on making products usable. The approach typically entails involving users in the design and testing of the system so that their feedback can be obtained. Prototypes are usually employed to do this and designs are modified in light of the user feedback.

Initially it may seem that the user-centered approach complicates the software development task, due to the need to make iterative refinements to the software in light of user feedback. However the benefits to be gained are considerable. The process promotes communication between users, managers and those developing the software and identifies problematic issues early on in the development schedule when it is much cheaper to implement changes.

***Principles of user-centred design***

* **Focus early** in the design process **on users and their tasks**
* **Measure users’ reactions and performance** to scenarios, manuals, simulations, and prototype are observed, recorded and analysed
* **Design iteratively**: when problems are found in user testing, fix them and carry out more tests
* **All usability factors must emerge together** and be under the responsibility of one control group.

***5.2 Data Gathering Techniques***

* **Questionnaires**
  + elicit specific information
  + can be YES / NO, multiple choice, comment
  + often used with other techniques
  + can give quantitative / qualitative data
  + good for answering specific questions from a large, dispersed group of people
* **Interviews**
  + forum for talking to people
  + can be structured, unstructured, or semi-structured
  + props, scenarios of use, prototypes can be used
  + good for exploring issues
  + can be time consuming, infeasible to visit everyone
* **Workshops / Focus Groups**
  + group interviews
  + good at gaining a consensus view and / or highlighting areas of conflict
* **Naturalistic Observation**
  + spend time with stakeholders in their day-to-day tasks, observing work as it happens
  + gain insight into stakeholders' tasks
  + good for understanding the nature / context of tasks
  + Requires time and commitment from a member of the design team, and can result in huge amounts of data
  + *ethnography* is one form of this
* **Studying Documentation**
  + procedures and rules are often written down in manuals
  + good source of data about the steps involved in an activity, and any regulations governing a task
  + not to be used in isolation
  + good for understanding legislation and getting background information
  + no stakeholder time, which is a limiting factor on the other techniques

**5.4.2 GOMS Model**

GOMS is a modeling technique (more specifically, a family of modeling techniques) that analyzes the user complexity of interactive systems. It is used by software designers to model user behavior. The user's behavior is modeled in terms of [Goals](http://www.cs.umd.edu/class/fall2002/cmsc838s/tichi/printer/goms.html#Goals), [Operators](http://www.cs.umd.edu/class/fall2002/cmsc838s/tichi/printer/goms.html#Operators), [Methods](http://www.cs.umd.edu/class/fall2002/cmsc838s/tichi/printer/goms.html#Methods) and [Selection rules](http://www.cs.umd.edu/class/fall2002/cmsc838s/tichi/printer/goms.html#Selection%20rules), which are described below in more detail. Briefly, a GOMS model consists of *Methods* that are used to achieve *Goals*. A *Method* is a sequential list of *Operators* that the user performs and (sub)*Goals* that must be achieved. If there is more than one *Method* which may be employed to achieve a *Goal*, a *Selection rule* is invoked to determine what *Method* to choose, depending on the context.

**Scope and Application**

A GOMS model provides the designer with a model of a user's behavior while performing well-known tasks. These models can be used for a variety of purposes, as follows

**Functionality Coverage -** If the designer has a list of likely user goals, GOMS models can be used to verify that a method exists to achieve each of these goals.

**Execution time -**GOMS models can predict the time it will take for the user to carry out a goal (assuming an expert user with no mistakes). This allows a designer to profile an application to locate bottlenecks, as well as compare different UI designs to determine which one allows users to execute tasks quicker.

**Help systems -**Since GOMS models are an explicit representation of expert user activity, they can assist in designing help systems and tutorials to assist users in achieving goals.

**Principles**

**Goals**

Goals are what the user is trying to accomplish. These can be defined at various levels of abstraction, from very high-level goals (e.g. WRITE-RESEARCH-PAPER) to low-level goals (e.g. DELETE-WORD). Higher-level goals are decomposable into subgoals, and are arranged hierarchically.

**Operators**

Operators are the elementary perceptual, motor or cognitive actions that are used to accomplish the goals (e.g. DOUBLE-CLICK-MOUSE, PRESS-INSERT-KEY). Operators are not decomposable: they are atomic elements in the GOMS model. Furthermore, it is generally assumed that each operator requires a fixed amount of time for the user to execute, and that this time interval is independent of context (e.g. CLICK-MOUSE button takes 0.20 seconds to execute).

**Methods**

Methods are the procedures that describe how to accomplish goals. A method is essentially an algorithm that the user has internalized that determines the sequence of subgoals and operators necessary to achieve the desired goal. For example, one method to accomplish the goal DELETE-WORD in the Emacs text editor would be to MOVE-MOUSE to the beginning of the word, and PRESS-ALT-D-KEY-COMBINATION (the use-mouse-delete-word method). Another method to accomplish the same goal could involve using the arrow keys to reach the beginning of the word (the use-arrows-delete-word method).

**Selection rules**

Selection rules specify which method should be used to satisfy a given goal, based on the context. Since there may be several different ways of achieving the same goal, selection rules represent the user's knowledge of which method must be applied to achieve the desired goal. Selection rules generally take the form of a conditional statement, such as "if the word to be deleted is less than 3 lines away from the current cursor location, then use the use-arrows-delete-word-method, else use the use-mouse-delete-word method".

**6. Evaluating User Interfaces**

***Types of Evaluation***

* **Formative** - *done at different stages of development to check that the product meets users needs*
* **Summative**, *assess the quality of a finished product*

***6.1 Reasons for Evaluating***

* **Understanding the real world -** Concerned with the work environment. Evaluations done for this reason are important during requirements gathering and also for checking that prototypes do actually fulfil users’ needs.
* **Comparing designs -** Comparisons may be made of actual designs or, at an earlier stage, design specifications
* **Engineering towards a target -** Evaluating the design with usability metrics
* **Checking conformance to a standard -**For example standards bodies testing that products conform to their standards.
  + - * 1. **USER SUPPORT**

There is often an implicit assumption that is an interactive system is properly designed it will be completely intuitive **(obvious)** to use and the user will require little or no help or training. This may be grand ideal but it is far from true with even the best designed systems currently available. A more helpful approach is to assume that the user will require assistance at various times and design this help into the system

**User support** refers to a range of services by which designers provide assistance to users of technology products such as mobile phones, televisions, computers, software products or other electronic or mechanical goods. In general, user support services attempt to help the user solve specific problems with a product(system)—rather than providing training, customization, or other support services

The type of assistance users require varies and is dependent on many factors for example

* + Familiarity with the system
  + The task
  + Age
  + Expertise
  + Among others

There are four main types of assistance that users require

* + **Quick reference-**thisis basically used as remainder to the user of the details of the tools he is basically familiar with and has used before e.g. commands previously used
  + **Task-specific help** is required when the user has encountered a problem in performing a particular task or when he is uncertain how to apply the tool
  + **Full explanation**-this is for more experienced and inquisitive users, which may include information the user perhaps doesn’t want
  + **Tutorial help**-for new users of a system and provides step by step instructions on how to accomplish some task

**Requirements of User Support**

Users have different requirements for support at different times. User support should be

* + available-can be accessed at any time
  + accurate-matching the actual behaviour of the system
  + Robust
  + Complete-all system features must be covered
  + consistent-different help should have the same things
  + flexible-various solutions, not be rigid
  + Unobtrusiveness-should not prevent the user from continuing with the normal work or interfere.

**Approaches to users support**

* **tutorial help**-allow users to work through the basics of an applications without being helped
* **on-line documentation** (integrated with application) for example *word help*
* **written documentation** (manuals or notices)-most common
* **web-based documentation**-use of video images
* **Wizards and assistants-**Thisaretask-specific tool that leads the user through’ the task step by step using info’ supplied by the user in response to the question