Unit 4

User interface analysis and design:

The visual part of a computer application or operating system through which a client interacts with a computer or software.

It determines how commands are given to the computer or the program and how data is displayed on the screen.

Types of User Interface

There are two main types of User Interface:

Text-Based User Interface or Command Line Interface

Graphical User Interface (GUI)

Text-Based User Interface: This method relies primarily on the keyboard. A typical example of this is UNIX.

Advantages

- Many and easier to customizations options.
- Typically capable of more important tasks.

Disadvantages

- Relies heavily on recall rather than recognition.
- Navigation is often more difficult.

Graphical User Interface (GUI): GUI relies much more heavily on the mouse. A typical example of this type of interface is any versions of the Windows operating systems.

GUI Characteristics

- Windows
- Icons
- Menus
- Pointing
- Graphics

Advantage

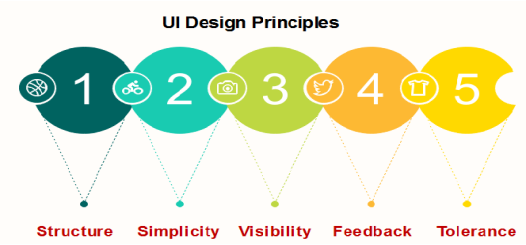
- Less expert knowledge is required to use it.
- Easier to Navigate and can look through folders quickly in a guess and check manner.

- The user may switch quickly from one task to another and can interact with several different applications.

Disadvantages

- Typically decreased options.
- Usually less customizable. Not easy to use one button for tons of different variations.

UI Design Principles



Structure: Design should organize the user interface purposefully, in the meaningful and usual based on precise, consistent models that are apparent and recognizable to users, putting related things together and separating unrelated things, differentiating dissimilar things and making similar things resemble one another. The structure principle is concerned with overall user interface architecture.

Simplicity: The design should make the simple, common task easy, communicating clearly and directly in the user's language, and providing good shortcuts that are meaningfully related to longer procedures.

Visibility: The design should make all required options and materials for a given function visible without distracting the user with extraneous or redundant data

Feedback: The design should keep users informed of actions or interpretation, changes of state or condition, and bugs or exceptions that are relevant and of interest to the user through clear, concise, and unambiguous language familiar to users.

Tolerance: The design should be flexible and tolerant, decreasing the cost of errors and misuse by allowing undoing and redoing while also preventing bugs wherever possible by tolerating varied inputs and sequences and by interpreting all reasonable actions.

Design Concepts:

In software development, there are many stages of planning and analysis before the project is finalized and development can formally begin. Design always comes before development, and functional design makes coding and maintenance very simple.

There are seven main principles to keep in mind in the design model in object-oriented programming (OOP):

- Abstraction

- Patterns
- Separation of data
- Modularity
- Data hiding
- Functional independence
- Refactoring

Abstraction & Patterns

- In abstraction, is the process of hiding complex properties or characteristics from the software itself to keep things more simplistic.
- This allows for a much higher level of efficiency for complex software designs since it allows the developers to list out only the necessary elements or objects required.
- In this principle, the developer will define the properties, type of functions, and the interface for each of said objects in the project.
- The developers will be able to hide the complicated and unnecessary details in the background while retaining core information in the foreground.

There are three main patterns:

Architectural, which is a high-level pattern type that can be defined as the overall formation and organization of the software system itself.

Design, which is a medium-level pattern type that is used by the developers to solve problems in the design stage of development. It can also affect how objects or components interact with one another.

And, finally, **idioms**, which are low-level pattern types, often known as coding patterns, and are used as a workaround means of setting up and defining how components will be interacting with the software itself without being dependent on the programming language.

Separation of Data & Modularity

This principle states that the software code must be separated into two sections called layers and components.

To ensure proper implementation, the two sections must have little to no overlap between them and must have a defined purpose for each component.

This principle allows each component to be developed, maintained, and reused independently of one another.

Modularity, on the other hand, refers to the idea of using predetermined code to increase overall efficiency and management of the current project. The software components will usually be divided into unique items known as modules. Their specific functions divide these modules. Modularity makes the systems easy to manage.

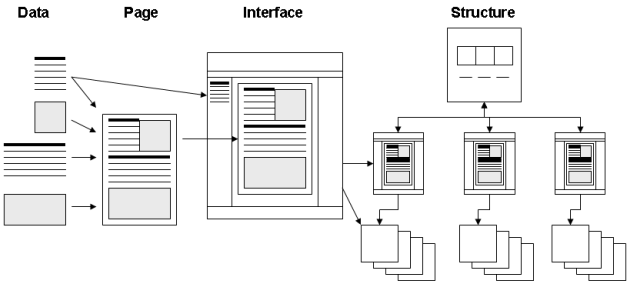
Hiding, Independence, & Refactoring

Also known as information hiding, data hiding allows modules to pass only the required information between themselves without sharing the internal structures and processing. The specific purpose of hiding the internal details of individual objects has several benefits.

Refactoring is the process of changing a software system in such a way that it does not alter the function of the code yet improves its internal structure.

Interface Analysis:

Interface Analysis is a business analysis elicitation technique that helps to identify interfaces between solutions/applications to determine the requirements for ensuring tha**t the components interact with one another effectively.**



Interface types range from user interfaces (human beings interacting directly with the system); interfaces to and from external applications; and interfaces to and from external hardware/gadgets.

The requirements that define how human beings interact with the system; how applications link to other applications and how hardware links to applications need to be defined for effective functioning of the system. Interface analysis helps in discovering the requirements needed to integrate software into its new environment.

Our services:

Analyze and advise the information interface between softwares and hardwares to ensure an effective connection.

Review software requirements specifications, software design description records and source code with hardware, operator, and software interface design documentation, for correctness, consistency, completeness, accuracy, and readability

Webapp interface design

https://www.slideshare.net/heminpatel8/web-application-design-147675133

UX workflow:

UX workflow is a step-by-step process designers must follow from conceptualization to design handoff. A typical UX workflow loosely follows the five stages of the design thinking process, but there is no specific workflow method.

How designers and organizations develop a UX workflow is a matter of preference, depending on multiple factors, including the product, organizational structure, policies, and tools, to name a few. Some workflows will include a few steps, while others might have ten or more.

Steps of a Typical UX Workflow

The following is a typical design workflow most UX teams use:

1. Defining the business need
2. Conducting research and gaining insights
3. Analyze research and ideate
4. Creating information architecture & user flows
5. Lo-fi prototyping
6. Hi-fi prototyping
7. Testing
8. Design handoff

Defining the Business Need

Defining the business need or project scope is a crucial first step. UX is about solving users' problems but within the context of the company and product.

UX designers will meet with the project manager and other stakeholders to discuss the business need and scope. This phase of the workflow might take several meetings and workshops to get input from all stakeholders.

The business need will include the following:

- Project scope
- Project roadmap
- Timeframe and deadlines
- Tasks and objectives
- User data and analytics
- Financial and technical constraints
- Stakeholders, roles, and responsibilities

Conducting Research and Gaining Insights

With a clear goal and purpose in mind, UX teams begin the research phase. Research methods will include:

- General user research

- Conducting interviews
- User focus groups
- Surveys
- Competitor research
- Market research

Analyze Research & Ideate

UX teams will analyze research insights to define:

- User personas
- Empathy and journey maps
- User problems and pain points
- Where competitors win and fail
- Business value opportunities

Teams can ideate to develop solutions with a clear picture of the users, market, problems, and business value opportunities. It's a collaborative brainstorming exercise often involving stakeholders from several departments like product, marketing, and engineering to get diverse ideas and perspectives.

Creating Information Architecture & User Flows

Using research results, UX designers begin listing and organizing the screens they'll need to design. Using these lists, they can create the information architecture or sitemap to define user flows and navigation.

Lo-Fi Prototyping

With information architecture and user flows defined, UX designers begin hand sketching wireframes to create low-fidelity paper prototypes. Paper prototyping is a collaborative effort where UX designers gather to simulate different user flows and identify the elements and components the product will need.

Once design teams have exhausted paper prototyping, they create digital wireframes and low-fidelity prototypes using a design tool. These lo-fi digital prototypes use simple click/tap interactions to test navigation and user flows.

Hi-Fi Prototyping

UI designers convert wireframes to mockups that resemble the final product's aesthetics before adding interactivity to create functioning high-fidelity prototypes.

With UXPin, designers can build fully functioning high-fidelity prototypes with advanced interactions, animations, conditional formatting, variables, data capture and validation, expressions, and even give elements (like buttons) states – features you cannot get from other leading design tools.

Testing

We've put testing at step seven in this UX workflow, but ultimately, designers begin testing from the very beginning. They might not always test with participants, but designers will constantly experiment to validate ideas and concepts.

But the most critical testing happens once design teams have working prototypes. Late usability testing with end-users produces meaningful feedback for designers to make changes, test, and iterate until the product is error-free and working as intended.

Design Handoff

The final design handoff to the development team is a critical and often tense part of any designer's UX workflow. If designers forget deliverables or the testing isn't thorough enough, it could cost the organization time and money to fix it!

Like testing, the design handoff starts early in the design process. Product designers, UX teams, and engineers meet periodically throughout the project to ensure designs meet technical constraints and designers document their work correctly.

Integrating UX and Agile development:

How the UX and Agile development mapped together in agile requirement gathering, planning, Modelling,Designing, deployment like wise the UX flow happens