UNIT – IV

User Interface Design

User Interface Golden rules

- **Strive for consistency** - Consistent sequences of actions should be required in similar situations. Identical terminology should be used in prompts, menus, and help screens. Consistent commands should be employed throughout.
- **Enable frequent users to use short-cuts** - The user's desire to reduce the number of interactions increases with the frequency of use. Abbreviations, function keys, hidden commands, and macro facilities are very helpful to an expert user.
- **Offer informative feedback** - For every operator action, there should be some system feedback. For frequent and minor actions, the response must be modest, while for infrequent and major actions, the response must be more substantial.
- **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, a sense of relief, the signal to drop contingency plans and options from their minds, and this indicates that the way ahead is clear to prepare for the next group of actions.
- **Offer simple error handling** - As much as possible, design the system so the user will not make a serious error. If an error is made, the system should be able to detect it and offer simple, comprehensible mechanisms for handling the error.
- **Permit easy reversal of actions** - This feature relieves anxiety, since the user knows that errors can be undone. Easy reversal of actions encourages exploration of unfamiliar options. The units of reversibility may be a single action, a data entry, or a complete group of actions.
- **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.
- **Reduce short-term memory load** - The limitation of human information processing in short-term memory requires the displays to 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.

User Interface Analysis

1. Understanding User Requirements:
User Research: Conduct surveys, interviews, and usability studies to understand user needs and expectations.
User Personas: Create personas representing different user types to guide design decisions.
2. Defining Use Cases:
User Stories: Break down functionality into user-centric stories that describe specific tasks or interactions.
3. Information Architecture:
Card Sorting: Organize content and features into categories to establish a logical structure.
Sitemaps: Create a visual representation of the software's structure and hierarchy.
4. Wireframing:
Low-Fidelity Wireframes: Sketch out basic layouts and interactions.
High-Fidelity Wireframes: Create more detailed representations using design tools.
5. Prototyping:
Interactive Prototypes: Develop clickable prototypes for user testing and feedback.
Usability Testing: Gather insights on how users interact with the prototype.
6. Visual Design:
Style Guides: Establish guidelines for visual elements like colors, typography, and iconography.
Mockups: Develop high-fidelity visual representations of the software's interface.
7. Accessibility Considerations:
Ensure the software is accessible to users with disabilities.
Adhere to accessibility standards (e.g., WCAG) to make the interface inclusive.
8. Consistency Across Platforms:
Ensure a consistent user experience across different devices and platforms.
Adapt the UI for responsiveness and compatibility.
9. Collaboration with Developers:
Work closely with developers to ensure the UI design is feasible and can be implemented effectively.
Use design systems or component libraries for consistency in implementation.
10. Usability Testing:
Conduct usability testing throughout the development process to identify and address user experience issues.
Use feedback to refine and improve the UI design.
11. Performance Considerations:
Optimize the UI for performance to ensure smooth interactions and response times .Consider factors like page load times and responsiveness.
12. Security:
Ensure that the UI design aligns with security best practices.

Address potential vulnerabilities in the user interface.

UI Design Process:

1. Understand the Users and Context:

User Research: Conduct surveys, interviews, and observations to understand the target audience and their needs.
Contextual Inquiry: Gain insights by observing users in their natural environment.

2. Define the Requirements:
User Personas: Create fictional characters representing different user types.
User Stories: Define specific, concrete use cases from the user's perspective.

3. Design the Information Architecture:
Card Sorting: Organize content and features into categories based on user input.
Sitemaps: Create a hierarchical diagram representing the structure of the website or application.

4. Create Wireframes:
Low-Fidelity Wireframes: Basic sketches to outline the structure and layout.
High-Fidelity Wireframes: Detailed representations, often created using design tools.

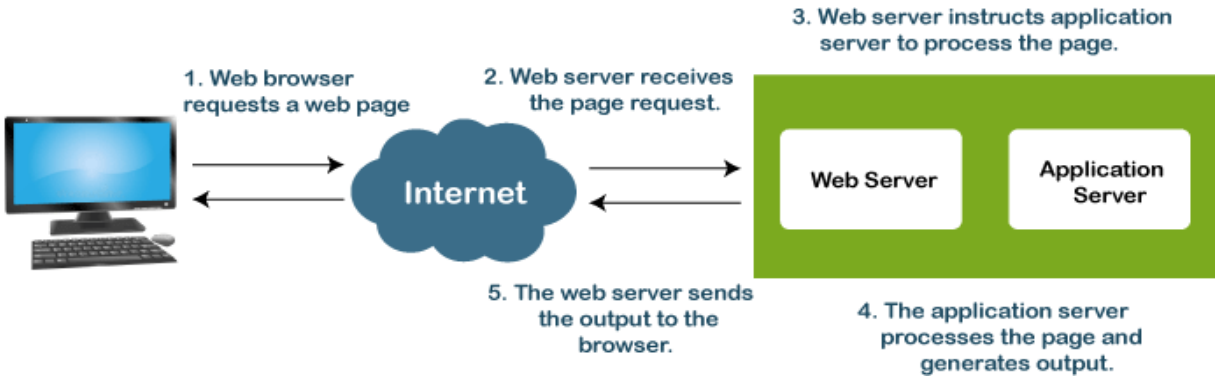
5. Prototype:
Interactive Prototypes: Build a clickable model of the design for user testing.
Usability Testing: Gather feedback on the prototype to identify potential issues.

6. Visual Design:
Style Guides: Define the visual elements, such as colors, fonts, and icons.
Mockups: Create high-fidelity representations of the final product.

7. Development and Implementation:
Collaboration with Developers: Work closely with developers to ensure the design is implemented accurately.
Iterative Testing: Test the UI during the development phase for any issues

Web Application:
A web-application is an application program that is usually stored on a remote server, and users can access it through the use of **Software** known as **web-browser.**

The Flow of the Web Application



1. In general, a user sends a request to the web-server using web browsers such as **Google Chrome, Microsoft Edge, Firefox**, etc over the **internet**.
2. Then, the request is forwarded to the appropriate web **application server** by the **web-server**.
3. Web application server performs the requested operations/ tasks like **processing the database, querying the databases; produces** the result of the requested data.
4. The obtained result is sent to the web-server by the web application server along with the requested data/information or processed data.
5. The web server responds to the user with the requested or processed data/information and provides the result to the user's screen .

Mobile interface design

Mobile interface design refers to the process of creating visual and interactive elements for applications and websites that are specifically tailored for mobile devices such as smartphones and tablets. It encompasses the design of the user interface (UI) and user experience (UX) to ensure that users can interact with the application or website seamlessly on smaller screens.

Mobile Interface Design Principles

1.Clarity and Simplicity:
Keep the interface simple and easy to understand. Avoid unnecessary elements that may clutter the screen. Prioritize clarity in design and communication.

2. Consistency:
Maintain a consistent design language throughout the application. Consistency in layout, colors, typography, and interactions helps users understand and navigate the interface more easily.

3. Prioritization of Content:
Prioritize important content and actions. Consider the hierarchy of information and ensure that crucial elements are easily accessible.

4. Intuitive Navigation:
Design navigation in a way that feels natural and intuitive to users. Mobile interfaces often rely on gestures, so ensure that users can easily navigate through the app with common gestures like swiping and tapping.

5. Thumb-Friendly Design:
Place interactive elements within easy reach of the user's thumb. Users should be able to navigate and interact with the app comfortably using one hand.

6. Responsive Design:
Ensure that the interface adapts seamlessly to different screen sizes and orientations. A responsive design provides a consistent experience across a variety of devices.

7. Feedback and Affordance:
Provide clear feedback for user actions. Use visual and audio cues to indicate that a button has been pressed or an action has been completed. Affordance refers to making elements look like they can be interacted with, making the interface more user-friendly.

8. Minimize User Input:
Reduce the number of steps and input fields required for users to complete tasks.
Simplify forms and utilize input options such as auto-complete and input masks.
9. Performance Optimization:
Optimize the performance of the application to ensure fast loading times and smooth interactions. Slow-loading content can frustrate users and lead to a poor experience.

10. Readable Typography:
Choose legible fonts and appropriate text sizes. Ensure that text is easily readable on various screen sizes and resolutions.

11. Touch-Friendly Buttons:
Design interactive elements, such as buttons, with an appropriate size to facilitate easy tapping. Provide enough spacing between elements to prevent accidental taps.

12. Accessibility:
Design with accessibility in mind. Ensure that users with disabilities can navigate and use the app effectively. This includes considerations for screen readers, color contrast, and alternative text for images.

13. User Testing:
Conduct user testing to gather feedback on the usability of the interface. Real user feedback can reveal insights into how users interact with the app and help identify areas for improvement.

Implementation issues

Software engineering involves various stages from requirements gathering to design, coding, testing, and maintenance. Throughout these stages, several implementation issues can arise. Here are some common implementation issues in software engineering:

Reuse

1. Code Reusability:
Issue: Difficulty in identifying and reusing existing code or components.
Solution: Implement modular and well-documented code. Utilize design patterns and frameworks to encapsulate reusable functionality.

2. Library and Component Management:
Issue: Managing external libraries and components can lead to versioning conflicts or dependency issues.
Solution: Use package managers or dependency management tools. Keep libraries up-to-date and adhere to versioning conventions.

3. Knowledge Transfer:
Issue: Lack of knowledge transfer and documentation on reusable components.
Solution: Document code thoroughly. Establish a knowledge-sharing culture within the development team. Use code reviews and pair programming.

4. Design for Reusability:
Issue: Components not designed with reusability in mind.
Solution: Encapsulate functionalities into modular components. Follow SOLID principles. Strive for a balance between generality and specificity.

5. Testing Reusable Components:
Issue: Ensuring that reused components work correctly in different contexts.
Solution: Implement thorough unit testing for reusable components. Conduct integration tests to validate the compatibility of reused code.

Configuration Management:

1. Version Control:
Issue: Difficulty in managing and tracking changes to the codebase.
Solution: Use version control systems (e.g., Git, SVN) to track changes, manage branches, and collaborate efficiently. Establish branching and merging strategies
.
2. Build and Deployment Configuration:
Issue: Inconsistencies in build and deployment configurations.
Solution: Use configuration files to manage build settings. Automate build and deployment processes. Utilize containerization technologies like Docker for consistent deployment environments.

3. Environment Configuration:
Issue: Configurations that are hard-coded, leading to inflexibility in different environments.
Solution: Use configuration files or environment variables to store environment-specific settings. Implement a configuration management system.

4. Dependency Management:
Issue: Unmanaged dependencies causing conflicts or security vulnerabilities.
Solution: Regularly update dependencies. Use tools like package managers to manage dependencies. Conduct regular security audits.

5. Documentation:
Issue: Inadequate documentation of configurations and changes.
Solution: Maintain detailed documentation for configuration settings, changes, and deployment processes. Ensure that documentation is regularly updated.

6. Rollback Mechanism:
Issue: Lack of a reliable mechanism to roll back changes in case of issues.
Solution: Implement a rollback strategy. Maintain backups and versioned releases to facilitate rollback if needed.

Software Quality

Software quality product is defined in term of its fitness of purpose. That is, a quality product does precisely what the users want it to do. For software products, the fitness of use is generally explained in terms of satisfaction of the requirements laid down in the SRS document. Although "fitness of purpose" is a satisfactory interpretation of quality for many devices such as a car, a table fan, a grinding machine, etc.for software products, "fitness of purpose" is not a wholly satisfactory definition of quality.

Example: Consider a functionally correct software product. That is, it performs all tasks as specified in the SRS document. But, has an almost unusable user interface. Even though it may be functionally right, we cannot consider it to be a quality product.

The modern view of a quality associated with a software product several quality methods such as the following:

Portability: A software device is said to be portable, if it can be freely made to work in various operating system environments, in multiple machines, with other software products, etc.

Usability: A software product has better usability if various categories of users can easily invoke the functions of the product.

Reusability: A software product has excellent reusability if different modules of the product can quickly be reused to develop new products.

Correctness: A software product is correct if various requirements as specified in the SRS document have been correctly implemented.

Maintainability: A software product is maintainable if bugs can be easily corrected as and when they show up, new tasks can be easily added to the product, and the functionalities of the product can be easily modified, etc.

McCall's quality factors

McCall's quality factors, introduced by John McCall in the 1970s, are a set of software quality attributes that serve as a framework for evaluating and measuring the quality of a software product. These factors help identify and define various aspects of software quality.
McCall's model consists of three main categories: product revision, product transition, and product operation. Within these categories, there are eleven quality factors

1. Product Revision Factors:

1. Correctness: The degree to which the software product meets its specifications and fulfills the user's requirements without errors.

2. Reliability:The ability of the software to perform its intended functions without failure over a specified period.

3. Efficiency:The ability of the software to perform its functions with minimal resource utilization, such as processing time and memory usage.

4. Integrity:The degree to which the software safeguards against unauthorized access or alterations of its data.

5. Usability: The ease with which users can learn, interact with, and use the software to achieve their goals.

2. Product Transition Factors:

1. Portability:The ease with which the software can be transferred from one environment or platform to another.

2. Reusability:The extent to which software components can be reused in different applications or parts of the same application.

3. Interoperability:The ability of the software to interact and operate with other specified software systems.

3. Product Operation Factors:

1. Maintainability:The ease with which the software can be modified or enhanced to adapt to changing requirements or correct defects.

2. Flexibility: The degree to which the software can be adapted or extended to accommodate future changes in requirements.

3. Testability:The ease with which the software can be tested to ensure that it meets its specified requirements.

ISO 9126 quality factors

ISO/IEC 9126 is an international standard for software product quality. The standard defines a set of characteristics and sub-characteristics that can be used to evaluate and measure the quality of software.

ISO/IEC 9126 is structured into three main categories: product quality, external quality, and internal quality. Within these categories, there are six primary quality characteristics, each with sub-characteristics.

The six primary quality characteristics are:

1. Functionality:The degree to which the software provides functions that meet stated and implied needs when used under specified conditions.

Sub-characteristics:
Suitability
Accuracy
Interoperability
Compliance
Security

2. Reliability:The ability of the software to maintain a specified level of performance when used under specified conditions.

Sub-characteristics:
Maturity
Fault Tolerance
Recoverability
Reliability

3. Usability:The extent to which the software can be understood, learned, used, and attractive to the user when used under specified conditions.

Sub-characteristics:
Understandability
Learnability
Operability
Attractiveness

4. Efficiency: The degree to which the software uses resources in relation to the amount of work accomplished under specified conditions.

Sub-characteristics:
Time Behavior
Resource Utilization

5. Maintainability:The degree of effectiveness and efficiency with which the software can be modified to correct defects, modified, or adapted to changes in the environment.

Sub-characteristics:
Analyzability
Modifiability
Stability
Testability

6. Portability: The degree of effectiveness and efficiency with which the software can be transferred from one hardware, software, or operational environment to another.

Sub-characteristics:
Adaptability
Installability
Co-Existence
Replaceability