

UNIT – 4

Unit IV:

User Interface Design: The Golden Rules, User Interface Analysis and Design, Interface Analysis, Interface Design Steps, WebApp Interface Design, Design Evaluation.

WebApp Design: WebApp Design Quality, Design Goal, A Design Pyramid for WebApps, WebApp Interface Design, Aesthetic Design, Content Design, Architecture Design, Navigation Design, Component-Level Design, Object-Oriented Hypermedia Design Method(OOHMD).

4.0 USER INTERFACE DESIGN

What is it? User interface design creates an effective communication medium between a human and a computer. Following a set of interface design principles, design identifies interface objects and actions and then creates a screen layout that forms the basis for a user interface prototype.

Who does it? A software engineer designs the user interface by applying an iterative process that draws on predefined design principles.

Why is it important? If software is difficult to use, if it forces you into mistakes, or if it frustrates your efforts to accomplish your goals, you won't like it, regardless of the computational power it exhibits, the content it delivers, or the functionality it offers. The interface has to be right because it molds a user's perception of the software.

What are the steps? User interface design begins with the identification of user, task, and environmental requirements. Once user tasks have been identified, user scenarios are created and analyzed to define a set of interface objects and actions. These form the basis for the creation of screen layout that depicts graphical design and placement of icons, definition of descriptive screen text, specification and titling for windows, and specification of major and minor menu items. Tools are used to prototype and ultimately implement the design model, and the result is evaluated for quality.

What is the work product? User scenarios are created and screen layouts are generated. An interface prototype is developed and modified in an iterative fashion.

How do I ensure that I've done it right? An interface prototype is "test driven" by the users, and feedback from the test drive is used for the next iterative modification of the prototype.

4.1 THE GOLDEN RULES

The three golden rules on interface design are

1. Place the user in control.
2. Reduce the user's memory load.
3. Make the interface consistent.

These golden rules actually form the basis for a set of user interface design principles that guide this important aspect of software design.

4.1.1 Place the User in Control: During a requirements-gathering session for a major new information system, a key user was asked about the attributes of the window-oriented graphical interface. User wanted to control the computer, not have the computer control her. Most interface constraints and restrictions that are imposed by a designer are intended to simplify the mode of interaction. The result may be an interface that is easy to build, but frustrating to use. Mandel defines a number of design principles that allow the user to maintain control:

Define interaction modes in a way that does not force a user into unnecessary or undesired actions. An interaction mode is the current state of the interface. For example, if spell check is selected in a word-processor menu, the software moves to a spell-checking mode. There is no reason to force the user to remain in spell-checking mode if the user desires to make a small text edit along the way. The user should be able to enter and exit the mode with little or no effort.

Provide for flexible interaction. Because different users have different interaction preferences, choices should be provided. For example, software might allow a user to interact via keyboard commands, mouse movement, a digitizer pen, a multi touch screen, or voice recognition commands.

Allow user interaction to be interruptible and undoable. Even when involved in a sequence of actions, the user should be able to interrupt the sequence to do something else (without losing the work that had been done). The user should also be able to “undo” any action.

Streamline interaction as skill levels advance and allow the interaction to be customized. Users often find that they perform the same sequence of interactions repeatedly. It is worthwhile to design a “macro” mechanism that enables an advanced user to customize the interface to facilitate interaction.

Hide technical internals from the casual user. The user interface should move the user into the virtual world of the application. The user should not be aware of the operating system, file management functions, or other arcane computing technology. In essence, the interface should never require that the user interact at a level that is “inside” the machine (e.g., a user should never be required to type operating system commands from within application software).

Design for direct interaction with objects that appear on the screen. The user feels a sense of control when able to manipulate the objects that are necessary to perform a task in a manner similar to what would occur if the object were a physical thing. For example, an application interface that allows a user to “stretch” an object is an implementation of direct manipulation.

4.1.2 Reduce the User's Memory Load: The more a user has to remember, the more error-prone the interaction with the system will be. It is for this reason that a well-designed user interface does not tax the user's memory.

Reduce demand on short-term memory. When users are involved in complex tasks, the demand on short-term memory can be significant. The interface should be designed to reduce the requirement to remember past actions, inputs, and results.

Establish meaningful defaults. The initial set of defaults should make sense for the average user, but a user should be able to specify individual preferences.

Define shortcuts that are intuitive. When mnemonics are used to accomplish a system function (e.g., alt-P to invoke the print function), the mnemonic should be tied to the action in a way that is easy to remember.

The visual layout of the interface should be based on a real-world metaphor. For example, a bill payment system should use a checkbook and check register metaphor to guide the user through the bill paying process. This enables the user to rely on well-understood visual cues, rather than memorizing an arcane interaction sequence.

Disclose information in a progressive fashion. The interface should be organized hierarchically. That is, information about a task, an object, or some behavior should be presented first at a high level of abstraction. More detail should be presented after the user indicates interest with a mouse pick.

4.1.3 Make the Interface Consistent: The interface should present and acquire information in a consistent fashion. This implies that (1) all visual information is organized according to design rules that are maintained throughout all screen displays, (2) input mechanisms are constrained to a limited set that is used consistently throughout the application, and (3) mechanisms for navigating from task to task are consistently defined and implemented.

Allow the user to put the current task into a meaningful context. Many interfaces implement complex layers of interactions with dozens of screen images. It is important to provide indicators (e.g., window titles, graphical icons, consistent color coding) that enable the user to know the context of the work at hand. In addition, the user should be able to determine where he has come from and what alternatives exist for a transition to a new task.

Maintain consistency across a family of applications. A set of applications (or products) should all implement the same design rules so that consistency is maintained for all interaction. If past interactive models have created user expectations, do not make changes unless there is a compelling reason to do so. Once a particular interactive sequence has become a de facto standard the user expects this in every application he encounters. A change will cause confusion.

4.2 USER INTERFACE ANALYSIS AND DESIGN

The overall process for analyzing and designing a user interface begins with the creation of different models of system function. Tools are used to prototype and ultimately implement the design model, and the result is evaluated by end users for quality.

4.2.1 Interface Analysis and Design Models: Four different models come into play when a user interface is to be analyzed and designed. To build an effective user interface, “all design should begin with an understanding of the intended users, including profiles of their age, gender, physical abilities, education, cultural or ethnic background, motivation, goals and personality . In addition, users can be categorized as:

Novices. No syntactic knowledge of the system and little semantic knowledge of the application or computer usage in general.

Knowledgeable, intermittent users. Reasonable semantic knowledge of the application but relatively low recall of syntactic information necessary to use the interface.

Knowledgeable, frequent users. Good semantic and syntactic knowledge that often leads to the “power-user syndrome”; that is, individuals who look for shortcuts and abbreviated modes of interaction.

The user’s **mental model** (system perception) is the image of the system that end users carry in their heads. For example, if the user of a particular word processor were asked to describe its operation, the system perception would guide the response.

The implementation model combines the outward manifestation of the computer based system (look and feel interface), coupled with all supporting information (books, manuals, videotapes, help) that describes interface syntax and semantics. When the implementation model and the user’s mental model are coincident, users generally feel comfortable with the software and use it effectively. In essence, these models enable the interface designer to satisfy a key element of the most important principle of user interface design: “Know the user, know the tasks.”

4.2.2 The Process: The analysis and design process for user interfaces is iterative and can be represented using a spiral model. Referring to Figure 11.1, the four distinct framework activities: (1) interface analysis and modeling, (2) interface design, (3) interface construction, and (4) interface validation. The spiral shown in Figure 11.1 implies that each of these tasks will occur more than once, with each pass around the spiral representing additional elaboration of requirements and the resultant design.

In most cases, the construction activity involves prototyping—the only practical way to validate what has been designed. Interface analysis focuses on the profile of the users who will interact with the system. Skill level, business understanding, and general receptiveness to the new system are recorded; and different user categories are defined. For each user category, requirements are elicited. In essence, you work to understand the system perception for each class of users.

Once general requirements have been defined, a more detailed task analysis is conducted. Those tasks that the user performs to accomplish the goals of the system are identified, described, and elaborated.

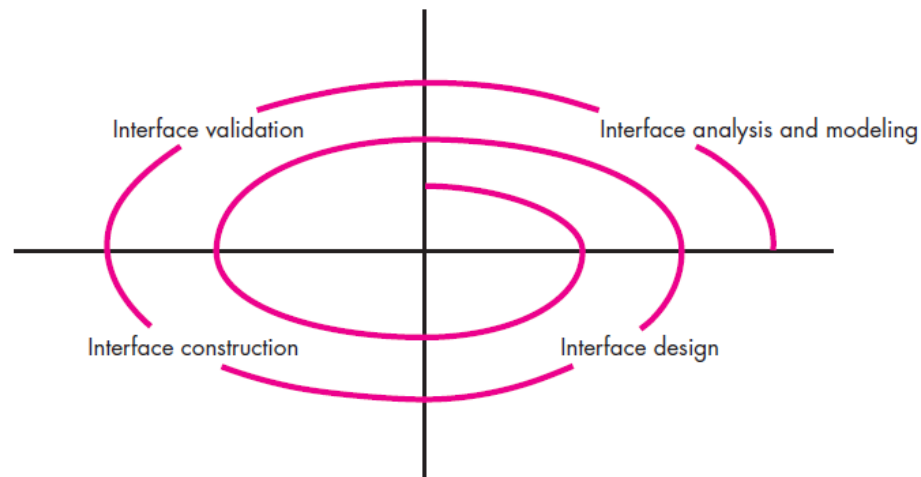


Fig 11.1: The user interface design process

Finally, analysis of the user environment focuses on the physical work environment. Among the questions to be asked are

- Where will the interface be located physically?
- Will the user be sitting, standing, or performing other tasks unrelated to the interface?
- Does the interface hardware accommodate space, light, or noise constraints?
- Are there special human factors considerations driven by environmental factors?

The goal of interface design is to define a set of interface objects and actions (and their screen representations) that enable a user to perform all defined tasks in a manner that meets every usability goal defined for the system.

Interface construction normally begins with the creation of a prototype that enables usage scenarios to be evaluated. As the iterative design process continues, a user interface tool kit may be used to complete the construction of the interface.

Interface validation focuses on (1) the ability of the interface to implement every user task correctly, to accommodate all task variations, and to achieve all general user requirements; (2) the degree to which the interface is easy to use and easy to learn, and (3) the users' acceptance of the interface as a useful tool in their work.

Subsequent passes through the process elaborate task detail, design information, and the operational features of the interface.

4.3 INTERFACE ANALYSIS

A key tenet of all software engineering process models is: understand the problem before you attempt to design a solution. In the case of user interface design, understanding the problem means understanding (1) the people (end users) who will interact with the system through the interface, (2) the tasks that end users must perform to do their work, (3) the content that is presented as part of the interface, and (4) the environment in which these tasks will be conducted. We examine these elements of interface analysis with the intent of establishing a solid foundation for the design tasks that follow.

4.3.1 User Analysis: The only way that you can get the mental image and the design model to converge is to work to understand the users themselves as well as how these people will use the system. Information from a broad array of sources can be used to accomplish this:

User Interviews. The most direct approach, members of the software team meet with end users to better understand their needs, motivations, work culture, and a myriad of other issues. This can be accomplished in one-on-one meetings or through focus groups.

Sales input. Sales people meet with users on a regular basis and can gather information that will help the software team to categorize users and better understand their requirements.

Marketing input. Market analysis can be invaluable in the definition of market segments and an understanding of how each segment might use the software in subtly different ways.

Support input. Support staff talks with users on a daily basis. They are the most likely source of information on what works and what doesn't, what users like and what they dislike, what features generate questions and what features are easy to use.

The following set of questions will help you to better understand the users of a system:

- Are users trained professionals, technicians, clerical, or manufacturing workers?
- What level of formal education does the average user have?
- Are the users capable of learning from written materials or have they expressed a desire for classroom training?
- Are users expert typists or keyboard phobic?
- What is the age range of the user community?
- Will the users be represented predominately by one gender?
- How are users compensated for the work they perform?
- Do users work normal office hours or do they work until the job is done?
- Is the software to be an integral part of the work users do or will it be used only occasionally?
- What is the primary spoken language among users?
- What are the consequences if a user makes a mistake using the system?
- Are users experts in the subject matter that is addressed by the system?
- Do users want to know about the technology that sits behind the interface?

Once these questions are answered, you'll know who the end users are, what is likely to motivate and please them, how they can be grouped into different user classes or profiles, what their mental models of the system are, and how the user interface must be characterized to meet their needs.

4.3.2 Task Analysis and Modeling: The goal of task analysis is to answer the following questions:

- What work will the user perform in specific circumstances?
- What tasks and subtasks will be performed as the user does the work?
- What specific problem domain objects will the user manipulate as work is performed?
- What is the sequence of work tasks—the workflow?
- What is the hierarchy of tasks?

To answer these questions, you must use techniques that are applied to the user interface. **Use cases.** When used as part of task analysis, the use case is developed to show how an end user performs some specific work-related task. The use case provides a basic description of one important work task for the computer-aided design system. From it, you can extract tasks, objects, and the overall flow of the interaction.

Task elaboration. The stepwise elaboration is (also called functional decomposition or stepwise refinement) a mechanism for refining the processing tasks that are required for software to accomplish some desired function.

Task analysis can be applied in two ways. An interactive, computer-based system is often used to replace a manual or semi manual activity. To understand the tasks that must be performed to accomplish the goal of the activity, you must understand the tasks that people currently perform (when using a manual approach) and then map these into a similar (but not necessarily identical) set of tasks that are implemented in the context of the user interface.

Alternatively, you can study an existing specification for a computer-based solution and derive a set of user tasks that will accommodate the user model, the design model, and the system perception. Regardless of the overall approach to task analysis, you must first define and classify tasks. Each of the major tasks can be elaborated into subtasks.

Object elaboration. Rather than focusing on the tasks that a user must perform, you can examine the use case and other information obtained from the user and extract the physical objects that are used by the interior designer. These objects can be categorized into classes. Attributes of each class are defined, and an evaluation of the actions applied to each object provide a list of operations. The user interface analysis model would not provide a literal implementation for each of these operations. However, as the design is elaborated, the details of each operation are defined.

Workflow analysis. When a number of different users, each playing different roles, makes use of a user interface, it is sometimes necessary to go beyond task analysis and object elaboration and apply workflow analysis. This technique allows you to understand how a work process is

completed when several people (and roles) are involved. Consider a company that intends to fully automate the process of prescribing and delivering prescription drugs. The entire process will revolve around a Web-based application that is accessible by physicians (or their assistants), pharmacists, and patients. Workflow can be represented effectively with a UML swimlane diagram (a variation on the activity diagram). See Figure 11.2

Regardless, the flow of events (shown in the figure) enables you to recognize a number of key interface characteristics:

1. Each user implements different tasks via the interface; therefore, the look and feel of the interface designed for the patient will be different than the one defined for pharmacists or physicians.
2. The interface design for pharmacists and physicians must accommodate access to and display of information from secondary information sources (e.g., access to inventory for the pharmacist and access to information about alternative medications for the physician).
3. Many of the activities noted in the swimlane diagram can be further elaborated using task analysis and/or object elaboration (e.g., Fills prescription could imply a mail-order delivery, a visit to a pharmacy, or a visit to a special drug distribution center).

Hierarchical representation. A process of elaboration occurs as you begin to analyze the interface. Once workflow has been established, a task hierarchy can be defined for each user type. The hierarchy is derived by a stepwise elaboration of each task identified for the user. For example, consider the following user task and subtask hierarchy.

User task: Requests that a prescription be refilled

- Provide identifying information.
- Specify name.
- Specify userid.
- Specify PIN and password.
- Specify prescription number.
- Specify date refill is required.

To complete the task, three subtasks are defined. One of these subtasks, provide identifying information, is further elaborated in three additional sub-subtasks.

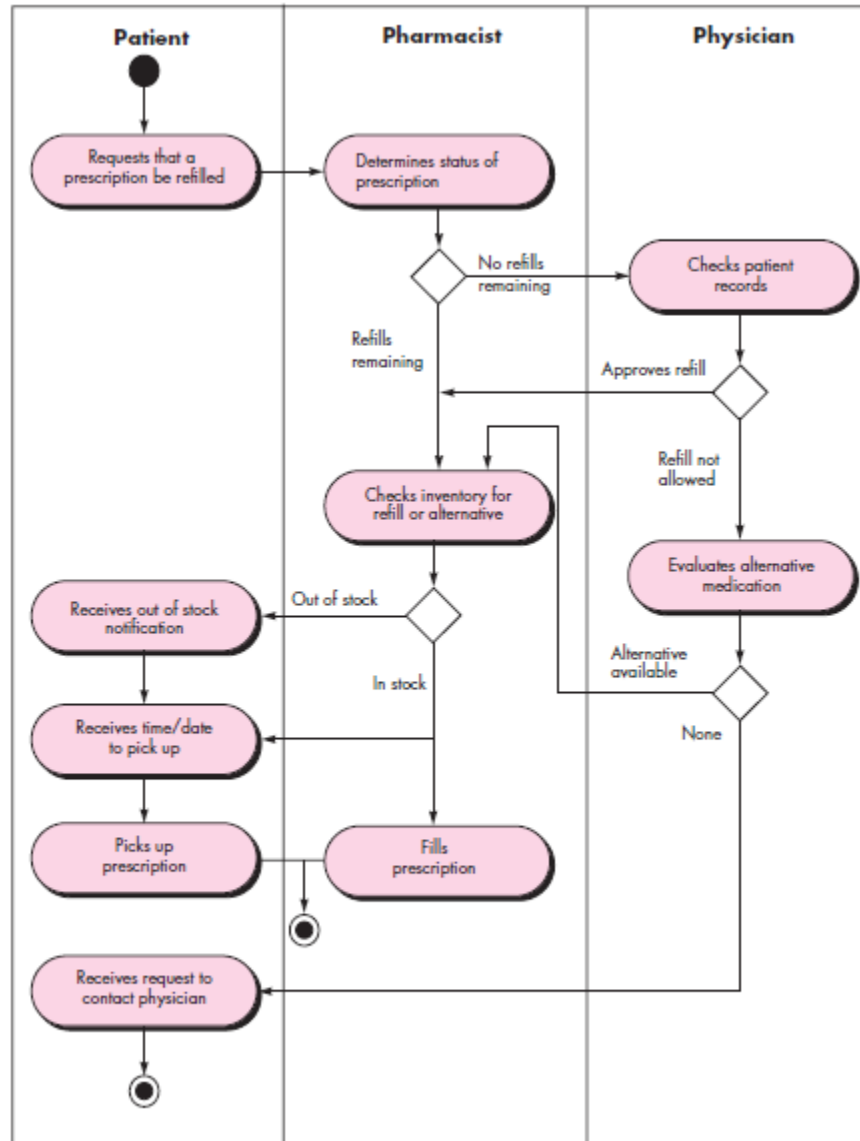


Fig 11.2: Swimlane diagram for prescription refill function

4.3.3 Analysis of Display Content: The user tasks lead to the presentation of a variety of different types of content. For modern applications, display content can range from character-based reports (e.g., a spreadsheet), graphical displays (e.g., a histogram, a 3-D model, a picture of a person), or specialized information (e.g., audio or video files).

These data objects may be

(1) generated by components (unrelated to the interface) in other parts of an application, (2) acquired from data stored in a database that is accessible from the application, or (3) transmitted from systems external to the application in question.

During this interface analysis step, the format and aesthetics of the content (as it is displayed by the interface) are considered. Among the questions that are asked and answered are:

- Are different types of data assigned to consistent geographic locations on the screen (e.g., photos always appear in the upper right-hand corner)?
- Can the user customize the screen location for content?
- Is proper on-screen identification assigned to all content?
- If a large report is to be presented, how should it be partitioned for ease of understanding?
- Will mechanisms be available for moving directly to summary information for large collections of data?
- Will graphical output be scaled to fit within the bounds of the display device that is used?
- How will color be used to enhance understanding?
- How will error messages and warnings be presented to the user?

The answers to these (and other) questions will help you to establish requirements for content presentation.

4.3.4 Analysis of the Work Environment: Hackos and Redish stated the importance of work environment analysis as:

People do not perform their work in isolation. They are influenced by the activity around them, the physical characteristics of the workplace, the type of equipment they are using, and the work relationships they have with other people. If the products you design do not fit into the environment, they may be difficult or frustrating to use.

4.4 INTERFACE DESIGN STEPS

Once interface analysis has been completed, all tasks (or objects and actions) required by the end user have been identified in detail and the interface design activity commences. Interface design is an iterative process. Although many different user interface design models (e.g., have been proposed, all suggest some combination of the following steps:

1. Using information developed during interface analysis define interface objects and actions (operations).
2. Define events (user actions) that will cause the state of the user interface to change. Model this behavior.
3. Depict each interface state as it will actually look to the end user.
4. Indicate how the user interprets the state of the system from information provided through the interface.

4.4.1 Applying Interface Design Steps: The definition of interface objects and the actions that are applied to them is an important step in interface design. To accomplish this, user scenarios are parsed. That is, a use case is written. Nouns (objects) and verbs (actions) are isolated to create a list of objects and actions.

Once the objects and actions have been defined and elaborated iteratively, they are categorized by type. Target, source, and application objects are identified. A source object (e.g., a report icon) is dragged and dropped onto a target object (e.g., a printer icon).

When you are satisfied that all important objects and actions have been defined (for one design iteration), screen layout is performed. Like other interface design activities, screen layout is an interactive process in which graphical design and placement of icons, definition of descriptive screen text, specification and titling for windows, and definition of major and minor menu items are conducted.

4.4.2 User Interface Design Patterns: Graphical user interfaces have become so common that a wide variety of user interface design patterns has emerged. As an example of a commonly encountered interface design problem, consider a situation in which a user must enter one or more calendar dates, sometimes months in advance. A vast array of interface design patterns has been proposed over the past decade.

4.4.3 Design Issues: As the design of a user interface evolves, four common design issues almost always surface: system response time, user help facilities, error information handling, and command labeling

Response time. System response time is the primary complaint for many interactive applications. In general, system response time is measured from the point at which the user performs some control action (e.g., hits the return key or clicks a mouse) until the software responds with desired output or action.

System response time has two important characteristics: length and variability. If system response is too long, user frustration and stress are inevitable. Variability refers to the deviation from average response time.

Help facilities. Almost every user of an interactive, computer-based system requires help now and then. In most cases, however, modern software provides online help facilities that enable a user to get a question answered or resolve a problem without leaving the interface. A number of design issues must be addressed when a help facility is considered:

- Will help be available for all system functions and at all times during system interaction? Options include help for only a subset of all functions and actions or help for all functions.
- How will the user request help? Options include a help menu, a special function key, or a HELP command.
- How will help be represented? Options include a separate window, a reference to a printed document (less than ideal), or a one- or two-line suggestion produced in a fixed screen location.
- How will the user return to normal interaction? Options include a return button displayed on the screen, a function key, or control sequence.
- How will help information be structured? Options include a “flat” structure in which all information is accessed through a keyword, a layered hierarchy of information that provides increasing detail as the user proceeds into the structure, or the use of hypertext.

Error handling. Error messages and warnings are “bad news” delivered to users of interactive systems when something has gone awry. At their worst, error messages and warnings impart useless or misleading information and serve only to increase user frustration.

In general, every error message or warning produced by an interactive system should have the following characteristics:

- The message should describe the problem in jargon that the user can understand.
- The message should provide constructive advice for recovering from the error.
- The message should indicate any negative consequences of the error (e.g., potentially corrupted data files) so that the user can check to ensure that they have not occurred (or correct them if they have).
- The message should be accompanied by an audible or visual cue. That is, a beep might be generated to accompany the display of the message, or the message might flash momentarily or be displayed in a color that is easily recognizable as the “error color.”
- The message should be “nonjudgmental.” That is, the wording should never place blame on the user.

Menu and command labeling. The typed command was once the most common mode of interaction between user and system software and was commonly used for applications of every type. Today, the use of window-oriented, point-and-pick interfaces has reduced reliance on typed commands. A number of design issues arise when typed commands or menu labels are provided as a mode of interaction:

- Will every menu option have a corresponding command?
- What form will commands take? Options include a control sequence (e.g., alt-P), function keys, or a typed word.
- How difficult will it be to learn and remember the commands? What can be done if a command is forgotten?
- Can commands be customized or abbreviated by the user?
- Are menu labels self-explanatory within the context of the interface?
- Are submenus consistent with the function implied by a master menu item?

Application accessibility. As computing applications become *ubiquitous*, software engineers must ensure that interface design encompasses mechanisms that enable easy access for those with special needs. Accessibility for users (and software engineers) who may be *physically challenged* is an imperative for ethical, legal, and business reasons. A variety of accessibility guidelines - many designed for Web applications but often applicable to all types of software—provide detailed suggestions for designing interfaces that achieve varying levels of accessibility. Others provide specific guidelines for “assistive technology” that addresses the needs of those with visual, hearing, mobility, speech, and learning impairments.

Internationalization. Software engineers and their managers invariably underestimate the effort and skills required to create user interfaces that accommodate the needs of different locales and languages. Too often, interfaces are designed for one locale and language and then jury-rigged to work in other countries. The challenge for interface designers is to create “globalized” software. That is, user interfaces should be designed to accommodate a generic core of

functionality that can be delivered to all who use the software. Localization features enable the interface to be customized for a specific market.

A variety of internationalization guidelines are available to software engineers. These guidelines address broad design issues (e.g., screen layouts may differ in various markets) and discrete implementation issues (e.g., differential alphabets may create specialized labeling and spacing requirements). The Unicode standard has been developed to address the daunting challenge of managing dozens of natural languages with hundreds of characters and symbols.

4.5 WEBAPP INTERFACE DESIGN

WebApp interface need to answer three primary questions for the end user:

Where am I? The interface should (1) provide an indication of the WebApp that has been accessed and (2) inform the user of her location in the content hierarchy.

What can I do now? The interface should always help the user understand his current options—what functions are available, what links are live, what content is relevant?

Where have I been, where am I going? The interface must facilitate navigation. Hence, it must provide a “map” of where the user has been and what paths may be taken to move elsewhere within the WebApp. An effective WebApp interface must provide answers for each of these questions as the end user navigates through content and functionality.

4.5.1 Interface Design Principles and Guidelines: The user interface of a WebApp is its “first impression.” Because of the sheer volume of competing WebApps in virtually every subject area, the interface must “grab” a potential user immediately.

Effective interfaces do not concern the user with the inner workings of the system. Effective applications and services perform a maximum of work, while requiring a minimum of information from users. The designer of the WebApp should anticipate that the user might request a download of the driver and should provide navigation facilities that allow this to happen without requiring the user to search for this capability.

Communication. The interface should communicate the status of any activity initiated by the user. Communication can be obvious (e.g., a text message) or subtle (e.g., an image of a sheet of paper moving through a printer to indicate that printing is under way). The interface should also communicate user status (e.g., the user’s identification) and her location within the WebApp content hierarchy.

Consistency. The use of navigation controls, menus, icons, and aesthetics (e.g., color, shape, layout) should be consistent throughout the WebApp

Controlled autonomy. The interface should facilitate user movement throughout the WebApp, but it should do so in a manner that enforces navigation conventions that have been established for the application. For example, navigation to secure portions of the WebApp should be controlled by userID and password.

Efficiency. The design of the WebApp and its interface should optimize the user's work efficiency, not the efficiency of the developer who designs and builds it or the client server environment that executes it.

Flexibility. The interface should be flexible enough to enable some users to accomplish tasks directly and others to explore the WebApp in a somewhat random fashion.

Focus. The WebApp interface (and the content it presents) should stay focused on the user task(s) at hand.

Fitt's law. "The time to acquire a target is a function of the distance to and size of the target" If a sequence of selections or standardized inputs (with many different options within the sequence) is defined by a user task, the first selection (e.g., mouse pick) should be physically close to the next selection.

Human interface objects. A vast library of reusable human interface objects has been developed for WebApps. Use them. Any interface object that can be "seen, heard, touched or otherwise perceived" by an end user can be acquired from any one of a number of object libraries.

Latency reduction. Rather than making the user wait for some internal operation to complete (e.g., downloading a complex graphical image), the WebApp should use multitasking in a way that lets the user proceed with work as if the operation has been completed. In addition to reducing latency, delays must be acknowledged so that the user understands what is happening. This includes (1) providing audio feedback when a selection does not result in an immediate action by the WebApp, (2) displaying an animated clock or progress bar to indicate that processing is under way, and (3) providing some entertainment (e.g., an animation or text presentation) while lengthy processing occurs.

Learnability. A WebApp interface should be designed to minimize learning time, and once learned, to minimize relearning required when the WebApp is revisited. In general the interface should emphasize a simple, intuitive design that organizes content and functionality into categories that are obvious to the user.

Metaphors. An interface that uses an interaction metaphor is easier to learn and easier to use, as long as the metaphor is appropriate for the application and the user. A metaphor should call on images and concepts from the user's experience, but it does not need to be an exact reproduction of a real-world experience.

Maintain work product integrity. A work product must be automatically saved so that it will not be lost if an error occurs. To avoid data loss, a WebApp should be designed to autosave all user-specified data. The interface should support this function and provide the user with an easy mechanism for recovering "lost" information.

Readability. All information presented through the interface should be readable by young and old. The interface designer should emphasize readable type styles, font sizes, and color background choices that enhance contrast.

Track state. When appropriate, the state of the user interaction should be tracked and stored so that a user can logoff and return later to pick up where she left off. In general, cookies can be

designed to store state information. However, cookies are a controversial technology, and other design solutions may be more palatable for some users.

Visible navigation. A well-designed WebApp interface provides “the illusion that users are in the same place, with the work brought to them”

- Reading speed on a computer monitor is approximately 25 percent slower than reading speed for hardcopy. Therefore, do not force the user to read voluminous amounts of text, particularly when the text explains the operation of the WebApp or assists in navigation.
- Avoid “under construction” signs—an unnecessary link is sure to disappoint.
- Users prefer not to scroll. Important information should be placed within the dimensions of a typical browser window.
- Navigation menus and head bars should be designed consistently and should be available on all pages that are available to the user. The design should not rely on browser functions to assist in navigation.
- Aesthetics should never supersede functionality. For example, a simple button might be a better navigation option than an aesthetically pleasing, but vague image or icon whose intent is unclear.
- Navigation options should be obvious, even to the casual user. The user should not have to search the screen to determine how to link to other content or services.

A well-designed interface improves the user’s perception of the content or services provided by the site. It need not necessarily be flashy, but it should always be well structured and ergonomically sound.

4.5.2 Interface Design Workflow for WebApps: Information contained within the requirements model forms the basis for the creation of a screen layout that depicts graphical design and placement of icons, definition of descriptive screen text, specification and titling for windows, and specification of major and minor menu items. Tools are then used to prototype and ultimately implement the interface design model. The following tasks represent a rudimentary workflow for WebApp interface design:

- 1. Review information contained in the requirements model and refine as required.**
- 2. Develop a rough sketch of the WebApp interface layout.** An interface prototype (including the layout) may have been developed as part of the requirements modeling activity. If the layout already exists, it should be reviewed and refined as required. If the interface layout has not been developed, you should work with stakeholders to develop it at this time.
- 3. Map user objectives into specific interface actions.** For the vast majority of WebApps, the user will have a relatively small set of primary objectives. These should be mapped into specific interface actions as shown in Figure 11.4. In essence, you must answer the following question: “How does the interface enable the user to accomplish each objective?”

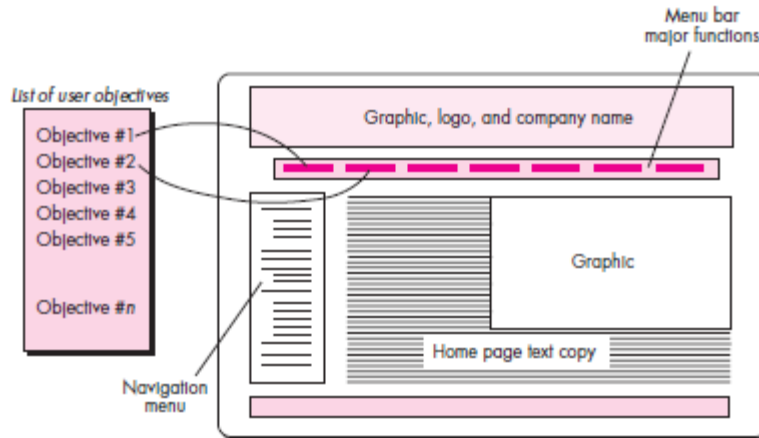


Fig 11.4: Mapping user objectives into interface actions

4. Define a set of user tasks that are associated with each action. Each interface action (e.g., “buy a product”) is associated with a set of user tasks. These tasks have been identified during requirements modeling. During design, they must be mapped into specific interactions that encompass navigation issues, content objects, and WebApp functions.

5. Storyboard screen images for each interface action. As each action is considered, a sequence of storyboard images (screen images) should be created to depict how the interface responds to user interaction. Content objects should be identified (even if they have not yet been designed and developed), WebApp functionality should be shown, and navigation links should be indicated.

6. Refine interface layout and storyboards using input from aesthetic design. In most cases, you’ll be responsible for rough layout and storyboarding, but the aesthetic look and feel for a major commercial site is often developed by artistic, rather than technical, professionals. Aesthetic design is integrated with the work performed by the interface designer.

7. Identify user interface objects that are required to implement the interface. This task may require a search through an existing object library to find those reusable objects (classes) that are appropriate for the WebApp interface. In addition, any custom classes are specified at this time.

8. Develop a procedural representation of the user’s interaction with the interface. This optional task uses UML sequence diagrams and/or activity diagrams to depict the flow of activities (and decisions) that occur as the user interacts with the WebApp.

9. Develop a behavioral representation of the interface. This optional task makes use of UML state diagrams to represent state transitions and the events that cause them. Control mechanisms (i.e., the objects and actions available to the user to alter a WebApp state) are defined.

10. Describe the interface layout for each state. Using design information developed in Tasks 2 and 5, associate a specific layout or screen image with each WebApp state described in Task

11. Refine and review the interface design model. Review of the interface should focus on usability. It is important to note that the final task set you choose should be adapted to the special requirements of the application that is to be built.

4.6 DESIGN EVALUATION

The user interface evaluation cycle takes the form shown in Figure 11.5. After the design model has been completed, a first-level prototype is created. The prototype is evaluated by the user, who provides you with direct comments about the efficacy of the interface. In addition, if formal evaluation techniques are used (e.g., questionnaires, rating sheets), you can extract information from these data (e.g., 80 percent of all users did not like the mechanism for saving data files). Design modifications are made based on user input, and the next level prototype is created. The evaluation cycle continues until no further modifications to the interface design are necessary.

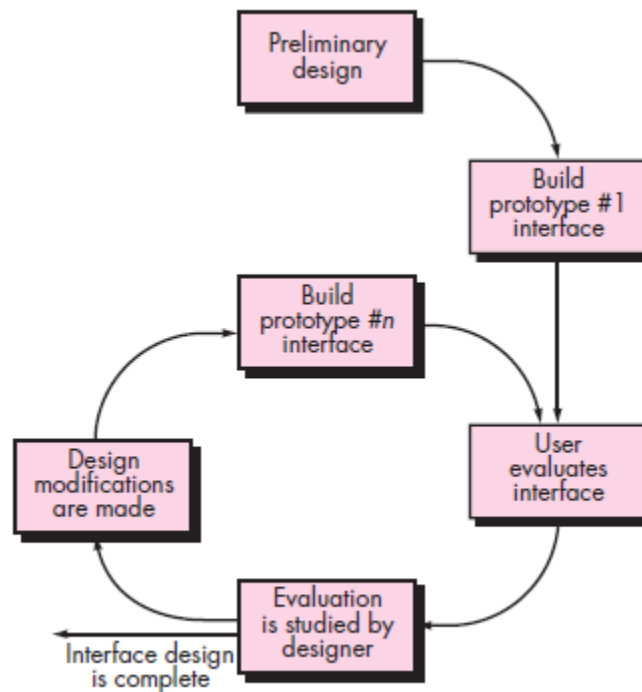


Fig 11.5: The interface design evaluation cycle

The prototyping approach is effective, but is it possible to evaluate the quality of a user interface before a prototype is built? If you identify and correct potential problems early, the number of loops through the evaluation cycle will be reduced and development time will shorten. If a design model of the interface has been created, a number of evaluation criteria can be applied during early design reviews:

1. The length and complexity of the requirements model or written specification of the system and its interface provide an indication of the amount of learning required by users of the system.
2. The number of user tasks specified and the average number of actions per task provide an indication of interaction time and the overall efficiency of the system.
3. The number of actions, tasks, and system states indicated by the design model imply the memory load on users of the system.
4. Interface style, help facilities, and error handling protocol provide a general indication of the complexity of the interface and the degree to which it will be accepted by the user.

Once the first prototype is built, you can collect a variety of qualitative and quantitative data that will assist in evaluating the interface. To collect qualitative data, questionnaires can be distributed to users of the prototype. Questions can be: (1) simple yes/no response, (2) numeric response, (3) scaled (subjective) response, (4) Likert scales (e.g., strongly agree, somewhat agree), (5) percentage (subjective) response, or (6) open-ended.

If quantitative data are desired, a form of time-study analysis can be conducted.

4.7 WEBAPP DESIGN

What is it? Design for WebApps encompasses technical and nontechnical activities that include: establishing the look and feel of the WebApp, creating the aesthetic layout of the user interface, defining the overall architectural structure, developing the content and functionality that reside within the architecture, and planning the navigation that occurs within the WebApp.

Who does it? Web engineers, graphic designers, content developers, and other stakeholders all participate in the creation of a WebApp design model.

Why is it important? Design allows you to create a model that can be assessed for quality and improved before content and code are generated, tests are conducted, and end users become involved in large numbers. Design is the place where WebApp quality is established.

What are the steps? WebApp design encompasses six major steps that are driven by information obtained during requirements modeling. Content design uses the content model (developed during analysis) as the basis for establishing the design of content objects. Aesthetic design (also called graphic design) establishes the look and feel that the end user sees. Architectural design focuses on the overall hypermedia structure of all content objects and functions. Interface design establishes the layout and interaction mechanisms that define the user interface. Navigation design defines how the end user navigates through the hypermedia structure, and component design represents the detailed internal structure of functional elements of the WebApp.

What is the work product? A design model that encompasses content, aesthetics, architecture, interface, navigation, and component-level design issues is the primary work product that is produced during WebApp design.

How do I ensure that I've done it right? Each element of the design model is reviewed in an effort to uncover errors, inconsistencies, or omissions. In addition, alternative solutions are considered, and the degree to which the current design model will lead to an effective implementation is also assessed.

4.7.1 Web App Design Quality: Design is the engineering activity that leads to a high-quality product. In fact, the user’s perception of “goodness” might be more important than any technical discussion of WebApp quality . Olsina and his colleagues have prepared a “quality requirement tree” that identifies a set of technical attributes—usability, functionality, reliability, efficiency, and maintainability—that lead to high-quality WebApps. Figure 13.1 summarizes their work.

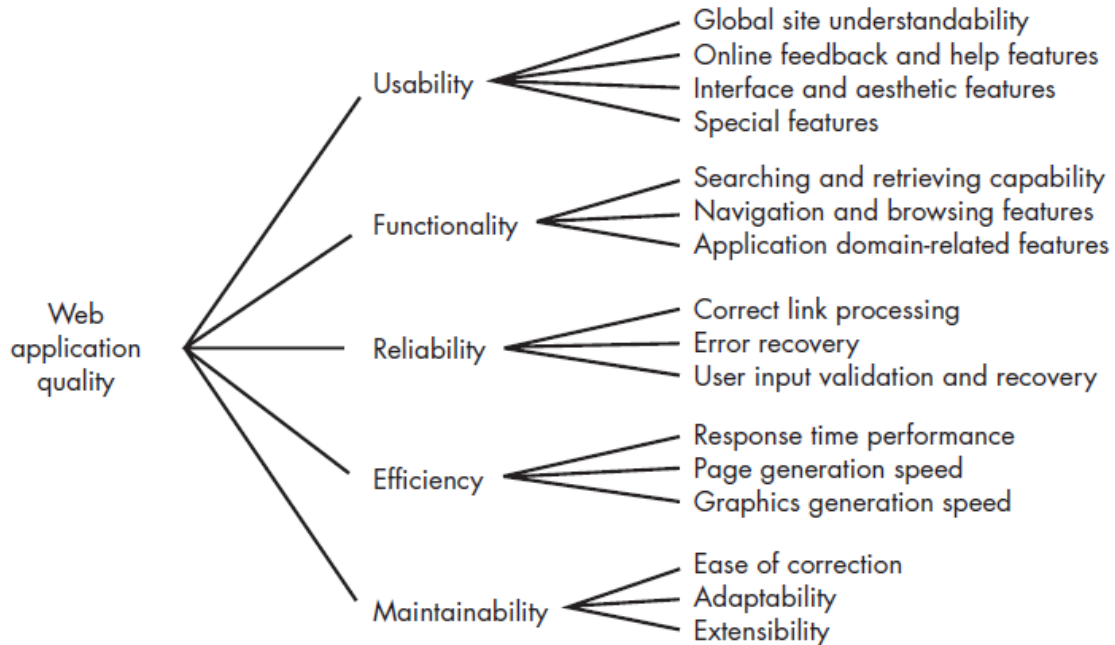


Figure 13.1: Quality requirements Tree

Security. WebApps have become heavily integrated with critical corporate and government databases. E-commerce applications extract and then store sensitive customer information. For these and many other reasons, WebApp security is paramount in many situations. The key measure of security is the ability of the WebApp and its server environment to rebuff unauthorized access and/or thwart an outright malevolent attack.

Availability. Even the best WebApp will not meet users’ needs if it is unavailable. In a technical sense, availability is the measure of the percentage of time that a WebApp is available for use. The typical end user expects WebApps to be available 24/7/365. Anything less is deemed unacceptable.³ But “up-time” is not the only indicator of availability. Offutt suggests that “using features available on only one browser or one platform” makes the WebApp unavailable to those with a different browser/platform configuration. The user will invariably go elsewhere.

Scalability. Can the WebApp and its server environment be scaled to handle 100, 1000, 10,000, or 100,000 users? Will the WebApp and the systems with which it is interfaced handle significant variation in volume or will responsiveness drop dramatically? It is not enough to build a WebApp that is successful. It is equally important to build a WebApp that can accommodate the burden of success and become even more successful.

Time-to-market. Although time-to-market is not a true quality attribute in the technical sense, it is a measure of quality from a business point of view. The first WebApp to address a specific market segment often captures a disproportionate number of end users.

4.7.2 Design Goals: *Jean Kaiser* suggests a set of design goals that are applicable to virtually every WebApp regardless of application domain, size, or complexity:

Simplicity. Better the web design should be moderation and simple. Content should be informative but succinct and should use a delivery mode (e.g., text, graphics, video, audio) that is appropriate to the information that is being delivered. Aesthetics should be pleasing, but not overwhelming (e.g., too many colors tend to distract the user rather than enhancing the interaction). Functions should be easy to use and easier to understand.

Consistency. Content should be constructed consistently. . Graphic design (aesthetics) should present a consistent look across all parts of the WebApp. Architectural design should establish templates that lead to a consistent hypermedia structure. Interface design should define consistent modes of interaction, navigation, and content display. Navigation mechanisms should be used consistently across all WebApp elements

Identity. The aesthetic, interface, and navigational design of a WebApp must be consistent with the application domain for which it is to be built. You should work to establish an identity for the WebApp through the design.

Robustness Based on the identity that has been established, a WebApp often makes an implicit “promise” to a user. The user expects robust content and functions that are relevant to the user’s needs. If these elements are missing or insufficient, it is likely that the WebApp will fail.

Navigability The navigation should be simple and consistent. It should also be designed in a manner that is intuitive and predictable. That is, the user should understand how to move about the WebApp without having to search for navigation links or instructions.

Visual Appeal. Beauty (visual appeal) is undoubtedly in the eye of the beholder, but many design characteristics do contribute to visual appeal.

Compatibility. A WebApp will be used in a variety of environments and must be designed to be compatible with each.

4.7.3 A Design Pyramid For Webapps

What is WebApp design? The creation of an effective design will typically require a diverse set of skills. Sometimes, for small projects, a single developer may need to be multi-skilled. For larger projects, it may be advisable and/or feasible to draw on the expertise of specialists: Web engineers, graphic designers, content developers, programmers, database specialists, information architects, network engineers, security experts, and testers. Drawing on these diverse skills allows the creation of a model that can be assessed for quality and improved before content and code are generated, tests are conducted, and end-users become involved in large numbers. If analysis is where WebApp quality is established, then design is where the

quality is truly embedded. The appropriate mix of design skills will vary depending upon the nature of the WebApp. Figure 13.2 depicts a design pyramid for WebApps. Each level of the pyramid represents a design action.

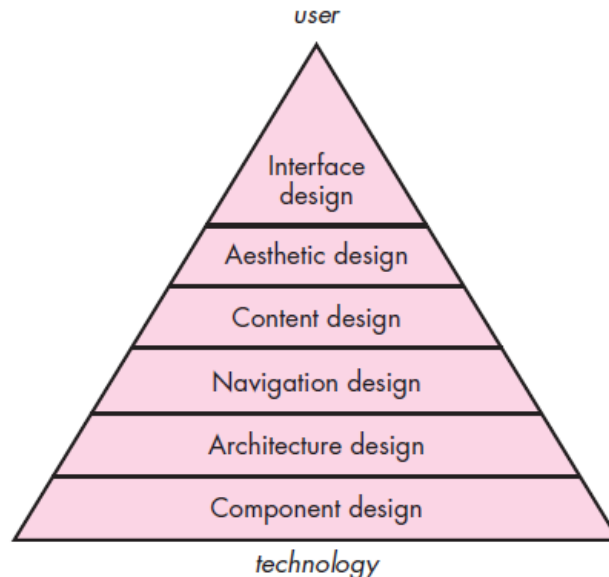


Fig 13.2: A design pyramid for WebApps

4.7.4 Webapp Interface Design: One of the challenges of interface design for WebApps is the indeterminate nature of the user's entry point. The objectives of a WebApp interface are to: (1) establish a consistent window into the content and functionality provided by the interface, (2) guide the user through a series of interactions with the WebApp, and (3) organize the navigation options and content available to the user. To achieve a consistent interface, you should first use aesthetic design to establish a coherent "look." This encompasses many characteristics, but must emphasize the layout and form of navigation mechanisms. To guide user interaction, you may draw on an appropriate metaphor⁵ that enables the user to gain an intuitive understanding of the interface. To implement navigation options, you can select from one of a number of interaction mechanisms:

- **Navigation menus**—keyword menus (organized vertically or horizontally) that list key content and/or functionality. These menus may be implemented so that the user can choose from a hierarchy of subtopics that is displayed when the primary menu option is selected.
- *Graphic icons*—button, switches, and similar graphical images that enable the user to select some property or specify a decision.
- *Graphic images*—some graphical representation that is selectable by the user and implements a link to a content object or WebApp functionality.

4.7.5 AESTHETIC DESIGN

Aesthetic design, also called *graphic design*, is an artistic endeavor that complements the technical aspects of WebApp design. Without it, a WebApp may be functional, but unappealing.

4.7.5.1 Layout Issues: Like all aesthetic issues, there are no absolute rules when screen layout is designed. However, a number of general layout guidelines are worth considering:

Don't be afraid of white space. It is inadvisable to pack every square inch of a Web page with information. The resulting clutter makes it difficult for the user to identify needed information or features and create visual chaos that is not pleasing to the eye.

Emphasize content. After all, that's the reason the user is there. Nielsen suggests that the typical Web page should be 80 percent content with the remaining real estate dedicated to navigation and other features.

Organize layout elements from top-left to bottom-right. The vast majority of users will scan a Web page in much the same way as they scan the page of a book—top-left to bottom-right. If layout elements have specific priorities, high-priority elements should be placed in the upper-left portion of the page real estate.

Group navigation, content, and function geographically within the page. Humans look for patterns in virtually all things. If there are no discernable patterns within a Web page, user frustration is likely to increase (due to unnecessary searching for needed information).

Don't extend your real estate with the scrolling bar. Although scrolling is often necessary, most studies indicate that users would prefer not to scroll. It is better to reduce page content or to present necessary content on multiple pages.

Consider resolution and browser window size when designing layout. Rather than defining fixed sizes within a layout, the design should specify all layout items as a percentage of available space.

4.7.5.2 Graphic Design Issues: Graphic design considers every aspect of the look and feel of a WebApp. The graphic design process begins with layout and proceeds into a consideration of global color schemes; text types, sizes, and styles; the use of supplementary media (e.g., audio, video, animation); and all other aesthetic elements of an application.

4.8 CONTENT DESIGN

Content design focuses on two different design tasks. First, a design representation for *content objects* and the mechanisms required to establish their relationship to one another is developed. Second, the *information* within a specific content object is created.

4.8.1 Content Objects: A content object has attributes that include content-specific information (normally defined during WebApp requirements modeling) and implementation-specific attributes that are specified as part of design.

4.8.2 Content Design Issues: Once all content objects are modeled, the information that each object is to deliver must be authored and then formatted to best meet the customer's needs. Content authoring is the job of specialists in the relevant area who design the content object by providing an outline of information to be delivered and an indication of the types of generic content objects (e.g., descriptive text, graphic images, photographs) that will be used to deliver the information. Aesthetic design may also be applied to represent the proper look and feel for the content.

As content objects are designed, they are "chunked" to form WebApp pages. The number of content objects incorporated into a single page is a function of user needs, constraints imposed by download speed of the Internet connection, and restrictions imposed by the amount of scrolling that the user will tolerate.

4.9 ARCHITECTURE DESIGN

Content architecture focuses on the manner in which content objects (or composite objects such as Web pages) are structured for presentation and navigation. WebApp architecture addresses the manner in which the application is structured to manage user interaction, handle internal processing tasks, effect navigation, and present content.

In most cases, architecture design is conducted in parallel with interface design, aesthetic design, and content design. Because the WebApp architecture may have a strong influence on navigation, the decisions made during this design action will influence work conducted during navigation design.

4.9.1 Content Architecture: The design of content architecture focuses on the definition of the overall hypermedia structure of the WebApp. Although custom architectures are sometimes created, you always have the option of choosing from four different content structures

Linear structures (Figure 13.4) are encountered when a predictable sequence of interactions (with some variation or diversion) is common.

Grid structures (Figure 13.5) are an architectural option that you can apply when WebApp content can be organized categorically in two (or more) dimensions.

Hierarchical structures (Figure 13.6) are undoubtedly the most common WebApp architecture.

A networked or "pure web" structure (Figure 13.7) is similar in many ways to the architecture that evolves for object-oriented systems.

The architectural structures can be combined to form composite structures.

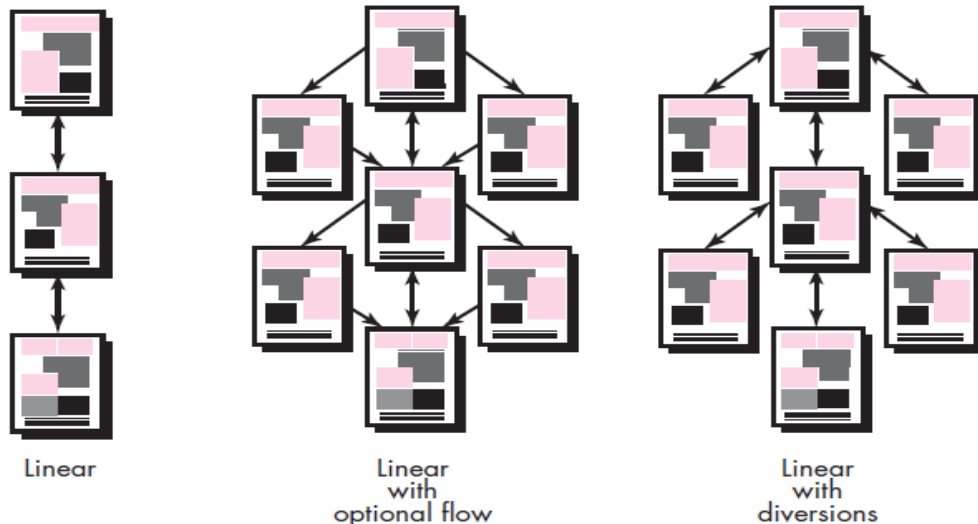


Fig 13.4: Linear Structures

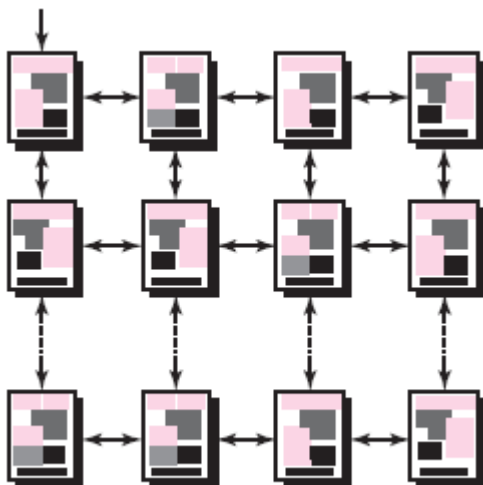


Fig 13.5: Grid Structure

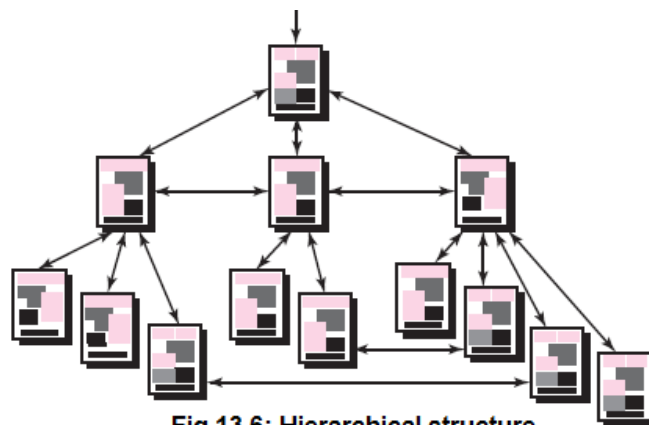


Fig 13.6: Hierarchical structure

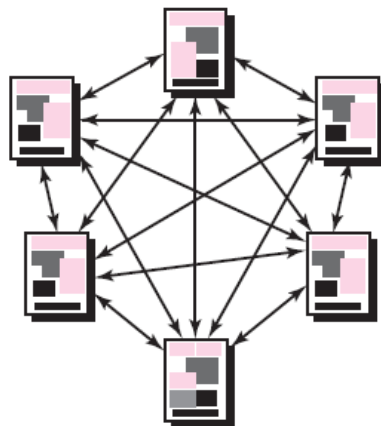


Fig 13.7: Network structure

4.9.2 WebApp Architecture: Jacyntho and his colleagues describe the basic characteristics of this infrastructure in the following manner:

Applications should be built using layers in which different concerns are taken into account; in particular, application data should be separated from the page's contents (navigation nodes) and these contents, in turn, should be clearly separated from the interface look-and-feel (pages).

The three-layer design architecture that decouples interface from navigation and from application behavior. They argue that keeping interface, application, and navigation separate simplifies implementation and enhances reuse. The Model-View-Controller (MVC) architecture is one of a number of suggested WebApp infrastructure models that decouple the user interface

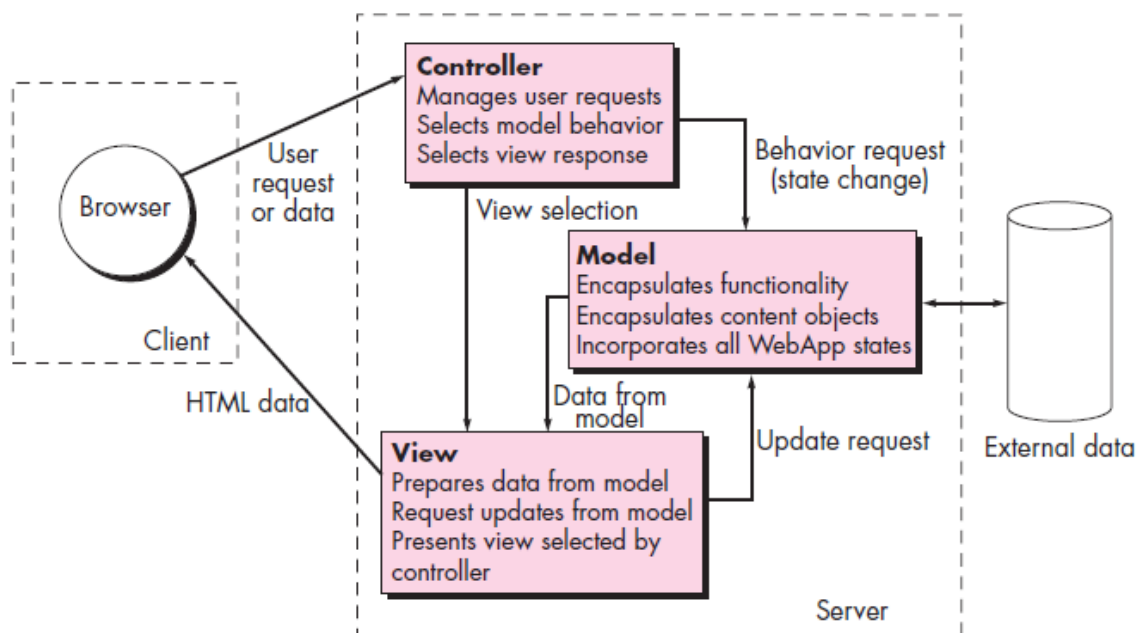


Fig 13.8: The MVC architecture

from the WebApp functionality and informational content. The *model* (sometimes referred to as the “model object”) contains all application-specific content and processing logic, including all content objects, access to external data/information sources, and all processing functionality that is application specific. The *view* contains all interface specific functions and enables the presentation of content and processing logic, including all content objects, access to external data/information sources, and all processing functionality required by the end user. The *controller* manages access to the model and the view and coordinates the flow of data between them. In a WebApp, “the view is updated by the controller with data from the model based on user input”. A schematic representation of the MVC architecture is shown in Figure 13.8.

4.10 NAVIGATION DESIGN

Once the WebApp architecture has been established and the components (pages, scripts, applets, and other processing functions) of the architecture have been identified, you must define navigation pathways that enable users to access WebApp content and functions. To accomplish this, you should (1) identify the semantics of navigation for different users of the site, and (2) define the mechanics (syntax) of achieving the navigation.

4.10.1 Navigation Semantics: Like many WebApp design actions, navigation design begins with a consideration of the user hierarchy and related use cases developed for each category of user (actor). Each actor may use the WebApp somewhat differently and therefore have different navigation requirements. A series of navigation semantic units (NSUs)—“a set of information and related navigation structures that collaborate in the fulfillment of a subset of related user requirements. The overall navigation structure for a WebApp may be organized as a hierarchy of NSUs.

4.10.2 Navigation Syntax: As design proceeds, your next task is to define the mechanics of navigation. A number of options are available as you develop an approach for implementing each NSU:

- Individual navigation link—includes text-based links, icons, buttons and switches, and graphical metaphors. You must choose navigation links that are appropriate for the content and consistent with the heuristics that lead to high-quality interface design.
- Horizontal navigation bar—lists major content or functional categories in a bar containing appropriate links. In general, between four and seven categories are listed.
- Vertical navigation column—(1) lists major content or functional categories, or (2) lists virtually all major content objects within the WebApp. If you choose the second option, such navigation columns can “expand” to present content objects as part of a hierarchy (i.e., selecting an entry in the original column causes an expansion that lists a second layer of related content objects).
- Tabs—a metaphor that is nothing more than a variation of the navigation bar or column, representing content or functional categories as tab sheets that are selected when a link is required.
- Site maps—provide an all-inclusive table of contents for navigation to all content objects and functionality contained within the WebApp. In addition to choosing the mechanics of navigation, you should also establish appropriate navigation conventions and aids.

4.11 COMPONENT-LEVEL DESIGN

Modern WebApps deliver increasingly sophisticated processing functions that (1) perform localized processing to generate content and navigation capability in a dynamic fashion, (2) provide computation or data processing capability that are appropriate for the WebApp’s business domain, (3) provide sophisticated database query and access, and (4) establish data

interfaces with external corporate systems. To achieve these (and many other) capabilities, you must design and construct program components that are identical in form to software components for traditional software.

4.12 OBJECT-ORIENTED HYPERMEDIA DESIGN METHOD (OOHDM)

One of the most widely discussed WebApp design methods— OOHDM. *Daniel Schwabe* and his colleagues originally proposed the Object-Oriented Hypermedia Design Method (OOHDM), which is composed of four different design activities: conceptual design, navigational design, abstract interface design, and implementation. A summary of these design activities is shown in Figure 13.10


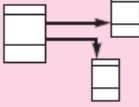


	 Conceptual design	 Navigational design	 Abstract interface design	 Implementation
Work products	Classes, subsystems, relationships, attributes	Nodes links, access structures, navigational contexts, navigational transformations	Abstract interface objects, responses to external events, transformations	Executable WebApp
Design mechanisms	Classification, composition, aggregation, generalization specialization	Mapping between conceptual and navigation objects	Mapping between navigation and perceptible objects	Resource provided by target environment
Design concerns	Modeling semantics of the application domain	Takes into account user profile and task. Emphasis on cognitive aspects.	Modeling perceptible objects, implementing chosen metaphors. Describe interface for navigational objects.	Correctness; application performance; completeness

Fig 13.10: Summary of the OOHDM method.

4.12.1 Conceptual Design for OOHDM: OOHDM conceptual design creates a representation of the subsystems, classes, and relationships that define the application domain for the WebApp. UML may be used to create appropriate class diagrams, aggregations, and composite class representations, collaboration diagrams, and other information that describes the application domain. The class diagrams, aggregations, and related information developed as part of WebApp analysis are reused during conceptual design to represent relationships between classes.

4.12.2 Navigational Design for OOHDM: Navigational design identifies a set of “objects” that are derived from the classes defined in conceptual design. A series of “navigational classes” or “nodes” are defined to encapsulate these objects. UML may be used to create appropriate use

cases, state charts, and sequence diagrams—all representations that assist you in better understanding navigational requirements. In addition, design patterns for navigational design may be used as the design is developed. OOHDM uses a predefined set of navigation classes—nodes, links, anchors, and access structures.

Access structures are more elaborate and include mechanisms such as a WebApp index, a site map, or a guided tour.

4.12.3 Abstract Interface Design and Implementation: The abstract interface design action specifies the interface objects that the user sees as WebApp interaction occurs. A formal model of interface objects, called an abstract data view (ADV), is used to represent the relationship between interface objects and navigation objects, and the behavioral characteristics of interface objects.

The ADV model defines a “static layout” that represents the interface metaphor and includes a representation of navigation objects within the interface and the specification of the interface objects (e.g., menus, buttons, icons) that assist in navigation and interaction. In addition, the ADV model contains a behavioral component that indicates how external events “trigger navigation and which interface transformations occur when the user interacts with the application”.

The OOHDM implementation activity represents a design iteration that is specific to the environment in which the WebApp will operate.