You:

Each of us has encountered a design problem and silently thought: I wonder if anyone has developed a solution for this? The answer is almost always— yes! The problem is finding the solution; ensuring that it does, in fact, fit the problem you've encountered; understanding the constraints that may restrict the manner in which the solution is applied; and finally, translating the proposed solution into your design environment. But what if the solution were codified in some manner? What if there was a standard way of describing a problem (so you could look it up), and an organized method for representing the solution to the problem? It turns out that software problems have been codified and described using a standardized template, and solutions to them (along with constraints) have been proposed. Called design patterns, 347 CHAPTER PATTERN D - B ESIGN ASED 12 What is it? Pattern-based design creates a new application by finding a set of proven solutions to a clearly delineated set of problems. Each problem and its solution is described by a design pattern that has been cataloged and vetted by other software engineers who have encountered the problem and implemented the solution while designing other applications. Each design pattern provides you with a proven approach to one part of the problem to be solved. Who does it? A software engineer examines each problem encountered for a new application and then attempts to find a relevant solution by searching one or more patterns repositories. Why is it important? Have you ever heard the phrase "reinventing the wheel"? It happens all the time in software development, and it's a waste of time and energy. By using existing design patterns, you can acquire a proven solution for a specific problem. As each pattern is applied, solutions are integrated and the application to be built moves closer to a complete design. Q UICK L OOK What are the steps? The requirements model is examined in order to isolate the hierarchical set of problems to be solved. The problem space is partitioned so that subsets of problems associated with specific software functions and features can be identified. Problems can also be organized by type: architectural, componentlevel, algorithmic, user interface, etc. Once a subset of problems is defined, one or more pattern repositories are searched to determine if an existing design pattern, represented at an appropriate level of abstraction, exists. Patterns that are applicable are adapted to the specific needs of the software to be built. Custom problem solving is applied in situations for which no patterns can be found. What is the work product? A design model that depicts the architectural structure, user interface, and component-level detail is developed. How do I ensure that I've done it right? As each design pattern is translated into some element of the design model, work products are reviewed for clarity, correctness, completeness, and consistency with requirements and with one another. K E Y C ONCEPTS design mistakes . . . . .359 forces . . . . . .349 frameworks . . .352 granularity . . . .369 pattern languages . . . .353 patterns architectural . .360 behavioral . . .351 this codified method for describing problems and their solution allows the software engineering community to capture design knowledge in a way that enables it to be reused. The early history of software patterns begins not with a computer scientist but a building architect, Christopher Alexander, who recognized that a recurring set of problems were encountered whenever a building was designed. He characterized these recurring problems and their solutions as patterns, describing them in the following manner [Ale77]: Each pattern describes a problem that occurs over and over again in our environment and then describes the core of the solution to that problem in such a way that you can use the solution a million times over without ever doing it the same way twice. Alexander's ideas were first translated into the software world in books by Gamma [Gam95], Buschmann [Bus96], and their many colleagues.1 Today, dozens of pattern repositories exist, and pattern-based design can be applied in many different application domains. 348 PART TWO MODELING componentlevel . . . . . . .362 generative . . .350 creational . . . .350 structural . . . .351 user interface . . . .366 WebApps . . . .368 12.1 DESIGN PATTERNS A design pattern can be characterized as "a three-part rule which expresses a relation between a certain context, a problem, and a solution" [Ale79]. For software design, context allows the reader to understand the environment in which the problem resides and what solution might be appropriate within that environment. A set of requirements, including limitations and constraints, acts as a system of forces that influences how the problem can be interpreted within its context and how the solution can be effectively applied. To better understand these concepts, consider a situation2 in which a person must travel between New York and Los Angeles. In this context, travel will occur within an industrialized country (the United States), using an existing transportation infrastructure (e.g., roads, airlines, railways). The system of forces that will affect the way in which the travel problem is solved will include: how quickly the person wants to get from New York to LA, whether the trip will include site-seeing or stopovers, how much money the person can spend, whether the trip is intended to accomplish a specific purpose, and the personal vehicles the person has at her disposal. Given these forces, the problem (traveling from New York to LA) can be better defined. For example, investigation (requirements gathering) indicates that the person has very little money, owns only a bicycle (and is an avid cyclist), wants to make the trip to raise money for her favorite charity, and has plenty of time to spare. The solution to the problem, given the context and the system of forces, might be a cross-country 1 Earlier discussions of software patterns do exist, but these two classic books were the first cohesive treatments of the subject. 2 This example has been adapted from [Cor98]. Forces are those characteristics of the problem and attributes of the solution that constrain the way in which the design can be developed. bike trip. If the forces were different (e.g., travel time must be minimized and the purpose of the trip is a business meeting), another solution might be more appropriate. It is reasonable to argue that most problems have multiple solutions, but that a solution is effective only if it is appropriate within the context of the existing problem. It is the system of forces that causes a designer to choose a specific solution. The intent is to provide a solution that best satisfies the system of forces, even when these forces are contradictory. Finally, every solution has consequences that may have an impact on other aspects of the software and may themselves become part of the system of forces for other problems to be solved within the larger system. Coplien [Cop05] characterizes an effective design pattern in the following way: • It solves a problem: Patterns capture solutions, not just abstract principles or strategies. • It is a proven concept: Patterns capture solutions with a track record, not theories or speculation. • The solution isn’t obvious: Many problem-solving techniques (such as software design paradigms or methods) try to derive solutions from first principles. The best patterns generate a solution to a problem indirectly—a necessary approach for the most difficult problems of design. • It describes a relationship: Patterns don’t just describe modules, but describe deeper system structures and mechanisms. • The pattern has a significant human component (minimize human intervention). All software serves human comfort or quality of life; the best patterns explicitly appeal to aesthetics and utility. Stated even more pragmatically, a good design pattern captures hard-earned, pragmatic design knowledge in a way that enables others to reuse that knowledge “a million times over without ever doing it the same way twice.” A design pattern saves you from “reinventing the wheel,” or worse, inventing a “new wheel” that is slightly out of round, too small for its intended use, and too narrow for the ground it will roll over. Design patterns, if used effectively, will invariably make you a better software designer ANSWER THE QUESTION : Explain the need for pattern based software design.in short

----

ChatGPT:

Pattern-based software design is needed to address the common challenges and recurring problems encountered in software development. By codifying solutions in the form of design patterns, the software engineering community can benefit in several ways:

1. **Efficiency and Reusability:** Design patterns provide proven solutions to well-defined problems. Instead of reinventing solutions for similar problems in different projects, developers can reuse established patterns, saving time and effort.

2. **Knowledge Transfer:** Patterns serve as a means of capturing and transferring design knowledge. They provide a shared vocabulary and understanding within the software development community, facilitating communication and collaboration among team members.

3. **Consistency and Best Practices:** Design patterns embody best practices and design principles. By following established patterns, developers ensure a consistent and reliable approach to solving specific types of problems, leading to higher-quality software.

4. **Avoiding Mistakes:** Design patterns have been vetted and validated by other experienced software engineers. Using these patterns helps developers avoid common pitfalls and mistakes, contributing to more robust and reliable software systems.

5. **Adaptability:** Patterns are adaptable to different contexts and can be customized to fit specific project requirements. This flexibility allows developers to tailor solutions while still benefitting from the underlying design principles.

6. **Human-Centric Design:** Effective design patterns consider the human component, addressing not just technical aspects but also factors related to user experience, aesthetics, and overall quality of life. This holistic approach enhances the usability and acceptance of software systems.

In summary, pattern-based software design is essential for promoting efficiency, consistency, and knowledge sharing within the software development community. It enables developers to leverage established solutions, avoid common pitfalls, and deliver higher-quality software in a more systematic and efficient manner.

You:

A design pattern can be characterized as “a three-part rule which expresses a relation between a certain context, a problem, and a solution” [Ale79]. For software design, context allows the reader to understand the environment in which the problem resides and what solution might be appropriate within that environment. A set of requirements, including limitations and constraints, acts as a system of forces that influences how the problem can be interpreted within its context and how the solution can be effectively applied. To better understand these concepts, consider a situation2 in which a person must travel between New York and Los Angeles. In this context, travel will occur within an industrialized country (the United States), using an existing transportation infrastructure (e.g., roads, airlines, railways). The system of forces that will affect the way in which the travel problem is solved will include: how quickly the person wants to get from New York to LA, whether the trip will include site-seeing or stopovers, how much money the person can spend, whether the trip is intended to accomplish a specific purpose, and the personal vehicles the person has at her disposal. Given these forces, the problem (traveling from New York to LA) can be better defined. For example, investigation (requirements gathering) indicates that the person has very little money, owns only a bicycle (and is an avid cyclist), wants to make the trip to raise money for her favorite charity, and has plenty of time to spare. The solution to the problem, given the context and the system of forces, might be a cross-country 1 Earlier discussions of software patterns do exist, but these two classic books were the first cohesive treatments of the subject. 2 This example has been adapted from [Cor98]. Forces are those characteristics of the problem and attributes of the solution that constrain the way in which the design can be developed. bike trip. If the forces were different (e.g., travel time must be minimized and the purpose of the trip is a business meeting), another solution might be more appropriate. It is reasonable to argue that most problems have multiple solutions, but that a solution is effective only if it is appropriate within the context of the existing problem. It is the system of forces that causes a designer to choose a specific solution. The intent is to provide a solution that best satisfies the system of forces, even when these forces are contradictory. Finally, every solution has consequences that may have an impact on other aspects of the software and may themselves become part of the system of forces for other problems to be solved within the larger system. Coplien [Cop05] characterizes an effective design pattern in the following way: • It solves a problem: Patterns capture solutions, not just abstract principles or strategies. • It is a proven concept: Patterns capture solutions with a track record, not theories or speculation. • The solution isn’t obvious: Many problem-solving techniques (such as software design paradigms or methods) try to derive solutions from first principles. The best patterns generate a solution to a problem indirectly—a necessary approach for the most difficult problems of design. • It describes a relationship: Patterns don’t just describe modules, but describe deeper system structures and mechanisms. • The pattern has a significant human component (minimize human intervention). All software serves human comfort or quality of life; the best patterns explicitly appeal to aesthetics and utility. Stated even more pragmatically, a good design pattern captures hard-earned, pragmatic design knowledge in a way that enables others to reuse that knowledge “a million times over without ever doing it the same way twice.” A design pattern saves you from “reinventing the wheel,” or worse, inventing a “new wheel” that is slightly out of round, too small for its intended use, and too narrow for the ground it will roll over. Design patterns, if used effectively, will invariably make you a better software designer. ANSWER THE QUESTION : Describe design patterns with its needs.in short

----

ChatGPT:

Design patterns are structured solutions to recurring design problems in software development. They consist of three essential parts: **context, problem, and solution**. The context defines the environment in which the problem exists, and the problem is the recurring issue to be addressed. The solution, shaped by a set of requirements and constraints (system of forces), offers an effective resolution within the given context.

Needs for Design Patterns:

1. **Efficiency and Reusability:** Design patterns provide proven, reusable solutions, saving time and effort by avoiding the need to reinvent solutions for common problems.

2. **Knowledge Transfer:** They capture and transfer design knowledge within the software development community, creating a shared vocabulary and understanding.

3. **Consistency and Best Practices:** Design patterns embody best practices, ensuring a consistent and reliable approach to solving specific types of problems.

4. **Avoiding Mistakes:** Patterns help developers avoid common pitfalls and mistakes by offering solutions that have been vetted and validated by experienced professionals.

5. **Adaptability:** Patterns can be customized to fit specific project requirements, providing flexibility while maintaining adherence to underlying design principles.

6. **Human-Centric Design:** Effective design patterns consider human aspects, enhancing the usability and quality of life in software systems.

In essence, design patterns, when used effectively, enable developers to leverage proven solutions, avoid mistakes, and deliver higher-quality software in a systematic and efficient manner. They save developers from reinventing solutions and contribute to a more skilled and proficient software design process.

You:

Every Web page has a limited amount of “real estate” that can be used to support non- functional aesthetics, navigation features, informational content, and user-directed functionality. The development of this real estate is planned during aesthetic design. Like all aesthetic issues, there are no absolute rules when screen layout is de- signed. 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 [Nie00] 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.6 If layout elements have specific 380 PART TWO MODELING What interaction mechanisms are available to WebApp designers? ? Not every Web engineer (or software engineer) has artistic (aesthetic) talent. If you fall into this category, hire an experienced graphic designer for aesthetic design work. ? uote: “We find that people quickly evaluate a site by visual design alone.” Stanford Guidelines for Web Credibility 6 There are exceptions that are cultural and language-based, but this rule does hold for most users. pre75977_ch13.qxd 11/27/08 5:47 PM Page 380 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 dis- cernable 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 scroll- ing 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 [Nie00]. ANSWER THE QUESTION : Describe various layout issues in design of web applications.

ChatGPT:

Several layout issues need careful consideration in the design of web applications to ensure an effective and user-friendly experience. Here are various layout issues:

1. **White Space Management:**
   - **Principle:** Don't be afraid of white space.
   - **Rationale:** Avoid overcrowding the web page, as it makes it difficult for users to identify information and features. White space enhances readability and reduces visual clutter.

2. **Content Emphasis:**
   - **Principle:** Emphasize content.
   - **Rationale:** Prioritize content, allocating about 80% of the page to actual information. This ensures that users can quickly find the reason they visited the web page in the first place.

3. **Top-Left to Bottom-Right Organization:**
   - **Principle:** Organize layout elements from top-left to bottom-right.
   - **Rationale:** Users typically scan a web page in a manner similar to reading a book, from top-left to bottom-right. Place high-priority elements in the upper-left portion to align with user scanning behavior.

4. **Geographical Grouping:**
   - **Principle:** Group navigation, content, and function geographically within the page.
   - **Rationale:** Users look for patterns, and an organized layout enhances user experience. Logical grouping reduces user frustration by making information easier to find.

5. **Scrolling Bar Management:**

- **Principle:** Don't extend your real estate with the scrolling bar.
- **Rationale:** While scrolling is sometimes necessary, users generally prefer not to scroll excessively. It is better to reduce page content or present it on multiple pages to maintain a more user-friendly experience.

6. **Resolution and Browser Window Size Consideration:**

    - **Principle:** Consider resolution and browser window size when designing layout.
    - **Rationale:** Design layouts to be flexible, specifying sizes as percentages of available space rather than fixed dimensions. This ensures a consistent and responsive layout across various devices and screen sizes.

7. **Aesthetic Design Considerations:**

    - **Principle:** Not every web engineer has artistic talent; consider hiring a graphic designer for aesthetic design work.
    - **Rationale:** Aesthetic design plays a crucial role in user perception and engagement. If lacking artistic skills, collaborating with a graphic designer can enhance the overall visual appeal of the web application.

By addressing these layout issues, web designers can create web applications that are visually appealing, user-friendly, and optimized for effective information presentation.

You:

The Model-View-Controller (MVC) architecture [Kra88]9 is one of a number of suggested WebApp infrastructure models that decouple the user interface from the 386 PART TWO MODELING FIGURE 13.7 Network structure 9 It should be noted that MVC is actually an architectural design pattern developed for the Smalltalk environment (see www.cetus-links.org/oo_smalltalk.html) and can be used for any interactive application. pre75977_ch13.qxd 11/27/08 5:47 PM Page 386 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, in- cluding 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” [WMT02]. A schematic representation of the MVC architecture is shown in Figure 13.8. Referring to the figure, user requests or data are handled by the controller. The con- troller also selects the view object that is applicable based on the user request. Once the type of request is determined, a behavior request is transmitted to the model, which implements the functionality or retrieves the content required to accommodate the request. The model object can access data stored in a corporate database, as part of a local data store, or as a collection of independent files. The data developed by the model must be formatted and organized by the appropriate view object and then transmitted from the application server back to the client-based browser for display on the customer’s machine. In many cases, WebApp architecture is defined within the context of the develop- ment environment in which the application is to be implemented. If you have further interest, see [Fow03] for a discussion of development environments and their role in the design of Web application architectures. ANSWER THE QUESTION : Explain MVC architecture in detail.

ChatGPT:

The Model-View-Controller (MVC) architecture is a design pattern that aims to decouple the user interface from the functionality and content of a web application. This separation enhances modularity, maintainability, and scalability. Here's a detailed explanation of each component in the MVC architecture:

1. **Model:**
   - **Definition:** The model represents the application's data and business logic. It encapsulates the content objects, access to external data/information sources, and all processing functionality that is specific to the application.
   - **Responsibilities:**
     - Manages application data.
     - Implements business logic and processing.
     - Communicates with external data sources.

2. **View:**
   - **Definition:** The view is responsible for presenting the data to the user and handling the user interface. It encapsulates all interface-specific functions and is concerned with the presentation of content and processing logic.
   - **Responsibilities:**
     - Displays information to the user.
     - Interacts with the user interface.
     - Receives input and forwards it to the controller.

3. **Controller:**
   - **Definition:** The controller manages the interaction between the model and the view. It handles user requests, updates the model based on user input, and selects the appropriate view to present the data. It acts as an intermediary that coordinates the flow of data between the model and the view.
   - **Responsibilities:**
     - Receives and processes user input.
     - Updates the model based on user actions.
     - Selects the appropriate view for presentation.
     - Manages the flow of data between the model and the view.

MVC Workflow:

1. User input or data is received by the controller.
2. The controller selects the appropriate view based on the user's request.
3. The controller sends a behavior request to the model to perform the required functionality or retrieve content.
4. The model processes the request, accessing data from external sources if necessary.
5. The model returns the processed data to the controller.
6. The controller formats and organizes the data and updates the selected view.
7. The updated view is transmitted to the client-based browser for display.

Benefits of MVC:

- **Modularity:** Each component (model, view, controller) can be developed and maintained independently, promoting modularity and ease of development.
- **Separation of Concerns:** Clear separation between data, presentation, and control logic enhances code maintainability and understandability.
- **Reusability:** Components can be reused in different parts of the application or in other projects.

- **Scalability:** Easy to scale and modify individual components without affecting the entire system.

The MVC architecture is widely used in web application development for its ability to provide a structured and organized approach to building interactive and maintainable applications.

You:

Hundreds of user interface (UI) patterns have been proposed in recent years. Most fall within one of the following 10 categories of patterns (discussed with a represen- tative example8) as described by Tidwell [Tid02] and vanWelie [Wel01]: Whole UI. Provide design guidance for top-level structure and navigation throughout the entire interface. Pattern: TopLevelNavigation Brief description: Used when a site or application implements a number of major functions. Provides a top-level menu, often coupled with a logo or 364 PART TWO MODELING 8 An abbreviated pattern template is used here. Full pattern descriptions (along with dozens of other patterns) can be found at [Tid02] and [Wel01]. Applying Patterns The scene: Informal discussion during the design of a software increment that implements sensor control via the Internet for SafeHomeAssured.com. The players: Jamie (responsible for design) and Vinod (SafeHomeAssured.com chief system architect). The conversation: Vinod: So how is the design of the camera control interface coming along? Jamie: Not too bad. I've designed most of the capability to connect to the actual sensors without too many problems. I've also started thinking about the interface for the users to actually move, pan, and zoom the cameras from a remote Web page, but I'm not sure I've got it right yet. Vinod: What have you come up with? Jamie: Well, the requirements are that the camera control needs to be highly interactive—as the user moves the control, the camera should move as soon as possible. So, I was thinking of having a set of buttons laid out like a normal camera, but when the user clicks them, it controls the camera. Vinod: Hmmm. Yeah, that would work, but I'm not sure it's right—for each click of a control you need to wait for the whole client-server communication to occur, and so you won't get a good sense of quick feedback. Jamie: That's what I thought—and why I wasn't very happy with the approach, but I'm not sure how else I might do it. Vinod: Well, why not just use the InteractiveDeviceControl pattern! Jamie: Uhmmm—what's that? I haven't heard of it? Vinod: It's basically a pattern for exactly the problem you are describing. The solution it proposes is basically to create a control connection to the server with the device, through which control commands can be sent. That way you don't need to send normal HTTP requests. And the pattern even shows how you can implement this using some simple AJAX techniques. You have some simple client-side JavaScript that communicates directly with the server and sends the commands as soon as the user does anything. Jamie: Cool! That's just what I needed to solve this thing. Where do I find it? Vinod: It's available in an online repository. Here's the URL. Jamie: I'll go check it out. Vinod: Yep—but remember to check the consequences field for the pattern. I seem to remember that there was something in there about needing to be careful about issues of security. I think it might be because you are creating a separate control channel and so bypassing the normal Web security mechanisms. Jamie: Good point. I probably wouldn't have thought of that! Thanks. SAFEHOME pre75977_ch12.qxd 11/27/08 3:58 PM Page 364 CHAPTER 12 PATTERN-BASED DESIGN 365 identifying graphic, that enables direct navigation to any of the system's major functions. Details: Major functions (generally limited to between four and seven func- tion names) are listed across the top of the display (vertical column formats are also possible) in a horizontal line of text. Each name provides a link to the appropriate function or information source. Often used with the BreadCrumbs pattern discussed later. Navigation elements: Each function/content name represents a link to the appropriate function or content. Page layout. Address the general organization of pages (for websites) or distinct screen displays (for interactive applications). Pattern: CardStack Brief description: Used when a number of specific subfunctions or content categories related to a feature or function must be selected in random order. Provides the appearance of a stack of tabbed cards, each selectable with a mouse click and each representing specific subfunctions or content categories. Details: Tabbed cards are a well-understood metaphor and are easy for the user to manipulate. Each tabbed card (divider) may have a slightly different format. Some may require input and have buttons or other navigation mech- anisms; others may be informational. May be combined with other patterns such as DropDownList, Fill-in-the-Blanks, and others. Navigation elements: A mouse click on a tab causes the appropriate card to appear. Navigation features within the card may also be present, but in general, these should initiate a function that is related to card data, not cause an actual link to some other display. Forms and input. Consider a variety of design techniques for completing form- level input. Pattern: Fill-in-the-Blanks Brief description: Allow alphanumeric data to be entered in a “text box.” Details: Data may be entered within a text box. In general, the data are validated and processed after some text or graphic indicator (e.g., a button containing “go,” “submit,” “next”) is picked. In many cases this pattern can be combined with drop-down list or other patterns (e.g., SEARCH <drop down list> FOR <fill-in-the-blanks text box>). Navigation elements: A text or graphic indicator that initiates validation and processing. Tables. Provide design guidance for creating and manipulating tabular data of all kinds. pre75977_ch12.qxd 11/27/08 3:58 PM Page 365 Pattern: SortableTable Brief description: Display a long list of records that can be sorted by selecting a toggle mechanism for any column label. Details: Each row in the table represents a complete record. Each column represents one field in the record. Each column header is actually a selec- table button that can be toggled to initiate an ascending or descending sort on the field associated with the column for all records displayed. The table is generally resizable and may have a scrolling mechanism if the number of records is larger than available window space. Navigation elements: Each column header initiates a sort on all records. No other navigation is provided, although in some cases, each record may itself contain navigation links to other content or functionality. Direct data manipulation. Address data editing, modification, and transformation. Pattern: BreadCrumbs Brief description: Provides a full navigation path when the user is working with a complex hierarchy of pages or display screens. Details: Each page or display screen is given a unique identifier. The naviga- tion path to the current location is specified in a predefined location for every display. The path takes the form: home>major topic page>subtopic page> specific page>current page. Navigation elements: Any of the entries within the bread crumbs display can be used as a pointer to link back to a higher level of the hierarchy. Navigation. Assist the user in navigating through hierarchical menus, Web pages, and interactive display screens. Pattern: EditInPlace Brief description: Provide simple text editing capability for certain types of content in the location that it is displayed. No need for the user to enter a text editing function or mode explicitly. Details: The user sees content on the display that must be changed. A mouse double click on the content indicates to the system that editing is desired. The content is highlighted to signify that editing mode is available and the user makes appropriate changes. Navigation elements: None. Searching. Enable content-specific searches through information maintained within a website or contained by persistent data stores that are accessible via an interactive application. Pattern: SimpleSearch Brief description: Provides the ability to search a website or persistent data source for a simple data item described by an alphanumeric string. 366 PART TWO MODELING pre75977_ch12.qxd 11/27/08 3:58 PM Page 366 Details: Provides the ability to search either locally (one page or one file) or globally (entire site or complete database) for the search string. Generates a list of “hits” in order of their probability of meeting the user’s needs. Does not provide multiple item searches or special Boolean operations (see advanced search pattern). Navigation elements: Each entry in the list of hits represents a navigation link to the data referenced by the entry. Page elements. Implement specific elements of a Web page or display screen. Pattern: Wizard Brief description: Takes the user through a complex task one step at a time, providing guidance for the completion of the task through a series of simple window displays. Details: Classic example is a registration process that contains four steps. The wizard pattern generates a window for each step, requesting specific information from the user one step at a time. Navigation elements: Forward and back navigation allows the user to revisit each step in the wizard process. E-commerce. Specific to websites, these patterns implement recurring elements of e-commerce applications. Pattern: ShoppingCart Brief description: Provides a list of items selected for purchase. Details: Lists item, quantity, product code, availability (in stock, out of stock), price, delivery information, shipping costs, and other relevant purchase infor- mation. Also provides ability to edit (e.g., remove, change quantity). Navigation elements: Contains ability to proceed with shopping or go to checkout. Miscellaneous. Patterns that do not easily fit into one of the preceding cate- gories. In some cases, these patterns are domain dependent or occur only for specific classes of users. Pattern: ProgressIndicator Brief description: Provides an indication of progress when an operation takes longer than n seconds. Details: Represented as an animated icon or a message box that contains some visual indication (e.g., a rotating “barber pole,” a slider with a percent complete indicator) that processing is under way. May also contain a text content indication of the status of processing. Navigation elements: Often contains a button that allows the user to pause or cancel processing. CHAPTER 12 PATTERN-BASED DESIGN 367 pre75977_ch12.qxd 11/27/08 3:58 PM Page 367 Each of the preceding example patterns (and all patterns within each category) would also have a complete component-level design, including design classes, attributes, operations, and interfaces. A comprehensive discussion of user interface patterns is beyond the scope of this book. If you have further interest, see [Duy02], [Bor01], [Tid02], and [Wel01] for further information. ANSWER THE QUESTION : Explain user interface design patterns in detail

----

ChatGPT:

User Interface (UI) design patterns are reusable solutions to common design problems in user interface development. They provide a structured approach to solving specific design challenges and enhance the user experience. Here is an explanation of various UI design patterns along with representative examples:

1. **TopLevelNavigation (Whole UI):**
    - **Description:** Provides design guidance for top-level structure and navigation throughout the entire interface.
    - **Example:** A top-level menu with major functions, often coupled with a logo, enabling direct navigation to system functions.
2. **CardStack (Page layout):**
    - **Description:** Used when specific subfunctions or content categories related to a feature or function must be selected in random order. Presents a stack of tabbed cards, each selectable with a mouse click.
    - **Example:** Tabs representing subfunctions or content categories, allowing users to select and navigate between them.
3. **Fill-in-the-Blanks (Forms and input):**
    - **Description:** Allows alphanumeric data to be entered in a "text box." Data is validated and processed after user input.
    - **Example:** Text box for entering data, followed by a button (e.g., "Submit") for validation and processing.
4. **SortableTable (Tables):**
    - **Description:** Displays a long list of records that can be sorted by selecting a toggle mechanism for any column label.
    - **Example:** A table with sortable columns where users can toggle to arrange data in ascending or descending order.
5. **BreadCrumbs (Direct data manipulation):**
    - **Description:** Provides a full navigation path when the user is working with a complex hierarchy of pages or display screens.
    - **Example:** Home > Major topic page > Subtopic page > Specific page > Current page, displaying the user's navigation path.
6. **EditInPlace (Direct data manipulation):**

- **Description:** Provides simple text editing capability for certain types of content in the location that it is displayed. No need for the user to enter a text editing function explicitly.
- **Example:** Double-clicking on content allows users to edit text directly in its displayed location.

7. **SimpleSearch (Searching):**

    - **Description:** Provides the ability to search a website or persistent data source for a simple data item described by an alphanumeric string.
    - **Example:** A search box allowing users to enter a search string and receive a list of hits related to the query.

8. **Wizard (Page elements):**

    - **Description:** Takes the user through a complex task one step at a time, providing guidance for the completion of the task through a series of simple window displays.
    - **Example:** A registration process with multiple steps, guiding the user through each step in a sequential manner.

9. **ShoppingCart (E-commerce):**

    - **Description:** Provides a list of items selected for purchase, displaying relevant information such as quantity, price, and delivery details.
    - **Example:** A shopping cart displaying selected items, allowing users to edit and proceed to checkout.

10. **ProgressIndicator (Miscellaneous):**

    - **Description:** Provides an indication of progress when an operation takes longer than a specified duration.
    - **Example:** An animated icon or message box with a visual indicator (e.g., progress bar) showing the status of processing.

These UI design patterns help designers and developers address common challenges, maintain consistency, and improve the overall usability of web applications. Each pattern includes details such as its purpose, components, and navigation elements, providing a guide for effective implementation.

You:

When we use the term language, the first thing that comes to mind is either a natu- ral language (e.g., English, Spanish, Chinese) or a programming language (e.g., C, Java). In both cases the language has a syntax and semantics that are used to com- municate ideas or procedural instructions in an effective manner. When the term language is used in the context of design patterns, it takes on a slightly different meaning. A pattern language encompasses a collection of patterns, each described using a standardized template (Section 12.1.3) and interrelated to show how these patterns collaborate to solve problems across an application domain.4 In a natural language, words are organized into sentences that impart meaning. The structure of sentences is described by the language's syntax. In a pattern lan- guage, design patterns are organized in a way that provides a "structured method of describing good design practices within a particular domain."5 uote: "Patterns are half- baked—meaning you always have to finish them yourself and adapt them to your own environment." Martin Fowler 4 Christopher Alexander originally proposed pattern languages for building architecture and urban planning. Today, pattern languages have been developed for everything from the social sciences to the software engineering process. 5 This

Wikipedia description can be found at http://en.wikipedia.org/wiki/Pattern_language. Design Pattern Template Pattern name—describes the essence of the pattern in a short but expressive name Problem—describes the problem that the pattern addresses Motivation—provides an example of the problem Context—describes the environment in which the problem resides including the application domain Forces—lists the system of forces that affect the manner in which the problem must be solved; includes a discussion of limitations and constraints that must be considered Solution—provides a detailed description of the solution proposed for the problem Intent—describes the pattern and what it does Collaborations—describes how other patterns contribute to the solution Consequences—describes the potential trade-offs that must be considered when the pattern is implemented and the consequences of using the pattern Implementation—identifies special issues that should be considered when implementing the pattern Known uses—provides examples of actual uses of the design pattern in real applications Related patterns—cross-references related design patterns pre75977_ch12.qxd 11/27/08 3:58 PM Page 353 In a way, a pattern language is analogous to a hypertext instruction manual for problem solving in a specific application domain. The problem domain under con- sideration is first described hierarchically, beginning with broad design problems as- sociated with the domain and then refining each of the broad problems into lower levels of abstraction. In a software context, broad design problems tend to be archi- tectural in nature and address the overall structure of the application and the data or content that serve it. Architectural problems are refined to lower levels of abstrac- tion, leading to design patterns that solve subproblems and collaborate with one another at the component (or class) level. Rather than a sequential list of patterns, a pattern language represents an interconnected collection in which the user can begin with a broad design problem and “burrow down” to uncover specific problems and their solutions. Dozens of pattern languages have been proposed for software design [Hil08]. In most cases, the design patterns that are part of pattern language are stored in a Web- accessible patterns repository (e.g., [Boo08], [Cha03], [HPR02]). The repository pro- vides an index of all design patterns and contains hypermedia links that enable the user to understand the collaborations between patterns. ANSWER THE QUESTION : Write a short note on pattern languages and repositories.

ChatGPT:

Pattern Languages and Repositories:

In the context of design patterns, a pattern language refers to a structured collection of design patterns, each described using a standardized template. Unlike natural languages or programming languages, a pattern language is a way of organizing design knowledge within a specific application domain. It provides a systematic and interconnected approach to problem-solving, where patterns collaborate to address design challenges at various levels of abstraction.

Components of a Design Pattern Template:

1. **Pattern name:** Describes the essence of the pattern in a short but expressive name.
2. **Problem:** Describes the problem that the pattern addresses.
3. **Motivation:** Provides an example illustrating the problem.
4. **Context:** Describes the environment in which the problem resides, including the application domain.
5. **Forces:** Lists the system of forces influencing the problem and discusses limitations and constraints.
6. **Solution:** Provides a detailed description of the proposed solution.
7. **Intent:** Describes the pattern's purpose and functionality.
8. **Collaborations:** Describes how other patterns contribute to the solution.
9. **Consequences:** Describes potential trade-offs and consequences of using the pattern.
10. **Implementation:** Identifies special considerations for implementing the pattern.
11. **Known uses:** Provides examples of real applications using the design pattern.

12. **Related patterns:** Cross-references other design patterns.

A pattern language, therefore, offers a hierarchical structure, starting with broad architectural problems and progressively refining them into more specific, lower-level design patterns that collaborate at the component or class level.

Pattern Repositories: Pattern repositories are online collections or databases that store design patterns, making them accessible to the design community. These repositories serve as valuable resources for designers and developers seeking solutions to common design problems. Examples of pattern repositories include Boostrap Patterns [Boo08], The Hillside Group's Pattern Repository [HPR02], and the Portland Pattern Repository [Cha03].

Key Aspects of Pattern Repositories:

1. **Indexing:** Patterns are indexed for easy search and retrieval.
2. **Hypermedia Links:** Repositories often include hypermedia links that allow users to navigate and understand collaborations between patterns.
3. **Accessibility:** Patterns are made accessible to a wide audience, facilitating knowledge sharing.
4. **Real-world Examples:** Repositories may include real-world examples of pattern implementations.
5. **Community Contributions:** They encourage contributions from the design community, fostering a collaborative approach to pattern development.

In summary, pattern languages and repositories play a crucial role in organizing, sharing, and evolving design knowledge within the software development community, promoting best practices and effective problem-solving strategies.

You:

Design Focus In earlier chapters I emphasized a design progression that begins by considering architecture, component-level issues, and user interface representations. At each step, the problems you consider and the solutions you propose begin at a high level of abstraction and slowly become more detailed and specific. Stated another way, design focus becomes “narrower” as you move further into design. The problems (and solutions) you will encounter when designing an information architecture for a WebApp are different from the problems (and solutions) that are encountered when performing interface design. Therefore, it should come as no surprise that patterns for WebApp design can be developed for different levels of design focus, so that you can address the unique problems (and related solutions) that are encountered at each level. WebApp patterns can be categorized using the following levels of design focus: • Information architecture patterns relate to the overall structure of the information space, and the ways in which users will interact with the information. • Navigation patterns define navigation link structures, such as hierarchies, rings, tours, and so on. • Interaction patterns contribute to the design of the user interface. Patterns in this category address how the interface informs the user of the conse- quences of a specific action, how a user expands content based on usage 368 PART TWO MODELING Your focus becomes “narrower” the further you move into design. pre75977_ch12.qxd 11/27/08 3:58 PM Page 368 context and user desires, how to best describe the destination that is implied by a link, how to inform the user about the status of an ongoing interaction, and interface-related issues. • Presentation patterns assist in the presentation of content as it is presented to the user via the interface. Patterns in this category address how to organize user interface control functions for better usability, how to show the relation- ship between an interface action and the content objects it affects, and how to establish effective content hierarchies. • Functional patterns define the workflows, behaviors, processing, commu- nication, and other algorithmic elements within a WebApp. In most cases, it would be fruitless to explore the collection of information architec- ture patterns when a problem in interaction design is encountered. You would ex- amine interaction patterns, because that is the design focus that is relevant to the work being performed. 12.6.2 Design Granularity When a problem involves “big picture” issues, you should attempt to develop solutions (and use relevant patterns) that focus on the big picture. Conversely, when the focus is very narrow (e.g., uniquely selecting one item from a small set of five or fewer items), the solution (and the corresponding pattern) is targeted quite nar- rowly. In terms of the level of granularity, patterns can be described at the follow- ing levels: • Architectural patterns. This level of abstraction will typically relate to patterns that define the overall structure of the WebApp, indicate the rela- tionships among different components or increments, and define the rules for specifying relationships among the elements (pages, packages, compo- nents, subsystems) of the architecture. • Design patterns. These address a specific element of the design such as an aggregation of components to solve some design problem, relationships among elements on a page, or the mechanisms for effecting component-to- component communication. An example might be the Broadsheet pattern for the layout of a WebApp home page. • Component patterns. This level of abstraction relates to individual small- scale elements of a WebApp. Examples include individual interaction elements (e.g., radio buttons), navigation items (e.g., how might you format links?) or functional elements (e.g., specific algorithms). It is also possible to define the relevance of different patterns to different classes of applications or domains. For example, a collection of patterns (at different levels of design focus and granularity) might be particularly relevant to e-business. ANSWER THE QUESTION : Describe design focus and design granularity in web application design patterns.

ChatGPT:

Design Focus and Design Granularity in Web Application Design Patterns:

In web application design, the approach to problem-solving involves different levels of focus and granularity. Design patterns are organized and categorized based on these aspects to address specific challenges at various stages of the design process.

Design Focus: Design focus refers to the level of abstraction and the specific aspect of the design process that is being considered. In the context of web application design patterns, different levels of design focus include:

1. **Information Architecture Patterns:**
    - Relate to the overall structure of the information space.
    - Address how users interact with information within the application.
2. **Navigation Patterns:**
    - Define navigation link structures such as hierarchies, rings, and tours.
    - Focus on guiding users through the application's content.
3. **Interaction Patterns:**
    - Contribute to the design of the user interface.
    - Address how the interface communicates consequences of actions, expands content based on user behavior, describes destination links, and informs users about ongoing interactions.
4. **Presentation Patterns:**
    - Assist in presenting content to users through the interface.
    - Address usability issues, organize user interface control functions, and establish effective content hierarchies.

5. **Functional Patterns:**

    - Define workflows, behaviors, processing, communication, and other algorithmic elements within the web application.

Each category of patterns aligns with a specific design focus, allowing designers to choose the most relevant patterns based on the nature of the problem at hand.

Design Granularity: Design granularity refers to the level of detail and specificity in the design patterns. In web application design, patterns can be described at different levels of granularity:

1. **Architectural Patterns:**

    - Address the overall structure of the web application.
    - Define relationships among different components, increments, and rules for specifying relationships.

2. **Design Patterns:**

    - Address specific elements of the design, such as aggregations of components, relationships among elements on a page, or mechanisms for component-to-component communication.
    - Examples include layout patterns for web application home pages.

3. **Component Patterns:**

    - Focus on individual small-scale elements of a web application.
    - Examples include interaction elements like radio buttons, navigation items, or specific functional elements.

Relevance to Different Classes of Applications or Domains: Patterns can also be relevant to specific classes of applications or domains. For instance, a collection of patterns at different levels of granularity and focus might be particularly suited for e-business applications.

In summary, considering design focus and granularity helps designers select the most appropriate design patterns based on the nature and scope of the design problem within a web application.

You:

Like many WebApp design actions, navigation design begins with a consideration of the user hierarchy and related use cases (Chapter 5) developed for each category of user (actor). Each actor may use the WebApp somewhat differently and therefore have different navigation requirements. In addition, the use cases developed for each actor will define a set of classes that encompass one or more content objects or WebApp functions. As each user interacts with the WebApp, she encounters a series of navigation semantic units (NSUs)—"a set of information and related naviga- tion structures that collaborate in the fulfillment of a subset of related user requirements" [Cac02]. An NSU is composed of a set of navigation elements called ways of navigating (WoN) [Gna99]. A WoN represents the best navigation pathway to achieve a naviga- tional goal for a specific type of user. Each WoN is organized as a set of navigational nodes (NN) that are connected by navigational links. In some cases, a navigational link may be another NSU. Therefore, the overall navigation structure for a WebApp may be organized as a hierarchy of NSUs. To illustrate the development of an NSU, consider the use case Select SafeHome Components: Use Case: Select SafeHome Components The WebApp will recommend product components (e.g., control panels, sensors, cam- eras) and other features (e.g., PC-based functionality implemented in software) for each room and exterior entrance. If I request alternatives, the WebApp will provide them, if they exist. I will be able to get descriptive and pricing information for each product component. The WebApp will create and display a bill-of-materials as I select various components. I'll be able to give the bill-of-materials a name and save it for future reference (see use case Save Configuration). The underlined items in the use-case description represent classes and content objects that will be incorporated into one or more NSUs that will enable a new customer to perform the scenario described in the Select SafeHome Components use case. Figure 13.9 depicts a partial semantic analysis of the navigation implied by the Select SafeHome Components use case. Using the terminology intro- duced earlier, the figure also represents a way of navigating (WoN) for the 388 PART TWO MODELING uote: “Just wait, Gretel, until the moon rises, and then we shall see the crumbs of bread which I have strewn about, they will show us our way home again.” Hansel and Gretel An NSU describes the navigation requirements for each use case. In essence, the NSU shows how an actor moves between content objects or WebApp functions. pre75977_ch13.qxd 11/27/08 5:47 PM Page 388 SafeHomeAssured.com WebApp. Important problem domain classes are shown along with selected content objects (in this case the package of content objects named CompDescription, an attribute of the ProductComponent class). These items are navigation nodes. Each of the arrows represents a navigation link10 and is labeled with the user-initiated action that causes the link to occur. You can create an NSU for each use case associated with each user role. For example, a new customer for SafeHomeAssured.com may have three different use cases, all resulting in access to different information and WebApp functions. An NSU is created for each goal. During the initial stages of navigation design, the WebApp content architecture is assessed to determine one or more WoN for each use case. As noted earlier, a WoN identifies navigation nodes (e.g., content) and then links that enable navigation between them. The WoN are then organized into NSUs. ANSWER THE QUESTION : Explain navigation semantics with an example of web application.

ChatGPT:

Navigation Semantics in Web Application Design:

Navigation semantics in web application design involve the consideration of user hierarchies, use cases, and the development of navigation structures that fulfill user requirements. One way to represent this is through Navigation Semantic Units (NSUs), which are sets of information and related navigation structures collaborating to fulfill a subset of user requirements.

Components of Navigation Semantic Units (NSUs):

1. **Ways of Navigating (WoN):** These are sets of navigation elements that represent the best navigation pathway to achieve a specific navigational goal for a particular type of user.

2. **Navigational Nodes (NN):** These are the navigation elements or content objects that form the building blocks of a WoN. They represent specific pieces of information or functionalities.

3. **Navigational Links:** These links connect navigational nodes and form the pathways along which users navigate through the web application.

Example of Navigation Semantic Unit (NSU):

Consider the use case "Select SafeHome Components" for a web application related to home security. The use case involves recommending product components (e.g., control panels, sensors, cameras) for different rooms and entrances. Users should be able to view alternatives, access descriptive and pricing information, create a bill-of-materials, and save configurations for future reference.

Partial Semantic Analysis:

A partial semantic analysis of the navigation implied by the "Select SafeHome Components" use case involves identifying important problem domain classes, content objects, and the relationships between them. The analysis is represented as a WoN for the SafeHomeAssured.com WebApp. Each navigational link is labeled with the user-initiated action that triggers the link.

Key Elements in the Analysis:

1. **Problem Domain Classes:** These are classes representing entities in the application's problem domain, such as ProductComponent.

2. **Content Objects:** For example, the CompDescription package, which is an attribute of the ProductComponent class.

3. **Navigational Links:** Arrows representing links between content objects based on user-initiated actions.

Outcome:

This analysis helps create an NSU for the "Select SafeHome Components" use case, outlining the navigation nodes and links that users will encounter while interacting with the web application. The NSU captures how an actor (user) moves between content objects or web application functions to achieve specific goals.

In summary, navigation semantics involve understanding user goals, developing ways of navigating, and organizing them into NSUs that guide users through the web application based on their interactions and use cases.

You:

The best designers in any field have an uncanny ability to see patterns that charac- terize a problem and corresponding patterns that can be combined to create a solu- tion. The software developers at Microsoft [Mic04] discuss this when they write: While pattern-based design is relatively new in the field of software development, indus- trial technology has used pattern-based design for decades, perhaps even centuries. Catalogs of mechanisms and standard configurations provide design elements that are used to engineer automobiles, aircraft, machine tools, and robots. Applying pattern- based design to software development promises the same benefits to software as it does to industrial technology: predictability, risk mitigation, and increased productivity. Throughout the design process, you should look for every opportunity to apply existing design patterns (when they meet the needs of the design) rather than creat- ing new ones. 12.2.1 Pattern-Based Design in Context Pattern-based design is not used in a vacuum. The concepts and techniques dis- cussed for architectural, component-level, and user interface design (Chapters 9 through 11) are all used in conjunction with a pattern-based approach. In Chapter 8, I noted that a set of quality guidelines and attributes serve as the basis for all software design decisions. The decisions themselves are influenced by a set of fundamental design concepts (e.g., separation of concerns, stepwise refine- ment, functional independence) that are achieved using heuristics that have evolved over many decades, and best practices (e.g., techniques, modeling notation) that 354 PART TWO MODELING WebRef For a listing of useful patterns languages see c2.com/ppr/ titles.html. Additional information can be obtained at hillside.net/ patterns/. If you can't find a pattern language that addresses your problem domain, look for analogies in another set of patterns. pre75977_ch12.qxd 11/27/08 3:58 PM Page 354 have been proposed to make design easier to perform and more effective as a basis for construction. The role of pattern-based design in all of this is illustrated in Figure 12.1. A soft- ware designer begins with a requirements model (either explicit or implied) that presents an abstract representation of the system. The requirements model describes the problem set, establishes the context, and identifies the system of forces that hold sway. It may imply the design in an abstract manner, but the requirements model does little to represent the design explicitly. As you begin your work as a designer, it’s always important to keep quality attributes in mind. These attributes (e.g., a design must implement all explicit require- ments addressed in the requirements model) establish a way to assess software qual- ity but do little to help you actually achieve it. The design you create should exhibit the fundamental design concepts discussed in Chapter 8. Therefore, you should ap- ply proven techniques for translating the abstractions contained in the requirements model into a more concrete form that is the software design. To accomplish this, you’ll use the methods and modeling tools available for architectural, component- level, and interface design. But only when you’re faced with a problem, context, and system of forces that have not been solved before. If a solution already exists, use it! And that means applying a pattern-based design approach. CHAPTER 12 PATTERN-BASED DESIGN 355 Design begins Consider design concepts Extract problem, context forces Requirements model Consider design quality attributes Begin pattern-based design tasks Apply other design methods and notation yes no Addressed by pattern? Design model FIGURE 12.1 Pattern-based design in context pre75977_ch12.qxd 11/27/08 3:58 PM Page 355 12.2.2 Thinking in Patterns In an excellent book on pattern-based design, Shalloway and Trott [Sha05] comment on a “new way of thinking” when one uses patterns as part of the design activity: I had to open my mind to a new way of thinking. And when I did so, I heard [Christopher] Alexander say that “good software design cannot be achieved simply by adding together performing parts.” Good design begins by considering context—the big picture. As context is evaluated, you extract a hierarchy of problems that must be solved. Some of these problems will be global in nature, while others will address specific features and functions of the software. All will be affected by a system of forces that will influence the nature of the solution that is proposed. Shalloway and Trott [Sha05] suggest the following approach6 that enables a designer to think in patterns: 1. Be sure you understand the big picture—the context in which the software to be built resides. The requirements model should communicate this to you. 2. Examining the big picture, extract the patterns that are present at that level of abstraction. 3. Begin your design with “big picture” patterns that establish a context or skeleton for further design work. 4. “Work inward from the context” [Sha05] looking for patterns at lower levels of abstraction that contribute to the design solution. 5. Repeat steps 1 to 4 until the complete design is fleshed out. 6. Refine the design by adapting each pattern to the specifics of the software you’re trying to build. It’s important to note that patterns are not independent entities. Design patterns that are present at a high level of abstraction will invariably influence the manner in which other patterns are applied at lower levels of abstraction. In addition, patterns often collaborate with one another. The implication —when you select an architec- tural pattern, it may very well influence the component-level design patterns you choose. Likewise, when you select a specific interface design pattern, you are some- times forced to use other patterns that collaborate with it. To illustrate, consider the SafeHomeAssured.com WebApp. If you consider the big picture, the WebApp must address a number of fundamental problems such as: • How to provide information about SafeHome products and services • How to sell SafeHome products and services to customers • How to establish Internet-based monitoring and control of an installed security system 356 PART TWO MODELING Pattern- based design looks interesting for the problem I have to solve. How do I get started? ? 6 Based on the work of Christopher Alexander [Ale79]. pre75977_ch12.qxd 11/27/08 3:58 PM Page 356 Each of these fundamental problems can be further refined into a set of subprob- lems. For example How to sell via the Internet implies an E-commerce pattern that itself implies a large number of patterns at lower levels of abstraction. The E-commerce pattern (likely, an architectural pattern) implies mechanisms for setting up a customer account, displaying the products to be sold, selecting prod- ucts for purchase, and so forth. Hence, if you think in patterns, it is important to determine whether a pattern for setting up an account exists. If SetUpAccount is available as a viable pattern for the problem context, it may collaborate with other patterns such as BuildInputForm, ManageFormsInput, and Validate- FormsEntry. Each of these patterns delineates problems to be solved and solu- tions that may be applied. 12.2.3 Design Tasks The following design tasks are applied when a pattern-based design philosophy is used: 1. Examine the requirements model and develop a problem hierarchy. Describe each problem and subproblem by isolating the problem, the context, and the system of forces that apply. Work from broad problems (high level of abstraction) to smaller subproblems (at lower levels of abstraction). 2. Determine if a reliable pattern language has been developed for the problem domain. As I noted in Section 12.1.4, a pattern language addresses problems associated with a specific application domain. The SafeHome software team would look for a pattern language developed specifically for home security products. If that level of pattern language specificity could not be found, the team would partition the SafeHome software problem into a series of generic problem domains (e.g., digital device monitoring problems, user interface problems, digital video management problems) and search for appropriate pattern languages. 3. Beginning with a broad problem, determine whether one or more architectural patterns is available for it. If an architectural pattern is available, be certain to examine all collaborating patterns. If the pattern is appropriate, adapt the design solution proposed and build a design model element that adequately represents it. As I noted in Section 12.2.2, a broad problem for the SafeHomeAssured.com WebApp is addressed with an E-commerce pattern. This pattern will suggest a specific architecture for addressing e-commerce requirements. 4. Using the collaborations provided for the architectural pattern, examine subsystem or component-level problems and search for appropriate patterns to address them. It may be necessary to search through other pattern repositories as well as the list of patterns that corre- sponds to the architectural solution. If an appropriate pattern is found, adapt CHAPTER 12 PATTERN-BASED DESIGN 357 What are the tasks required to create a pattern-based design? ? pre75977_ch12.qxd 11/27/08 3:58 PM Page 357 the design solution proposed and build a design model element that adequately represents it. Be certain to apply step 7. 5. Repeat steps 2 through 5 until all broad problems have been addressed. The implication is to begin with the big picture and elaborate to solve problems at increasingly more detailed levels. 6. If user interface design problems have been isolated (this is almost always the case), search the many user interface design pattern repositories for appropriate patterns. Proceed in a manner similar to steps 3, 4, and 5. 7. Regardless of its level of abstraction, if a pattern language and/or patterns repository or individual pattern shows promise, compare the problem to be solved against the existing pattern(s) presented. Be certain to examine context and forces to ensure that the pattern does, in fact, provide a solution that is amenable to the problem. 8. Be certain to refine the design as it is derived from patterns using design quality criteria as a guide. Although this design approach is top-down, real-life design solutions are sometimes more complex. Gillis [Gil06] comments on this when he writes: Design patterns in software engineering are meant to be used in a deductive, rationalis- tic fashion. So you have this general problem or requirement, X, design pattern Y solves X, therefore use Y. Now, when I reflect on my own process—and I've got reason to believe that I'm not alone here—I find that it's more organic than that, more inductive than deductive, more bottom-up than top-down. Obviously, there's a balance to be achieved. When a project is in the initial bootstrap phase and I'm trying to make the jump from abstract requirements to a concrete design solution, I'll often perform a sort of breadth-first search . . . I've found design patterns to be helpful, allowing me to quickly frame up the design problem in concrete terms. In addition, the pattern-based approach must be used in conjunction with other soft- ware design concepts and techniques. 12.2.4 Building a Pattern-Organizing Table As pattern-based design proceeds, you may encounter trouble organizing and categorizing candidate patterns from multiple pattern languages and repositories. To help organize your evaluation of candidate patterns, Microsoft [Mic04] suggests the creation of a pattern-organizing table that takes the general form shown in Figure 12.2. A pattern-organizing table can be implemented as a spreadsheet model using the form shown in the figure. An abbreviated list of problem statements, organized by data/content, architecture, component-level, and user interface issues, is presented in the left-hand (shaded) column. Four pattern types—database, application, 358 PART TWO MODELING Entries in the table can be supplemented with an indication of the relative applicability of the pattern. pre75977_ch12.qxd 11/27/08 3:58 PM Page 358 implementation, and infrastructure—are listed across the top row. The names of candidate patterns are noted in the cells of the table. To provide entries for the organizing table, you'll search through pattern languages and repositories for patterns that address a particular problem statement. When one or more candidate patterns is found, it is entered in the row correspon- ding to the problem statement and the column that corresponds to the pattern type. The name of the pattern is entered as a hyperlink to the URL of the Web address that contains a complete description of the pattern. 12.2.5 Common Design Mistakes Pattern-based design can make you a better software designer, but it is not a panacea. Like all design methods, you must begin with first principles, emphasizing software quality fundamentals and ensuring that the design does, in fact, address the needs expressed by the requirements model. A number of common mistakes occur when pattern-based design is used. In some cases, not enough time has been spent to understand the underlying problem and its context and forces, and as a consequence, you select a pattern that looks right but is inappropriate for the solution required. Once the wrong pattern is selected, you refuse to see your error and force-fit the pattern. In other cases, the problem has forces that are not considered by the pattern you've chosen, resulting in a poor or CHAPTER 12 PATTERN-BASED

DESIGN 359 Problem statement ... Problem statement ... Problem statement ... User interface Problem statement ... Problem statement ... Problem statement ... Component-level Problem statement ... Problem statement ... Problem statement ... Architecture Problem statement ... Problem statement ... Problem statement ... PatternName(s) Database Data/Content Application Implementation Infrastructure FIGURE 12.2 A pattern- organizing table Source: Adapted from [Mic04]. Don't force a pattern, even if it addresses the problem at hand. If the context and forces are wrong, look for another pattern. pre75977_ch12.qxd 11/27/08 3:58 PM Page 359 erroneous fit. Sometimes a pattern is applied too literally and the required adapta- tions for your problem space are not implemented. Can these mistakes be avoided? In most cases the answer is "yes." Every good de- signer looks for a second opinion and welcomes review of her work. The review tech- niques discussed in Chapter 15 can help to ensure that the pattern-based design you've developed will result in a high-quality solution for the software problem to be solved. ANSWER THE QUESTION : Explain pattern based software design with an example.IN SHORT

ChatGPT:

Pattern-Based Software Design:

Pattern-based design involves using design patterns—reusable solutions to common problems—as building blocks in the software design process. Design patterns provide a structured and proven approach to solving specific design issues, promoting predictability, risk mitigation, and increased productivity. Here's an overview with an example:

Example: SafeHomeAssured.com WebApp

1. **Context and Problem Hierarchy:**
    - Understand the big picture or context of the software.
    - Extract patterns at a high level of abstraction from the requirements model.
    - Identify a hierarchy of problems and subproblems.
2. **Architectural Patterns:**
    - Search for reliable pattern languages or repositories.
    - Apply architectural patterns to address broad problems.
    - Collaborate with other patterns to form a cohesive solution.
3. **Component and User Interface Patterns:**
    - Search for patterns at lower levels to address subsystem or component-level problems.
    - Utilize user interface design patterns for interface-related issues.
4. **Thinking in Patterns:**
    - Open the mind to a new way of thinking.
    - Consider patterns at different levels of abstraction, working from the big picture to more detailed levels.
    - Collaborate patterns to form a comprehensive solution.
5. **Design Tasks:**
    - Develop a problem hierarchy.

- Determine pattern languages or repositories relevant to the problem domain.
- Apply architectural patterns, then move to subsystems and components.
- Search for user interface patterns if needed.
- Compare the problem against existing patterns, ensuring relevance.
- Refine the design using quality criteria.

6. **Pattern-Organizing Table:**

   - Create a table to organize and categorize candidate patterns.
   - Classify patterns by types such as database, application, implementation, and infrastructure.
   - Use hyperlinks to pattern descriptions.

7. **Common Design Mistakes:**

   - Avoid selecting patterns without understanding the problem context.
   - Don't force-fit a pattern; look for alternatives if the chosen pattern doesn't fit.
   - Adapt patterns to the specific problem space.
   - Seek second opinions and reviews to ensure a high-quality solution.

In summary, pattern-based design is an approach where existing design patterns are applied to solve software design problems, ensuring a structured and proven way of addressing common challenges.

You:

The best designers in any field have an uncanny ability to see patterns that charac- terize a problem and corresponding patterns that can be combined to create a solu- tion. The software developers at Microsoft [Mic04] discuss this when they write: While pattern-based design is relatively new in the field of software development, indus- trial technology has used pattern-based design for decades, perhaps even centuries. Catalogs of mechanisms and standard configurations provide design elements that are used to engineer automobiles, aircraft, machine tools, and robots. Applying pattern- based design to software development promises the same benefits to software as it does to industrial technology: predictability, risk mitigation, and increased productivity. Throughout the design process, you should look for every opportunity to apply existing design patterns (when they meet the needs of the design) rather than creat- ing new ones. 12.2.1 Pattern-Based Design in Context Pattern-based design is not used in a vacuum. The concepts and techniques dis- cussed for architectural, component-level, and user interface design (Chapters 9 through 11) are all used in conjunction with a pattern-based approach. In Chapter 8, I noted that a set of quality guidelines and attributes serve as the basis for all software design decisions. The decisions themselves are influenced by a set of fundamental design concepts (e.g., separation of concerns, stepwise refine- ment, functional independence) that are achieved using heuristics that have evolved over many decades, and best practices (e.g., techniques, modeling notation) that 354 PART TWO MODELING WebRef For a listing of useful patterns languages see c2.com/ppr/ titles.html. Additional information can be obtained at hillside.net/ patterns/. If you can’t find a pattern language that addresses your problem domain, look for analogies in another set of patterns. pre75977_ch12.qxd 11/27/08 3:58 PM Page 354 have been proposed to make design easier to perform and more effective as a basis for construction. The role of pattern-based design in all of this is illustrated in Figure 12.1. A soft- ware designer begins with a requirements model (either explicit or implied) that presents an abstract representation of the system. The requirements model describes the problem set, establishes the context, and identifies the system of forces that hold sway. It may imply the design in an abstract manner, but the requirements model does little to represent the design explicitly. As you begin your work as a designer, it’s always important to keep quality attributes in mind. These attributes (e.g., a design must implement all explicit require- ments addressed in the requirements model) establish a way to assess software qual- ity but do little to help you actually achieve it. The design you create should exhibit the fundamental design concepts discussed in Chapter 8. Therefore, you should ap- ply proven techniques for translating the abstractions contained in the requirements model into a more concrete form that is the software design. To accomplish this, you'll use the methods and modeling tools available for architectural, component- level, and interface design. But only when you're faced with a problem, context, and system of forces that have not been solved before. If a solution already exists, use it! And that means applying a pattern-based design approach. ANSWER THE QUESTION : Explain pattern based software design with an example.In short

----

ChatGPT:

Pattern-Based Software Design Overview:

Pattern-based software design involves utilizing design patterns—established solutions to common problems—as integral components in the software design process. It is a practice employed to enhance predictability, manage risks, and improve productivity. Here's a concise explanation along with an example:

Key Points:

1. **Industrial Roots:**
    - Design patterns have been integral to various industries like automobiles, aircraft, and machinery for centuries.
    - These patterns provide standard configurations and mechanisms for engineering solutions.
2. **Benefits in Software Development:**
    - Pattern-based design promises similar advantages in software, including predictability, risk mitigation, and increased productivity.
    - Microsoft emphasizes the application of existing design patterns over creating new ones.
3. **Integration with Software Design Concepts:**
    - Pattern-based design doesn't operate in isolation; it complements architectural, component-level, and user interface design concepts.
4. **Foundations in Quality Guidelines:**
    - Quality guidelines and attributes, established in Chapter 8, serve as the foundation for software design decisions.
    - Fundamental design concepts like separation of concerns and stepwise refinement play a crucial role.
5. **Role in Design Process:**
    - Illustrated in Figure 12.1, pattern-based design starts with a requirements model, offering an abstract representation of the system.
    - The requirements model outlines the problem, context, and system forces, but it doesn't explicitly represent the design.
    - Quality attributes guide the design, ensuring that it aligns with established principles.
6. **Application of Patterns:**

- Designers apply pattern-based design when faced with new problems, contexts, and forces that lack pre-existing solutions.
- Proven techniques are employed for translating abstract requirements into concrete software design elements.
- The overarching principle is to use existing patterns when available, avoiding unnecessary creation of new solutions.

Example:

- Imagine a scenario in web application development where a recurring problem is user authentication. Instead of devising a new method, a designer applies an established authentication design pattern. This pattern specifies how to securely verify user identities, providing a proven and reliable solution.

In essence, pattern-based software design involves leveraging established solutions to common problems, promoting efficiency and effectiveness in the design process.

You:

What is WebApp design? This simple question is more difficult to answer than one might believe. In our book [Pre08] on Web engineering, David Lowe and I discuss this when we write: 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 pyra- mid represents a design action that is described in the sections that follow. 13.4 WEBAPP INTERFACE DESIGN When a user interacts with a computer-based system, a set of fundamental principles and overriding design guidelines apply. These have been discussed in 378 PART TWO MODELING uote: “If a site is perfectly usable but it lacks an elegant and appropriate design style, it will fail.” Curt Cloninger pre75977_ch13.qxd 11/27/08 5:47 PM Page 378 Chapter 11.4 Although WebApps present a few special user interface design chal- lenges, the basic principles and guidelines are applicable. One of the challenges of interface design for WebApps is the indeterminate nature of the user’s entry point. That is, the user may enter the WebApp at a “home” loca- tion (e.g., the home page) or may be linked into some lower level of the WebApp architecture. In some cases, the WebApp can be designed in a way that reroutes the user to a home location, but if this is undesirable, the WebApp design must provide interface navigation features that accompany all content objects and are available regardless of how the user enters the system. 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 (Section 13.5) to establish a coherent “look.” This encompasses many characteristics, but must emphasize the layout and form of nav- igation 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 interac- tion mechanisms: • Navigation menus—keyword menus (organized vertically or horizontally) that list key content and/or functionality. These menus may be implemented so CHAPTER 13 WEBAPP DESIGN 379 Interface design Aesthetic design Content design Navigation design Architecture design Component design user technology FIGURE 13.2 A design pyramid for WebApps 4 Section 11.5 is dedicated to WebApp interface design. If you have not already done so, read it at this time.

5 In this context, a metaphor is a representation (drawn from the user's real-world experience) that can be modeled within the context of the interface. A simple example might be a slider switch that is used to control the auditory volume of an .mpg file. pre75977_ch13.qxd 11/27/08 5:47 PM Page 379 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. It is important to note that one or more of these control mechanisms should be provided at every level of the content hierarchy. ANSWER THE QUESTION : Draw and explain design pyramid for web applications.

ChatGPT:

I'm unable to draw images directly in this text-based format, but I can describe the design pyramid for web applications based on the information provided.

Design Pyramid for Web Applications:

The design pyramid for web applications is depicted as a multi-level pyramid, with each level representing a specific design action. Here are the levels of the pyramid along with their descriptions:

1. **WebApp Interface Design:**
    - At the top of the pyramid is WebApp Interface Design, emphasizing the importance of designing a user-friendly and consistent interface.
    - Subsections of this level include Aesthetic Design, Content Design, Navigation Design, Architecture Design, and Component Design.
2. **Aesthetic Design:**
    - Aesthetic design is the foundation, focusing on the visual appeal and coherence of the interface.
    - It involves creating a consistent and visually pleasing appearance for the web application.
3. **Content Design:**
    - Content design deals with structuring and presenting information on the interface.
    - This level ensures that the content is organized, relevant, and effectively communicated to the users.
4. **Navigation Design:**
    - Navigation design addresses the challenges of user interaction within the web application.
    - It considers how users can move through the application, especially when entering from different points.
5. **Architecture Design:**
    - Architecture design focuses on the overall structure and organization of the web application.
    - It deals with how different components and modules are arranged to achieve optimal functionality.

6. **Component Design:**
    - At the base of the pyramid is component design, which involves designing individual elements or modules of the web application.
    - This level ensures that each component is well-designed, contributing to the overall effectiveness of the application.

Explanation:

- The pyramid illustrates a hierarchical approach to web application design, with higher levels building upon the foundation provided by lower levels.
- Each level represents a specific aspect of design, and expertise from various specialists (Web engineers, graphic designers, content developers, etc.) may be required at different stages.
- The pyramid emphasizes that quality is embedded in the design phase, and a diverse set of skills is essential for creating an effective model before generating content and code.

This structured approach ensures that web applications are not only visually appealing but also functional, organized, and user-friendly.

You:

A number of design methods for Web applications have been proposed over the past decade. To date, no single method has achieved dominance.11 In this section I pres- ent a brief overview of one of the most widely discussed WebApp design methods— OOHDM. 390 PART TWO MODELING The site map should be accessible from every page. The map itself should be organized so that the structure of WebApp information is readily apparent. 11 In fact, relatively few Web developers use a specific method when designing a WebApp. Hopefully, this ad hoc approach to design will change as time passes. pre75977_ch13.qxd 11/27/08 5:47 PM Page 390 Daniel Schwabe and his colleagues [Sch95, Sch98b] originally proposed the Object-Oriented Hypermedia Design Method (OOHDM), which is composed of four different design activities: conceptual design, navigational design, abstract inter- face design, and implementation. A summary of these design activities is shown in Figure 13.10 and discussed briefly in the sections that follow. 13.10.1 Conceptual Design for OOHDM OOHDM conceptual design creates a representation of the subsystems, classes, and re- lationships that define the application domain for the WebApp. UML may be used12 to create appropriate class diagrams, aggregations, and composite class representations, collaboration diagrams, and other information that describes the application domain. As a simple example of OOHDM conceptual design, consider the SafeHomeAssured .com e-commerce application. A partial “conceptual schema” is shown in Figure 13.11. The class diagrams, aggregations, and related information developed as part of WebApp analysis are reused during conceptual design to represent relationships between classes. 13.10.2 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 CHAPTER 13 WEBAPP DESIGN 391 Work products Design mechanisms Design concerns Modeling semantics of the application domain Classes, subsystems, relationships, attributes Classification, composition, aggregation, generalization specialization Conceptual design Navigational design Abstract interface design Implementation Nodes links, access structures, navigational contexts, navigational transformations Mapping between conceptual and navigation objects Takes into account user profile and task. Emphasis on cognitive aspects. Abstract interface objects, responses to external events, transformations Mapping between navigation and perceptible objects Modeling perceptible objects, implementing chosen metaphors. Describe interface for navigational objects. Executable WebApp Resource provided by target environment Correctness; application performance; completeness FIGURE 13.10 Summary of the OOHDM method. Source: Adapted from [Sch95]. 12 OOHDM does not prescribe a specific notation; however, the use of UML is common when this method is applied. pre75977_ch13.qxd 11/27/08 5:47 PM Page 391 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 nav- igational design may be used as the design is developed. OOHDM uses a predefined set of navigation classes—nodes, links, anchors, and access structures [Sch98b]. Access structures are more elaborate and include mechanisms such as a WebApp index, a site map, or a guided tour. Once navigation classes are defined, OOHDM "structures the navigation space by grouping navigation objects into sets called contexts" [Sch98b]. A context includes a description of the local navigation structure, restriction imposed on the access of content objects, and methods (operations) required to effect access of content objects. A context template (analogous to CRC cards discussed in Chapter 6) is de- veloped and may be used to track the navigation requirements of each category of user through the various contexts defined in OOHDM. Doing this, specific navigation paths (what we called WoN in Section 13.8.1) emerge. 13.10.3 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. 392 PART TWO MODELING ProductComponent partNumber partName partType description price createNewItem( ) getDescription( ) getTechSpec BillOfMaterials identifier BoMList numberItems priceTotal addEntry( ) deleteEntry( ) editEntry( ) name( ) computePrice( ) BoMItem quantity partNumber partName partType price addtoList( ) deletefromList( ) getNextListEntry( ) Order orderNumber customerInfo billOfMaterials shippingInfo billingInfo Room roomName dimensions exteriorWindows exteriorDoors Sensor Camera Control Panel SoftFeature customer continues component selection customer requests purchase component recommendation requested customer selects component FIGURE 13.11 Partial conceptual schema for SafeHomeAssured.com pre75977_ch13.qxd 11/27/08 5:47 PM Page 392 The ADV model defines a "static layout" [Sch98b] 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 (similar to the UML state diagram) that indicates how external events "trigger navigation and which interface transformations occur when the user inter- acts with the application" [Sch01a]. The OOHDM implementation activity represents a design iteration that is spe- cific to the environment in which the WebApp will operate. Classes, navigation, and the interface are each characterized in a manner that can be constructed for the client-server environment, operating systems, support software, program- ming languages, and other environmental characteristics that are relevant to the problem. ANSWER THE QUESTION : Discuss the concept of Object Oriented Hypermedia Design Method (OOHDM).

ChatGPT:

The Object-Oriented Hypermedia Design Method (OOHDM) is a design methodology for web applications that was initially proposed by Daniel Schwabe and his colleagues. It consists of four main design activities: conceptual design, navigational design, abstract interface design, and implementation. Here is a brief overview of each of these activities:

1. **Conceptual Design:**
    - **Objective:** The goal of conceptual design in OOHDM is to create a representation of the subsystems, classes, and relationships that define the application domain for the web application.
    - **Method:** UML (Unified Modeling Language) is commonly used to create class diagrams, aggregations, and other representations that describe the application domain.
    - **Example:** For instance, in the case of an e-commerce application like SafeHomeAssured.com, conceptual design involves creating class diagrams that

represent relationships between classes such as products, orders, customers, etc.

2. **Navigational Design:**
    - **Objective:** Navigational design identifies a set of navigational classes or nodes derived from the classes defined in conceptual design. It focuses on creating navigation structures and defining how users move through the application.
    - **Method:** UML may be used for creating use cases, state charts, and sequence diagrams. OOHDM uses predefined navigation classes like nodes, links, anchors, and access structures.
    - **Example:** In the e-commerce application, navigational design involves defining how users navigate through product categories, make selections, and proceed to checkout.

3. **Abstract Interface Design:**
    - **Objective:** Abstract interface design specifies the interface objects that users interact with during their interactions with the web application. It involves creating an abstract data view (ADV) that represents the relationship between interface objects and navigation objects.
    - **Method:** ADV model defines a static layout representing the interface metaphor and includes behavioral components indicating how external events trigger navigation and interface transformations.
    - **Example:** For the e-commerce application, abstract interface design defines how the user interacts with buttons, menus, and icons to browse products and make purchases.

4. **Implementation:**
    - **Objective:** Implementation is the design iteration specific to the environment in which the web application operates. It characterizes classes, navigation, and interfaces in a manner that can be constructed for the target environment, including client-server architecture, operating systems, programming languages, etc.
    - **Method:** This involves adapting the design to the specific technical and environmental characteristics relevant to the implementation.
    - **Example:** In the e-commerce application, the implementation activity ensures that the design is compatible with the chosen server environment, databases, and programming languages.

Overall, OOHDM provides a structured approach to web application design, considering conceptual, navigational, and interface aspects, and tailoring the design to the implementation environment.

You:

Design Tasks The following design tasks are applied when a pattern-based design philosophy is used: 1. Examine the requirements model and develop a problem hierarchy. Describe each problem and subproblem by isolating the problem, the context, and the system of forces that apply. Work from broad problems (high level of abstraction) to smaller subproblems (at lower levels of abstraction). 2. Determine if a reliable pattern language has been developed for the problem domain. As I noted in Section 12.1.4, a pattern language addresses problems associated with a specific application domain. The SafeHome software team would look for a pattern language developed specifically for home security products. If that level of pattern language specificity could not be found, the team would partition the SafeHome software problem into a series of generic problem domains (e.g., digital device monitoring problems, user interface problems, digital video management problems) and search for appropriate pattern languages. 3. Beginning with a broad problem, determine whether one or more architectural patterns is available for it. If an architectural pattern is available, be certain to examine all collaborating patterns. If the pattern is appropriate, adapt the design solution proposed and build a design model element that adequately represents it. As I noted in Section 12.2.2, a broad problem for the SafeHomeAssured.com WebApp is addressed with an E-commerce pattern. This pattern will suggest a specific architecture for addressing e-commerce requirements. 4. Using the collaborations provided for the architectural pattern, examine subsystem or component-level problems and search for appropriate patterns to address them. It may be necessary to search through other pattern repositories as well as the list of patterns that corre- sponds to the architectural solution. If an appropriate pattern is found, adapt CHAPTER 12 PATTERN-BASED DESIGN 357 What are the tasks required to create a pattern-based design? ? pre75977_ch12.qxd 11/27/08 3:58 PM Page 357 the design solution proposed and build a design model element that adequately represents it. Be certain to apply step 7. 5. Repeat steps 2 through 5 until all broad problems have been addressed. The implication is to begin with the big picture and elaborate to solve problems at increasingly more detailed levels. 6. If user interface design problems have been isolated (this is almost always the case), search the many user interface design pattern repositories for appropriate patterns. Proceed in a manner similar to steps 3, 4, and 5. 7. Regardless of its level of abstraction, if a pattern language and/or patterns repository or individual pattern shows promise, compare the problem to be solved against the existing pattern(s) presented. Be certain to examine context and forces to ensure that the pattern does, in fact, provide a solution that is amenable to the problem. 8. Be certain to refine the design as it is derived from patterns using design quality criteria as a guide. Although this design approach is top-down, real-life design solutions are sometimes more complex. Gillis [Gil06] comments on this when he writes: Design patterns in software engineering are meant to be used in a deductive, rationalis- tic fashion. So you have this general problem or requirement, X, design pattern Y solves X, therefore use Y. Now, when I reflect on my own process—and I've got reason to believe that I'm not alone here—I find that it's more organic than that, more inductive than deductive, more bottom-up than top-down. Obviously, there's a balance to be achieved. When a project is in the initial bootstrap phase and I'm trying to make the jump from abstract requirements to a concrete design solution, I'll often perform a sort of breadth-first search . . . I've found design patterns to be helpful, allowing me to quickly frame up the design problem in concrete terms. In addition, the pattern-based approach must be used in conjunction with other soft- ware design concepts and techniques. ANSWER THE QUESTION : Explain various design tasks in pattern based software design philosophy.

ChatGPT:

In pattern-based software design philosophy, various design tasks are applied to leverage design patterns effectively. Here are the key tasks involved in creating a pattern-based design:

1. **Examine Requirements Model and Develop Problem Hierarchy:**

    - **Description:** Start by examining the requirements model and developing a problem hierarchy. Identify and isolate problems, contexts, and systems of forces. Begin with broad problems at a high level of abstraction and proceed to smaller subproblems at lower levels.

2. **Determine if a Reliable Pattern Language Exists:**

    - **Description:** Check if a reliable pattern language exists for the specific problem domain. If available, use it to address problems associated with the application domain. If not, partition the problem into generic domains and search for appropriate pattern languages.

3. **Identify Architectural Patterns:**

- **Description:** Starting with a broad problem, determine if one or more architectural patterns are available. Examine collaborating patterns and, if appropriate, adapt the design solution proposed by the pattern. For example, an e-commerce pattern might suggest a specific architecture for addressing e-commerce requirements.

4. **Examine Subsystem or Component-Level Problems:**

   - **Description:** Using collaborations provided for the architectural pattern, examine subsystem or component-level problems. Search for appropriate patterns to address them, including patterns from other repositories. Adapt the design solution proposed by the pattern.

5. **Repeat Steps for All Broad Problems:**

   - **Description:** Iterate through steps 2 to 5 until all broad problems have been addressed. Start with the big picture and elaborate to solve problems at increasingly detailed levels.

6. **Search for User Interface Design Patterns:**

   - **Description:** If user interface design problems are isolated, search user interface design pattern repositories for appropriate patterns. Proceed similarly to steps 3, 4, and 5 for architectural patterns.

7. **Compare Against Existing Patterns:**

   - **Description:** Regardless of the level of abstraction, if a pattern language or repository shows promise, compare the problem to be solved against existing patterns. Ensure that the pattern provides a solution amenable to the problem. Examine context and forces.

8. **Refine the Design and Apply Quality Criteria:**

   - **Description:** Refine the design as derived from patterns using design quality criteria as a guide. Ensure that the design solution aligns with established quality guidelines.

It's important to note that while the design approach is generally top-down, real-life design solutions may involve a more organic and bottom-up process. The pattern-based approach should be balanced with other software design concepts and techniques. The use of design patterns is meant to be rationalistic but should also be flexible enough to accommodate the complexity of real-world design scenarios.
