




# Ch.16 Pattern-Based Design





## 16.1 Design Patterns



*I wonder if anyone has  
**developed a solution** to for a  
**design problem** ?*

- **Design patterns** are a 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**.



**Please read Christopher Alexander 's  
description in 1977**



## 16.1 Design Patterns

- Basic Concepts

“a three-part rule which expresses a relation between a certain context, a problem, and a solution.”

- **Context:** allows the reader to understand
  - the environment in which the problem resides
  - what solution might be appropriate within that environment.
- **a system of forces:** a set of requirements, including limitations and constraints, influences
  - how the problem can be interpreted within its context
  - how the solution can be effectively applied.





## 16.1 Design Patterns

- Effective Patterns ( [Coplien, 2005](#) )

- **It solves a problem**

not just abstract principles or strategies.

- **It is a proven concept**

not theories or speculation.

- **The solution isn't obvious**

generate a solution to a problem indirectly

--a necessary approach for the most difficult problems of design.

- **It describes a relationship**

don't just describe modules, but describe deeper system structures and mechanisms.

- **The pattern has a significant human component**

(minimize human intervention)

explicitly appeal to aesthetics and utility.

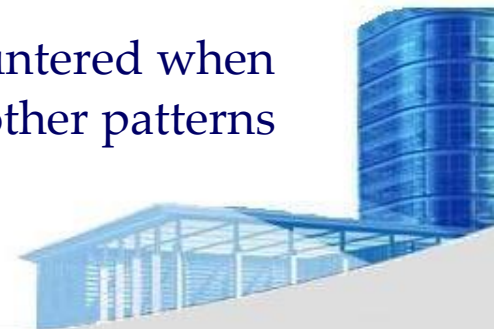




## 16.1 Design Patterns

- Kinds of Generative Patterns

- **Architectural patterns** describe broad-based design problems that are solved using a structural approach.
- **Data patterns** describe recurring data-oriented problems and the data modeling solutions that can be used to solve them.
- **Component patterns** (also referred to as *design patterns*) address problems associated with the development of subsystems and components, the manner in which they communicate with one another, and their placement within a larger architecture
- **Interface design patterns** describe common user interface problems and their solution with a system of forces that includes the specific characteristics of end-users.
- **WebApp patterns** address a problem set that is encountered when building WebApps and often incorporates many of the other patterns categories just mentioned.





## 16.1 Design Patterns

- Kinds of Patterns ( Gang of Four, 1995 )
  - **Creational patterns** focus on the “creation, composition, and representation of objects”
    - **Abstract factory pattern**: centralize decision of what **factory** to instantiate
    - **Factory method pattern**: centralize creation of an object of a specific type choosing one of several implementations
  - **Structural patterns** focus on problems and solutions associated with how classes and objects are organized and integrated to build a larger structure
    - **Adapter pattern**: 'adapts' one interface for a class into one that a client expects
    - **Aggregate pattern**: a version of the **Composite pattern** with methods for aggregation of children
  - **Behavioral patterns** address problems associated with the assignment of responsibility between objects and the manner in which communication is effected between objects
    - **Chain of responsibility pattern**: Command objects are handled or passed on to other objects by logic-containing processing objects
    - **Command pattern**: Command objects encapsulate an action and its parameters





## 16.1 Design Patterns

- Frameworks

- An **implementation-specific skeletal infrastructure** for design work.
- A “**reusable mini-architecture**” that provides the generic structure and behavior for a family of software abstractions, along with a context ... which specifies their collaboration and use within a given domain.”  
[Amb98]
- Not an architectural pattern, but rather a **skeleton with a collection of “plug points”** (also called hooks and slots) that enable it to be adapted to a specific problem domain.
  - **The plug points enable you to integrate problem specific classes or functionality within the skeleton.**







## 16.1 Design Patterns

- Describing a Pattern

☆ “ I N F O ” in Section 16.1.3		
Pattern name	Problem	Motivation
Context	Forces	Solution
Intent	Collaborations	Consequences
Implementation	Known uses	Related patterns

- Pattern Languages and Patterns Repositories

**Please read Section 16.1.4**

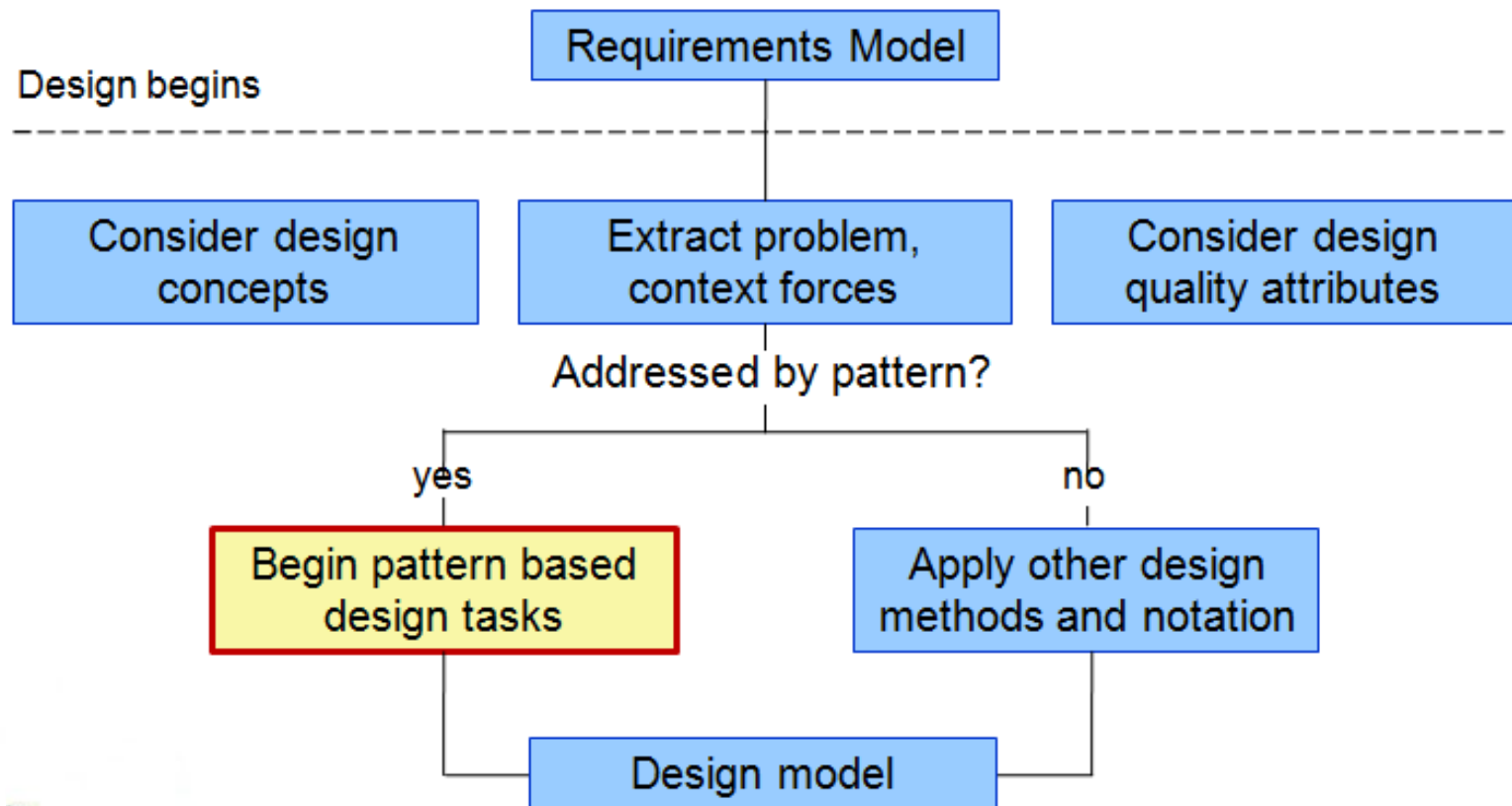






## 16.2 Pattern-Based Software Design

- Pattern-Based Design in Context



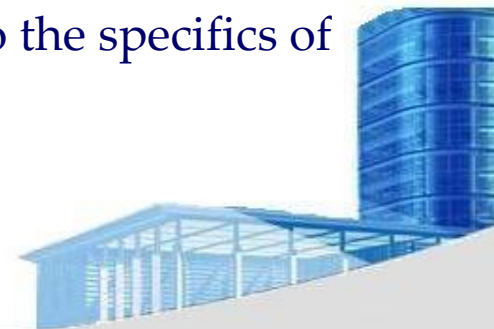


## 16.2 Pattern-Based Software Design

- Thinking in Patterns and Design Tasks

- Shalloway and Trott [Sha05] suggest the following approach :

- Step1: 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.
- Step2: Examining the big picture, extract the patterns that are present at that level of abstraction.
- Step3: Begin your design with ‘big picture’ patterns that establish a context or skeleton for further design work.
- Step4: “Work inward from the context” looking for patterns at lower levels of abstraction that contribute to the design solution.
- Step5: Repeat steps 1 to 4 until the complete design is fleshed out.
- Step6: Refine the design by adapting each pattern to the specifics of the software you’re trying to build.





## 16.2 Pattern-Based Software Design

- Design Tasks and Common Design Mistakes
  - Examine the requirements model and develop a problem hierarchy.
  - Determine if a reliable pattern language has been developed for the problem domain.
  - Beginning with a broad problem, determine whether one or more architectural patterns are available for it.
  - Using the collaborations provided for the architectural pattern, examine subsystem or component level problems and search for appropriate patterns to address them.
  - Repeat steps 2 through 5 until all broad problems have been addressed.
  - 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.
  - 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 refine the design as it is derived from patterns using design quality criteria as a guide.



## 16.2 Pattern-Based Software Design

- Pattern Organizing Table

	Database	Application	Implementation	Infrastructure
<b>Data/Content</b>				
Problem statement	PatternName(s)		PatternName(s)	
Problem statement		PatternName(s)		PatternName(s)
Problem statement	PatternName(s)			PatternName(s)
<b>Architecture</b>		PatternName(s)		
Problem statement		PatternName(s)		
Problem statement				PatternName(s)
Problem statement				
<b>Component-level</b>		PatternName(s)	PatternName(s)	
Problem statement		PatternName(s)	PatternName(s)	PatternName(s)
Problem statement				
Problem statement				
<b>User interface</b>		PatternName(s)	PatternName(s)	
Problem statement		PatternName(s)	PatternName(s)	
Problem statement		PatternName(s)	PatternName(s)	
Problem statement		PatternName(s)	PatternName(s)	



## 16.2 Pattern-Based Software Design

- Common Design Mistakes
  - Not enough time has been spent to understand the underlying problem, 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 erroneous fit.
  - Sometimes a pattern is applied too literally and the required adaptations for your problem space are not implemented.





## 16.3 Architectural Patterns

- Architectural Patterns address issues such as concurrency, persistence, and distribution

### Example: House & Kitchen pattern

- The Kitchen pattern and patterns it collaborates with address problems associated with the **storage and preparation** of food, the **tools** required to accomplish these tasks, and **rules** for placement of these tools relative to workflow in the room.
- The Kitchen pattern might address problems associated with **counter tops, lighting, wall switches, a central island, flooring, and so on.**
- There is **more than a single design** for a kitchen, but every design can be conceived within the context of the 'solution' suggested by the Kitchen pattern.







## 16.4 Component-Level Patterns

- Component-level design patterns provide a proven solution that addresses one or more sub-problems extracted from the requirement model.
- In many cases, design patterns of this type focus on some functional element of a system.

### Example: SafeHomeAssured.com

- The design sub-problem: How can we get product specifications and related information for any SafeHome device?
- Search-related patterns: counter AdvancedSearch, HelpWizard, SearchArea, SearchTips, SearchResults, SearchBox, .....







## 16.5 User Interface (UI) Patterns

- Design Focus

- **Whole UI.** Provide design guidance for top-level structure and navigation throughout the entire interface.
- **Page layout.** Address the general organization of pages (for Websites) or distinct screen displays (for interactive applications)
- **Forms and input.** Consider a variety of design techniques for completing form-level input.
- **Tables.** Provide design guidance for creating and manipulating tabular data of all kinds.
- **Direct data manipulation.** Address data editing, modification, and transformation.
- **Navigation.** Assist the user in navigating through hierarchical menus, Web pages, and interactive display screens.
- **Searching.** Enable content-specific searches through information maintained within a Web site or contained by persistent data stores that are accessible via an interactive application.
- **Page elements.** Implement specific elements of a Web page or display screen.
- **E-commerce.** Specific to Web sites, these patterns implement recurring elements of e-commerce applications.



## 16.6 Webapp Patterns

- Categories
  - **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 consequences of a specific action; how a user expands content based on usage 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 on-going 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 relationship between an interface action and the content objects it affects, and how to establish effective content hierarchies.
  - **Functional patterns** define the workflows, behaviors, processing, communications, and other algorithmic elements within a WebApp.





## 16.6 Webapp Patterns

- Categories
  - **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 consequences of a specific action; how a user expands content based on usage 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 on-going 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 relationship between an interface action and the content objects it affects, and how to establish effective content hierarchies.
  - **Functional patterns** define the workflows, behaviors, processing, communications, and other algorithmic elements within a WebApp.





## 16.7 Patterns for Mobile Apps

- Mobile User Interface Patterns
  - Check-in screens, Maps, Popovers, Sign-up flows, Custom Tab Navigation, Invitations
- Mobile User Interface Patterns
  - Active Objects
  - Applications Controller
  - Communicator
  - Data Transfer Object
  - Domain Model
  - Entity Translator
  - Lazy Acquisition
  - Model-View-Controller
  - Pagination
  - Reliable Sessions
  - Synchronization
  - Transaction Script

