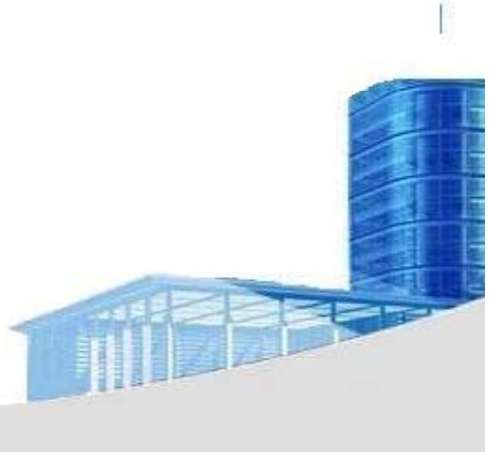
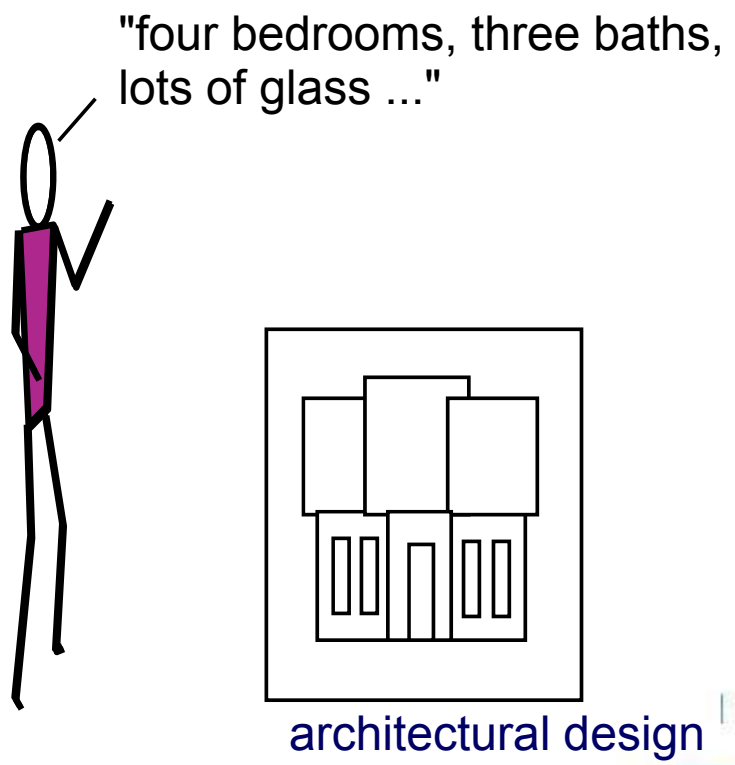
Ch.13 Architectural Design

April 27, 2015

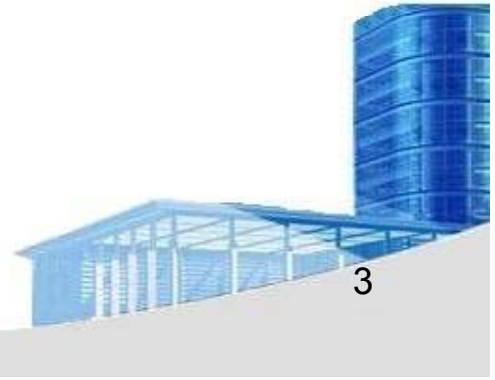


• Why Architecture?

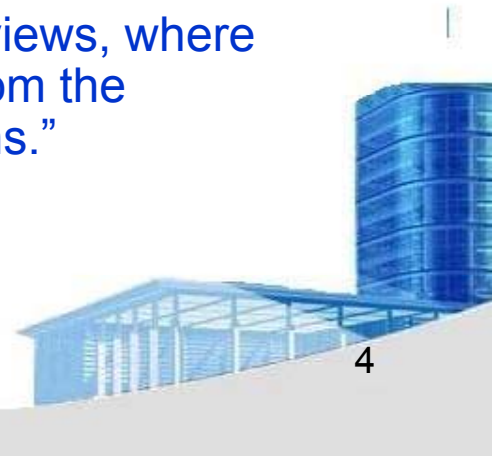
- The architecture is not the operational software. Rather, it is a representation that enables a software engineer to:
  - (1) ***analyze the effectiveness of the design*** in meeting its stated requirements,
  - (2) ***consider architectural alternatives*** at a stage when making design changes is still relatively easy, and
  - (3) ***reduce the risks*** associated with the construction of the software.



- **Why is Architecture Important?**
  - *Representations of software architecture are an enabler* for communication between all parties (stakeholders) interested in the development of a computer-based system.
  - *The architecture highlights early design decisions* that will have a profound impact on all software engineering work that follows and, as important, on the ultimate success of the system as an operational entity.
  - *Architecture "constitutes a relatively small, intellectually graspable mode* of how the system is structured and how its components work together" [BAS03].



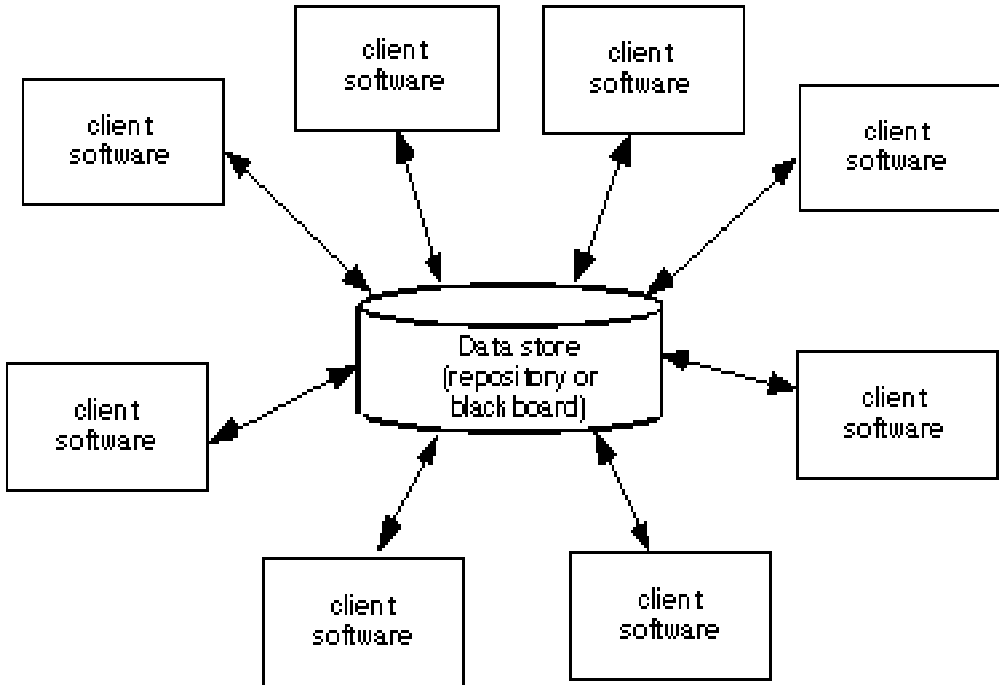
- **Architectural Descriptions**
  - The IEEE Computer Society has proposed IEEE-Std-1471-2000, Recommended Practice for Architectural Description of Software-Intensive System, [IEE00]
    - to establish a conceptual framework and vocabulary for use during the design of software architecture,
    - to provide detailed guidelines for representing an architectural description, and
    - to encourage sound architectural design practices.
  - The IEEE Standard defines an Architectural Description (AD) as a "a collection of products to document an architecture."
    - The description itself is represented using multiple views, where each view is "a representation of a whole system from the perspective of a related set of [stakeholder] concerns."



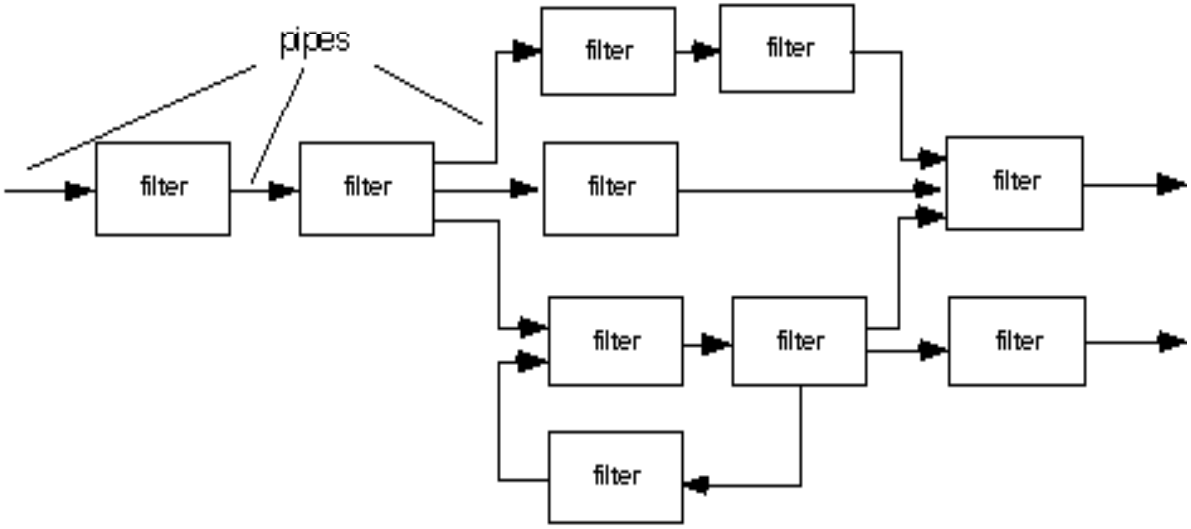
- **Architectural Genres**
  - *Genre(类型,样式)* implies a specific category within the overall software domain.
  - Within each category, you encounter a number of subcategories.
    - For example, within the genre of buildings, you would encounter the following general ***styles***: houses, condos(有独立产权公寓), apartment buildings, office buildings, industrial building, warehouses, and so on.
    - Within each general style, more specific styles might apply. Each style would have a structure that can be described using a set of predictable patterns.

- **Architectural Styles**
  - Each style describes a system category that encompasses:
  - (1) a *set of components* (e.g., a database, computational modules) that perform a function required by a system,
  - (2) a *set of connectors* that enable “communication, coordination and cooperation” among components,
  - (3) *constraints* that define how components can be integrated to form the system,
  - (4) *semantic models* that enable a designer to understand the overall properties of a system by analyzing the known properties of its constituent parts.
    - Data-centered architectures
    - Data flow architectures
    - Call and return architectures
    - Object-oriented architectures
    - Layered architectures

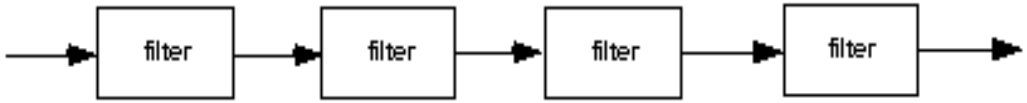
- **Data-Centered Architecture**



- **Data Flow Architecture**

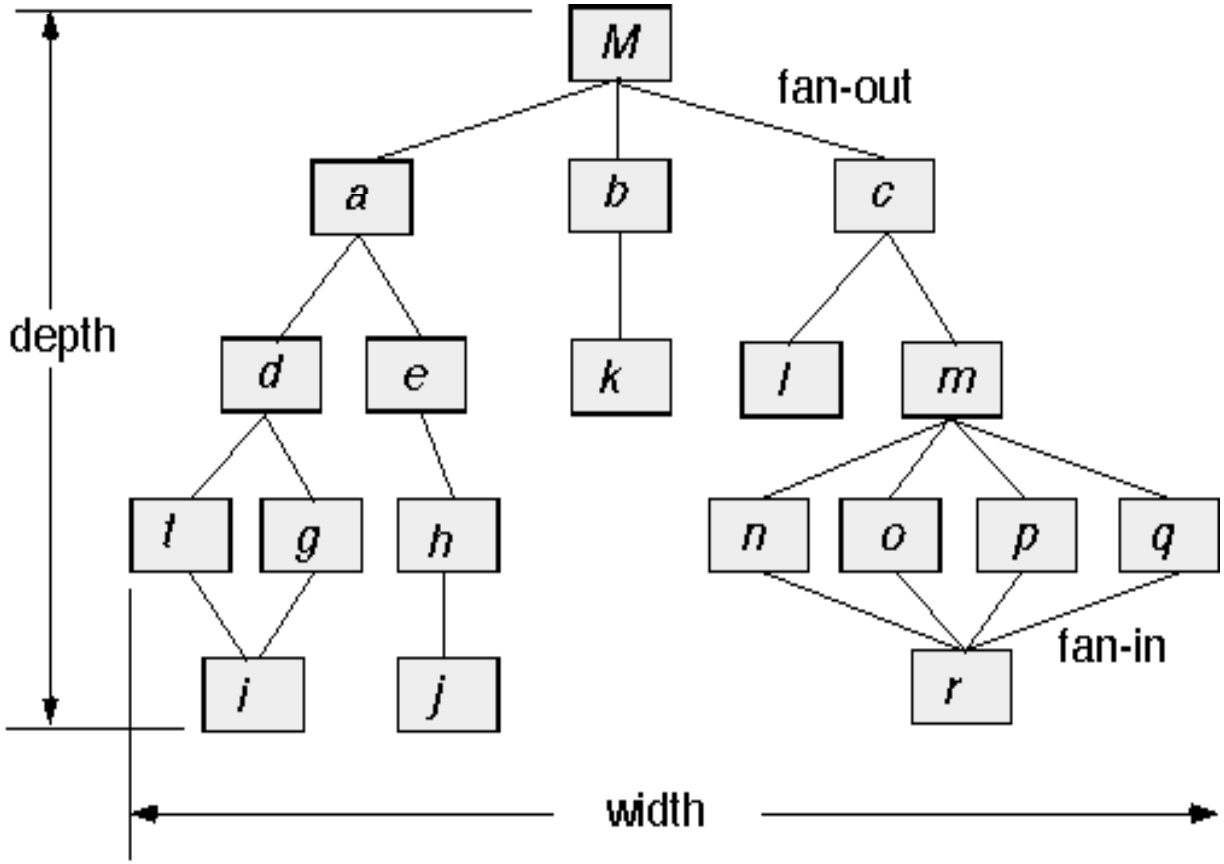


(a) pipes and filters

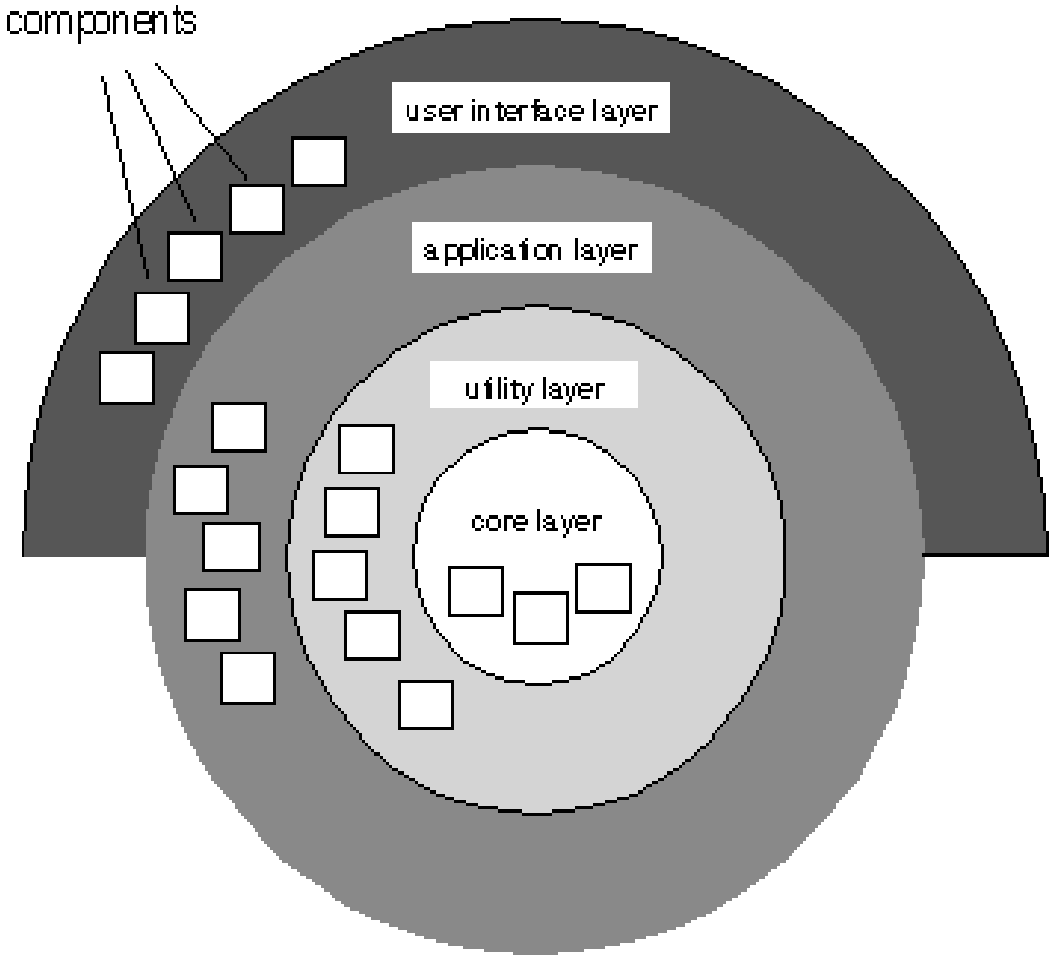


(b) batch sequential

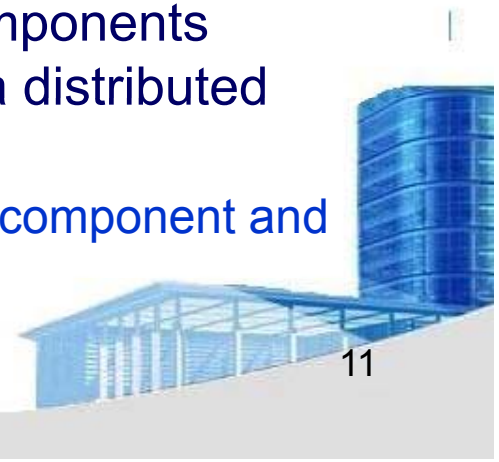
- **Call and Return Architecture**



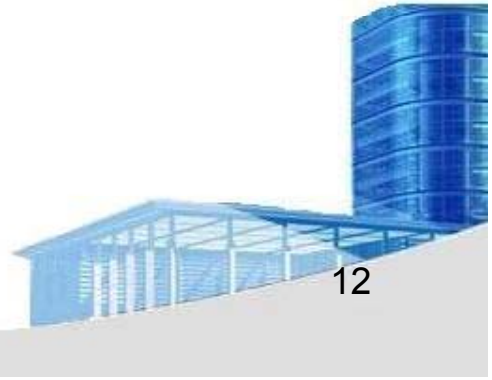
- **Layered Architecture**



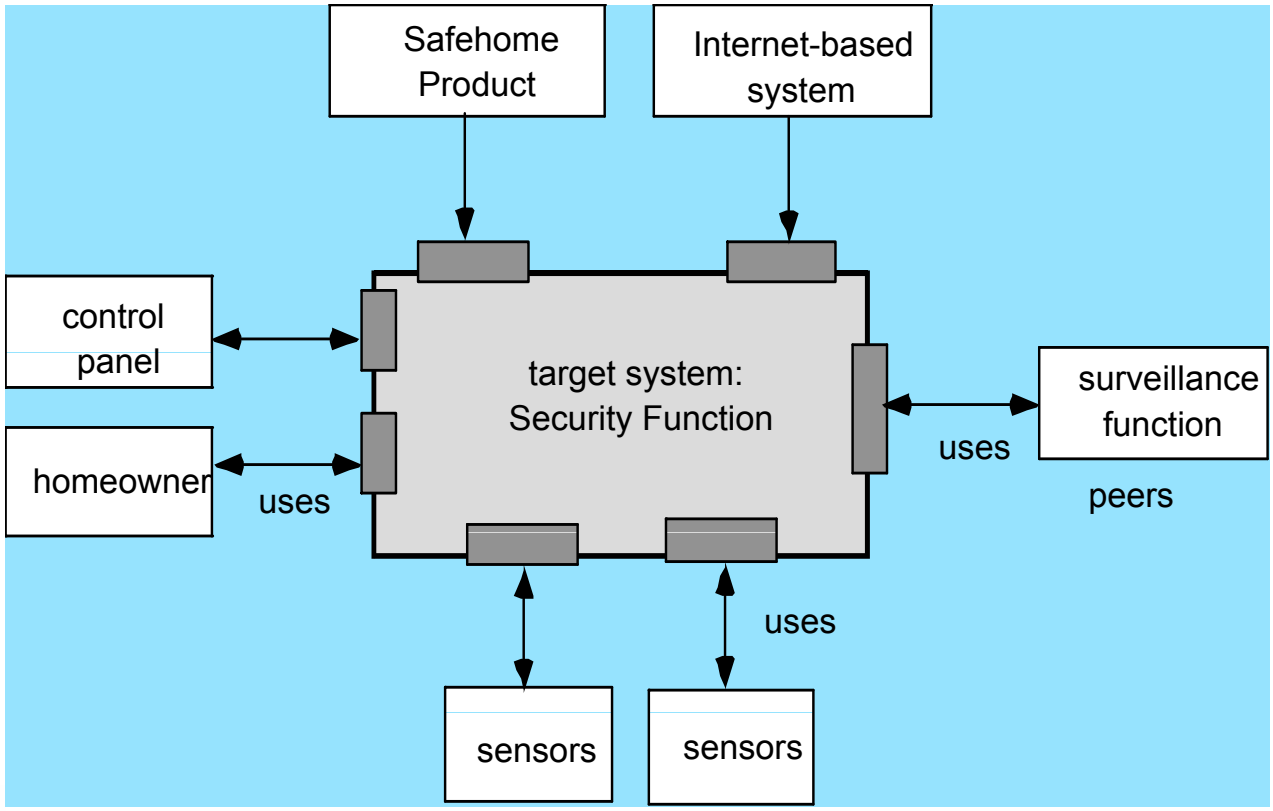
- **Architectural Patterns**
  - *Concurrency*—applications must handle multiple tasks in a manner that simulates parallelism
    - ***operating system process management*** pattern
    - ***task scheduler*** pattern
  - *Persistence*—Data persists if it survives past the execution of the process that created it. Two patterns are common:
    - a ***database management system*** pattern that applies the storage and retrieval capability of a DBMS to the application architecture
    - an ***application level persistence*** pattern that builds persistence features into the application architecture
  - *Distribution*— the manner in which systems or components within systems communicate with one another in a distributed environment
    - A ***broker*** acts as a 'middle-man' between the client component and a server component.



- **Architectural Design**
  - The software must be placed into context
    - the design should define the external entities (other systems, devices, people) that the software interacts with and the nature of the interaction
  - A set of architectural archetypes should be identified
    - An ***archetype(原型)*** is an abstraction (similar to a class) that represents one element of system behavior
  - The designer specifies the structure of the system by defining and refining software components that implement each ***archetype***



- **Architectural Context**



- **Archetypes**

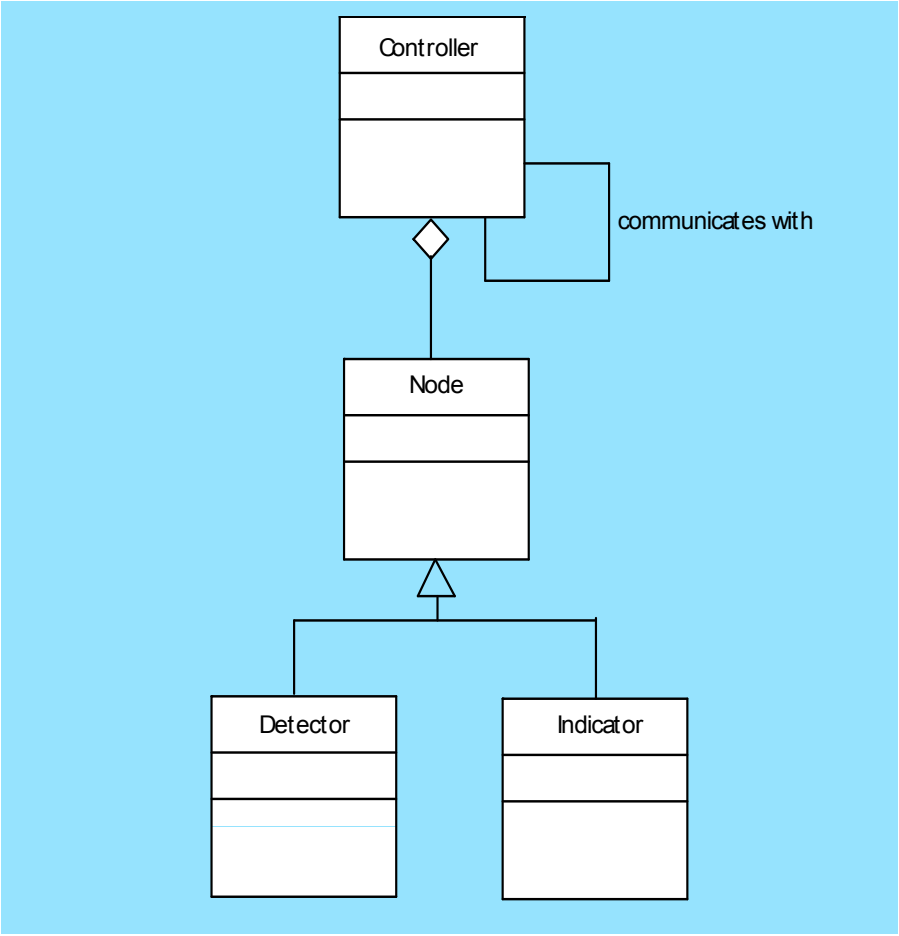
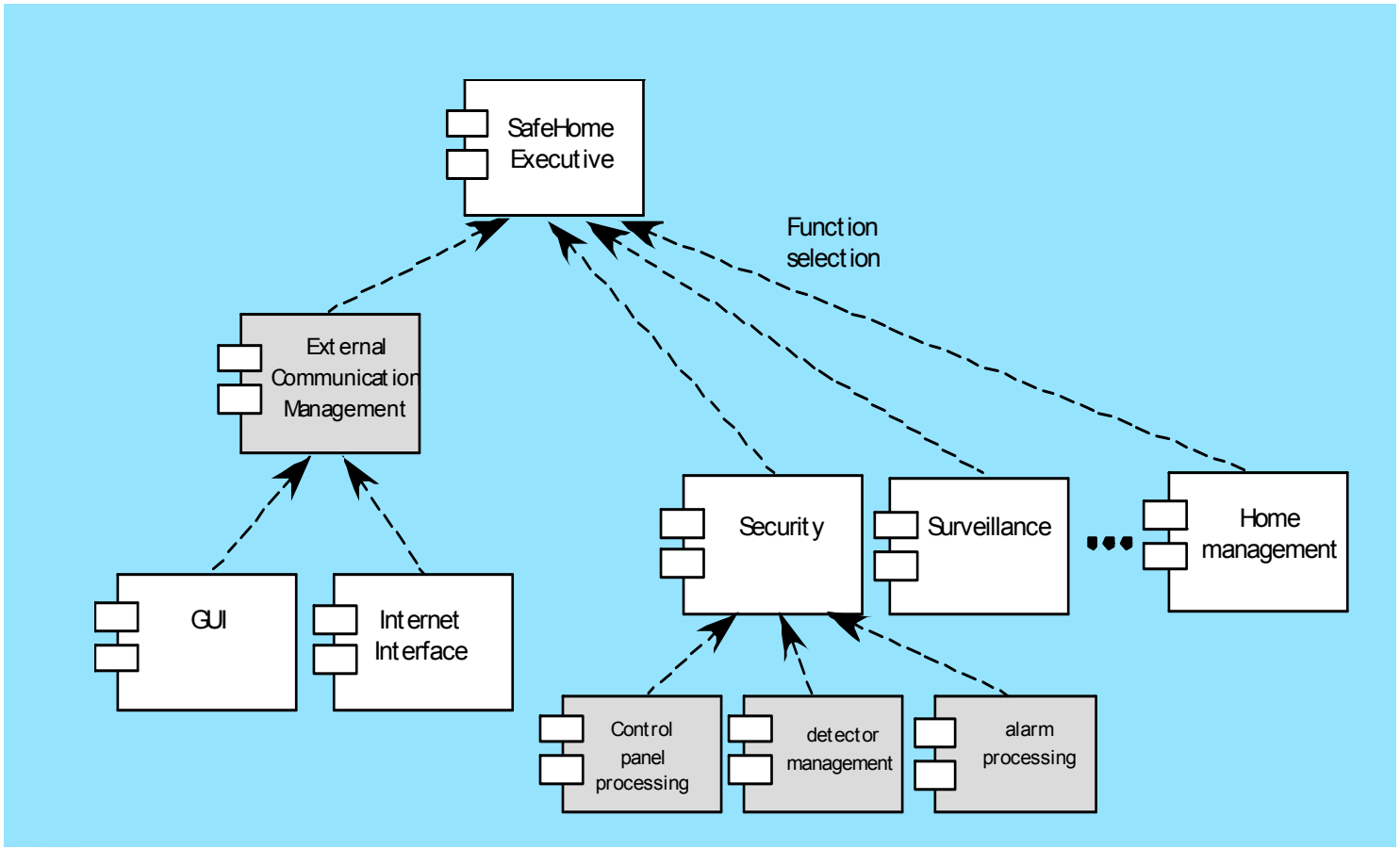
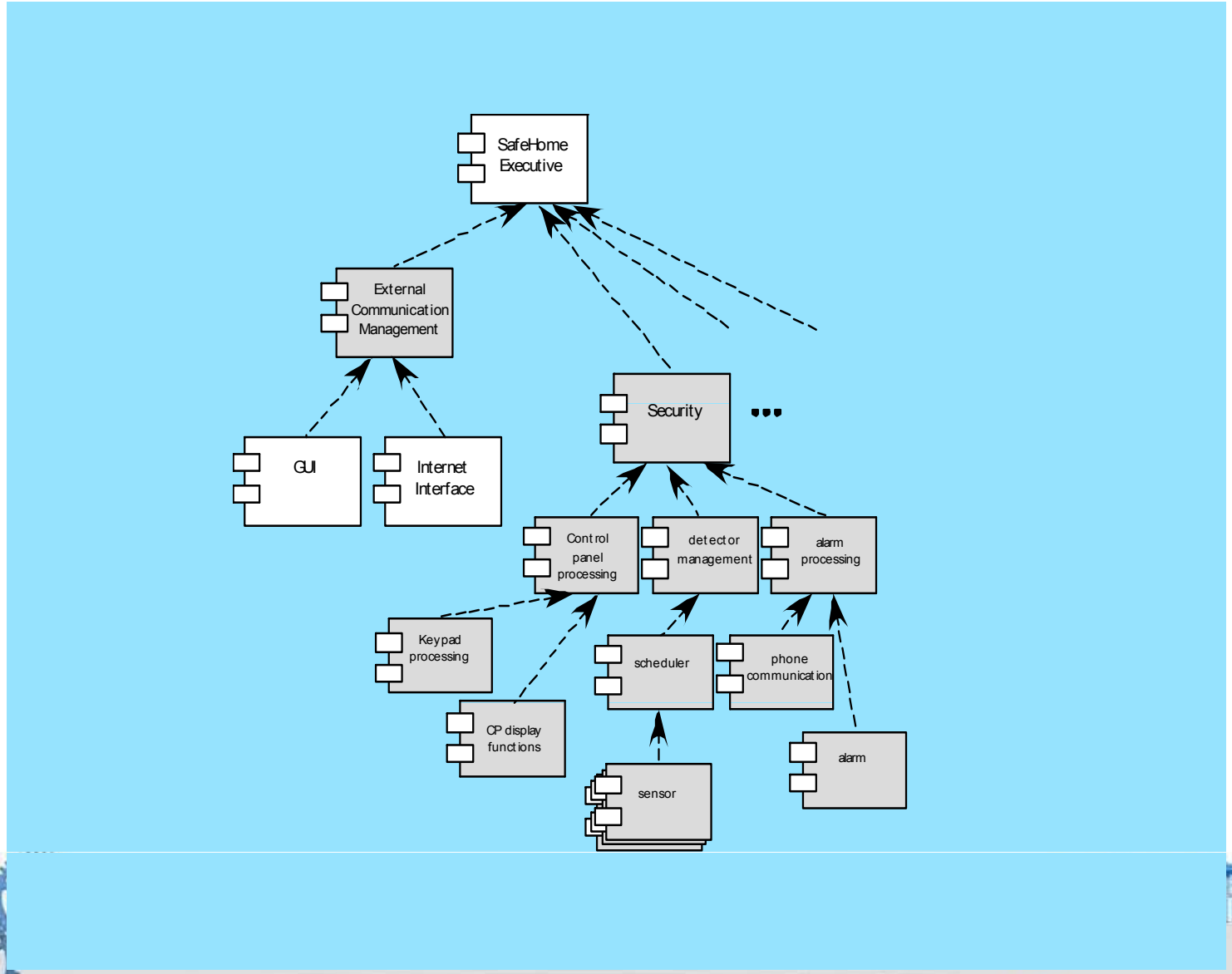


Figure 10.7 UML relationships for SafeHomesecurity function archetypes (adapted from [BOS00])

- **Component Structure**



• Refined Component Structure



- **Architectural Considerations**
  - *Economy* – The best software is uncluttered and relies on abstraction to reduce unnecessary detail.
  - *Visibility* – Architectural decisions and the reasons for them should be obvious to software engineers who examine the model at a later time.
  - *Spacing* – Separation of concerns in a design without introducing hidden dependencies.
  - *Symmetry* – Architectural symmetry implies that a system is consistent and balanced in its attributes.
  - *Emergence* – Emergent, self-organized behavior and control.

- **Architectural Decision Documentation**
  - Determine which information items are needed for each decision.
  - Define links between each decision and appropriate requirements.
  - Provide mechanisms to change status when alternative decisions need to be evaluated.
  - Define prerequisite(前提) relationships among decisions to support traceability.
  - Link significant decisions to architectural views resulting from decisions.
  - Document and communicate all decisions as they are made.

- **Architectural Tradeoff Analysis**

1）Collect scenarios.

2）Elicit requirements, constraints, and environment description.

3）Describe the architectural styles/patterns that have been chosen to address the scenarios and requirements:

  – module view →assignments with components
  – process view →system performance
  – data flow view →functional requirements

4）Evaluate quality attributes by considered each attribute in isolation(reliability, performance, security, maintainability, flexibility, testability, portability, reusability, and interoperability).

5）Identify the sensitivity of quality attributes to various architectural attributes for a specific architectural style.

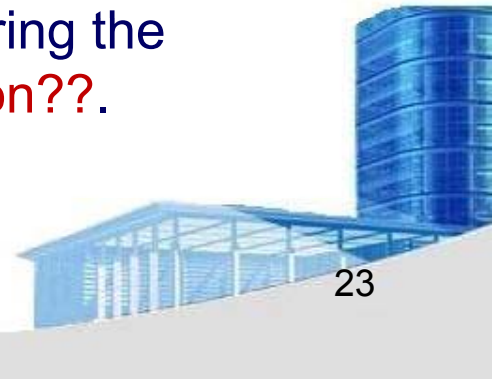
6）Critique（评论） candidate architectures (developed in step 3) using the sensitivity analysis conducted in step 5.

- **Architectural Complexity**
  - the overall complexity of a proposed architecture is assessed by considering the *dependencies* between components within the architecture [Zha98]
    - ***Sharing* dependencies** represent dependence relationships among consumers who use the same resource or producers who produce for the same consumers.
    - ***Flow* dependencies** represent dependence relationships between producers and consumers of resources.
    - ***Constrained* dependencies** represent constraints on the relative flow of control among a set of activities.

- **ADL**
  - *Architectural Description Language (ADL)* provides a semantics and syntax for describing a software architecture
  - Provide the designer with the ability to:
    - decompose architectural components
    - compose individual components into larger architectural blocks and
    - represent interfaces (connection mechanisms) between components.

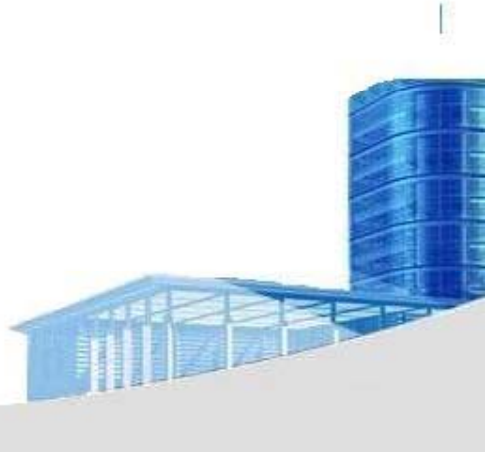
- **Architecture Reviews**
  - Assess the ability of the software architecture to meet the systems quality requirements and identify potential risks
  - Have the potential to reduce project costs by detecting design problems early
  - Often make use of experience-based reviews, prototype evaluation, and scenario reviews, and checklists（清单，检查表）

- **Pattern-Based Architecture Review**
  - Identify and discuss the quality attributes by walking through the use cases.
  - Discuss a diagram of system's architecture in relation to its requirements.
  - Identify the architecture patterns used and match the system's structure to the patterns' structure.
  - Use existing documentation and use cases to determine each pattern's effect on quality attributes.
  - Identify all quality issues raised by architecture patterns used in the design.
  - Develop a short summary of issues uncovered during the meeting and make revisions to the walking skeleton??.



- **Agility and Architecture**
  - To avoid rework, user stories are used to create and evolve an architectural model (walking skeleton) before coding
  - Hybrid models which allow software architects contributing users stories to the evolving storyboard
  - Well run agile projects include delivery of work products during each sprint
  - Reviewing code emerging from the sprint can be a useful form of architectural review

Ch.17 WebApp Design



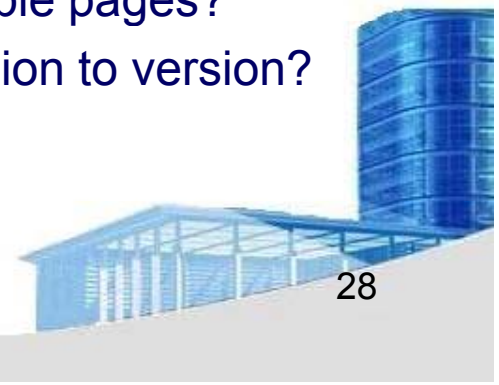
- **Design & WebApps**
  - "There are essentially two basic approaches to design: the artistic ideal of expressing yourself and the engineering ideal of solving a problem for a customer."

    Jakob Nielsen

  - *When should we emphasize WebApp design?*
    - when content and function are complex
    - when the size of the WebApp encompasses hundreds of content objects, functions, and analysis classes
    - when the success of the WebApp will have a direct impact on the success of the business

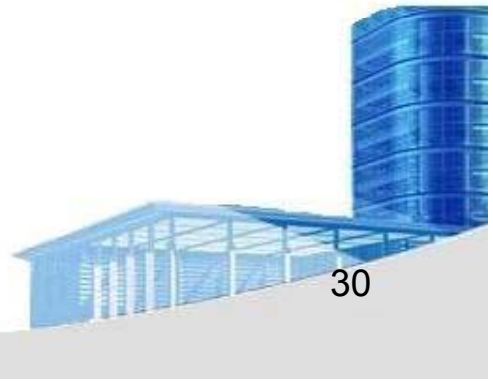
- **Design & WebApp Quality**
  - *Security*
    - Rebuff (挫败) external attacks
    - Exclude unauthorized access
    - Ensure the privacy of users/customers
  - *Availability*
    - the measure of the percentage of time that a WebApp is available for use→24/7/365
  - *Scalability*
    - **Can** the WebApp and the systems with which it is interfaced handle significant variation in user or transaction volume
  - *Time to Market*

- **Quality Dimensions for End-Users**
  - *Time*
    - How much has a Web site changed since the last upgrade?
    - How do you highlight the parts that have changed?
  - *Structural*
    - How well do all of the parts of the Web site hold together.
    - Are all links inside and outside the Web site working?
    - Do all of the images work?
    - Are there parts of the Web site that are not connected?
  - *Content*
    - Does the content of critical pages match what is supposed to be there?
    - Do key phrases exist continually in highly-changeable pages?
    - Do critical pages maintain quality content from version to version?
    - What about dynamically generated HTML pages?



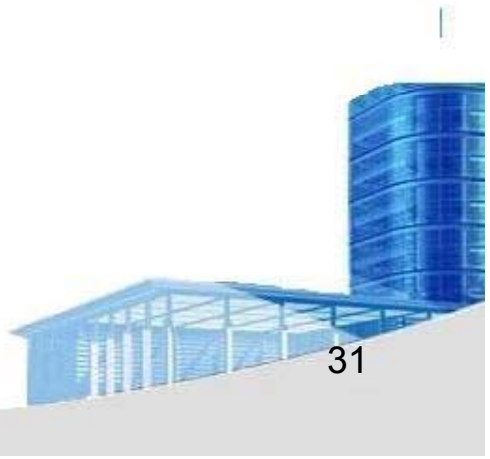
- **Quality Dimensions for End-Users**
  - *Accuracy and Consistency*
    - Are today's copies of the pages downloaded the same as yesterday's? Close enough?
    - Is the data presented accurate enough? How do you know?
  - *Response Time and **Latency** (潜在因素)*
    - Does the Web site server respond to a browser request within certain parameters?
    - In an E-commerce context, how is the end to end response time after a SUBMIT?
    - Are there parts of a site that are so slow the user declines to continue working on it?
  - *Performance*
    - Is the Browser-Web-Web site-Web-Browser connection quick enough?
    - How does the performance vary by time of day, by load and usage?
    - Is performance adequate for E-commerce applications?

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

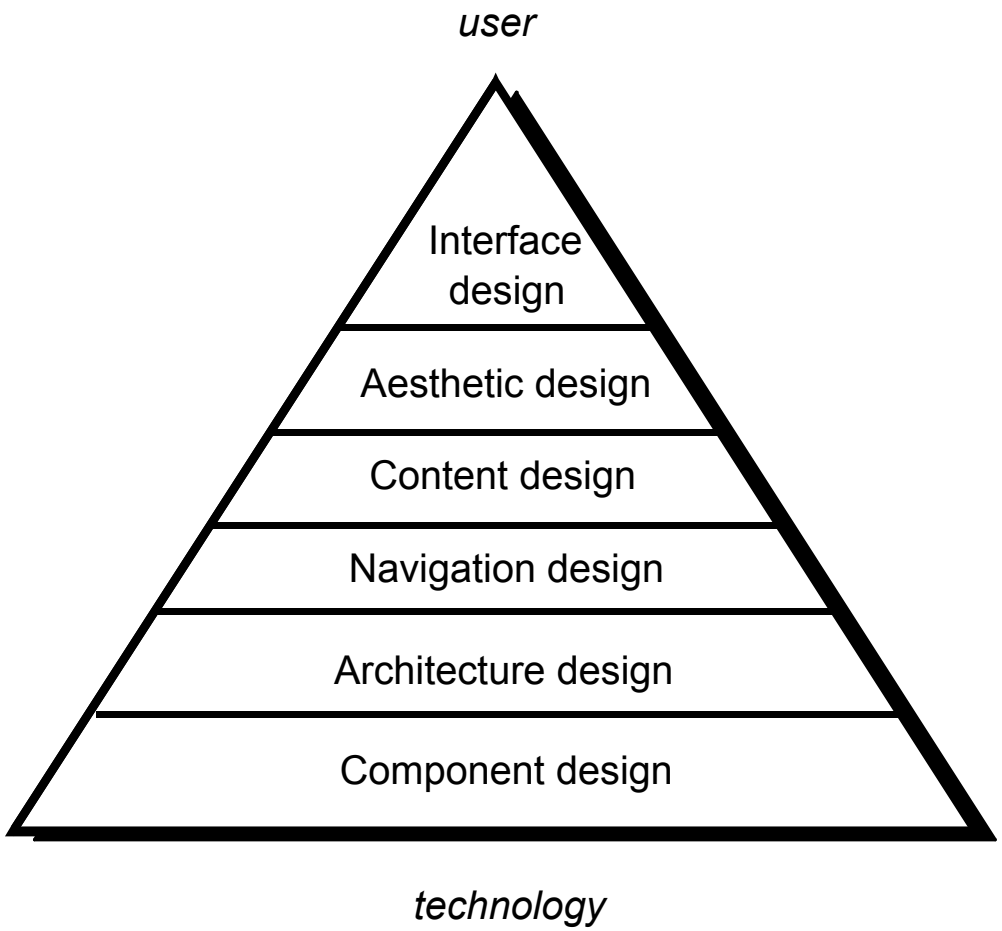


• WebApp Design Goals

- *Identity*
  - Establish an “identity” that is appropriate for the business purpose
- *Robustness*
  - The user expects robust content and functions that are relevant to the user’s needs
- *Navigability*
  - designed in a manner that is intuitive and predictable
- *Visual appeal*
  - the look and feel of content, interface layout, color coordination, the balance of text, graphics and other media, navigation mechanisms must appeal to end-users
- *Compatibility*
  - With all appropriate environments and configurations

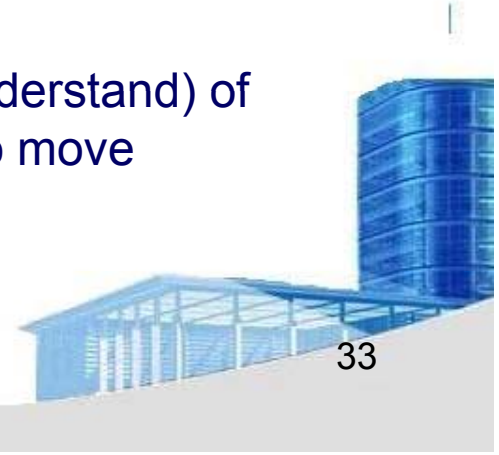


- **WebApp Design Pyramid**



• WebApp Interface Design

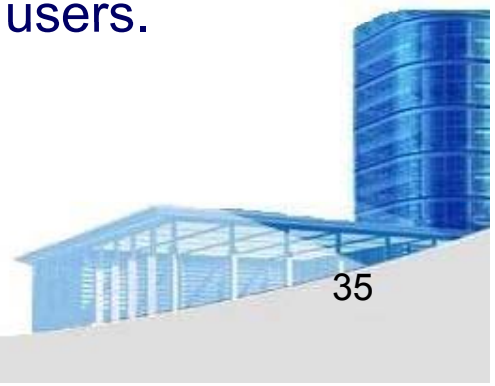
- *Where am I?* The interface should
  - provide an indication of the WebApp that has been accessed
  - inform the user of her location in the content hierarchy.
- *What can I do now?* The interface should always help the user understand his current options
  - what functions are available?
  - what links are live?
  - what content is relevant?
- *Where have I been, where am I going?* The interface must facilitate navigation.
  - Provide a “map” (implemented in a way that is easy to understand) of where the user has been and what paths may be taken to move elsewhere within the WebApp.



• WebApp Interface Design

- *Where an*
  - provic
  - inform
- *What can*
  understan
  - what fi
  - what li
  - what c
- *Where ha*
  facilitate r
  - Provid
    where
    elsewh

- **Effective WebApp Interfaces**
  - Bruce Tognozzi [TOG01] suggests…
    - *Effective interfaces are visually apparent and forgiving*, instilling in their users a sense of control. Users quickly see the breadth of their options, grasp how to achieve their goals, and do their work.
    - *Effective interfaces do not concern the user with the inner workings of the system.* Work is carefully and continuously saved, with full option for the user to undo any activity at any time.
    - *Effective applications and services perform a maximum of work*, while requiring a minimum of information from users.

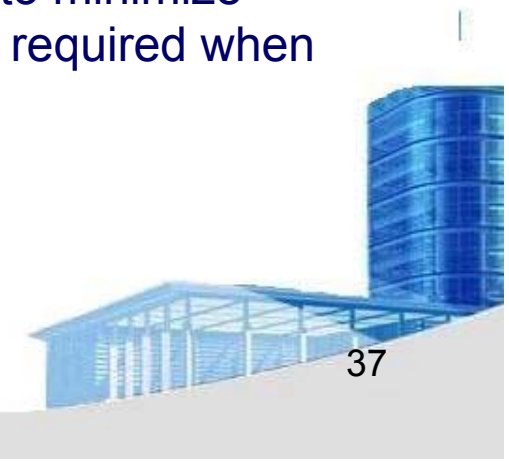


• Interface Design Principles - I

- *Anticipation*—A WebApp should be designed so that it anticipates(预见) the use's next move.
- *Communication*—The interface should communicate the status of any activity initiated by the user
- *Consistency*—The use of navigation controls, menus, icons, and aesthetics (e.g., color, shape, layout)
- *Controlled autonomy*—The interface should facilitate user movement throughout the WebApp, but it should do so in a manner that enforces navigation conventions that have been established for the application.
- *Efficiency*—The design of the WebApp and its interface should optimize the user's work efficiency, not the efficiency of the Web engineer who designs and builds it or the client-server environment that executes it.

• Interface Design Principles - II

- *Focus*—The WebApp interface (and the content it presents) should stay focused on the user task(s) at hand.
- *Fitt's Law*—"The time to acquire a target is a function of the distance to and size of the target."
- *Human interface objects*—A vast library of reusable human interface objects has been developed for WebApps.
- *Latency reduction*—The WebApp should use multi-tasking in a way that lets the user proceed with work as if the operation has been completed.
- *Learnability*— A WebApp interface should be designed to minimize learning time, and once learned, to minimize relearning required when the WebApp is revisited.



《System Design》

(8-10minute presentation +2minute- Q&A)

Speech Time: 08:30 on May 10, 2015

Grading Policy: Each group will evaluate the other groups' performances and fill in the *grading tables*. For each group, let p1 be the average points given by the other groups with the maximum and minimum points taken off; and let p2 be the points given by the instructor, the final points obtained will be (p1 + p2) / 2. The full mark = 50 points × number of participants

Tasks

- **Review** Ch.13, 17
- **Finish** “Problems and points to ponder” in Ch. 13, 17
- **Preview** Ch 18,20,21
- Prepare the **System Design Speech** on **May 10,** (**Sunday**, **8:30am, Room 7-102?**)
- **Submit System design Specification due May 13!**