Session 5

Design Modeling for WebApps (I)

- Chapter 8: WebApp Design
- Chapter 9: Interaction Design

Outcomes of This Session

- Understand the overall design process of WebApps.
- Understand how to use interface design principles and techniques in real Web projects.
- Understand how to use aesthetic design principles and techniques in real Web projects.

Design & WebApps

- When should we focus strongly on WebApp design?
  1. When **content and functions are complex**
  2. **When the size of the WebApp encompasses** hundreds of content objects, functions, and analysis classes
  3. When **multiple people become** involved in the design
  4. When the success of the WebApp will have a direct impact on the success of the business

Consequence of Design

- Move from *what it should do* to *how it should be done.*
- Design model
  - Provides sufficient information for the Web team to construct the final WebApp.
- Logical design
  - Involves the identification of those theoretical things that have to occur inside the WebApp - in order to meet stakeholder requirements and conform to the analysis model.
- Physical design
  - Maps our understanding of the logical design into the actual physical elements to be implemented as part of the WebApp – *such as Web servers, database server, Web pages and so on*.

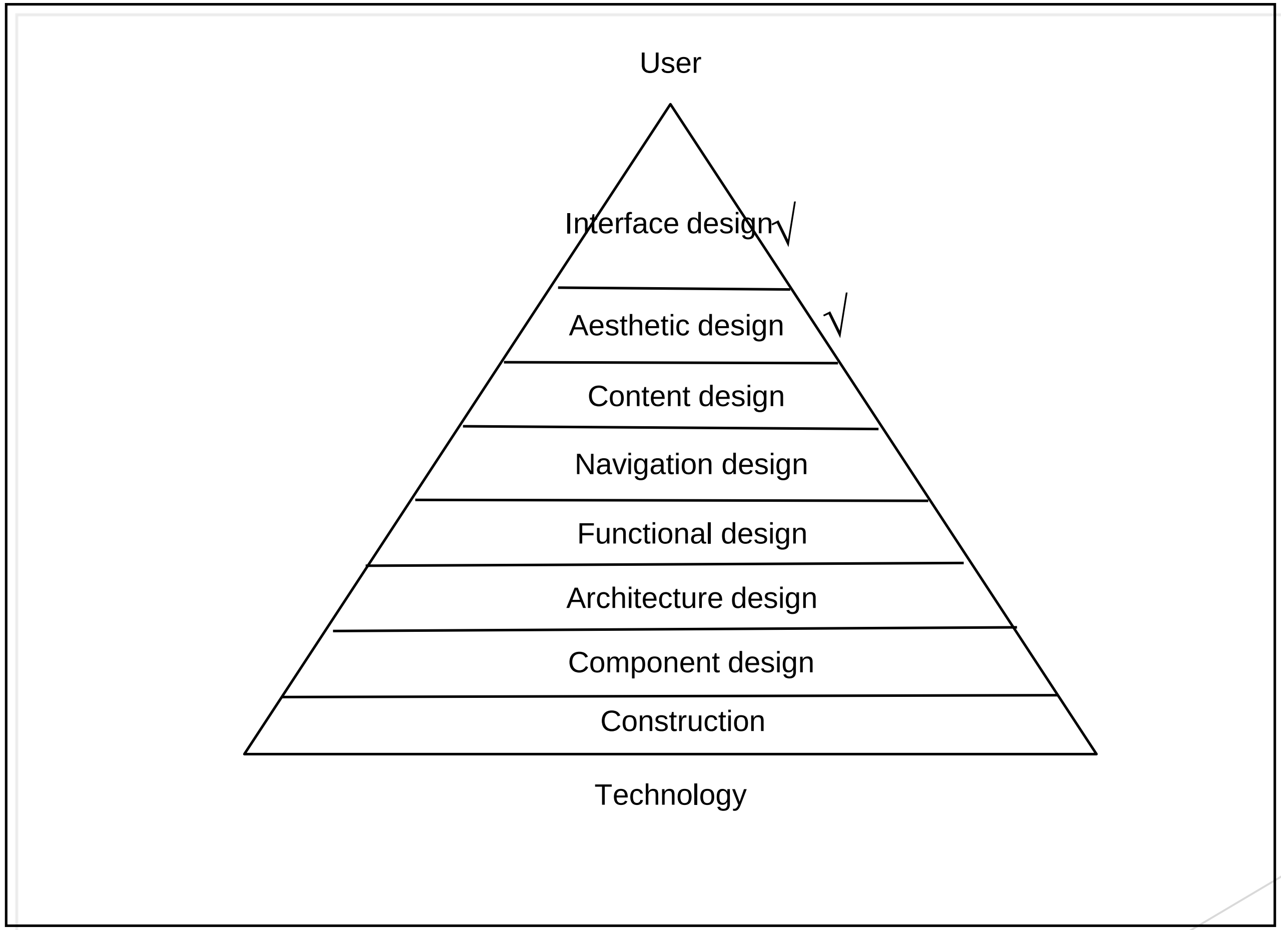
WebApp Design Goals!

- Simplicity!
  - “All things in moderation”
- 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 structure
  - Interface design should define consistency with regards to interaction, navigation and content display
  - Navigation mechanisms how users will navigate - 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 strong content and functions that are relevant to the user’s needs
- Navigation
  - 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
  - E.g. all major browsers

Elements of Design



Elements of Design

- *Interaction design*
  - Interface design: structure and organization of the user interface.
  - Aesthetic design: look and feel of the WebApp.
- *Information design*
  - Content design: layout, structure and outline for all WebApp content
  - Navigation design: navigation flow between content objects and WebApp functions
- *Functional design:* overall behavior and functionality supported by the WebApp
- *Technical design*
  - Architecture design: overall structure for the WebApp.
  - Component design: detailed processing logic required to implement functional component that support a complete set of WebApp functions.

WebApp Interface Design

- The "first impression" of a WebApp.
- A poorly designed interface will disappoint the potential user and may, in fact, cause the user to go elsewhere.
- ***Where am I?*** The interface should:
  - provide an indication of the WebApp that has been accessed.
  - inform the user of their location in the content hierarchy.
- *What can I do now?* The interface should always help the user understand their 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" of where the user has been and what paths may be taken to move elsewhere within the WebApp.

Interface Design Principles-I

- Anticipation—A WebApp should be designed so that it anticipates the user'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 pathways that have been established for the application.
- Efficiency—The design of the WebApp and its interface should optimize the user's work efficiency
- Focus—The WebApp interface (and the content it presents) should stay focused on the user task(s) at hand.

Interface Design Principles-II

- **Maintain work product integrity**—A work product (e.g., a form completed by the user, a user specified list) must be automatically saved so that it will not be lost if an error occurs.
- **Readability**—All information presented through the interface should be readable by young and old.
- **Track state**—When appropriate, the state of the user interaction should be tracked and stored so that a user can logoff and return later to pick up where she left off.
- **Visible navigation**—A well-designed WebApp interface provides "the illusion that users are in the same place, with the work brought to them."

Other complements to the above principles (see Section: Interface Design Principles and Guidelines of chapter 9 in the textbook)

Practical Design Guidelines

- Do not force the user to read voluminous amount of text.
- Avoid “under construction” signs
- Users prefer not to scroll
- Navigation menus and head bars should be designed consistently and should be available on all pages that are available to the user.
- Aesthetics should never supersede functionality.
- Navigation options should be obvious, even to the casual user.

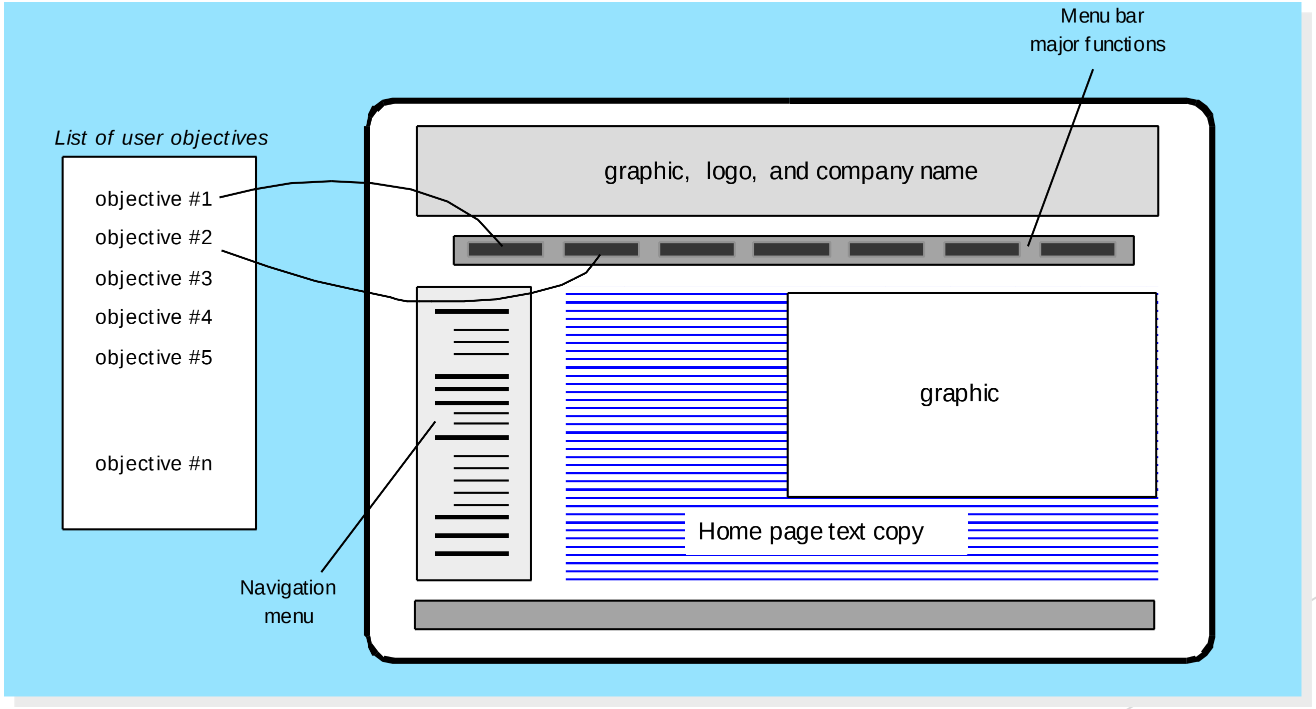
Interface Design Workflow-I

1. Review information contained in the analysis model and refine as required.
2. Develop a rough sketch of the WebApp interface layout.
3. Map user objectives into specific interface actions.
4. Define a set of user tasks that are associated with each action.
5. Develop screen images for each interface action.
6. Refine interface layout and screen images using input from aesthetic design.

Mapping User Objectives

Design is not just how it looks, but how it works!

Link the objectives with the interface functionality.



1. Your business objectives. The action you want visitors to take on the site.

2. User objectives. The desires or needs that they want to satisfy.

Map the objectives to your site!

Interface Design Workflow-II

7. Identify user interface objects that are required to implement the interface.
8. Develop a procedural representation of the user's interaction with the interface. (e.g. UML diagrams—activity and sequence diagram, include interface objects, operations and update your previous diagrams)
9. Develop a behavioral representation of the interface. (e.g. UML state diagram for user interface)
10. Describe the interface layout for each state. Relate each state with the layout and screen images
11. Refine and review the interface design model.

Aesthetic Design

- An artistic endeavor that complements the technical aspects of both interface design and content design.
- Without it, a WebApp may be useful, but unappealing!

Aesthetic Design

Some principles:

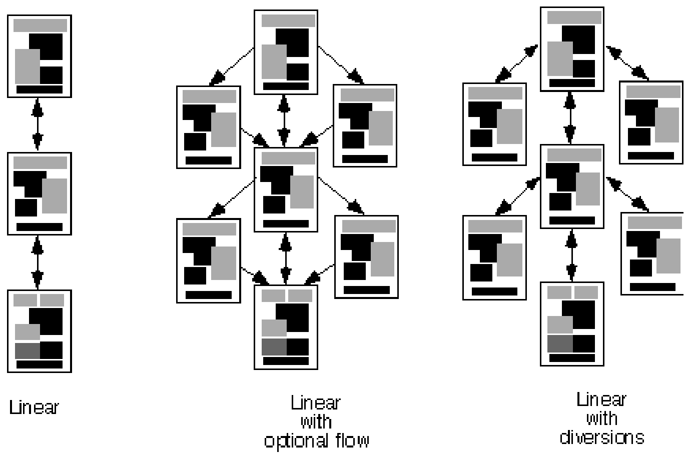
- Don’t be afraid of white space.
- Emphasize content – got something special, highlight it!
- Organize layout elements from top-left to bottom right.
- If you use photos, make them small format with the option to enlarge.
- Don’t extend your real estate with the scrolling bar.
- Consider resolution and browser window size when designing layout.
- Use a cascading style sheet (CSS) for consistency.

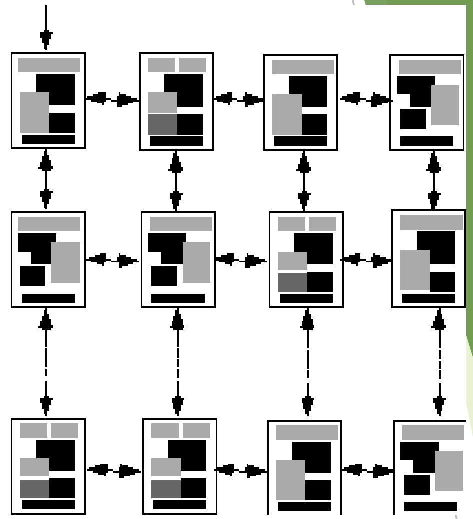
Information Architecture

- *“The structural design of an information space to facilitate task completion and intuitive access to content”*
- An Information architecture might be as simple as a site map.
  - Which represents the navigation structure.
- Or how the information is structured for the WebApp and how users might interact with that structure.
- There is a number of ways in which we can represent the architecture of the information and content of our Web App.

Information Structures

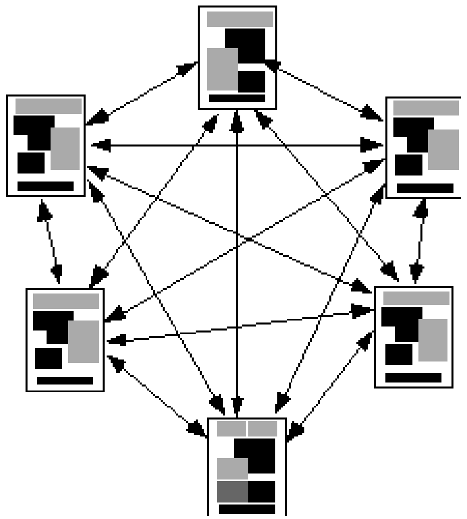
Linear structure

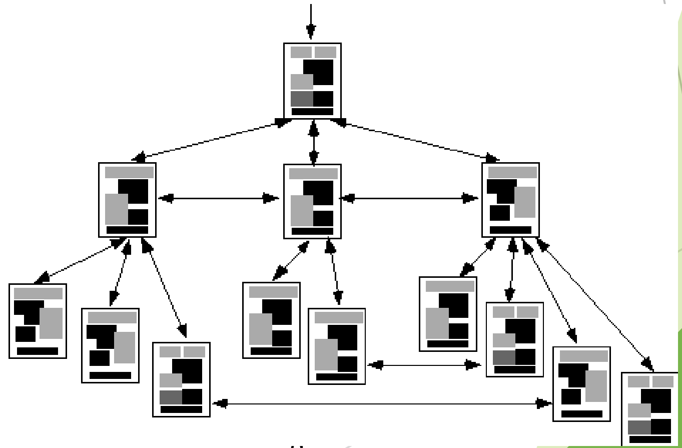




Grid structure

Network structure



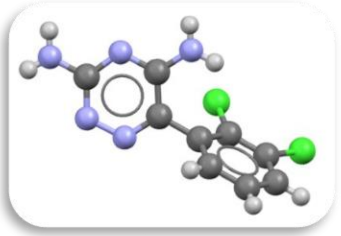


Hierarchical structure

Information Structures

- Linear structure: are encountered when we have a predictable sequence of interactions, although some variation or diversion is common.
- Matrix (or grid) structure: can be applied when WebApp content is organized categorically in two or more dimensions. It is useful only when highly regular content is encountered.
- Hierarchical structure: the most common WebApp architecture and is used to reflect natural information taxonomies.
- Network (or graph) structure: very effective in cross-connecting related aspects within a WebApp and allowing substantial navigation flexibility.

*Also, the architectural structures can be **combined** to form more complex structures.*



What Makes a Good Structure

Fan-out: a measure of the width of the navigation structure below a single node
(i.e. the number of outward links from a single node).

- The appropriate 'Fan-out' of the hierarchical structure should related to the:
  - Complexity of the WebApp options and how distinct the choices are.
- Avoid creating structures that are more than **three** to **four** layers deep.
- The hierarchy should be designed in a way that ensures that the categories are as individual as possible.

Blueprints: Adding Details to a Structure

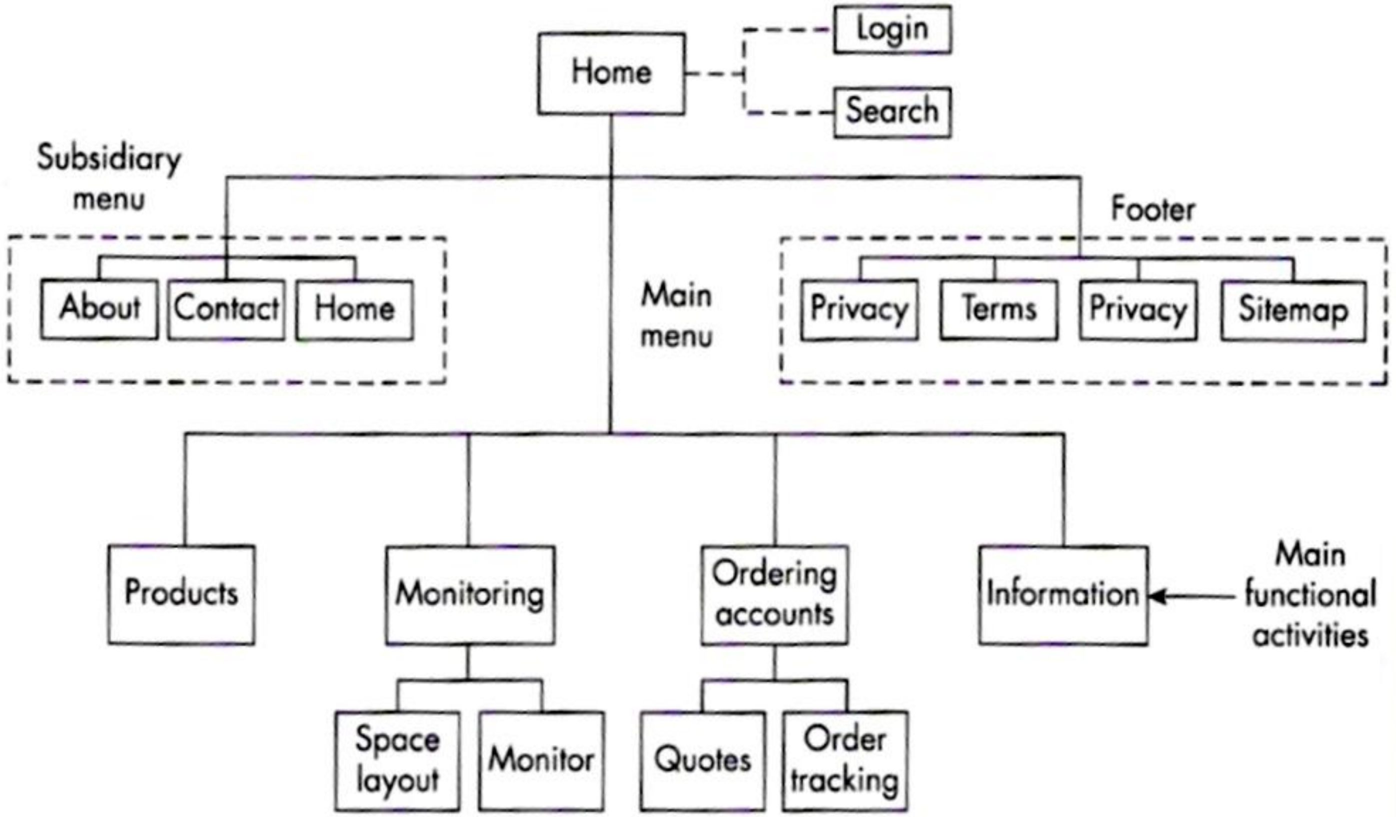
- Identifying the characteristics of the information architecture provides you with a starting point for information design.
- We can use a blueprint to show how the various content objects map into a specific information structure.
- A blueprint often looks like a site map. However, it also include additional information that aids in interpretation of the design, such as:
  - Whether content is dynamic or static.
  - Whether content is personalized for individual users.
  - What content objects are mapped to which Web pages.
  - What navigational paths will address given tasks.

What Form Does a Blueprint Take?

- The format will be defined by what characteristics we are trying to capture!
- A initial high-level blueprint might focus on the overall information structure.
- A second iteration might elaborate on structural information and provide insight into the nature of a search interface and how it allows access to content.
- A third iteration might emphasize
  how can we personalise our content.

Example

A preliminary information structure:



Which structure is this?

Example (cont'd)

A high-level blueprint:

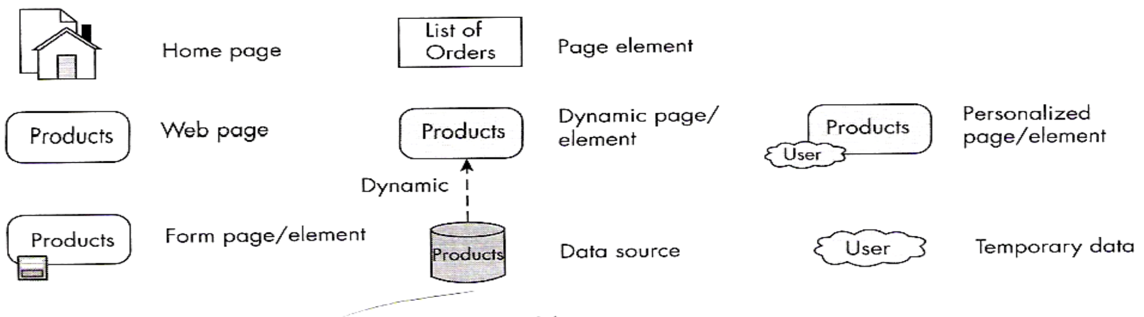
Dark cylinders = data sources

Red icons = dynamically constructed pages.

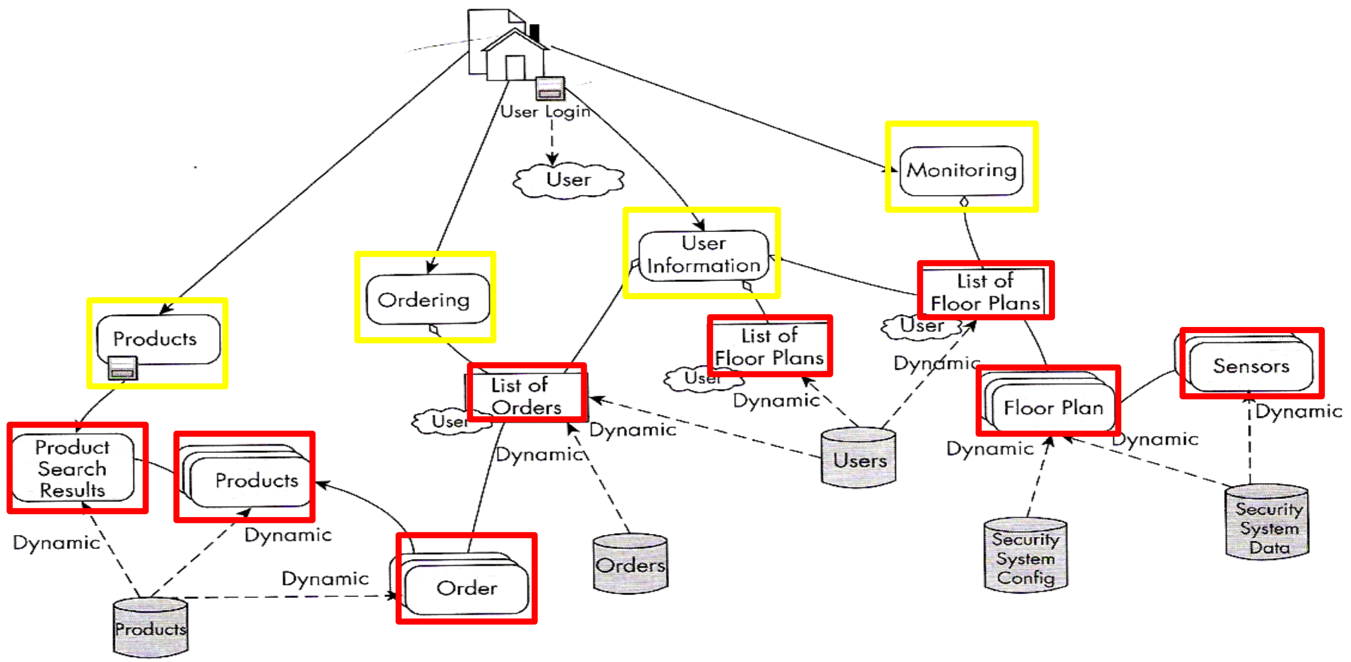
Yellow = our static main pages

25

(a) Example notation for use in blueprints. (b) High-level blueprint for SafeHomeAssured.com.



(a)



(b)

Wireframe Models

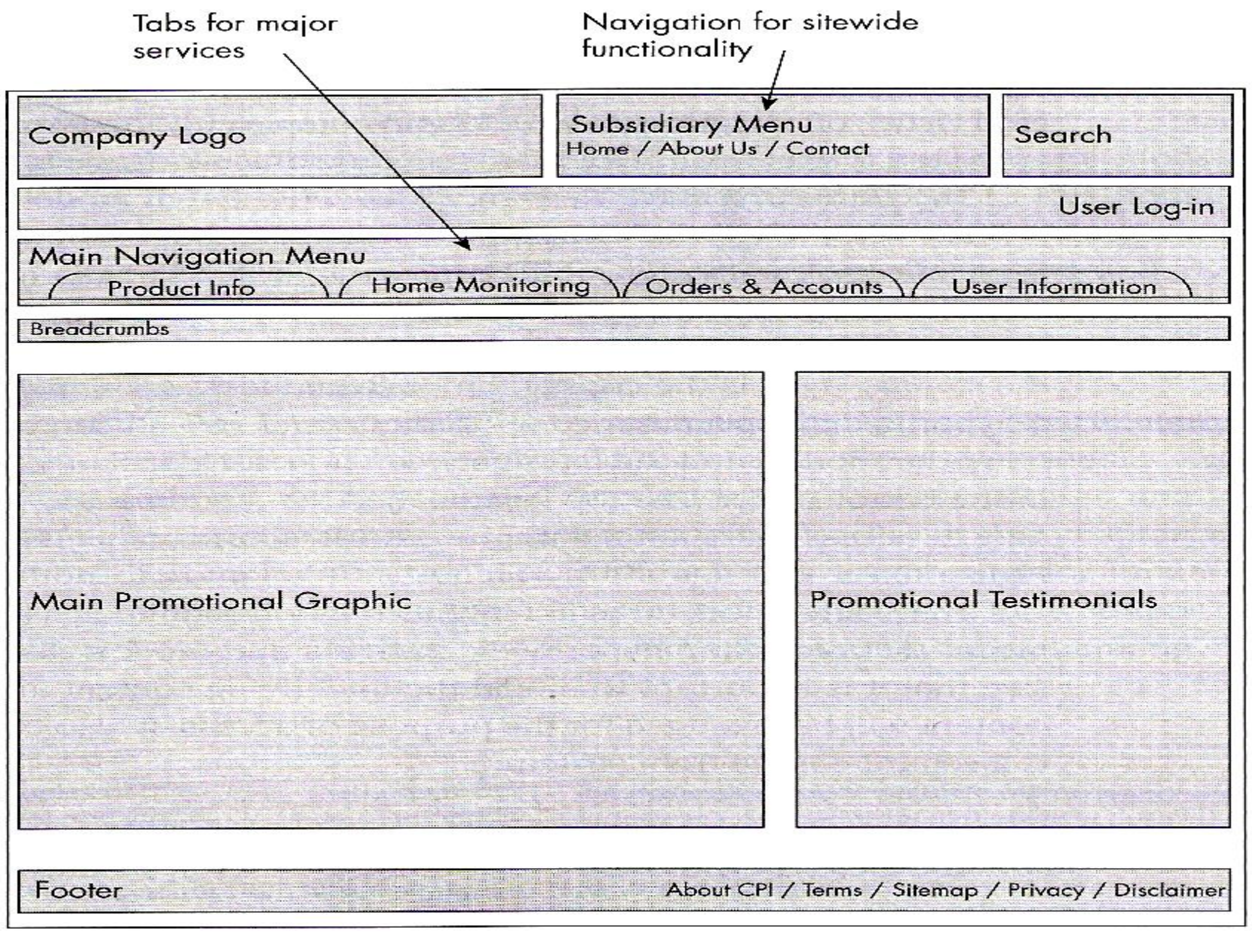
• Blueprints: are just one element of a design model for information structures.

• Wireframes: serve a parallel purpose in describing how an individual pages (or pages) within a WebApp might look conceptually.

• A wireframe does not depict the graphic design of a page.

• It does capture the core information and navigation elements that should be present on the page and the approximate arrangement of these elements.

A Wireframe Example

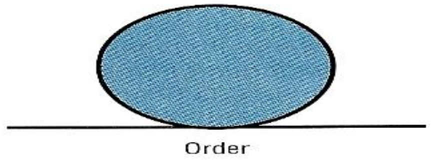


Design class: Class responsibilities and collaboration (CRC) modelling (do it in sprint planning stage)

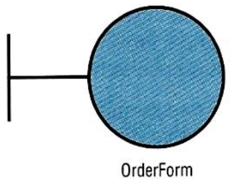
- For design class diagram with Blue, fish and black level use cases
- Sometime, with only fish and black level use cases
- Analysis classes have "responsibilities"
  - *Responsibilities* are the attributes and operations encapsulated by the class
- Analysis classes collaborate with one another
  - *Collaborators* are those classes that are required to provide a class with the information needed to complete a responsibility.
  - In general, a collaboration implies either a request for information or a request for some action.
  - [Ref: 2]

CRC modelling ([Ref: 2])

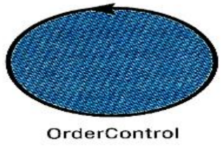
- Entity classes, also called model or business classes, are extracted directly from the statement of the problem (e.g., FloorPlan and Sensor).



- Boundary classes are used to create the interface (e.g., interactive screen or printed reports) that the user sees and interacts with as the software is used.



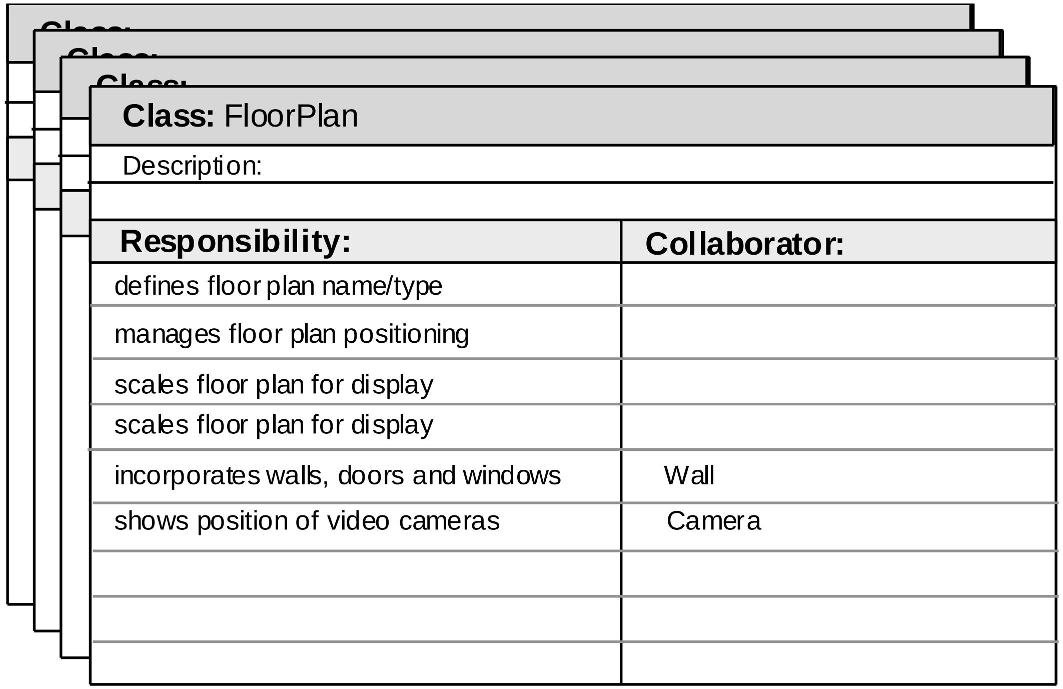
- Controller classes manage a "unit of work" from start to finish. That is, controller classes can be designed to manage
  - the creation or update of entity objects; the instantiation of boundary objects as they obtain information from entity objects; complex communication between sets of objects; validation of data communicated between objects or between the user and the application.



CRC (Ref:2) Responsibilities and Collaborations

- Responsibilities:
  - System intelligence should be distributed across classes to best address the needs of the problem
  - Each responsibility should be stated as generally as possible
  - Information and the behavior related to it should reside within the same class
  - Information about one thing should be localized with a single class, not distributed across multiple classes.
  - Responsibilities should be shared among related classes, when appropriate.
- Collaborations
  - Classes fulfill their responsibilities in one of two ways:
    - A class can use its own operations to manipulate its own attributes, thereby fulfilling a particular responsibility, or
    - a class can collaborate with other classes.
  - Collaborations identify relationships between classes
  - Collaborations are identified by determining whether a class can fulfill each responsibility itself

CRC card (Ref:2) (do it in sprint planning stage)



Class: FloorPlan

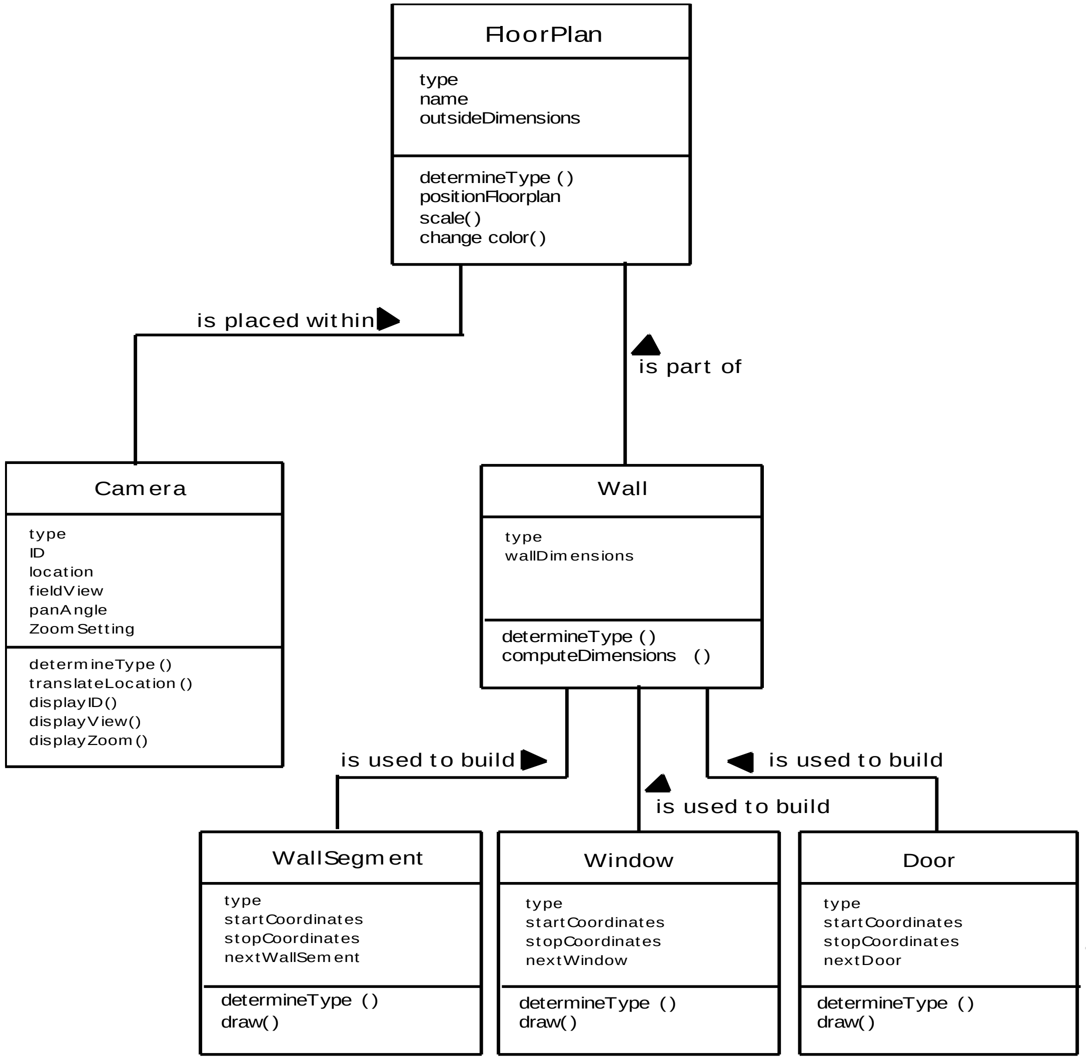
Description:

Responsibility:	**Collaborator:**
defines floor plan name/type	
manages floor plan positioning	
scales floor plan for display	
scales floor plan for display	
incorporates walls, doors and windows	Wall
shows position of video cameras	Camera

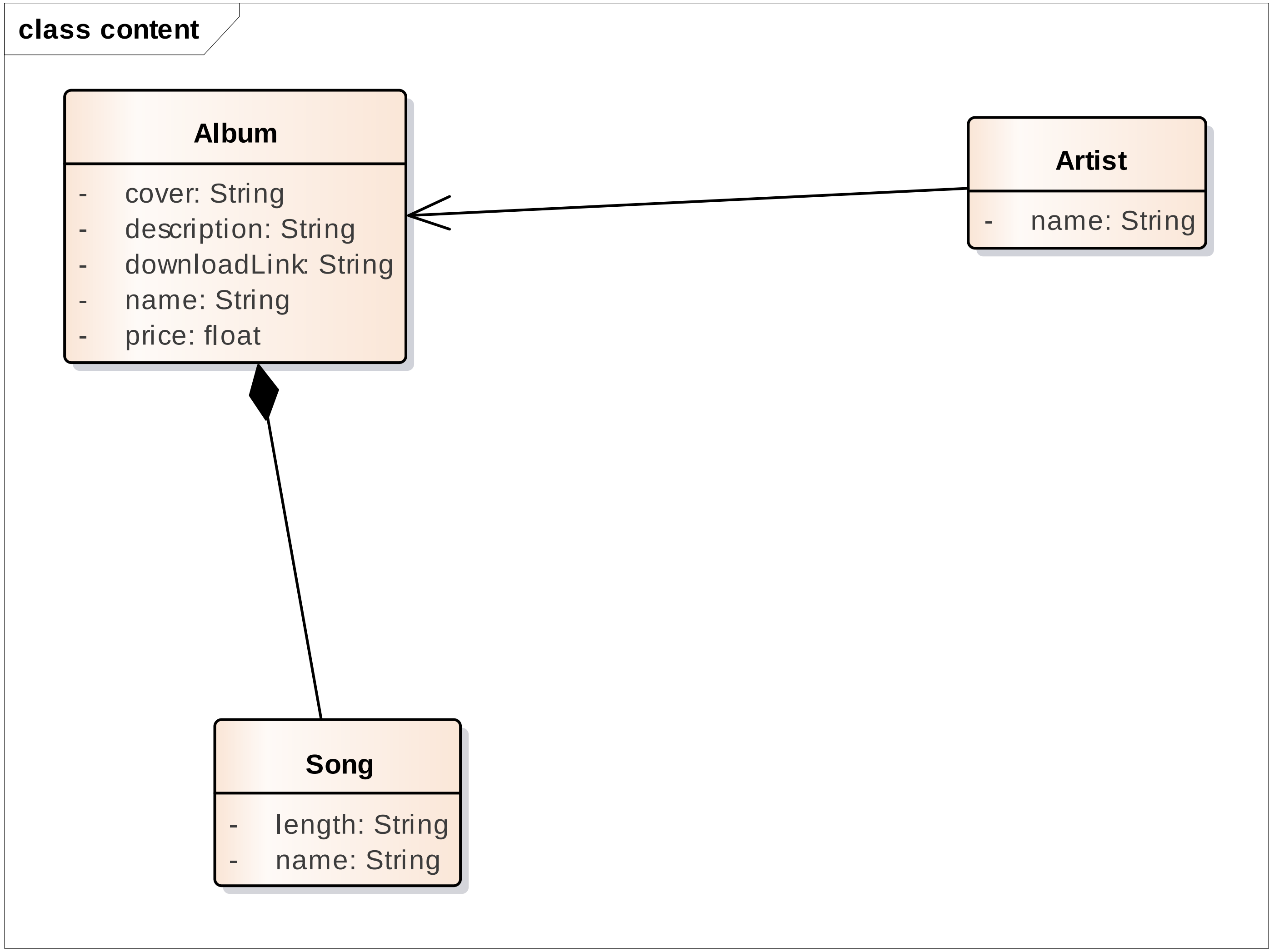
Identifying analysis classes using CRC (do it in sprint planning stage)

1. Select a use case, use problem domain class to identify a responsible class for this use case
2. Select a controller class from whom this object receive first message and handle system's internal process, you may need more than for other use cases
3. Create CRC cards by performing textual analyses from this use-cases description and role playing of this use case
4. Role play of this use case by thinking about what happens and which is helpful.
   I. Role playing involves thinking of your self as one of the instances (the role) and walking through what you would do.
   II. You can even have several people and let each of them play a class as you walk through a use case. It becomes a performance of the use case.
5. In the CRC card left side, write the actions/responsibility as the methods for this class, piece of information as attributes.
6. As you progress in role play, you may identify that this class needs information from other classes, need action to be completed. Write their **class-name (attribute name) format** on right side of CRC
7. Update the CRC cards of the other class or create new CRC cards for other class as you identify
8. Mention in those CRCs about other classes' responsibilities and attributes as you role play of this use-case
9. **Repeat 1-9 for all other use cases**
10. Review the structural model for missing and/or unnecessary classes, attributes, operations, and relationships.

CRC card (Ref:2)/Design Class diagram (do it in sprint planning stage)



Example content class model



Readings

R. S. Pressman and D. Lowe: *Web Engineering, A Practitioner's Approach,* McGraw-Hill, 2009*.*

- Chapter 8: WebApp Design
- Chapter 9: Interaction Design

(concentrate on the topics covered in the lecture)

Papers and other reading materials in "Week 5 Readings" folder on CloudDeakin.

- **## Essential Scrum—Kenneth S. Rubin**
- **## The agile age-Managing projects effectively using agile scrum---Brian vanderjack**