

Chapters 5-6-7

Requirements Engineering Highlights

Chapter 05:

Quality Function Deployment

- **Function deployment** determines the “value” (as perceived by the customer) of each function required of the system
- **Information deployment** identifies data objects and events
- **Task deployment** examines the behavior of the system
- **Value analysis** determines the relative priority of requirements

Planning Principles (from Chapter 04)

- **Principle #1.** *Understand the scope of the project.* It's impossible to use a roadmap if you don't know where you're going. Scope provides the software team with a destination.
- **Principle #2.** *Involve the customer in the planning activity.* The customer defines priorities and establishes project constraints.
- **Principle #3.** *Recognize that planning is iterative.* A project plan is never engraved in stone. As work begins, it very likely that things will change.
- **Principle #4.** *Estimate based on what you know.* The intent of estimation is to provide an indication of effort, cost, and task duration, based on the team's current understanding of the work to be done.

Planning Principles (from Chapter 04)

- **Principle #5.** *Consider risk as you define the plan.* If you have identified risks that have high impact and high probability, contingency planning is necessary.
- **Principle #6.** *Be realistic.* People don't work 100 percent of every day.
- **Principle #7.** *Adjust granularity as you define the plan.* Granularity refers to the level of detail that is introduced as a project plan is developed.
- **Principle #8.** *Define how you intend to ensure quality.* The plan should identify how the software team intends to ensure quality.
- **Principle #9.** *Describe how you intend to accommodate change.* Even the best planning can be obviated by uncontrolled change.
- **Principle #10.** *Track the plan frequently and make adjustments as required.* Software projects fall behind schedule one day at a time.

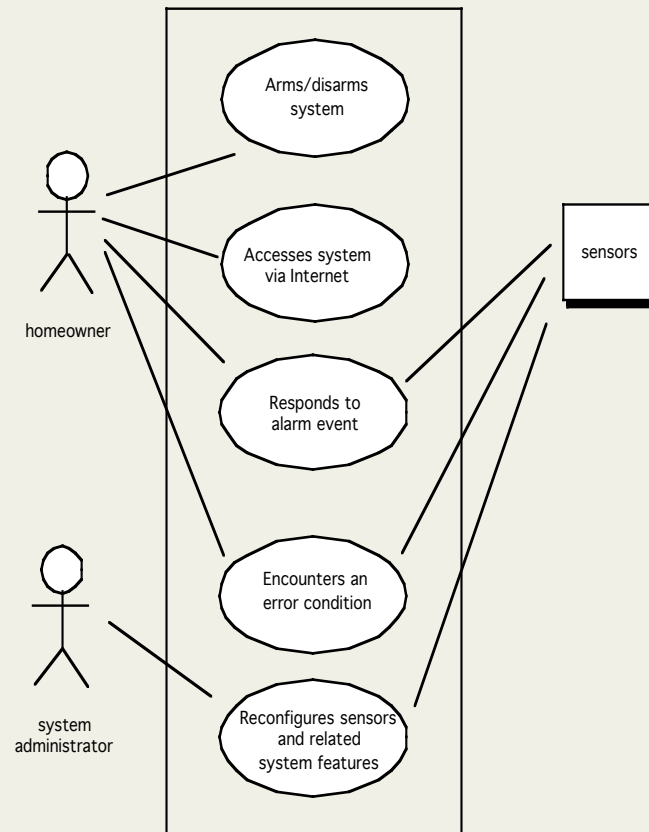
Requirements vs Expectations

- Customers “expect” the product to do certain things
 - Some are clearly stated expectations
 - Others are assumed but not stated
 - Figuring out the assumptions is where the real challenge lies
- Expectations lead to requirements specification
 - Inputs + action on inputs → outputs
 - Each has consequences for design & implementation
 - Some requirements may need to be re-thought, re-defined, or postponed
- This first “negotiation” with the customer must be managed with an eye toward realistic production
 - Get the requirements
 - Understand the implications
 - Assess feasibility
 - Then return with a proposal (release plan)
 - Validate plan

Chapter 05: UML & Use-Cases

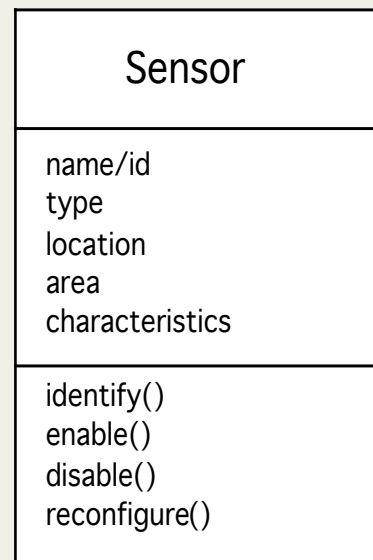
- UML (Unified Modeling Language) used to model many kinds of systems
- Based on concept of user interacting with system – hardware and software
- Use Case: A collection of user scenarios that describe the thread of usage of a system
- Each scenario is described from the point-of-view of an “actor” —a person or device that interacts with the software in some way
- Each scenario answers the following questions:
 - Who is the primary actor, the secondary actor (s)?
 - What are the actor’s goals?
 - What preconditions should exist before the story begins?
 - What main tasks or functions are performed by the actor?
 - What extensions might be considered as the story is described?
 - What variations in the actor’s interaction are possible?
 - What system information will the actor acquire, produce, or change?
 - Will the actor have to inform the system about changes in the external environment?
 - What information does the actor desire from the system?
 - Does the actor wish to be informed about unexpected changes?

Chapter 05: UML Use-Case Diagram

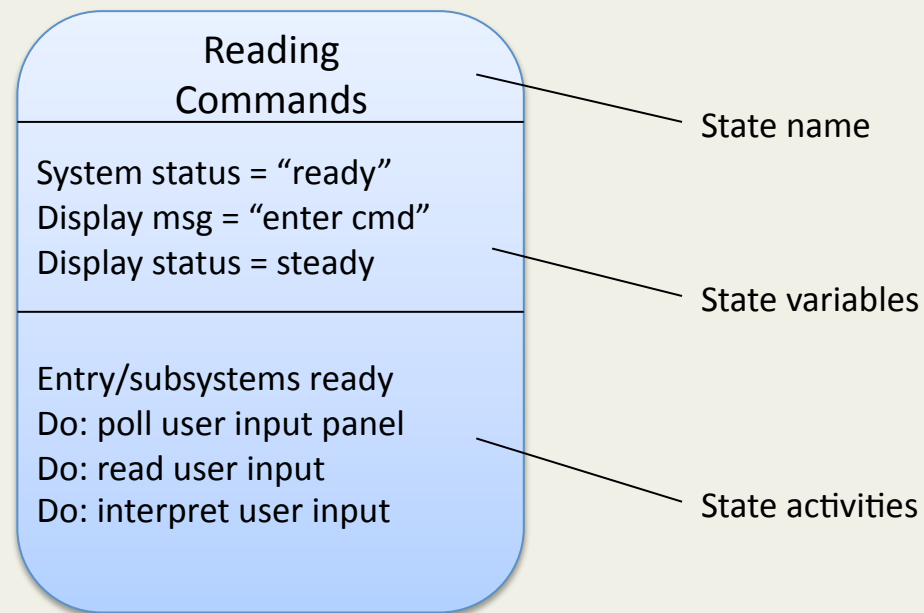


Chapter 05: UML Class Diagram

From the *SafeHome* system ...



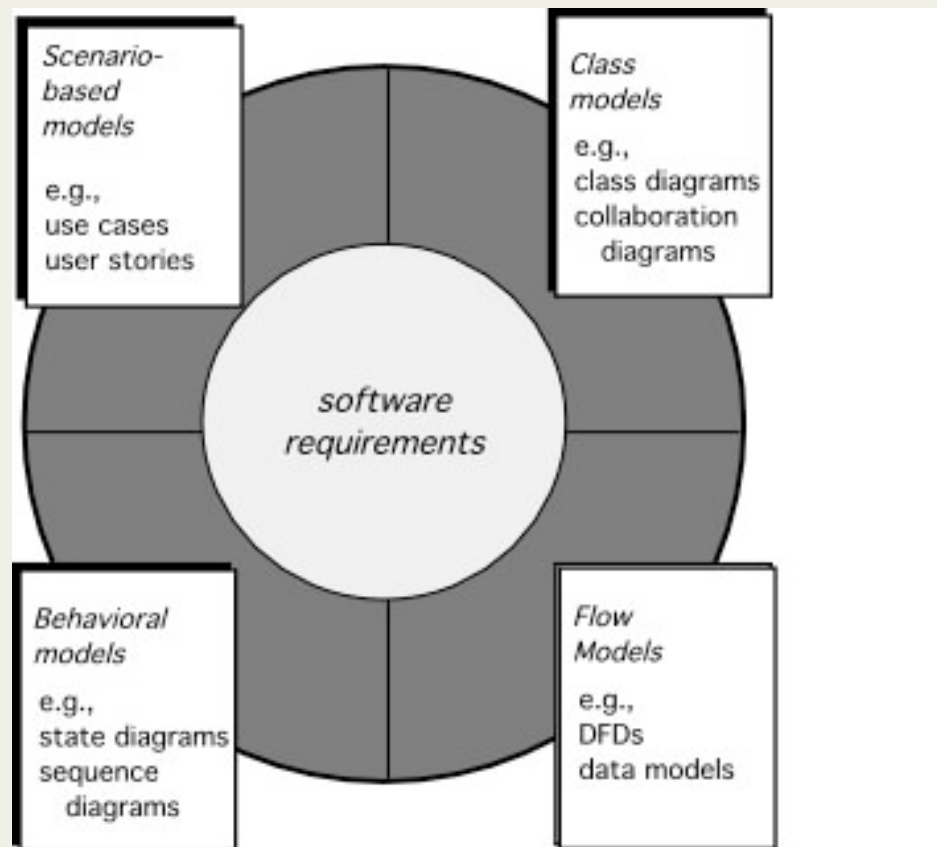
Chapter 05: UML State Diagram



Chapter 06: Modeling the Requirements

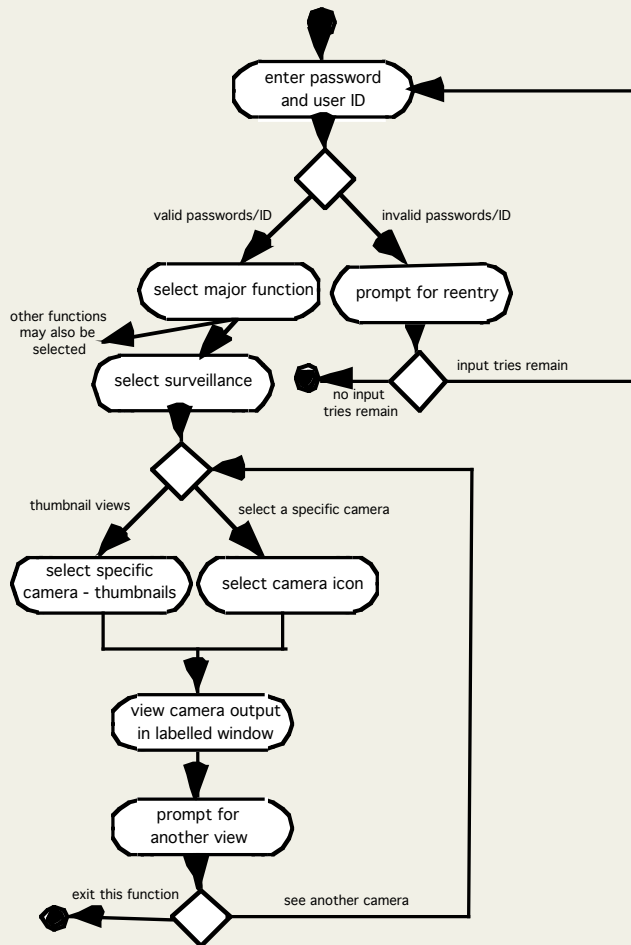
- Scenario-based models
- Data models
- Class-oriented models
- Flow-oriented models
- Behavioral models

DFD: Data Flow Diagram

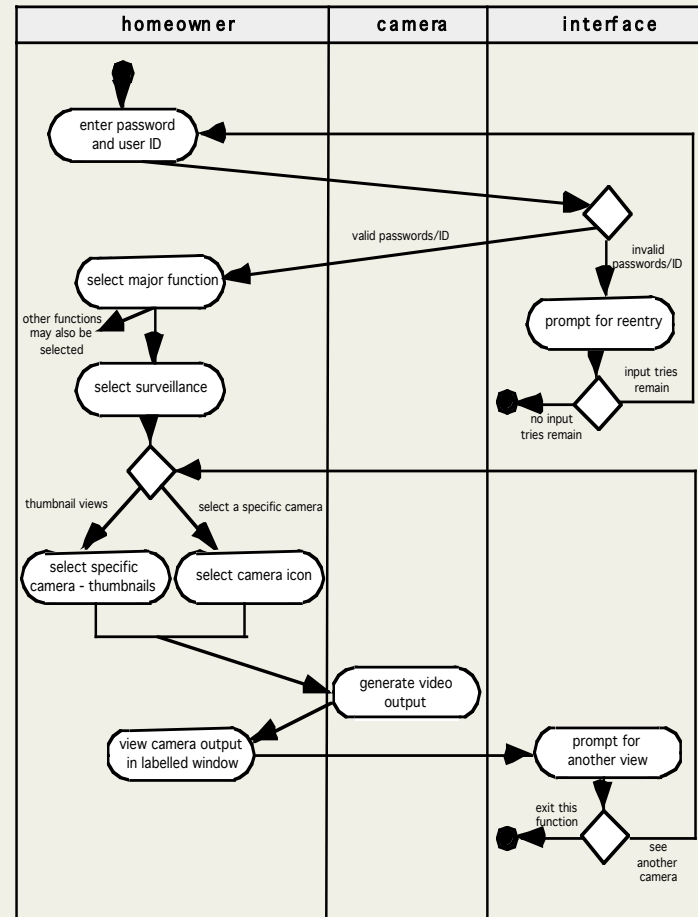


Chapter 06: UML Activity & Swimlane Diagrams

Activity Diagram



Swimlane Diagram



Chapter 06: Data Modeling

- examines data objects independently of processing
- focuses attention on the data domain
- creates a model at the customer's level of abstraction
- indicates how data objects relate to one another

Chapter 06: What is a Data Object?

- a representation of almost any composite information that must be understood by software.
 - *composite information*—something that has a number of different properties or attributes
- can be an **external entity** (e.g., anything that produces or consumes information), **a thing** (e.g., a report or a display), **an occurrence** (e.g., a telephone call) **or event** (e.g., an alarm), **a role** (e.g., salesperson), **an organizational unit** (e.g., accounting department), **a place** (e.g., a warehouse), or **a structure** (e.g., a file).
- The description of the data object incorporates the data object and all of its attributes.
- A data object encapsulates data only—there is no reference within a data object to operations that act on the data.

Chapter 06: Data Objects and Attributes

A data object contains a set of attributes that act as an aspect, quality, characteristic, or descriptor of the object

object: automobile

attributes:

make

model

body type

price

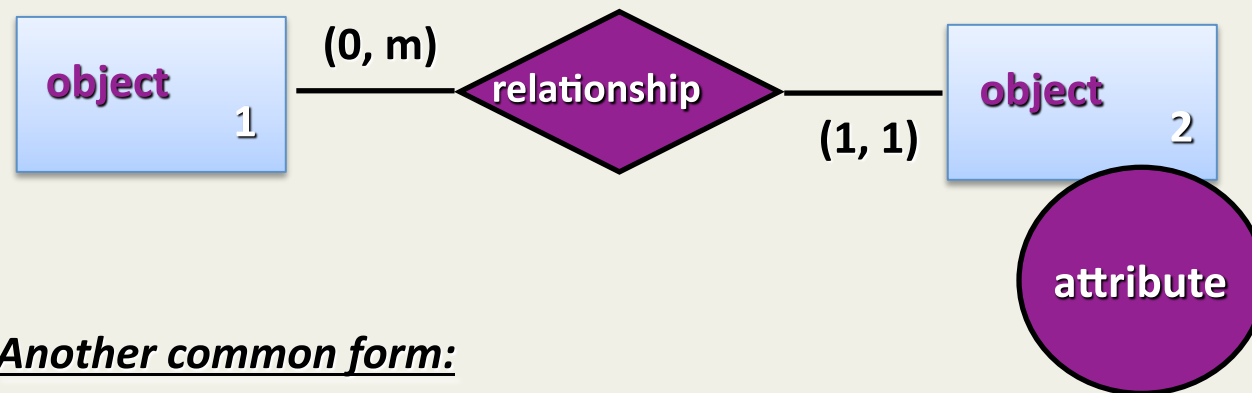
options code

Chapter 06: What is a Relationship?

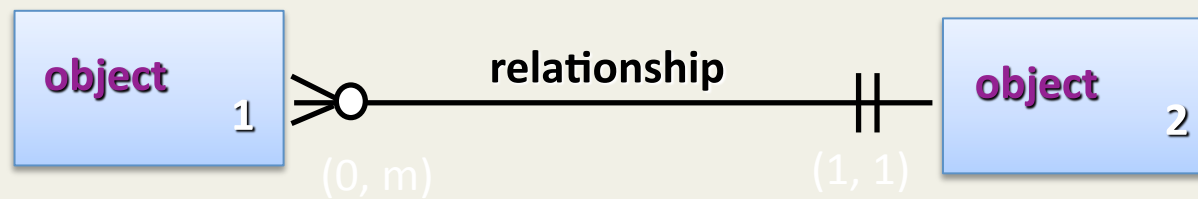
- Data objects are connected to one another in different ways.
 - A connection is established between **person** and **car** because the two objects are related.
 - A person *owns* a car
 - A person *is insured to drive* a car
- The relationships *owns* and *insured to drive* define the relevant connections between **person** and **car**.
- Several instances of a relationship can exist
- Objects can be related in many different ways

Chapter 06: ERD Notation (Entity-Relationship Diagram)

One common form:



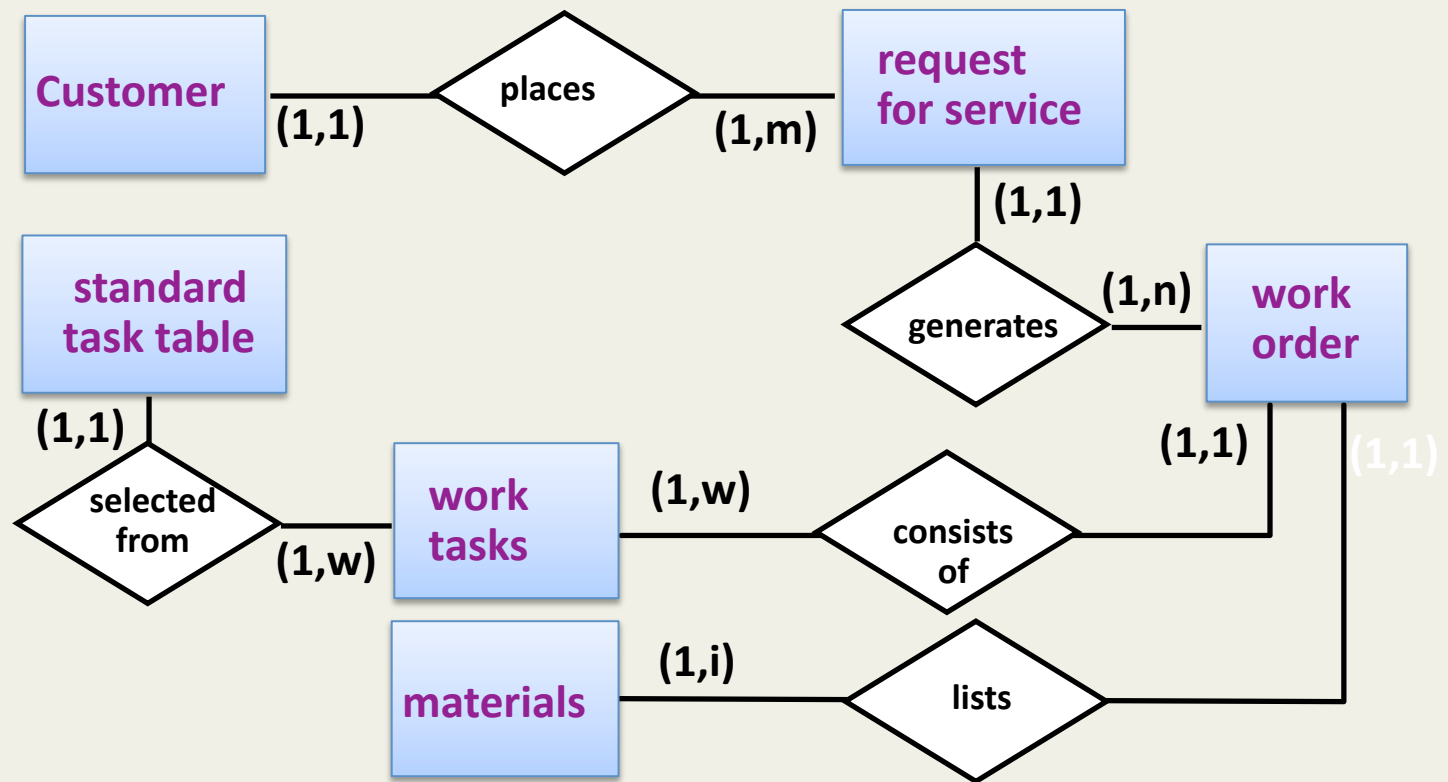
Another common form:



Chapter 06: Building an ERD

- *Level 1*—model all data objects (entities) and their “connections” to one another
- *Level 2*—model all entities and relationships
- *Level 3*—model all entities, relationships, and the attributes that provide further depth

Chapter 06: An ERD Example

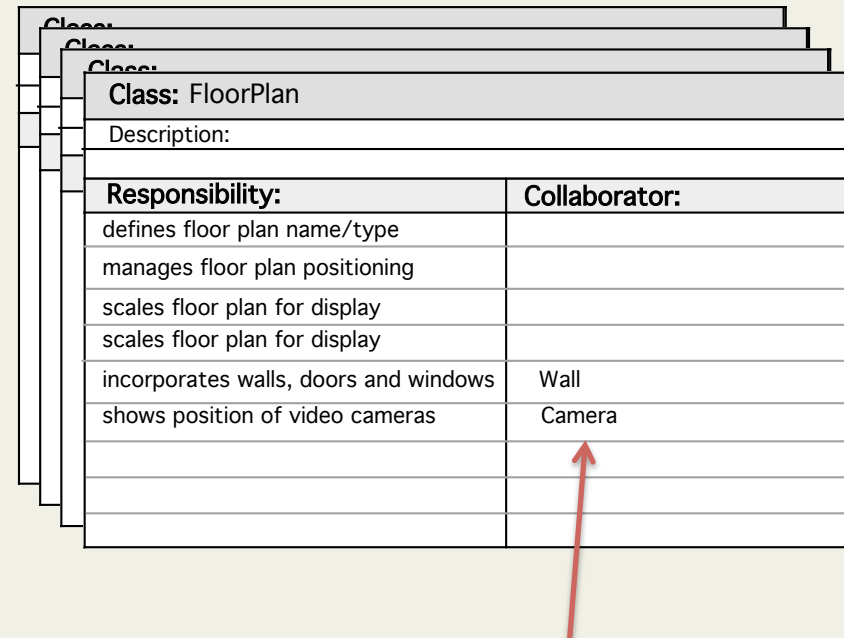


Chapter 06: Class-Based Modeling

- Class-based modeling represents:
 - **objects** that the system will manipulate
 - **operations** (also called methods or services) that will be applied to the objects to effect the manipulation
 - **relationships** (some hierarchical) between the objects
 - **collaborations** that occur between the classes that are defined.
- The elements of a class-based model include classes and objects, attributes, operations, CRC models, collaboration diagrams and packages.

Chapter 06: Class-Responsibility-Collaborator Modeling (CRC)

- *Class-responsibility-collaborator (CRC) modeling* [Wir90] provides a simple means for identifying and organizing the classes that are relevant to system or product requirements. Ambler [Amb95] describes CRC modeling in the following way:
 - A CRC model is really a collection of standard index cards that represent classes. The cards are divided into three sections. Along the top of the card you write the name of the class. In the body of the card you list the class responsibilities on the left and the collaborators on the right.

The diagram shows a stack of index cards representing CRC modeling. The top card is titled 'Class: FloorPlan' and is divided into three sections: a description section, a responsibility section, and a collaborator section. The responsibility section lists several tasks, and the collaborator section lists 'Wall' and 'Camera'. A red arrow points to the 'Camera' collaborator.

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

Other classes collaborating

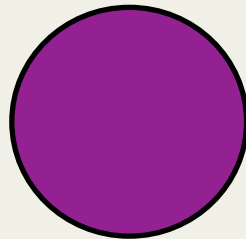
Chapter 07: Modeling Strategies

- *structured analysis* considers data and the processes that transform the data as separate entities.
 - Data objects are modeled in a way that defines their attributes and relationships.
 - Processes that manipulate data objects are modeled in a manner that shows how they transform data as data objects flow through the system.
- *object-oriented analysis* focuses on
 - the definition of classes and
 - the manner in which they collaborate with one another to effect customer requirements.
- Data Flow Diagrams
 - Represents how data objects are transformed as they move through the system

Chapter 07: Flow Modeling Notation



external entity



process

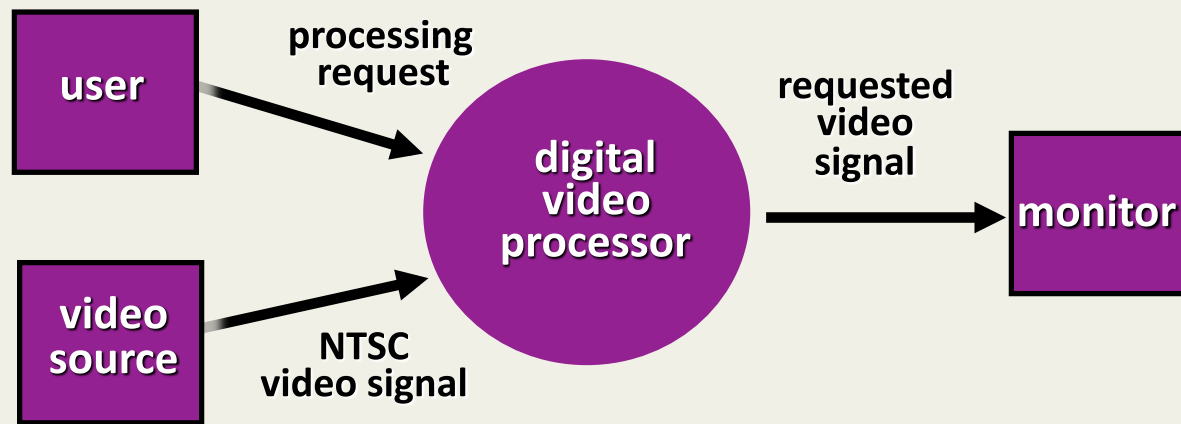


data flow



data store

Chapter 07: Data Flow Diagram Example



Chapter 07: Requirements Modeling for WebApps

Content Analysis. The full spectrum of content to be provided by the WebApp is identified, including text, graphics and images, video, and audio data. Data modeling can be used to identify and describe each of the data objects.

Interaction Analysis. The manner in which the user interacts with the WebApp is described in detail. Use-cases can be developed to provide detailed descriptions of this interaction.

Functional Analysis. The usage scenarios (use-cases) created as part of interaction analysis define the operations that will be applied to WebApp content and imply other processing functions. All operations and functions are described in detail.

Configuration Analysis. The environment and infrastructure in which the WebApp resides are described in detail.

The Configuration Model

- Server-side
 - Server hardware and operating system environment must be specified
 - Interoperability considerations on the server-side must be considered
 - Appropriate interfaces, communication protocols and related collaborative information must be specified
- Client-side
 - Browser configuration issues must be identified
 - Testing requirements should be defined

Navigation Modeling-I

- Should certain elements be easier to reach (require fewer navigation steps) than others? What is the priority for presentation?
- Should certain elements be emphasized to force users to navigate in their direction?
- How should navigation errors be handled?
- Should navigation to related groups of elements be given priority over navigation to a specific element.
- Should navigation be accomplished via links, via search-based access, or by some other means?
- Should certain elements be presented to users based on the context of previous navigation actions?
- Should a navigation log be maintained for users?

Navigation Modeling-II

- Should a full navigation map or menu (as opposed to a single “back” link or directed pointer) be available at every point in a user’s interaction?
- Should navigation design be driven by the most commonly expected user behaviors or by the perceived importance of the defined WebApp elements?
- Can a user “store” his previous navigation through the WebApp to expedite future usage?
- For which user category should optimal navigation be designed?
- How should links external to the WebApp be handled? overlaying the existing browser window? as a new browser window? as a separate frame?

Fact/Fallacy Tidbit

- Fact 26
The list of “derived requirements” can be 50x longer than the list of original requirements
- Discussion
 - Requirements inform the design; the design informs the solution; the solution creates new requirements
 - Complexity may increase as well
 - Requirements traceability is affected
 - Traceability involves mapping a customer requirement to each part of the design/code/test/documentation
 - Sometimes used for code analysis – identifying dangling code, for example (i.e., code that no longer meets a requirement)
 - Design “consequences” (implicit requirements) aren’t tied directly to an original customer requirement, so mapping is unclear at best
 - Each phase of the project and addition of further implicit requirements adds to the traceability problem; no just a simple linked-list problem but linked-lists of linked-lists – extremely complex

From Robert Glass, “Facts & Fallacies of Software Engineering”