

# Analysis

CSE 308: Software Engineering

# Topics Of Discussion

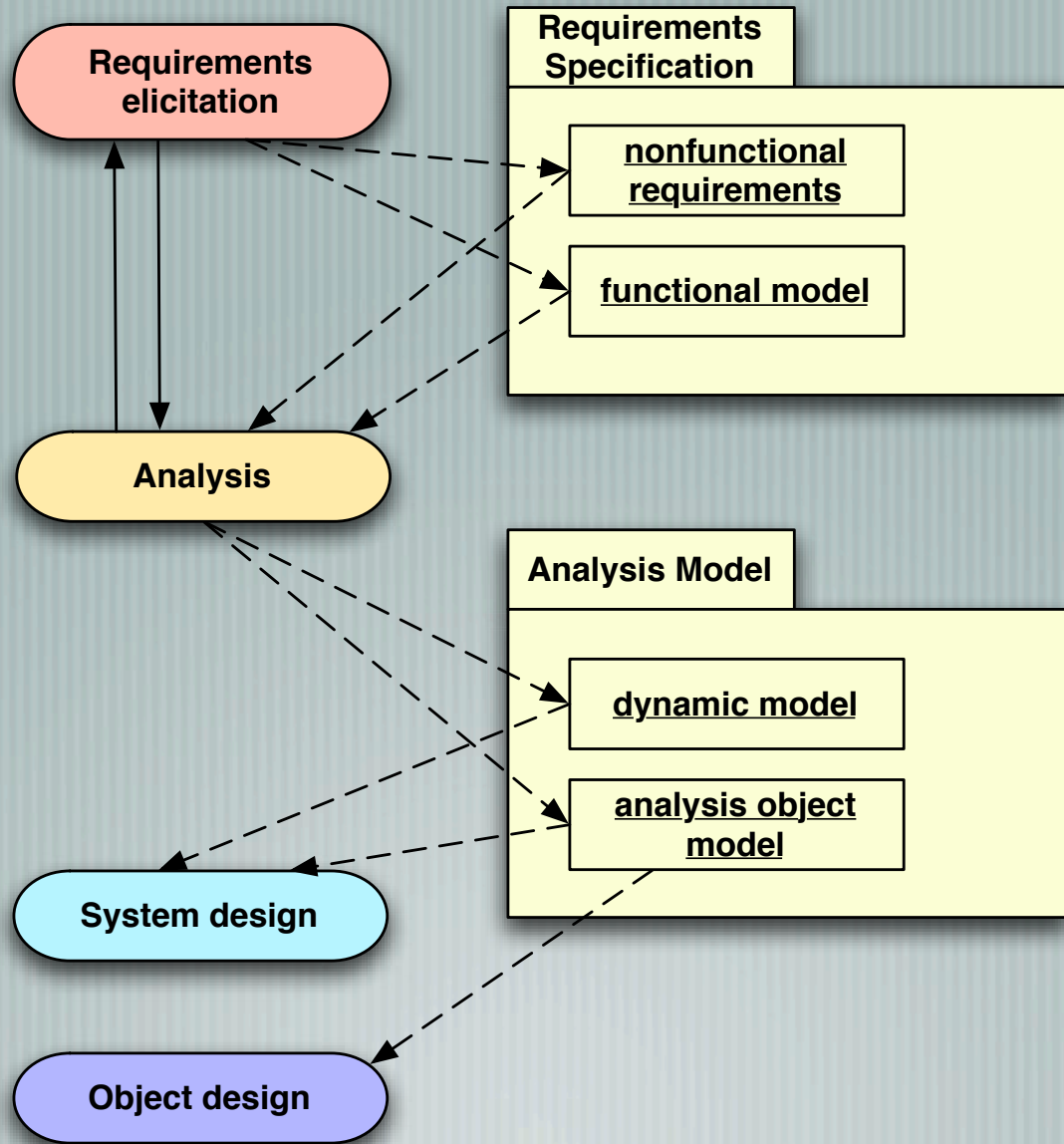
- [ Analysis Concepts
- [ Analysis Activities
- [ Managing Analysis

# Analysis

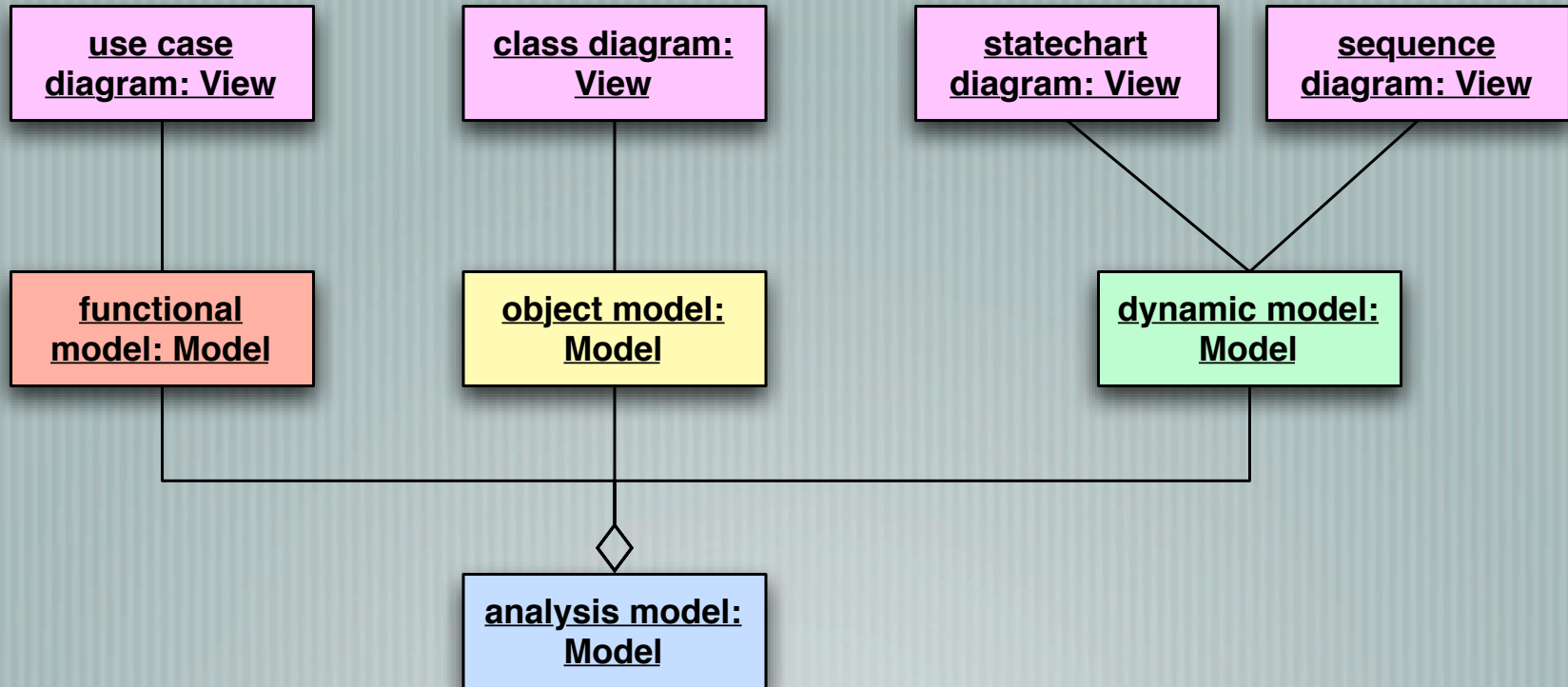
- [ Process of formalizing system requirements specification
- [ Results in a model that is:
  - correct
  - complete
  - consistent
  - unambiguous

# Analysis Overview

- [ Analysis focuses on producing a correct, complete, consistent, verifiable model of a system
- [ Analysis adds structure and formalization to requirements
  - this may lead to new insights and uncover errors
- [ Early formalization forces us to identify and resolve problems early in the development process



# Analysis Concepts



# Analysis Model Components

- [ Functional model
  - use cases and scenarios
- [ Analysis object model
  - class and object diagrams
- [ Dynamic model
  - statechart and sequence diagrams

# The Analysis Object Model

- [ Focuses on individual concepts that are manipulated by the system, including their properties and relationships
- [ Includes classes, attributes, and operations
- [ “Visual dictionary” of main user-level concepts
  - these are high-level abstractions, not design decisions!

# The Dynamic Model

- [ Focuses on system behavior
  - sequence diagrams represent object interactions
  - statecharts represent the behavior of a single object
- [ Assigns responsibilities to individual classes
  - may also identify new classes, associations, and attributes for the analysis object model
- [ High-level abstraction, like the analysis object model

# Analysis Activities

# Analysis Activities

- [ Purpose: transform use cases/scenarios into analysis model
- [ Activities include:
  - object identification
  - use case-object mapping
  - interaction modeling
  - identification of the “3 As”
  - modeling state-dependent behavior and inheritance

# Analysis Activities

— [ Identification of entity, boundary, and control objects

— [ Mapping use cases to objects

— [ Modeling object interactions

— [ Identification of associations, aggregates, and attributes

— [ Modeling state-dependent behavior

— [ Modeling inheritance relationships

— [ Reviewing the analysis model

# Entity, Boundary, Control Objects

- [ Comprise the analysis object model
- [ Entity objects: represent persistent information
- [ Boundary objects: represent actor-system interactions
- [ Control objects: realize use cases
- [ Ex. speed and direction (entity), speedometer (boundary), steering wheel and accelerator pedal (control)

# Entity, Boundary, Control Objects

- [ Entity, boundary, and control modeling distinguishes different (but related) concepts
  - Ex. time vs. a time display
- [ Leads to smaller, more specialized objects
  - separates interface from functionality

**<< entity >>  
Year**

**<< entity >>  
Month**

**<< entity >>  
Day**

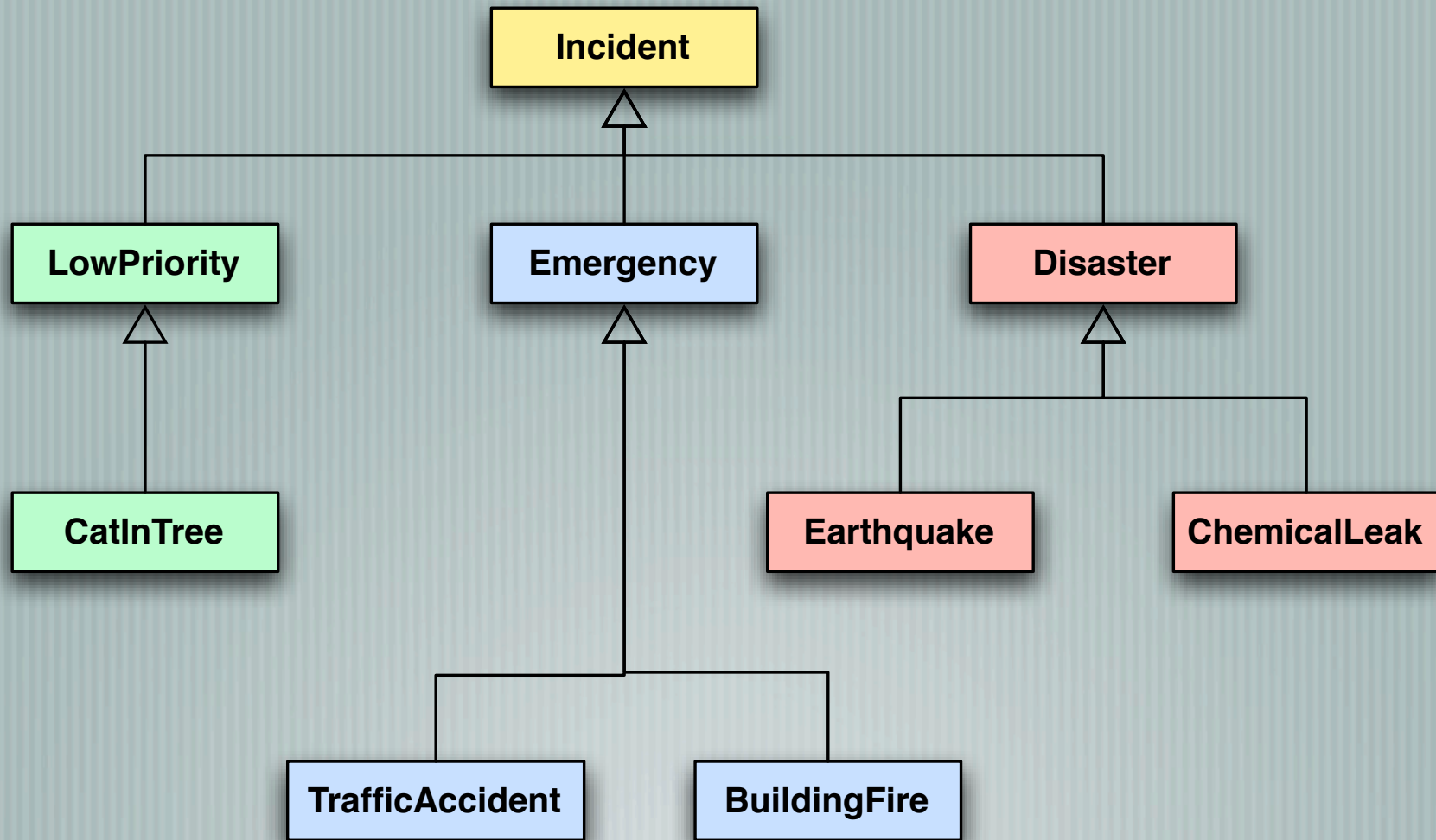
**<< control >>  
ChangeDateControl**

**<< boundary >>  
Button**

**<< boundary >>  
LCD\_Display**

# Generalization and Specialization

- [ Generalization identifies abstract concepts from lower-level (more detailed) ones
- [ Specialization identifies more specific concepts from abstract (higher-level) ones
- [ Relation to inheritance:
  - inheritance denotes the relationship
  - generalization/specialization identify those relationships



# Identifying Entity Objects

- [ Examine each use case to identify candidate objects
- [ Natural language provides heuristics for identifying objects, etc.
  - Map parts of speech to model components
  - heuristic works well for generating an initial object list
- [ Object model quality depends upon writing style
  - rephrase requirements specification to standardize terms

# Mapping Speech To Models

Part of speech	Model component	Examples
Proper noun	Instance	John, Mary
Common noun	Class	Employee, User
Doing verb	Operation	Creates, selects
Being verb	Inheritance	Is a kind of
Having verb	Aggregation	Has, includes
Modal verb	Constraints	Must be
Adjective	Attribute	Color, age

# Advanced Heuristics

— [ Likely entity objects are:

- terms that must be clarified to understand the use case
- recurring nouns in the use case
- real-world entities that the system must track
- real-world activities that the system must track
- data sources/sinks

# Object Naming

- [ Object names should be unique to promote standard terminology
- [ Select the names used by end users/domain specialists
- [ Each object should have a name and a brief description, plus an optional list of attributes and responsibilities
  - add additional detail as needed after model is stable

# Identifying Boundary Objects

- [ Boundary objects represent the system interface with actors
- [ Boundary objects express a coarse version of the user interface
  - details are likely to change as the interface evolves
  - this prevents costly/pointless analysis model updates

# Boundary Object Heuristics

- [ Identify UI controls that are needed to initiate the use case
- [ Identify data entry forms
- [ Identify system notices and messages
  - ex. acknowledgments, warnings
- [ Identify actor terminals to distinguish user interfaces
- [ **DO NOT** model visual aspects of the user interface
- [ **ALWAYS** use end user terms for describing interfaces

# Identifying Control Objects

- [ Control objects are responsible for coordinating boundary and entity objects
  - they usually don't have real-world counterparts
- [ Control objects are tightly tied to use cases
  - ex. collecting and dispatching information

# Control Object Heuristics

- [ Identify one control object per use case
- [ Identify one control object per actor in the use case
- [ A control object's life span should cover the entire use case or an entire user session
  - use case entry and exit conditions must be well-defined

# Analysis Activities

- [ Identify entity/boundary/control objects
- [ **Map use cases to objects**
- [ Model object interactions with CRC cards
- [ Identify associations/aggregates/attributes
- [ Model state-dependent behavior of objects
- [ Model Inheritance relationships among objects

# Mapping Use Cases to Objects

- [ A sequence diagram connects use cases with objects
  - shows how behavior is distributed among objects
- [ Columns represent participating objects
  - leftmost column is initiating actor
- [ Horizontal arrows represent messages
- [ Time proceeds vertically from top to bottom

# More on Sequence Diagrams

- [ Sequence diagrams also depict object lifetimes
  - Pre-existing objects are depicted at the top
  - Created objects are depicted within the diagram with the message << create >>
  - Objects that are destroyed are Xed out when they cease to exist
  - Dashed lines represent an object's lifespan

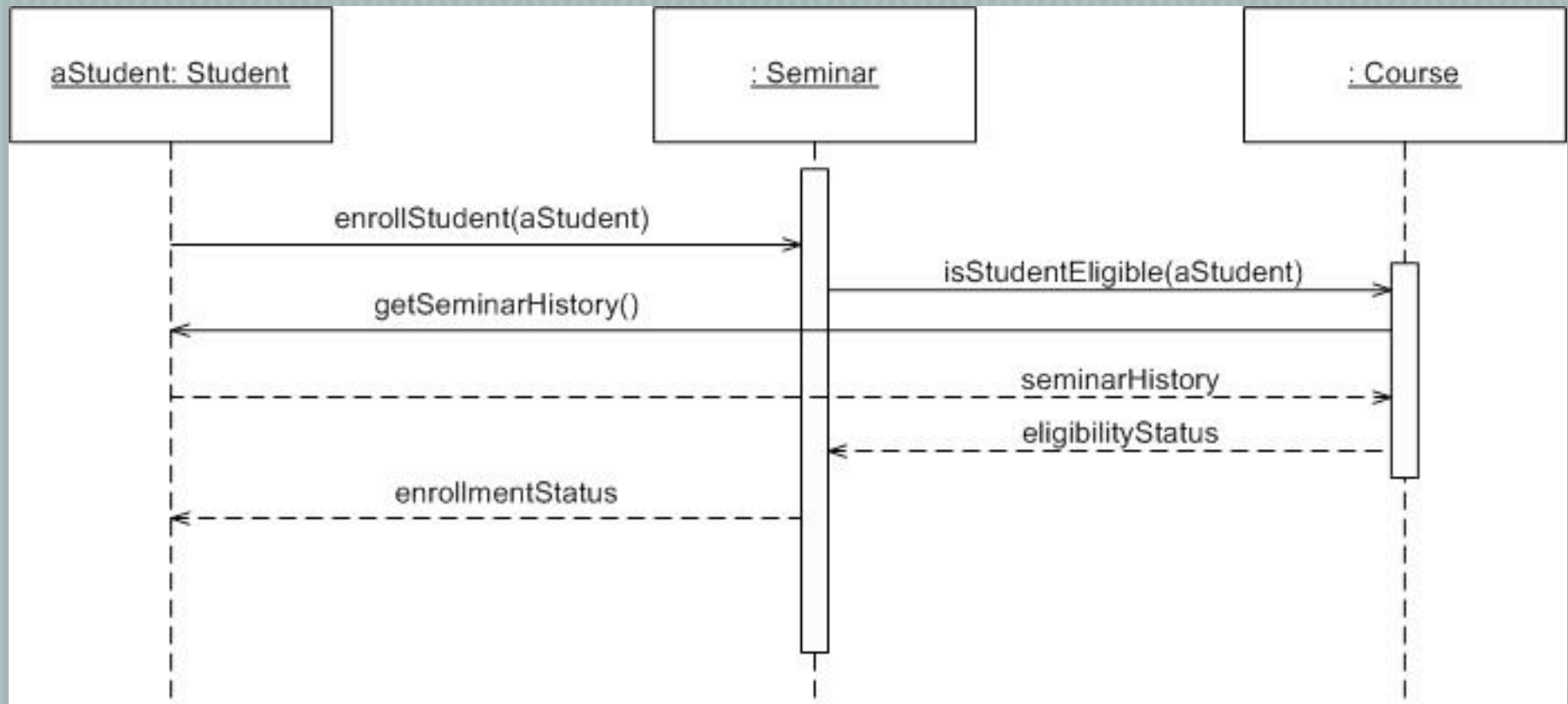
# Sequence Diagrams

- [ Second column generally represents a boundary object
- [ Third column is usually the primary control object
  
- [ Sequence diagrams distribute use case behavior
  - assign responsibilities to objects as operations
  - definitions shared among use cases must be identical

# Sequence Diagram Heuristics

- [ First/second/third column represent initiating actor/  
boundary/control objects
- [ Control objects are created by initiating boundary objects
- [ Boundary objects are created by control objects
- [ Entity objects are accessed by boundary/control objects
- [ Entity objects NEVER access boundary/control objects

# Sequence Diagram Example



Source: <http://www.agilemodeling.com/artifacts/sequenceDiagram.htm>

# Analysis Activities

- [ Identify entity/boundary/control objects
- [ Map use cases to objects
- [ **Model object interactions with CRC cards**
- [ Identify associations/aggregates/attributes
- [ Model state-dependent behavior of objects
- [ Model Inheritance relationships among objects

# Modeling Object Interactions

- [ CRC (class, responsibilities, and collaborators) cards represent objects
- [ Each card contains a class name, its responsibilities (on the left), and the helper classes it needs (on the right)
- [ Support the same activity as sequence diagrams
  - CRC cards are better for groups of developers

# Using CRC Cards

- [ Participants identify classes involved in a scenario
- [ One card per instance is put on the table
- [ Participants assign responsibilities to each class
- [ Cards are modified/pushed aside as the analysis proceeds
  - Cards are never thrown away, so they can be reused

**Class Name**

**Responsibilities**

**Collaborators**

**Student**

**Student number**

**Name**

**Address**

**Phone number**

**Enroll in a seminar**

**Drop a seminar**

**Request transcripts**

**Seminar**

Source: <http://www.agilemodeling.com/artifacts/crcModel.htm>

Enrollment	
Mark(s) received Average to date Final grade Student Seminar	Seminar

Transcript	
""See the prototype"" Determine average mark	Student Seminar Professor Enrollment

Student Schedule	
""See the prototype""	Seminar Professor Student Enrollment Room

Room	
Building Room number Type (Lab, class, ...) Number of Seats Get building name Provide available time slots	Building

Professor	
Name Address Phone number Email address Salary Provide information Seminars instructing	Seminar

Seminar	
Name Seminar number Fees Waiting list Enrolled students Instructor Add student Drop student	Student Professor

Student	
Name Address Phone number Email address Student number Average mark received Validate identifying info Provide list of seminars taken	Enrollment

Building	
Building Name Rooms Provide name Provide list of available rooms for a given time period	Room

Source: <http://www.agilemodeling.com/artifacts/crcModel.htm>

# Analysis Activities

- [ Identify entity/boundary/control objects
- [ Map use cases to objects
- [ Model object interactions with CRC cards
- [ **Identify associations/aggregates/attributes**
- [ Model state-dependent behavior of objects
- [ Model Inheritance relationships among objects

# Identifying Associations

- [ Use class diagrams to capture associations among objects
- [ Association: a relationship between 2 or more classes
- [ Associations:
  - clarify the analysis model through explication
  - enable the developer to discover boundary cases (exceptions that must be clarified)

# Association Properties

- [ Name: describes the association between classes (optional)
- [ Role: identifies the function of each class w.r.t. associations
  - added at each end of the association
- [ Multiplicity: identifies possible # of instances at each end
  - ex. \* = 0 or more, 1 = exactly 1, etc.

# Association Heuristics

- [ Examine verb phrases
- [ Name associations and roles as precisely as possible
- [ Use qualifiers to identify namespaces and key attributes
- [ Eliminate associations that can be derived elsewhere
- [ Reserve multiplicity until associations are stable
- [ Too many associations = model that is unreadable

# Identifying Aggregates

- [ Aggregations are a special type of association
  - denote a whole-part relationship
- [ Composition aggregation (solid diamond): existence of the parts is dependent upon the whole
  - ex. County depends on State
- [ Shared aggregation (hollow diamond): whole and part can exist independently
  - ex. FireEngine and FireStation

# Identifying Attributes

- [ Attributes are properties of individual objects
  - only model attributes that are relevant to the system
- [ Properties that are represented by objects are not attributes
- [ Identify as many associations as possible before identifying attributes

# Attribute Details

- [ Name: identifies attributes within an object
- [ Brief description
- [ Type: describes the legal values that an attribute can take
  
- [ Attributes are the least stable part of the object model

# Attribute Heuristics

- [ Examine possessive phrases
- [ Represent stored state as an attribute
- [ Describe each attribute
- [ Do not represent attributes as objects; use associations instead
- [ Don't describe fine details until the object model is stable

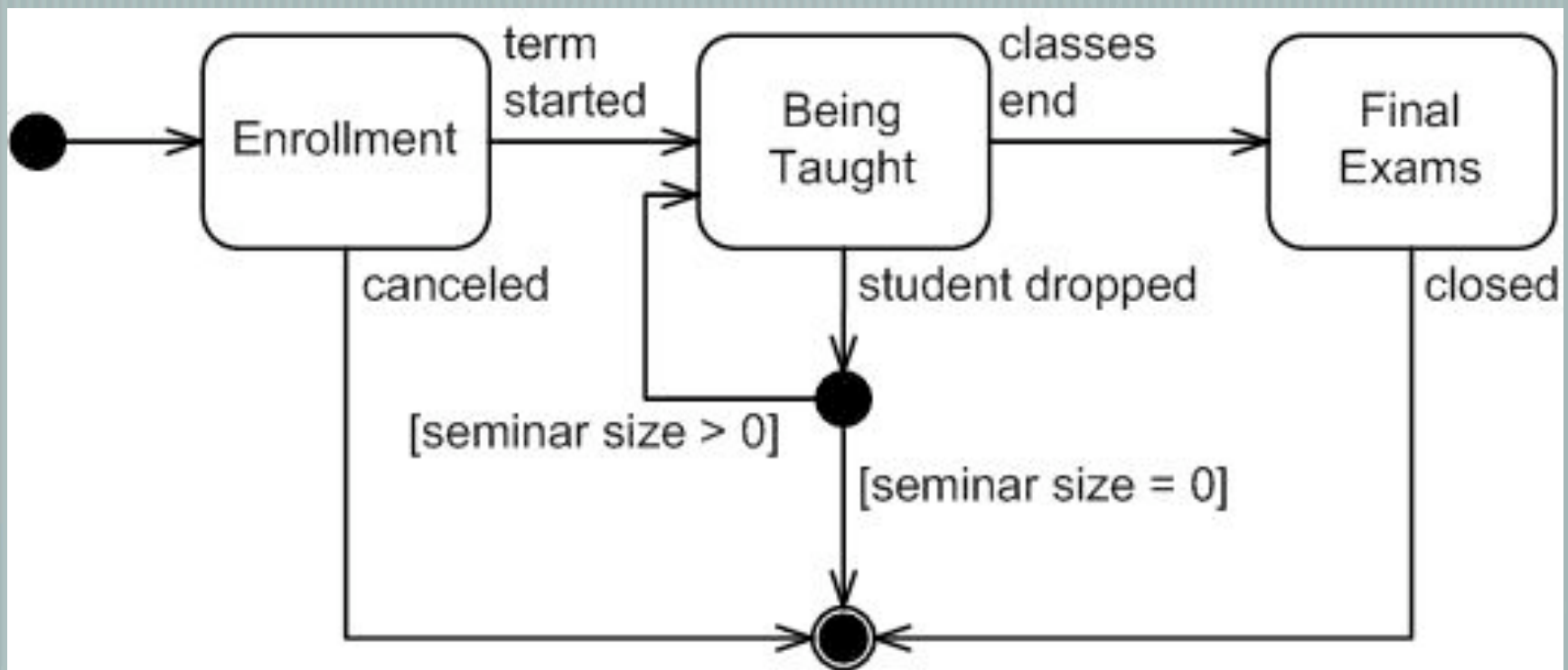
# Analysis Activities

- [ Identify entity/boundary/control objects
- [ Map use cases to objects
- [ Model object interactions with CRC cards
- [ Identify associations/aggregates/attributes
- [ **Model state-dependent behavior of objects**
- [ Model Inheritance relationships among objects

# State-Dependent Behavior

- [ Statechart diagrams represent behavior from the perspective of a single object
- [ Allows the developer to build a formal description of object behavior
  - helps to identify missing use cases
- [ Only objects with extended lifetimes and state-dependent behavior should be modeled this way

# UML State Machine Diagrams



Source: <http://www.agilemodeling.com/artifacts/stateMachineDiagram.htm>

# Analysis Activities

- [ Identify entity/boundary/control objects
- [ Map use cases to objects
- [ Model object interactions with CRC cards
- [ Identify associations/aggregates/attributes
- [ Model state-dependent behavior of objects
- [ **Model Inheritance relationships among objects**

# Modeling Inheritance

- [ Generalization eliminates redundancy from a model
- [ If two classes share attributes/behaviors, we consolidate those similarities into a(n abstract) superclass

# The Review Stage

# Analysis Summary

— [ Requirements elicitation is iterative and incremental

— [ General sequence:

1. Define use cases
2. Define participating objects
3. Define interactions and the 3 As
4. Consolidate model
5. Review model

# Analysis Review

- [ Analysis is the process of formalizing a requirements specification to ensure that it is correct, complete, consistent, and unambiguous
- [ Analysis tasks include object identification, mapping use cases to objects, defining object interactions, and defining object inheritance

# Analysis Model Review

- [ The analysis model is built incrementally and iteratively
- [ Once the model is stable, developers and clients review it
  - make sure requirements specification is correct, complete, consistent, and unambiguous
  - review requirements for realism and verifiability

# Is The Model Correct?

- [ Is the entity object glossary understandable by the user?
- [ Do abstract classes correspond to user-level concepts?
- [ Are all descriptions in accordance w/ user definitions?
- [ Do all entity/boundary objects have meaningful noun names?
- [ Do all use cases/control objects have meaningful verb names?
- [ Are all error cases described and handled?

# Is The Model Complete?

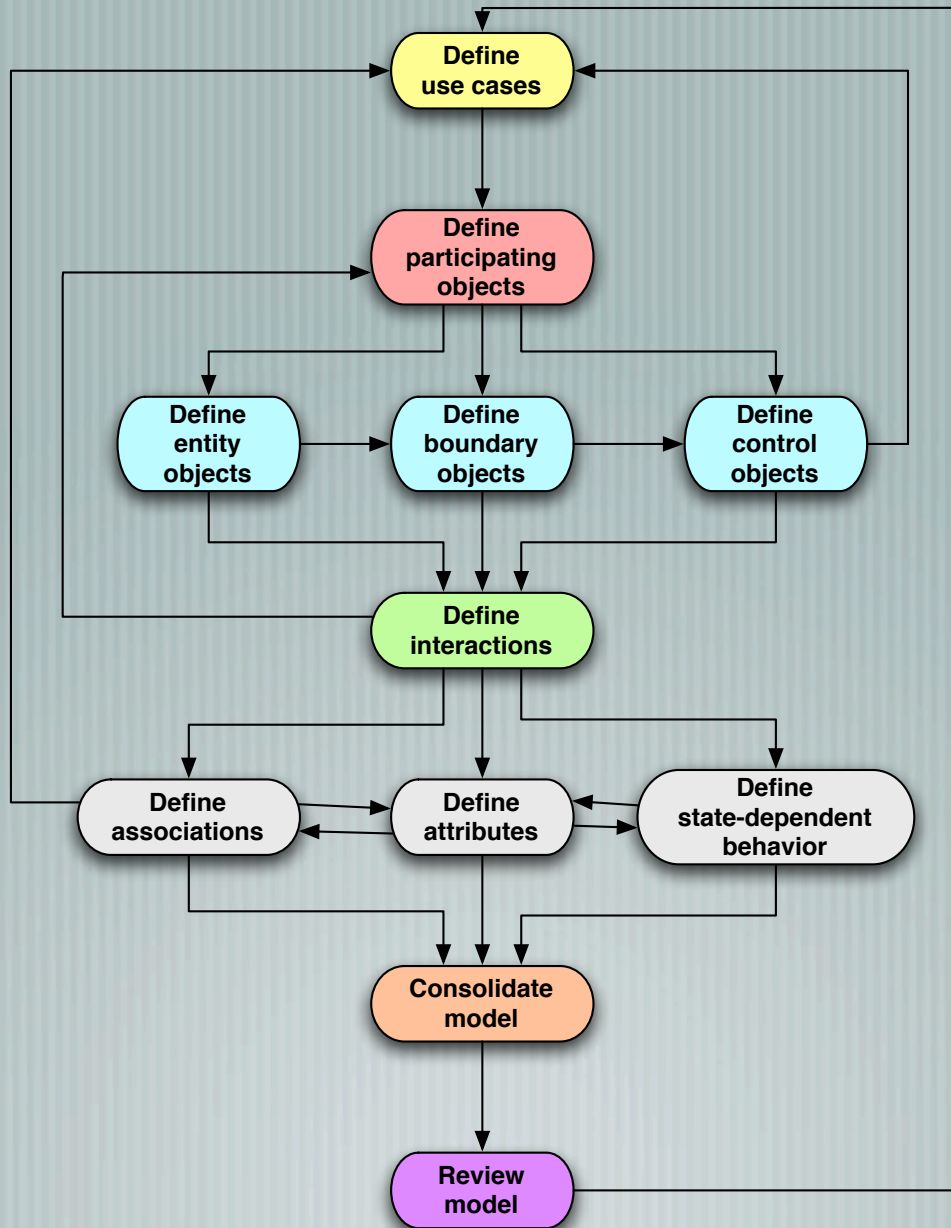
- [ For each object: is it needed by any use case? Where is it created/modified/destroyed? Can it be accessed?
- [ For each attribute: when is it set? What is its type? Should it be a qualifier?
- [ For each association: when is it traversed? Why was its specific multiplicity chosen?
- [ For each control object: does it have the necessary associations to access participating objects?

# Is The Model Consistent?

- [ Are there multiple classes/use cases with the same name?
- [ Do entities with similar names denote similar concepts?
- [ Are there objects with similar attributes/associations that are not in the same generalization hierarchy?

# Is The Model Realistic?

- [ Are there any novel features in the system?
  - Were any studies conducted or prototypes built to ensure their feasibility?
- [ Can the performance/reliability requirements be met?
  - Were these requirements verified by prototypes running on the selected hardware?



# Managing Analysis

- [ Documentation
- [ Role analysis
- [ Other management issues

# Documenting Analysis

- [ Recall the Requirements Analysis Document from last time
  - sections: introduction, current system, proposed system
- [ In the requirements elicitation phase, we already wrote sections 1 and 2, and part of 3 (functional/nonfunctional requirements and use case model)
- [ Now we can write the rest: object & dynamic models

# Documenting Analysis

- [ Object model section

- describes all objects with attributes and operations

- class diagrams illustrate object relationships

- [ Dynamic model section

- documents behavior: statecharts and sequence diagrams

# Assigning Responsibilities

- [ Three main types of roles:
  - information generation (ex. end user/domain expert)
  - integration: defines scope of system based on user requirements (ex. client)
  - review: validates RAD for correctness, etc.

# Typical Roles

— [ End user

— [ Client

— [ Analyst: application domain expert

— [ Architect

— [ Document editor

— [ Configuration manager

— [ Reviewer

# Communicating About Analysis

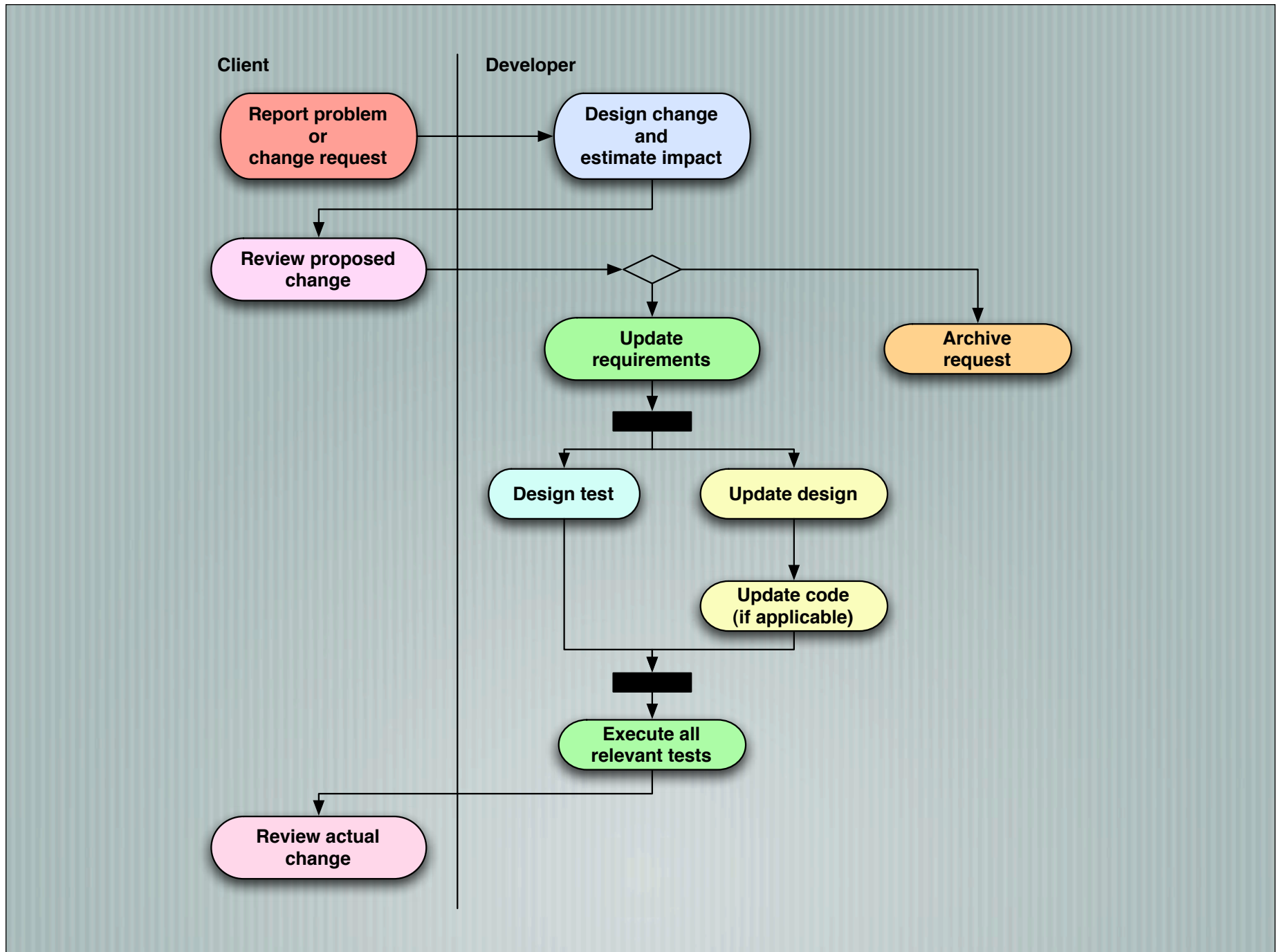
- [ Communication challenges include:
  - different backgrounds of participants
  - different stakeholder expectations
  - new teams
  - system evolution

# Communication Guidelines

- [ Define clear territories
  - includes defining roles and public/private forums
- [ Define clear objectives and success criteria
  - this is critical to resolving conflicts
- [ Brainstorm
  - may lead to shared, ad-hoc notations

# Client Sign-Off

- [ Acceptance of the analysis model by the client
- [ Includes agreement on:
  - a list of priorities (for development emphasis)
  - a revision process (for changing requirements)
  - a list of acceptance/rejection criteria
  - a schedule and budget



# Next Time

— [ System Design