

Lecture 16: Object-Oriented Design Methods

Kenneth M. Anderson
University of Colorado, Boulder
CSCI 4448/6448 – Lecture 16 – 10/18/2007

© University of Colorado, 2007

Goals for this Lecture

- Review various OO design methods (software development life cycles)
- Briefly review two additional UML diagrams
 - Activity Diagrams
 - State Diagrams

Background

- In Software Engineering:
 - “Process is King”
 - We want our activities to be coordinated and planned, e.g. “engineered”

Life Cycle Characteristics

- Life cycles make software development
 - predictable
 - repeatable
 - measurable
 - efficient
- High-quality processes should lead to high-quality products
 - at least it improves the odds of producing good software

Survey of OOA&D Methods

- Generalization
 - Taken from “SE: A Practitioner’s approach, 4th ed.” by Roger S. Pressman, McGraw-Hill, 1997
- The Booch Method
- The Jacobson Method
- The Rumbaugh Method
- The Unified Software Process

OO Methods in General...

- Obtain customer requirements for the OO System
 - Identify scenarios or use cases
 - Build a requirements model
- Select classes and objects using basic requirements
- Identify attributes and operations for each object
- Define structures and hierarchies that organize classes
- Build an object-relationship model
- Build an object-behavior model
- Review the OO analysis model against use cases
 - Once complete, move to design and implementation: These phases simply elaborate the previously created models with more and more detail, until it is possible to write code straight from the models

Detailed comparisons

- What follows is a barebones description of each method, detailed comparisons can be found in:
 - Graham, I. Object-Oriented Methods, Addison-Wesley, Third Edition, 2001

Background on OO Methods

- An OO Method should cover and include
 - requirements and business process modeling
 - a lightweight, customizable process framework
 - project management
 - component architecture
 - system specification
 - use cases, UML, architecture, etc.
 - component design and decomposition
 - testing throughout the life cycle
 - Software quality assurance
 - Configuration management

The Booch Method

- **Identify classes and objects**
 - Propose candidate objects
 - Conduct behavior analysis
 - Identify relevant scenarios
 - Define attributes and operations for each class
- **Identify the semantics of classes and objects**
 - Select scenarios and analyze
 - Assign responsibility to achieve desired behavior
 - Partition responsibilities to balance behavior
 - Select an object and enumerate its roles and responsibilities
 - Define operations to satisfy the responsibilities

Booch, continued

- **Identify relationships among classes and objects**
 - Define dependencies that exist between objects
 - Describe the role of each participating object
 - Validate by walking through scenarios
- **Conduct a series of refinements**
 - Produce appropriate diagrams for the work conducted above
 - Define class hierarchies as appropriate
 - Perform clustering based on class commonality
- **Implement classes and objects**
 - In analysis and design, this means specify everything!

The Jacobson Method

- Object-Oriented Software Engineering
 - Primarily distinguished by the use-case
 - Simplified model of Objectory
 - Objectory evolved into the Rational Unified Software Development Process
 - For more information on this Objectory precursor, see
 - Jacobson, I., Object-Oriented Software Engineering, Addison-Wesley, 1992.

Jacobson, continued

- **Identify the users of the system and their overall responsibilities**
- **Build a requirements model**
 - Define the actors and their responsibilities
 - Identify use cases for each actor
 - Prepare initial view of system objects and relationships
 - Review model using use cases as scenarios to determine validity
- Continued on next slide

Jacobson, continued

- **Build analysis model**
 - Identify interface objects using actor-interaction information
 - Create structural views of interface objects
 - Represent object behavior
 - Isolate subsystems and models for each
 - Review the model using use cases as scenarios to determine validity

The Rumbaugh Method

- Object Modeling Technique (OMT)
 - Rumbaugh, J. et al., Object-Oriented Modeling and Design, Prentice-Hall, 1991
- Analysis activity creates three models
 - Object model
 - Objects, classes, hierarchies, and relationships
 - Dynamic model
 - object and system behavior
 - Functional model
 - High-level Data-Flow Diagram

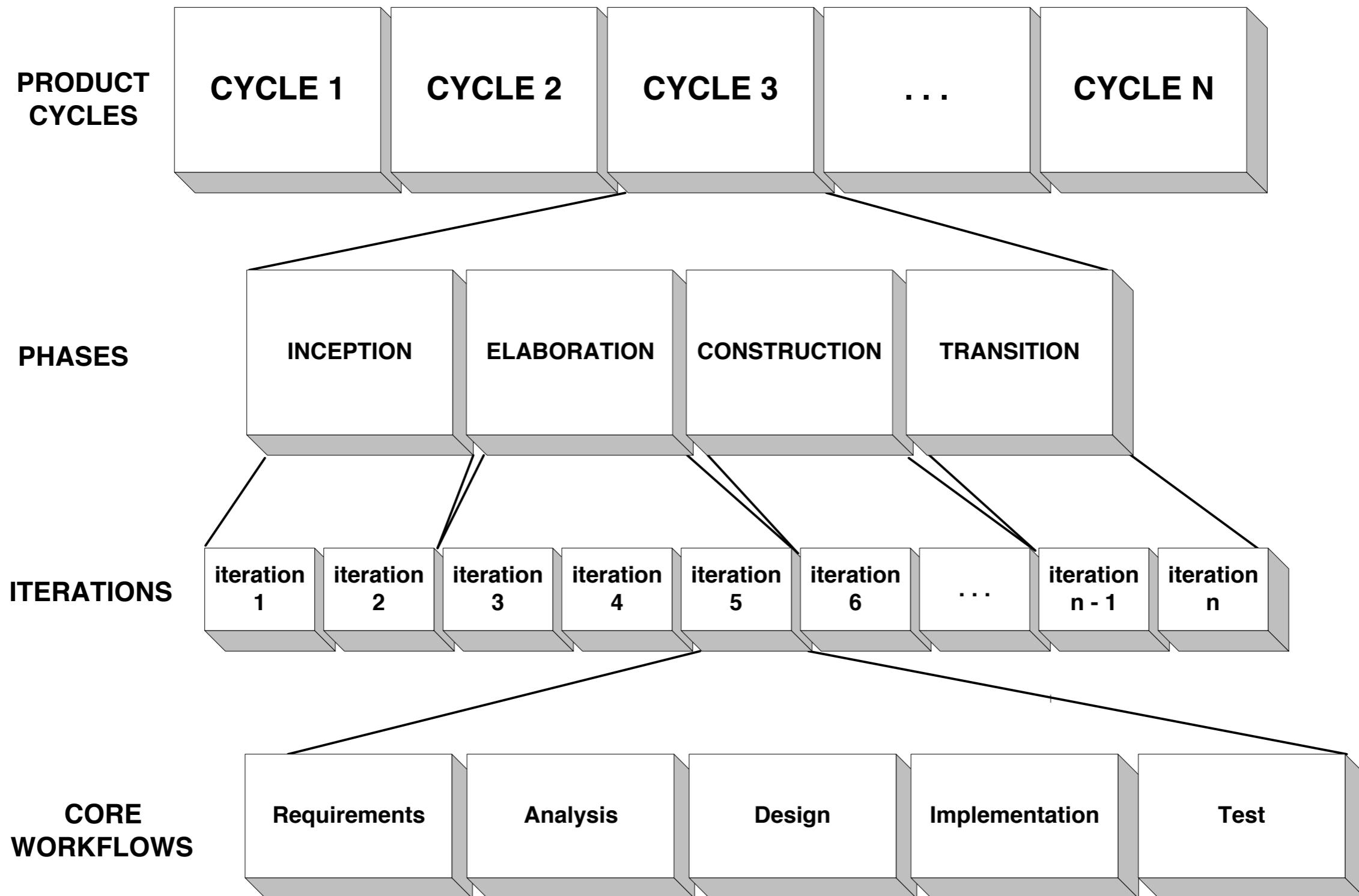
Rumbaugh, continued

- **Develop a statement of scope for the problem**
- **Build an object model**
 - Identify classes that are relevant for the problem
 - Define attributes and associations
 - Define object links
 - Organize object classes using inheritance
- **Develop a dynamic model**
 - Prepare scenarios
 - Define events and develop an event trace for each scenario
 - Construct an event flow diagram and a state diagram
 - Review behavior for consistency and completeness

Rumbaugh, continued

- **Construct a functional model for the system**
 - Identify inputs and outputs
 - Use data flow diagrams to represent flow transformations
 - Develop a processing specification for each process in the DFD
 - Specify constraints and optimization criteria
- Iterate

Rational Unified Process: Overview



Inception

- High-level planning for the project
- Determine the project's scope
- If necessary
 - Determine business case for the project
 - Estimate cost and projected revenue

Elaboration

- Develop requirements and initial design
- Develop Plan for Construction phase
- Risk-driven approach
 - Requirements Risks
 - Technological Risks
 - Skills Risks
 - Political Risks

Requirements Risks

- Is the project technically feasible?
- Is the budget sufficient?
- Is the timeline sufficient?
- Has the user really specified the desired system?
- Do the developers understand the domain well enough?

Dealing with Requirements Risks

- Construct models to record Domain and/or Design knowledge
 - Domain model (vocabulary)
 - Use Cases
 - Design model
 - Class diagrams
 - Activity diagrams
- Prototype construction

Dealing with Requirements Risks

- Begin by learning about the domain
 - Record and define jargon
 - Talk with domain experts and end-users
- Next construct use cases
 - What are the required external functions of the system?
 - Iterative process; Use Cases can be added as they are discovered

Dealing with Requirements Risks

- Finally, construct design model
 - Class diagrams identify key domain concepts and their high-level relationships
 - Activity diagrams highlight the domain's work practices
 - A major task here is identifying parallelism that can be exploited later
- Be sure to consolidate iterations into a final consistent model

Dealing with Requirements Risks

- Build prototypes
 - Used only to help understand requirements
 - Throw them all out!
 - Do not be tied to an implementation too early
 - Make use of rapid prototyping tools
 - 4th Generation Programming Languages
 - Scripting and/or Interpreted environments
 - UI Builders
 - Be prepared to educate the client as to the purpose of the prototype

Technology Risks

- Are you tied to a particular technology?
- Do you “own” that technology?
- Do you understand how different technologies interact?
- Techniques
 - Prototypes
 - Class diagrams, package diagrams
 - “Scouting” — evaluate technology early

Skill Risks

- Do the members of the project team have the necessary skills and background to tackle the project?
- If not your options are
 - Training
 - Consulting
 - Mentoring
 - Hiring new people

Political Risks

- How well does the proposed project mesh with corporate culture?
 - Consider the attempt to use Lotus Notes at Arthur Anderson
 - Lotus Notes attempts to promote collaboration
 - Arthur Anderson consultants compete with each other!
 - Consider e-mail: any employee can ignore the org chart and mail the CEO!

Political Risks, continued

- Will the project directly compete with another business unit?
- Will it be at odds with some higher level manager's business plan?
- Any of these can kill a project...
- Examples from students?

Reference

- Lotus Notes vs. Arthur Anderson
 - Orlikowski, W. J. (1992). "Learning from Notes: Organizational Issues in Groupware Implementation". Proceedings of ACM CSCW'92 Conference on Computer-Supported Cooperative Work: 362-369.
- If you are interested you can borrow my copy of the CSCW'92 proceedings to make a copy

Ending Elaboration

- Baseline architecture constructed
 - List of Use Cases (with estimates)
 - Domain Model
 - Technology Platform
- AND
 - Risks identified
 - Plan constructed
 - Use cases assigned to iterations

Construction

- Each iteration produces a software product that implements the assigned Use cases
 - Additional analysis and design may be necessary as the implementation details get addressed for the first time
- Extensive testing should be performed and the product should be released to (some subset of) the client for early feedback

Transition

- Final phase before release 1.0
- Optimizations can now be performed
 - Optimizing too early may result in the wrong part of the system being optimized
 - Largest boosts in performance come from replacing non-scalable algorithms or mitigating bottlenecks

Summary

- All OO Design Methods share a similar structure
 - The design method described in our textbook
 - is compatible with the methods discussed in this lecture
 - is lighter weight, allowing you to get to code more quickly
- The important point is to use a process that has you switching between A&D activities and coding in a fairly regular fashion
 - This helps you shift your perspective from “internal” to “external” and allows you to keep in mind your ultimate goal:
 - delivering a system that meets your customer’s needs

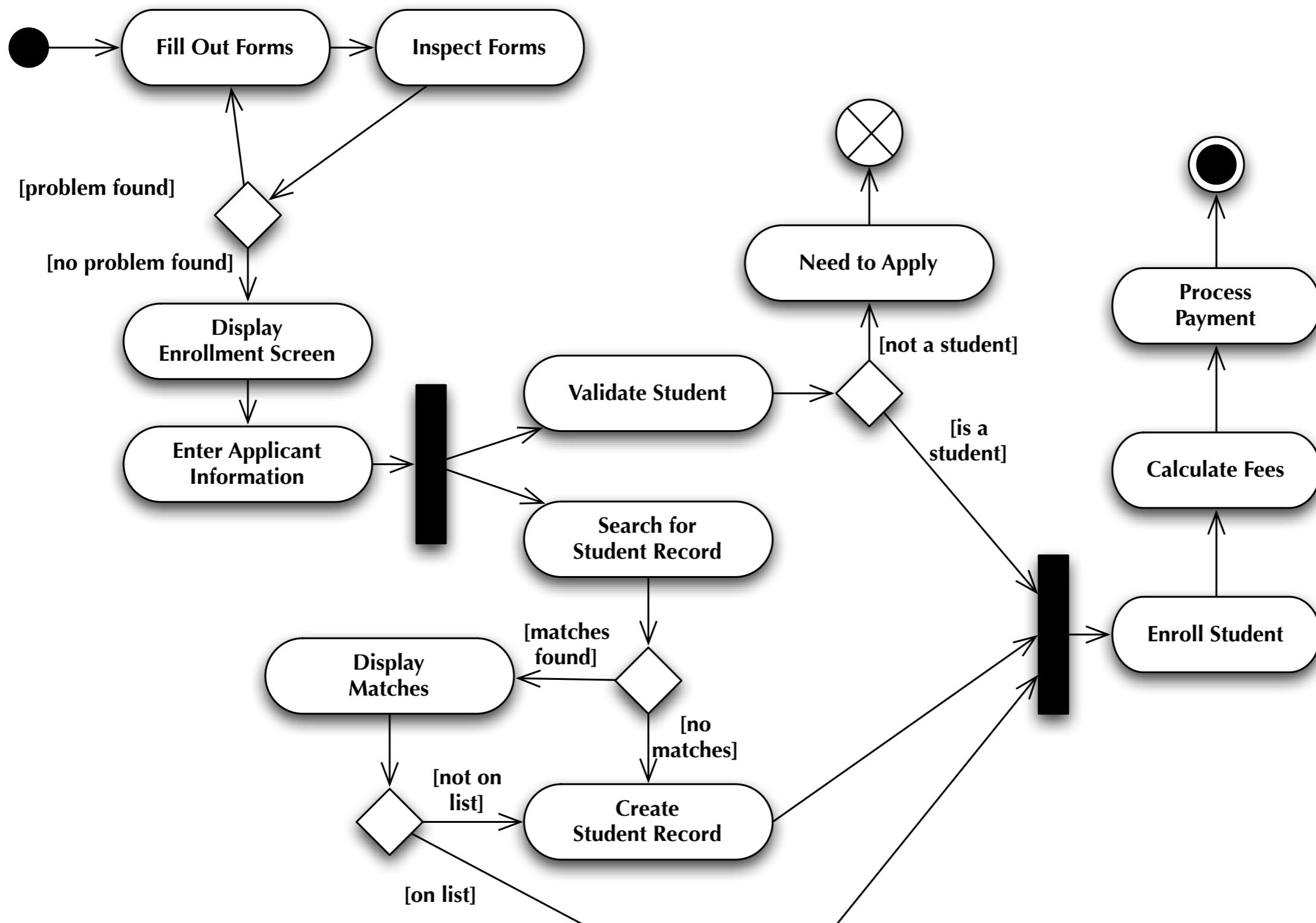
Two New UML Diagrams

- Activity Diagrams and State Diagrams
- Relationship to life cycles
 - They represent alternate ways to record/capture design information about your system. They can help you identify new classes and methods that
 - They are typically used in the following places in analysis and design
 - After use case creation: create an activity diagram for the use case
 - For each activity in the diagram: draw a sequence diagram
 - Add a class for each object in the sequence diagrams to your class diagram, add methods in sequence diagrams to relevant classes
 - Based on this information, see if you can partition an object's behavior into various categories (initializing, acquiring info, performing calcs, ...)
 - Create a state diagram for the object that documents these states and the transitions between them (transitions typically map to method calls)

Activity Diagrams

- Think “Flow Chart on Steroids”
 - Able to model complex, parallel processes with multiple ending conditions
- Constructs
 - Initial Node (circle)/Final Node (circle in circle)/Early Termination Node (circle with x through it)
 - Activity: Rounded Rectangle indicating an action of some sort either by a system or by a user
 - Flow: directed lines between activities and/or other constructs. Flows can be annotated with guards “[student on list]” that restrict its use
 - Fork/Join: Black bars that indicate activities that happen in parallel
 - Decision/Merge: Diamonds used to indicate conditional logic.
 - Swim Lanes: A way to layout the diagram to associate roles with activities

Example Activity Diagram

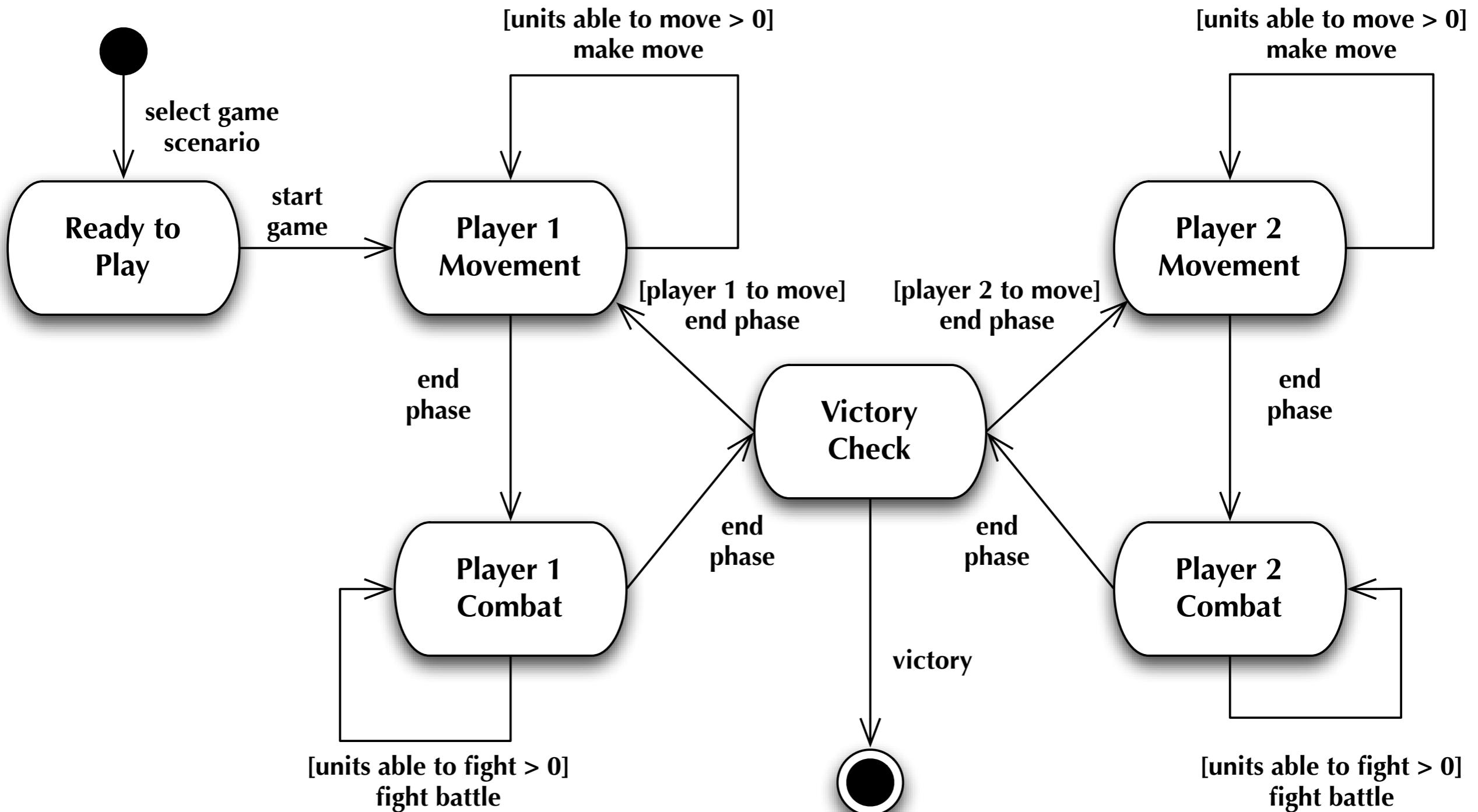


Example adapted from <<http://www.agilemodeling.com/artifacts/activityDiagram.htm>>. Copyright © 2003-2006 Scott W. Ambler

State Diagrams

- Shows the major states of an object or system
 - Each state appears as a rounded rectangle
 - Arrows indicate state transitions
 - Each transition has a name that indicates what triggers the transition (often times, this name corresponds to a method name)
 - Each transition may optionally have a guard that indicates a condition that must be true before the transition can be followed
 - A state diagram also has a start state and an end state
- State diagrams are useful if you have a class that has “partitions” in its behavior: it acts one way when it starts, another way when condition A occurs, another way when condition B occurs, and another way before it terminates

Example State Diagram



Wrapping Up

- We've finished the first half of the semester
 - Reviewed basic OO concepts, terminology and notations
 - Learned OO analysis and design techniques
 - Learned OO principles
 - Learned UML (class, sequence, activity, state, use case)
 - Laid the foundation for OO design patterns
 - Ended with a review of OO Design methods
- Second half of the semester
 - Design patterns, refactoring, test-driven design, and more