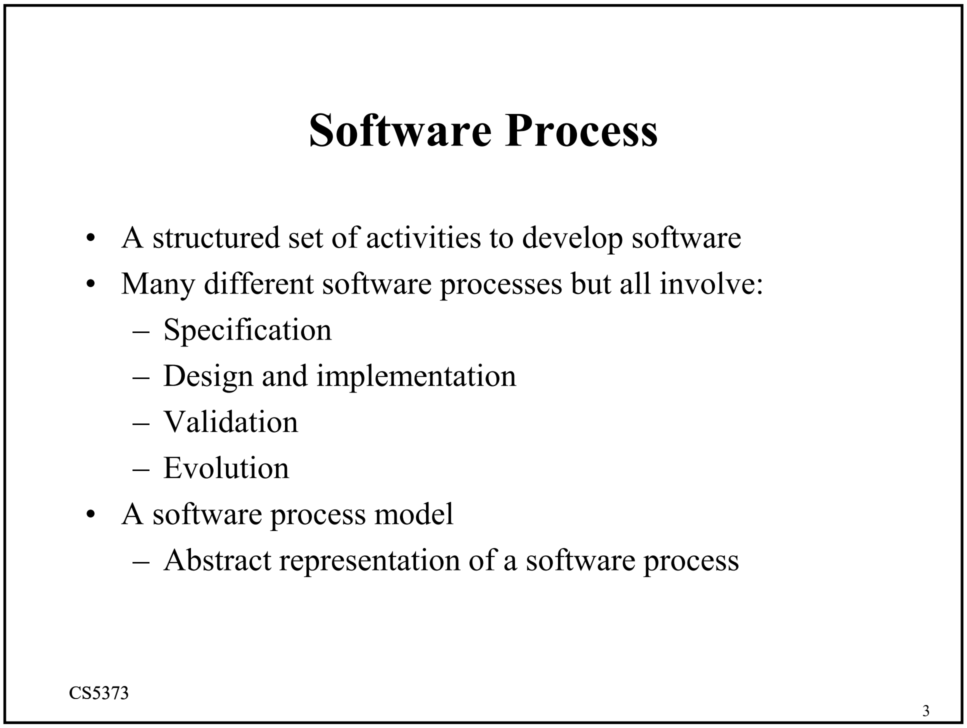
CS5373:
Software Modeling and Design

Lecture 1 - Introduction

H. Gomaa, “Chapters 1-5, Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures,” Cambridge University Press, February 2011

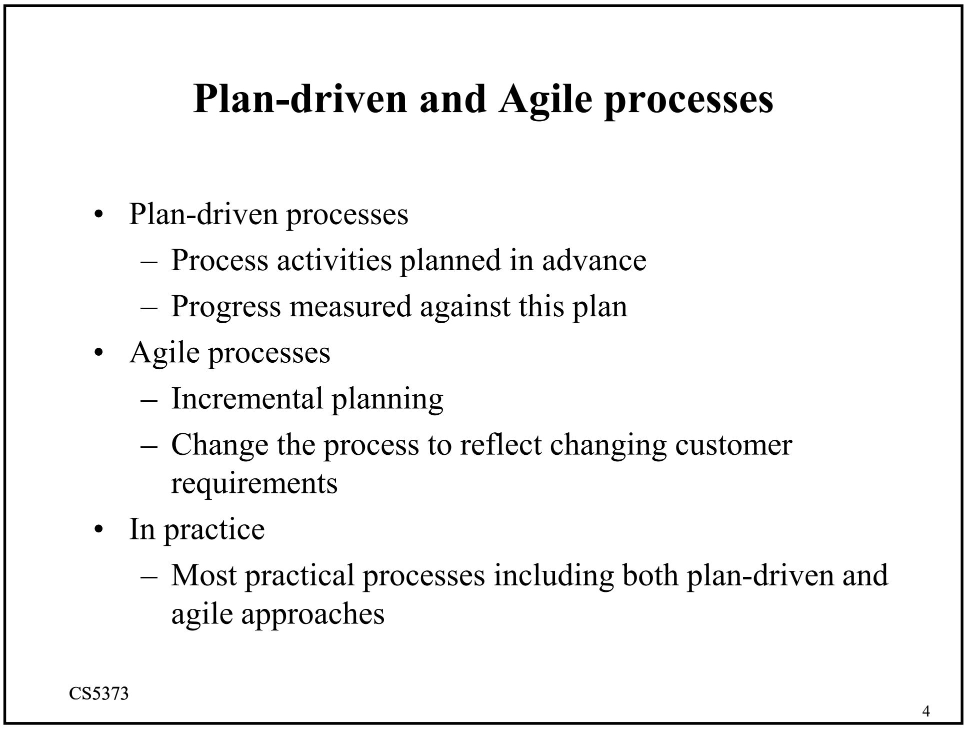
Overview

– Software Process
– Software Design Method: COMET



Software Process

- A structured set of activities to develop software
- Many different software processes but all involve:
  - Specification
  - Design and implementation
  - Validation
  - Evolution
- A software process model
  - Abstract representation of a software process



Plan-driven and Agile processes

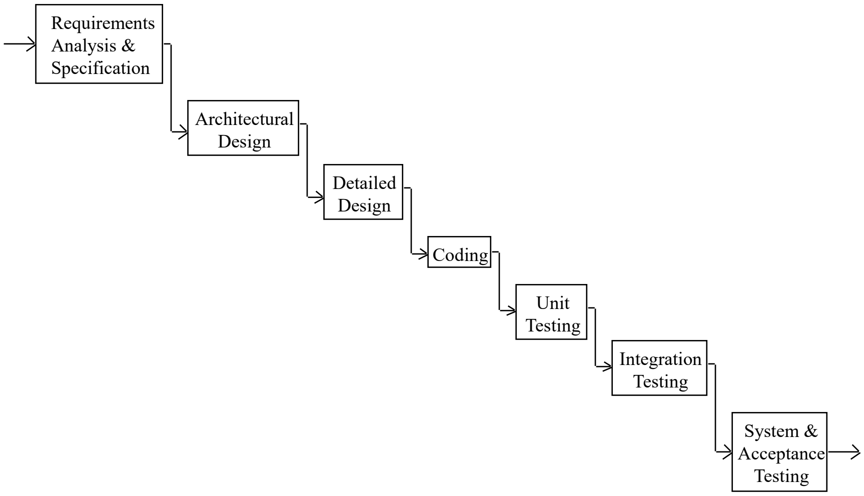
- Plan-driven processes
  - Process activities planned in advance
  - Progress measured against this plan
- Agile processes
  - Incremental planning
  - Change the process to reflect changing customer requirements
- In practice
  - Most practical processes including both plan-driven and agile approaches

Software life cycle models

- Waterfall – traditional
- Exploratory - throwaway prototyping
- Incremental - evolutionary prototyping
- Spiral – risk driven process model
- Agile – response to change request

Software Life Cycle

Fig 3.1: Waterfall Model



Requirements Analysis & Specification

Architectural Design

Detailed Design

Coding

Unit Testing

Integration Testing

System & Acceptance Testing

Waterfall model

- Characteristics
  - No iteration in a software life cycle
  - Only appropriate when the requirements are well-understood
  - Each phase clearly separated without overlap

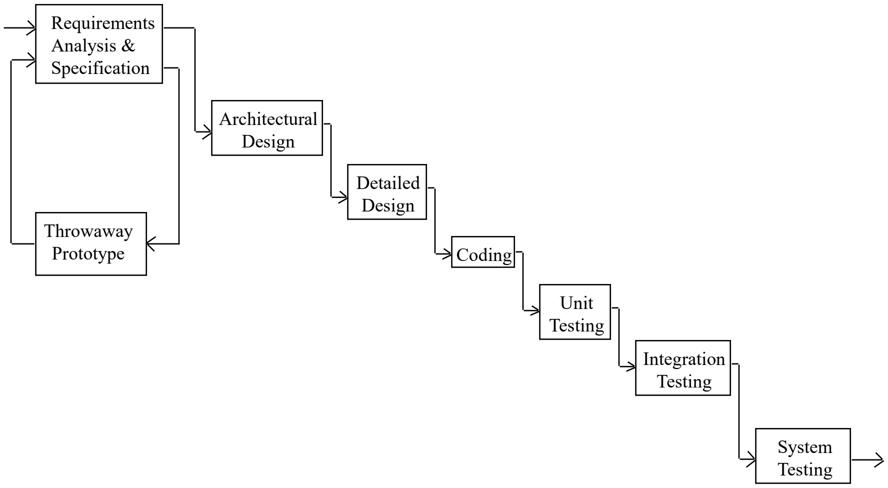
Limitations of Waterfall Model

- Lack of iteration
  - Limited iteration between phases is possible
- Software requirements tested late in life cycle
  - Throw-away Prototyping Life Cycle
- Operational system available late in life cycle
  - Incremental (evolutionary) Development Life Cycle

Prototyping (Exploratory) Process During Requirements Phase

- Problem
  - Software requirements tested late in life cycle
- Solution
  - Use throw-away prototyping
  - Bridges user/developer gap
- Approach
  - Develop prototype from draft requirements specification
  - Allows "hands-on" use of system before built
  - Revise requirements specification based on user feedback
- Prototyping applied for design phase as well

Fig. 3.3: Throwaway Prototyping (Exploratory) on Software Life Cycle



Requirements Analysis & Specification

Throwaway Prototype

Architectural Design

Detailed Design

Coding

Unit Testing

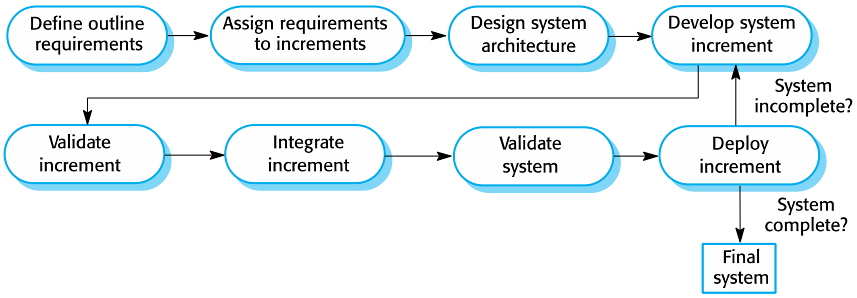
Integration Testing

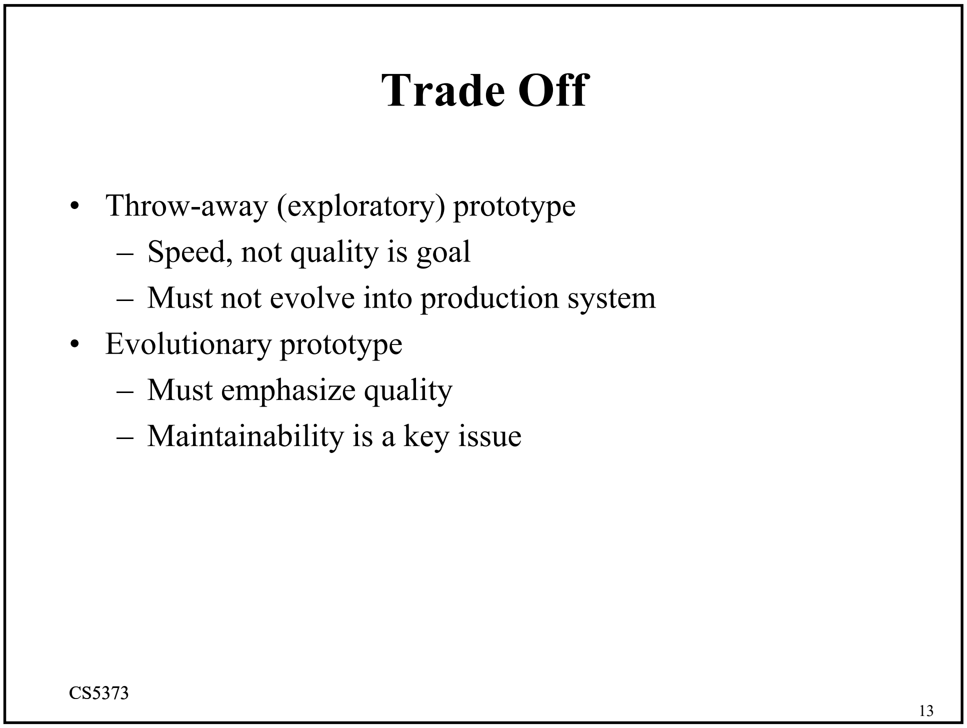
System Testing

Incremental Development

- Problem
  - Operational system available late in life cycle
- Solution
  - Use incremental development
- Approach
  - Develop subset of system - working early
  - Gradually build on

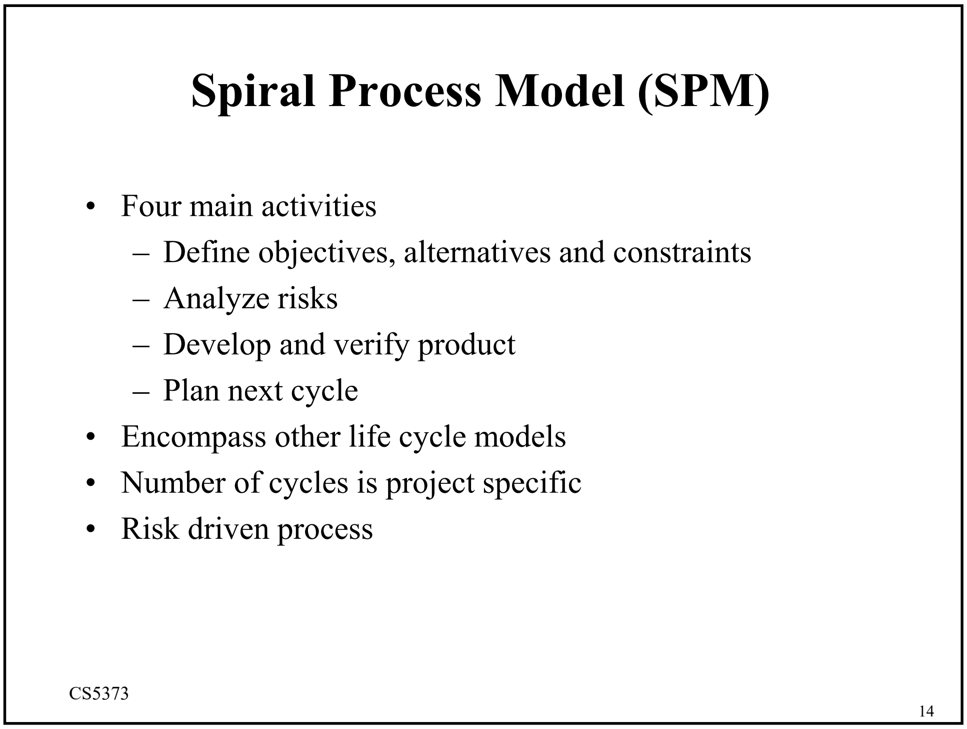
Incremental development





Trade Off

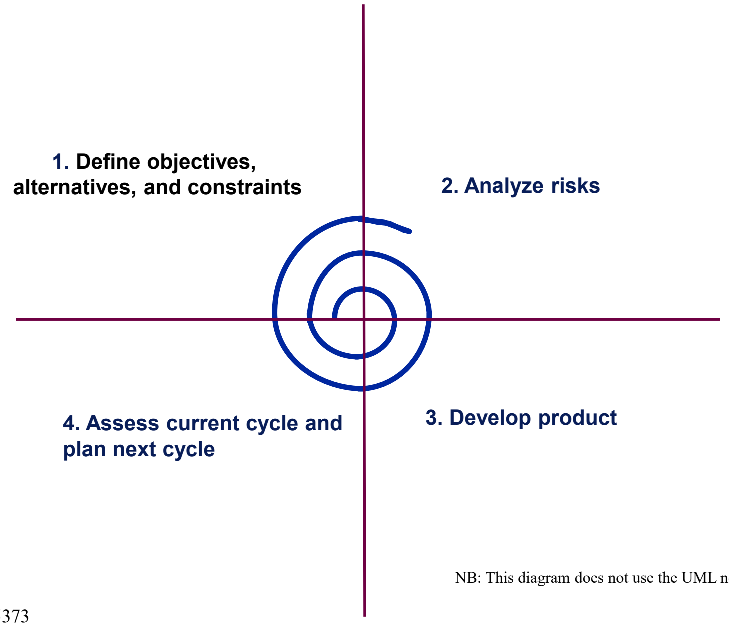
- Throw-away (exploratory) prototype
  - Speed, not quality is goal
  - Must not evolve into production system
- Evolutionary prototype
  - Must emphasize quality
  - Maintainability is a key issue



Spiral Process Model (SPM)

- Four main activities
  - Define objectives, alternatives and constraints
  - Analyze risks
  - Develop and verify product
  - Plan next cycle
- Encompass other life cycle models
- Number of cycles is project specific
- Risk driven process

Figure 3.7: The spiral process model



1. Define objectives, alternatives, and constraints

2. Analyze risks

3. Develop product

4. Assess current cycle and plan next cycle

NB: This diagram does not use the UML notation

Agile development

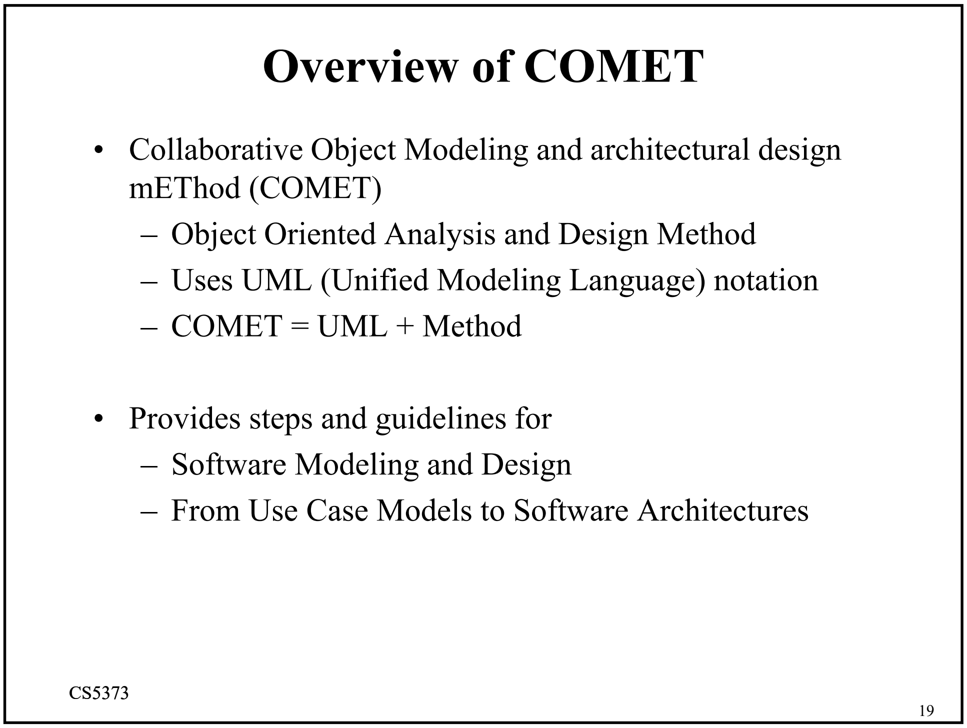
- Rapid development and delivery
  - Businesses operating in a fast-changing requirement
  - Plan-driven development
    - May not meet changing business needs
- Minimal documentation, and focus on working code

Some issues with agile methods

- Most software contracts based around a specification
- Agile methods appropriate for new software development
- Agile methods designed for small co-located teams
- The original developers not work on the system

Software Design Method: COMET

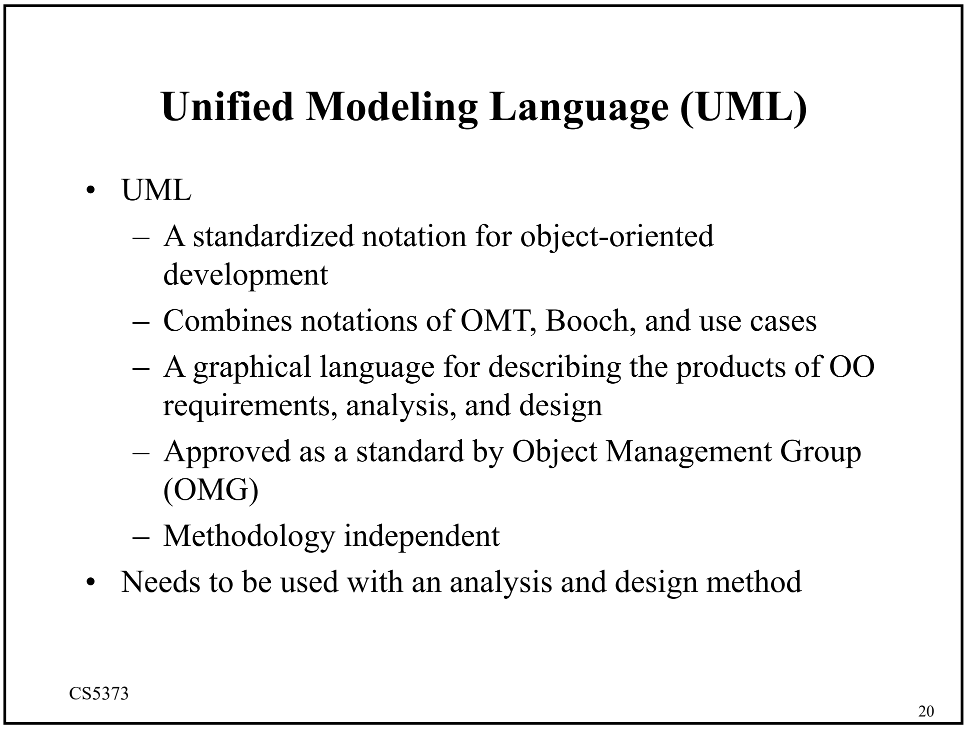
- Design concepts
  - Concurrent tasks, information hiding
- Design strategy
  - Develop requirements model
  - Develop analysis model
  - Develop design model
- Design structuring criteria
  - Object, subsystem, and task structuring criteria
- Design notation
  - UML (Unified Modeling Language)



Overview of COMET

- Collaborative Object Modeling and architectural design mEThod (COMET)
  - Object Oriented Analysis and Design Method
  - Uses UML (Unified Modeling Language) notation
  - COMET = UML + Method

- Provides steps and guidelines for
  - Software Modeling and Design
  - From Use Case Models to Software Architectures



Unified Modeling Language (UML)

- UML
  - A standardized notation for object-oriented development
  - Combines notations of OMT, Booch, and use cases
  - A graphical language for describing the products of OO requirements, analysis, and design
  - Approved as a standard by Object Management Group (OMG)
  - Methodology independent
- Needs to be used with an analysis and design method

Requirements Modeling

- Use Case Modeling
  - Define software functional requirements in terms of use cases and actors

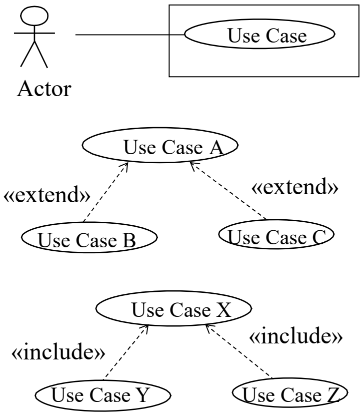


Figure 2.1 UML notation for use case diagram

Analysis Modeling

- Analysis Modeling consists of
  - Static Modeling
  - Dynamic Modeling
    - State Machine modeling using statecharts
    - Object interaction modeling

Analysis Modeling

- Static Modeling
  - Define structural relationships between classes
  - Depict classes and their relationships on class diagrams

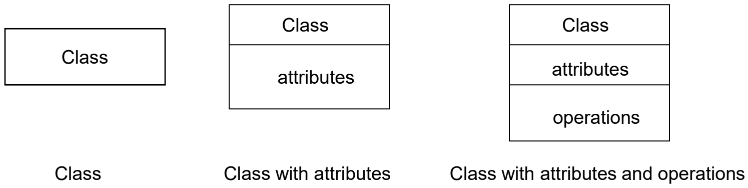


Figure 2.2 UML notation for classes

Analysis Modeling

- Dynamic Modeling
  - Define statecharts for state-dependent control objects

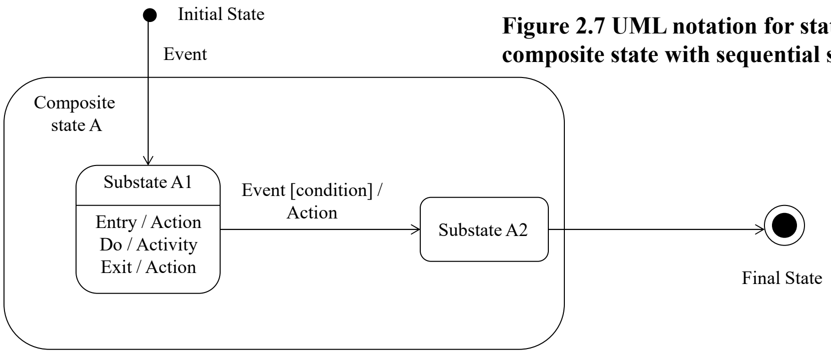
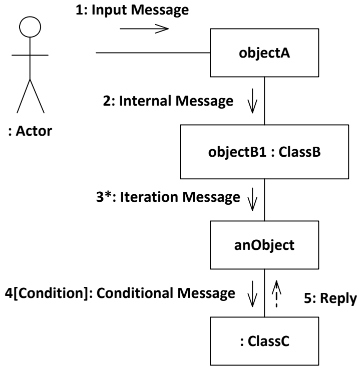


Figure 2.7 UML notation for statechart: composite state with sequential substates

Analysis Modeling

- Dynamic Modeling
  - Defines sequence of objects communicating with each other using communication diagrams or sequence diagrams

Figure 2.5: UML notation for communication diagram



Design Modeling

- Develop overall software architecture (Ch. 12)
  - Structure system into subsystems (Ch. 13)
- Design software architecture
  - Design object-oriented software architectures (Ch. 14)
  - Design client/server software architectures (Ch. 15)
  - Design service-oriented architectures (Ch. 16)
  - Design component-based software architectures (Ch. 17)
  - Design concurrent and real-time software architectures (Ch. 18)
  - Design software product line architectures (Ch. 19)