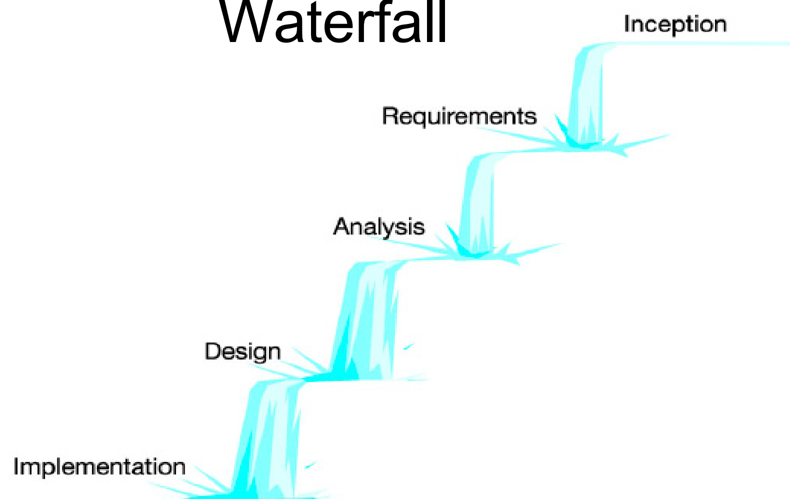
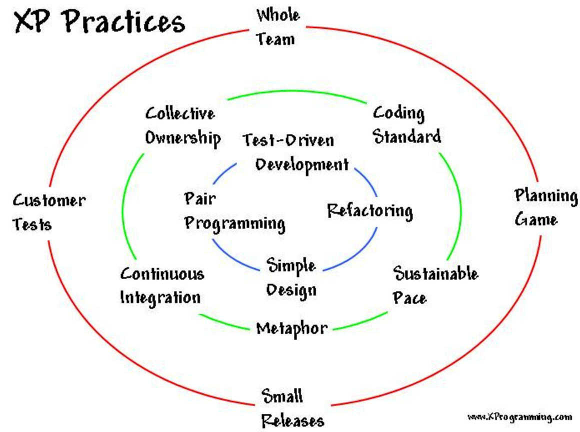
CIS*3430 Midterm Review

Where are we now…
in 42 slides or less

Waterfall

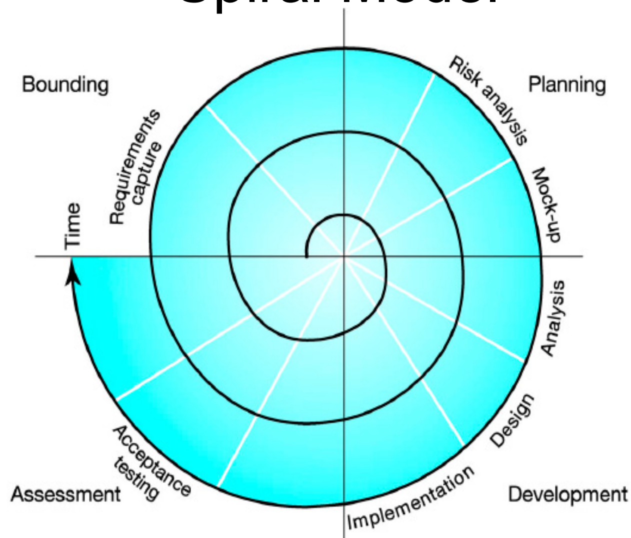


Extreme Programming

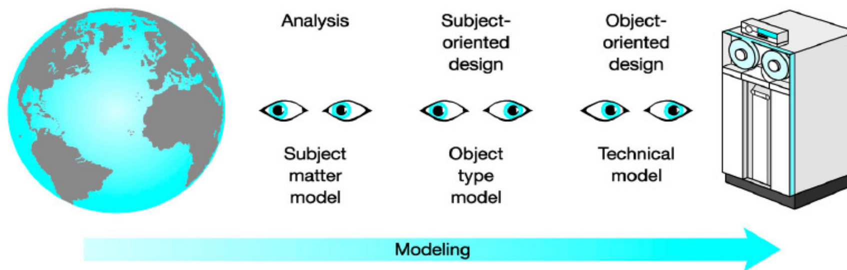


http://www.xprogramming.com/images/circles.jpg

Spiral Model

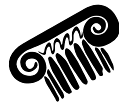


Three Models



What Are Requirements?

- Requirements are statements that would be unchanged even if the technology changed dramatically
- Contract between sponsor/stakeholders/user and developer
  - Every design element must be traceable to a formal requirement or use-case



Classic Requirements

- Definition:
  - A set of atomic statements about what the system
    - must do
    - should do
    - might do
- Example- rev 1:
  - The system must take a facial image, in any of the common image formats such as, today, JPEG, …, TIFF, produce a numerically-based profile and find other pictures, from selected databases, that are likely to be of the same person

Use Cases

- use case is an example of the system-to-be “in action”
  - “action” as seen from “outside the system looking in”, not the internal workings of the system
- use cases are text, not diagrams
- use cases describe interactions with the system
  - but written in “third person” as an observer factorial number of paths through system
- each pathway through a system called a scenario
  - can’t enumerate (let alone describe) all of them
- choose typical paths through system as 'main success' and then think about the alternates

A Use Case

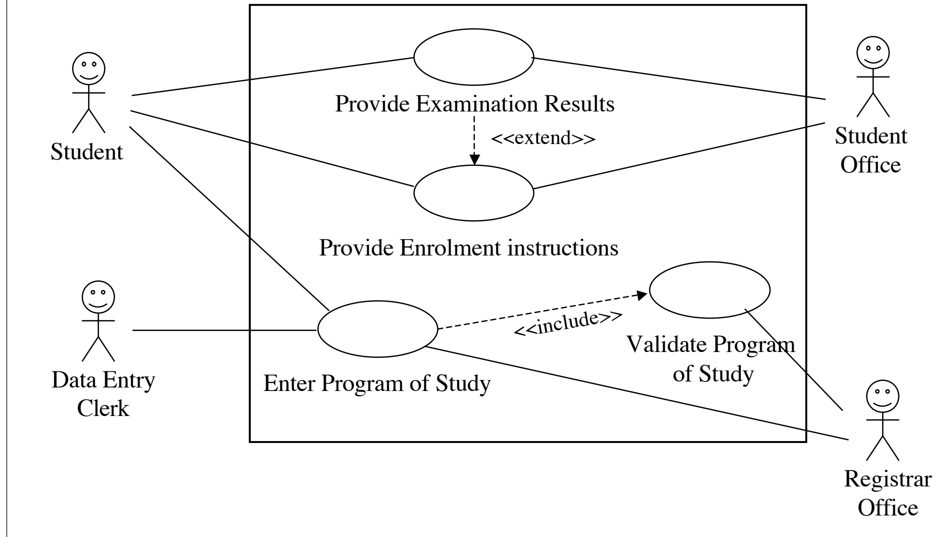
- Brief format

  Process Sale: A customer arrives at a checkout with items to purchase. The cashier uses the POS system to record each purchased item. The system presents a running total and line-item details. The customer enters payment information, which the system validates and records. The system updates inventory. The customer receives a receipt from the system and then leaves with the items.
- Casual format the same except has *Main Success Scenario* along with *Alternate Scenarios*

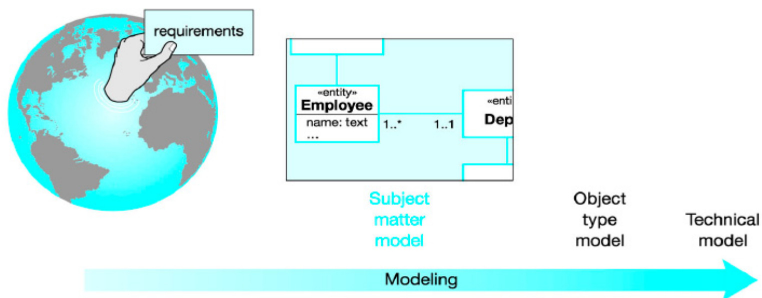
Actors: Three Types

- Primary
  - Main users of system services
  - For whom the system was built to “serve”
- Supporting
  - Provide a service to the system
  - Often another computer system external to the system under discussion (e.g. Credit Card information, PayPal, , etc.)
- Offstage
  - have interest in use case but not directly involved
  - e.g. government tax department

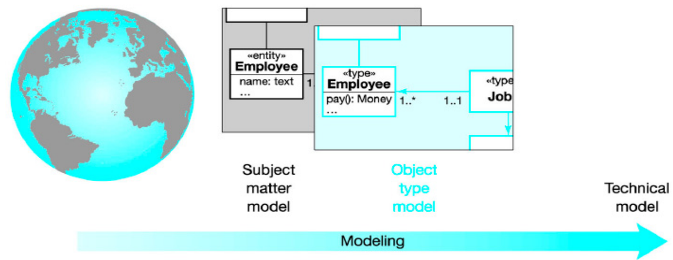
Use Case Diagrams: Example



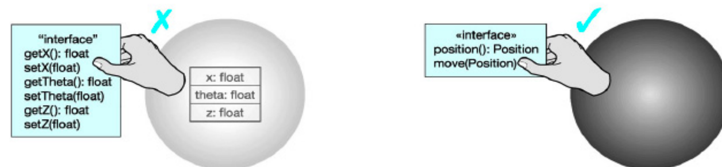
Subject Matter Model



Object Type Model



Inside Out or Outside In



Agile Development (our aim)

- Neither too much nor too little modelling
- Subject matter model represents entities, attributes and state
- Object Type model represents methods, message passing, outside-in design
- Technical model has classes and specific OO constructs
- Code a little, test a little

Chunks, cohesion and coupling

- Model must be:
  - Understandable
  - Reorganizable (Maintainable)
  - Divisible
- We have learned that for anything to be understandable and maintainable, we need
  - Chunks (modularity, componentization, …)
  - High cohesion
  - Low coupling

Finding Entities

- Nouns
- Talk the subject matter or to an expert
- Read what other experts have written
- Read the requirements document (remember, its an analysis input)
- Read proposals
- Listen to the voices in your head

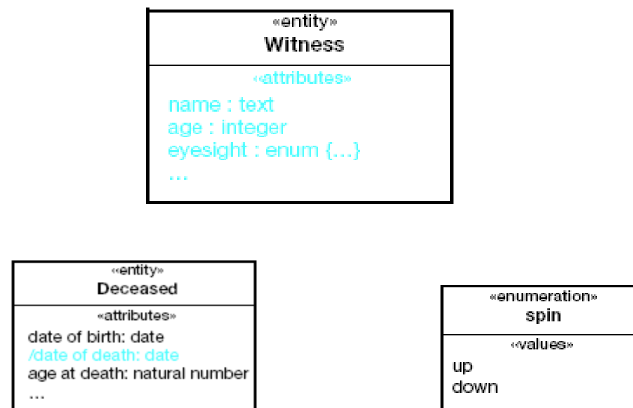
Assessing Entities

- Cohesion and Naming
- Relevance
- Identity
- Subject Matter Membership

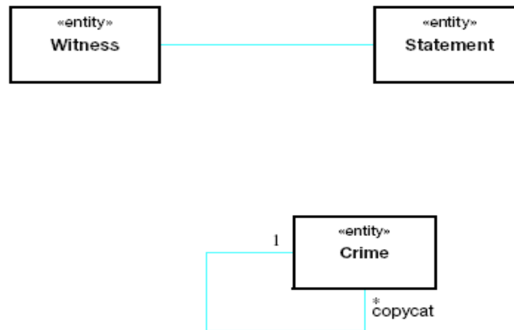
Secondary Model Elements: Properties and Connections

- Attributes
  - The relevant, measurable properties of the relevant entities
  - Intrinsic state
    - Remember state is our modeling friend
- Characterizing relationships
  - Association, aggregation and composition
  - Extrinsic state
    - State contributions from the identity of other entities rather than values

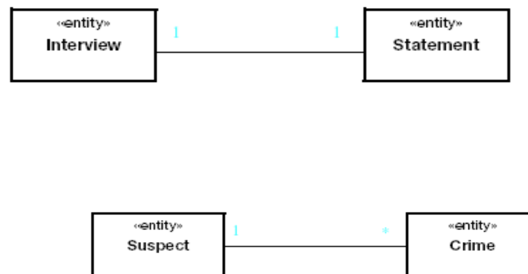
Depicting Attributes



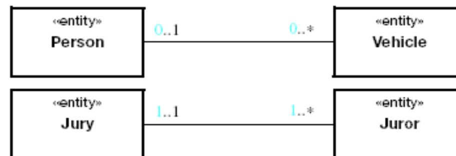
Depicting Associations



Multiplicity



More Multiplicity

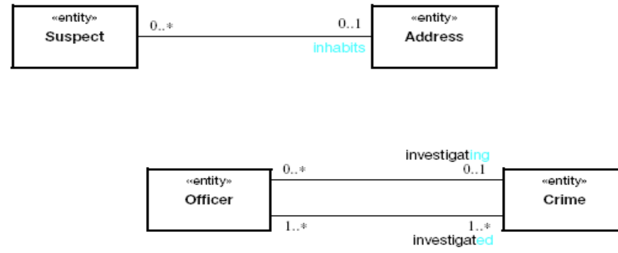


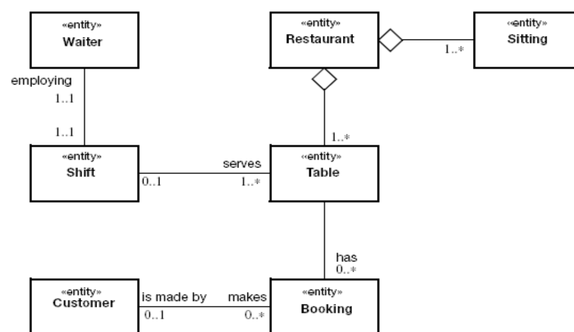
Many to many associations

We need to be careful with many-to-many associations for at least three reasons

- It's not obvious as to exactly what we mean by many-to-many
- They're difficult to implement
- They sometimes mask complimentary pairs of one-to-many associations

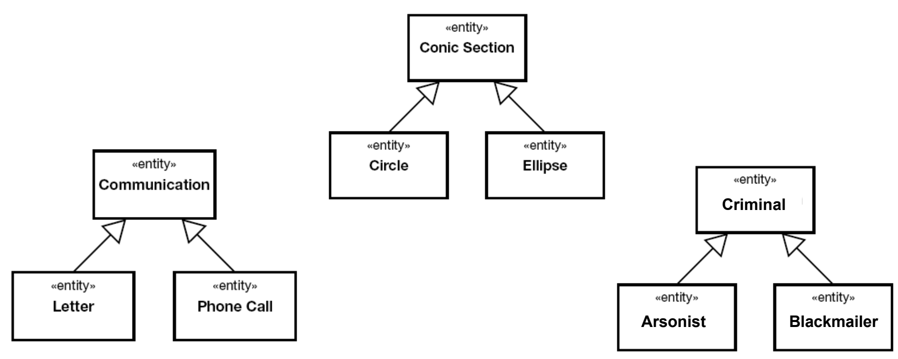
Roles





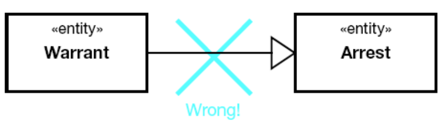
Generalizations

- Model entities that have features in common
  - Relationship represented by a open triangular arrowhead



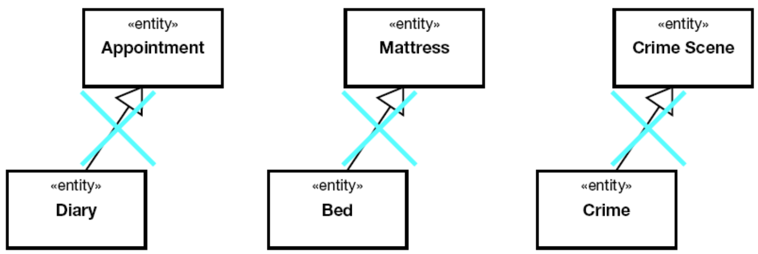
Incorrect Generalizations

- Fails the "is-a-kind-of" test



Should be an association (Warrant **leads to** arrest)

- Examples of "has-a" relationship

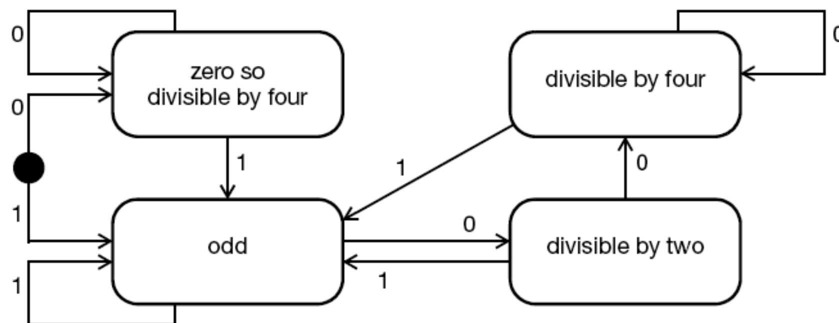


Should be aggregations

Basic State Machine Elements

- States
  - "history" or memory
- Events
  - the input
- Transition
  - events cause transitions from state to state
  - transition labeled by the event that caused the transition

Simple and Classic Example



The Elements of An Activity Diagram

- Actions
  - Like the steps of flowcharts
    - easily describable, fairly atomic thing that is to be done
  - Predefined and low level

- Flows
  - Of control – the stuff flowcharts
    - Which can be concurrent – the stuff of Petri nets
  - Of action-provoking data – the stuff of dataflow diagrams

First Model Summary

- We have greater, and consensus, understanding of the subject matter
- We have good ideas for types of object instances (but not yet for classes)
- We have good ideas for query services (but not yet for instance variables)
- We have good ideas for characterizing relationships
- We might have state transition constraints
- We have hints as to possible generalization and conformance structures

Design

- Requirements and analysis ensure you are building the right system
- Design ensures that you are building the system right
- A design is a plan for the system to be
- Complexity reduction, modularity, cohesion and coupling still major considerations

Possible Design Inputs

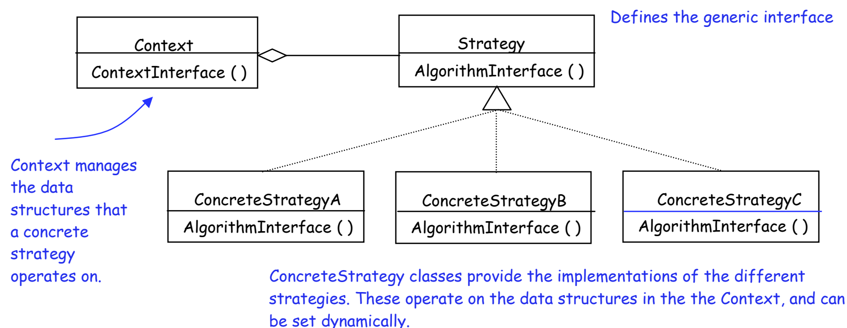
- Architecture proposals
- Technology choices
- Requirements
- Analysis models
- Design patterns
  - Strategy
  - Observer

Design Patterns

- You, as the designer have some understanding of design theory
  - "the peanut butter goes on FIRST"
- Stand on the shoulders of Giants
  - what works
  - what doesn't work and why
  - "don't put a rope handle on something you need to push"
- Young industry with rapidly changing technology
  - Design patterns represent the Giants

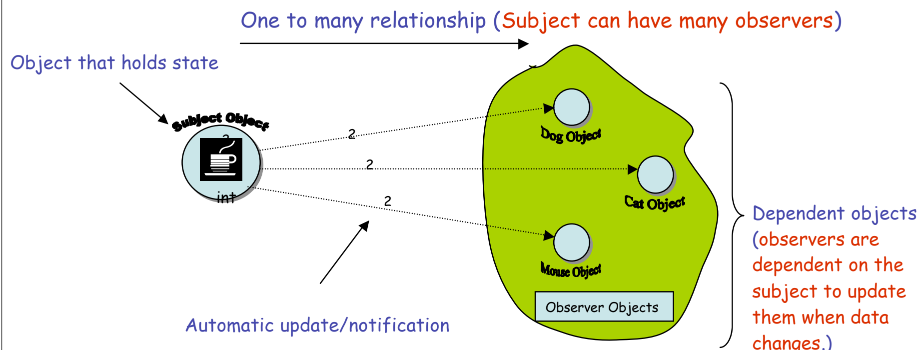
The Strategy Design Pattern

The **Strategy Design Pattern** defines a family of algorithms, encapsulates each one, and makes them interchangeable. Strategy lets the algorithms vary independently from the clients that use it.



The Observer Pattern Defined

The **Observer Pattern** defines a one-to-many dependency between objects so that when one object changes state, all of its dependents are notified and updated automatically.



Summary

- OO Basics
  - Abstraction
  - Encapsulation
  - Polymorphism
  - Inheritance
- OO Principles
  - Encapsulate what varies
  - Favour composition over inheritance
  - Program to interfaces not implementations
  - Loosely coupled designs between objects that interact
- OO Patterns
  - Strategy
  - Observer

Take away points

- OO basics alone don't make you a good OO designer
- Good OO designs are extensible, reusable and maintainable
- Patterns show how good OO designs have been created previously
- Patterns are the result of experience
- Patterns aren't code, they are general approaches
- Patterns are discovered,not invented
- Most patterns address issues of change in software

…Outside-In Design…

Design Goals (order is important):

- Cohesive and loosely-coupled objects
- in the right place
- exhibiting the right interfaces
- to the right clients
- defined/made by cohesive, loosely-coupled classes
- holding the right mixture of cohesive and loosely-coupled methods
- supported by sensible instance variables
- normalized via self-messaging
- normalized via component objects
- perhaps normalized via implementation inheritance

normalized means that a feature must be presented:

- in one place
- fully at that place
- and only at that place

Inheritance and Polymorphism

- Inheritance:
  - base class defined, and other classes derived from it
  - Code for the base class used for base objects, as well as objects of any derived classes
- Polymorphism:
  - Associate many meanings to one method name
  - changes made to method definitions in the derived classes
  - *changes apply to the software written for the base class*

CRC

"Class" Responsibility Collaboration

- Origins
  - Yet another excellent invention of Cunningham and Beck; at yet another OOPSLA
  - Interesting, among many other reasons, because it wasn't based on some older, legacy technique
- Still neglected and under-promoted
  - Because there is no UML diagram called a CRC diagram?