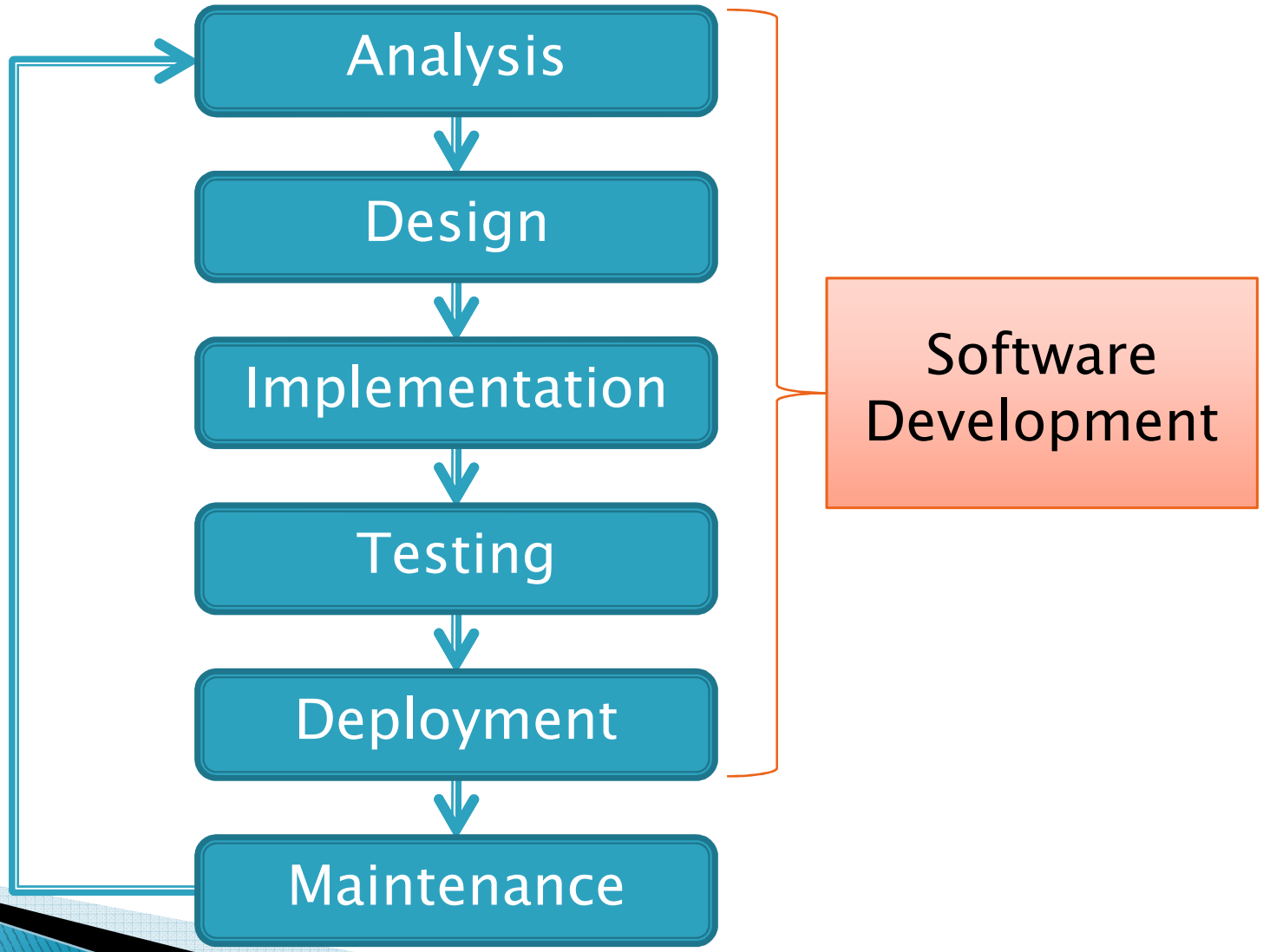
Review: Cohesion and Coupling, Mutable, Inheritance
Screen Layouts
Software methodologies – Extreme Programming
Object-Oriented Design – CRC Cards
– UML class diagrams

Software Development Methods

Software Life Cycle



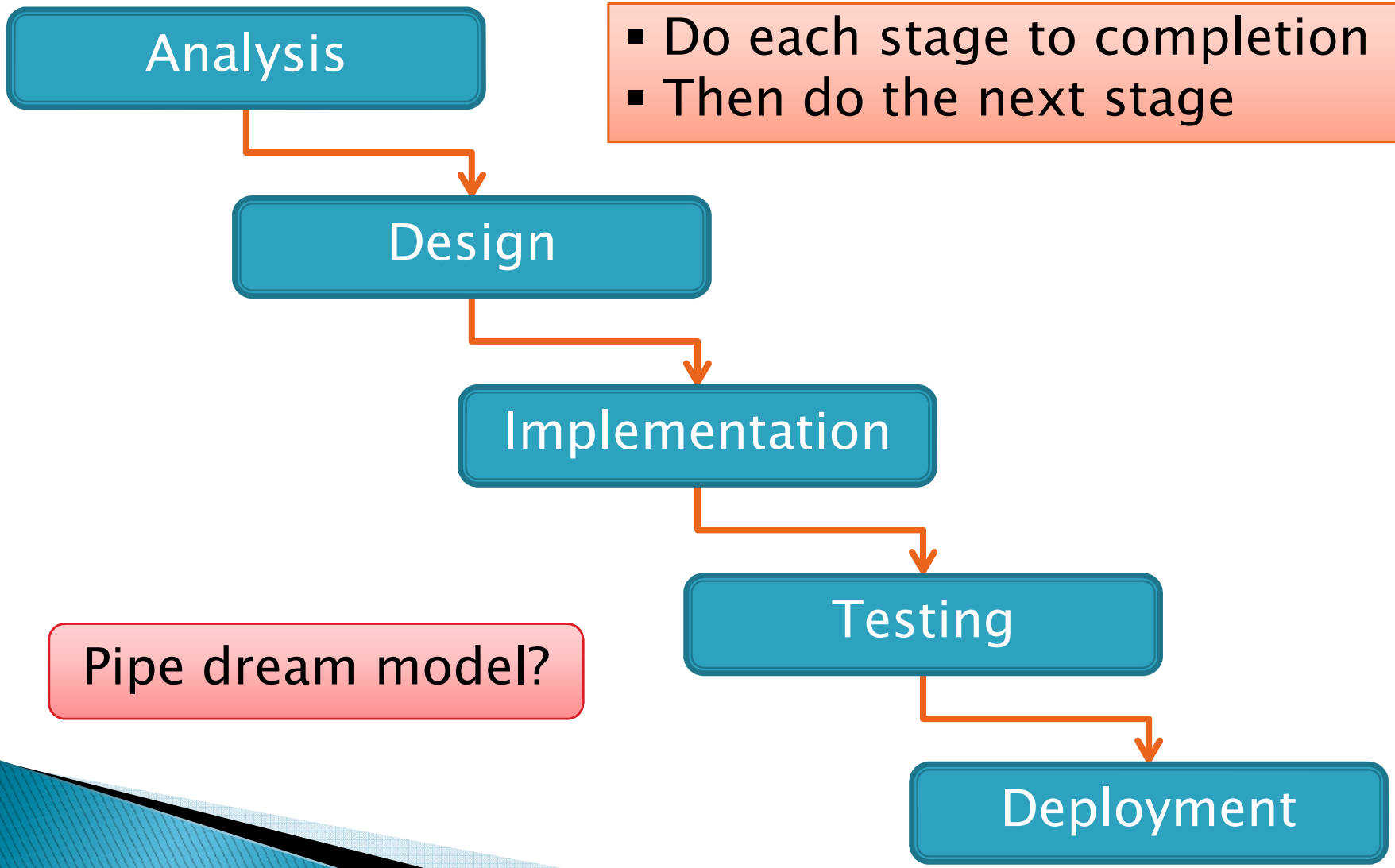
Formal Development Processes

- Standardized approaches intended to:
  - Reduce costs
  - Increase predictability of results

- Examples:
  - Waterfall model
  - Spiral model
  - “Rational Unified Process”

Waterfall Model

- Do each stage to completion
- Then do the next stage

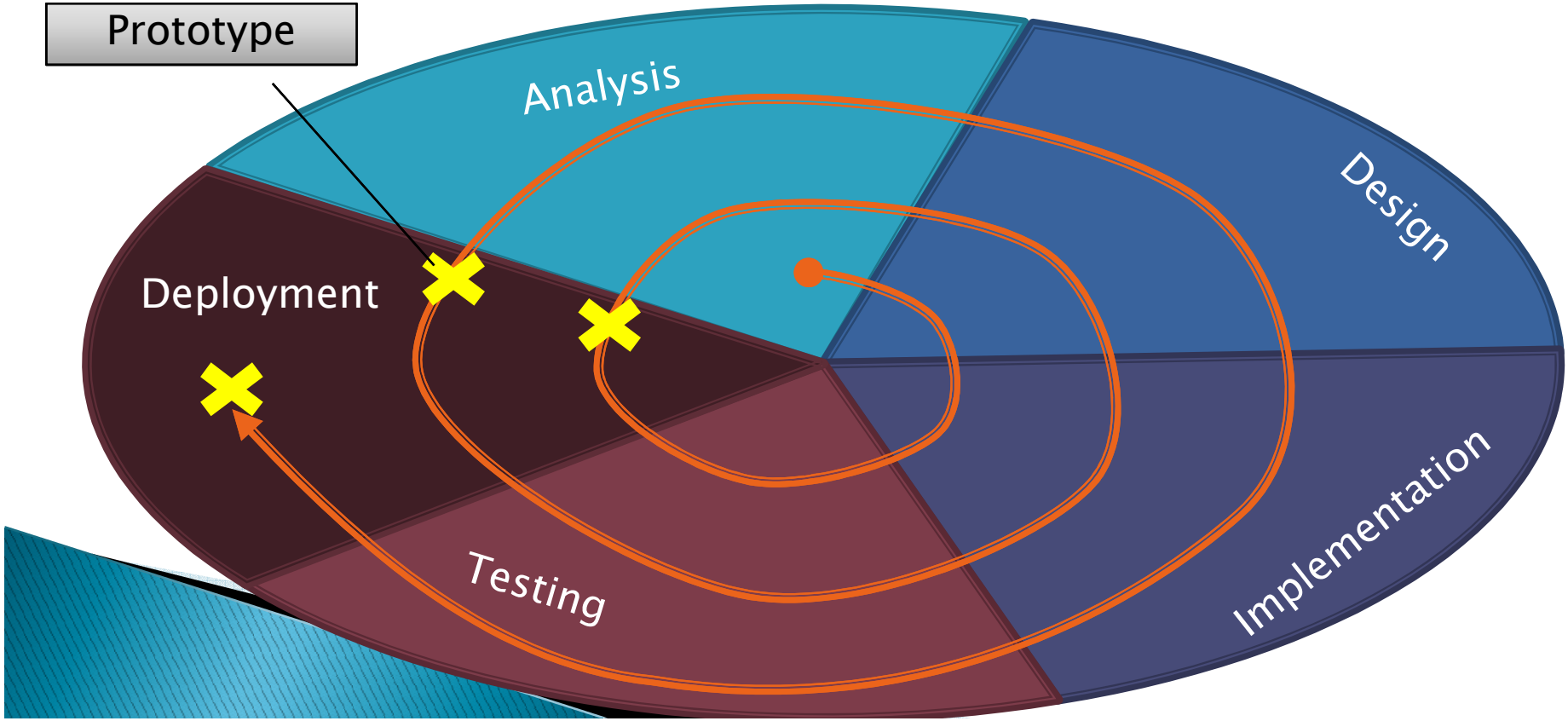


Pipe dream model?

Spiral Model

- Schedule overruns
- Scope creep

- Repeat phases in a cycle
- Produce a prototype at end of each cycle
- Get early feedback, incorporate changes



Extreme Programming—XP

- Like the spiral model with very short cycles
- Pioneered by Kent Beck
- One of several “agile” methodologies, focused on building high quality software quickly
- Rather than focus on rigid process, XP espouses 12 key practices…

The XP Practices

- Realistic planning
- Small releases
- Shared metaphors
- Simplicity
- Testing
- Refactoring
- Pair programming
- Collective ownership
- Continuous integration
- 40-hour week
- On-site customer
- Coding standards

When you see opportunity to make code better, do it

Use descriptive names

Q1

What is good object-oriented design?

- It starts with good classes…

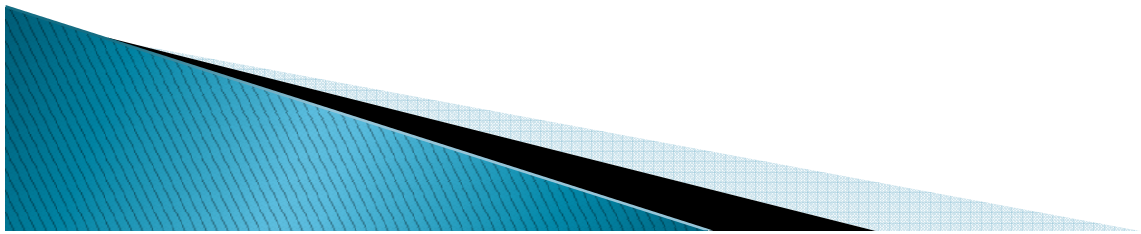
Good Classes Typically

- Often come from nouns in the problem description
- May…
  - Represent single concepts
    - `Circle`, `Investment`
  - Be abstractions of real-life entities
    - `BankAccount`, `TicTacToeBoard`
  - Be actors
    - `Scanner`, `CircleViewer`
  - Be utilities
    - `Math`

Q2

What Stinks? Bad Class Smells

- Can't tell what it does from its name
  - **PayCheckProgram**
- Turning a single action into a class
  - **ComputePaycheck**
- Name isn't a noun
  - **Interpolate**, **Spend**



Q3

Analyzing Quality of Class Design

- Cohesion
- Coupling

Cohesion

- A class should represent a single concept
- Public methods and constants should be cohesive
- Which is more cohesive?

CashRegister
double NICKEL_VALUE double DIME_VALUE double QUARTER_VALUE
void add(int nickels, int dimes, int quarters) …

CashRegister
void add(ArrayList<Coin> coins) …

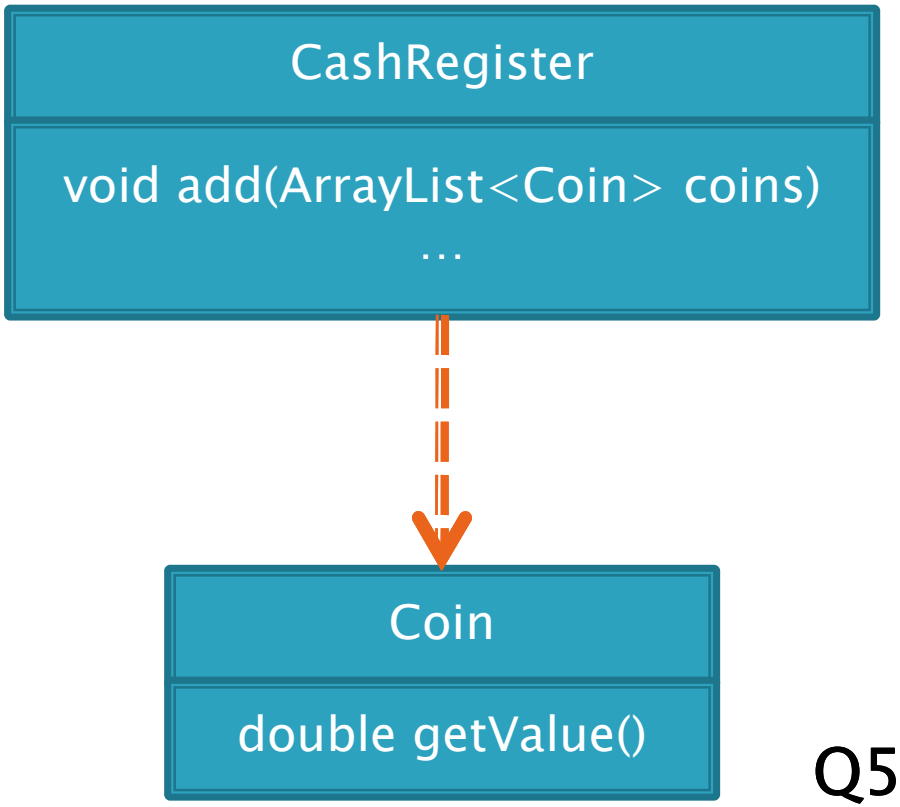
Coin
double getValue()

Q4

Dependency Relationship

- When one classes requires another class to do its job, the first class depends on the second

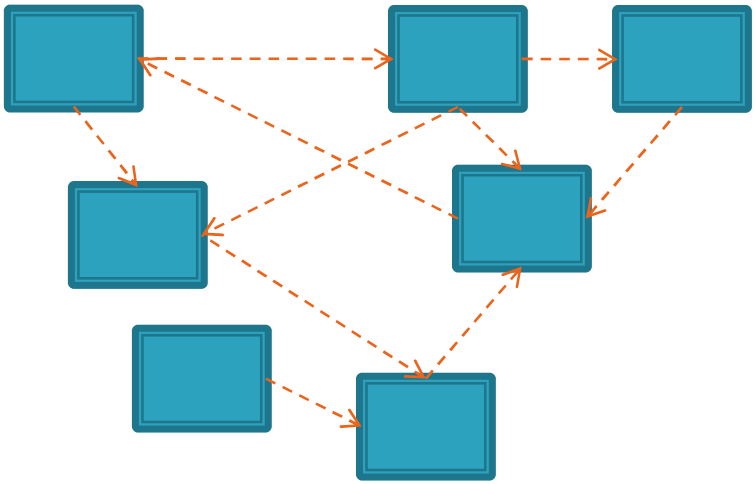
- Shown on UML diagrams as:
  - dashed line
  - with open arrowhead

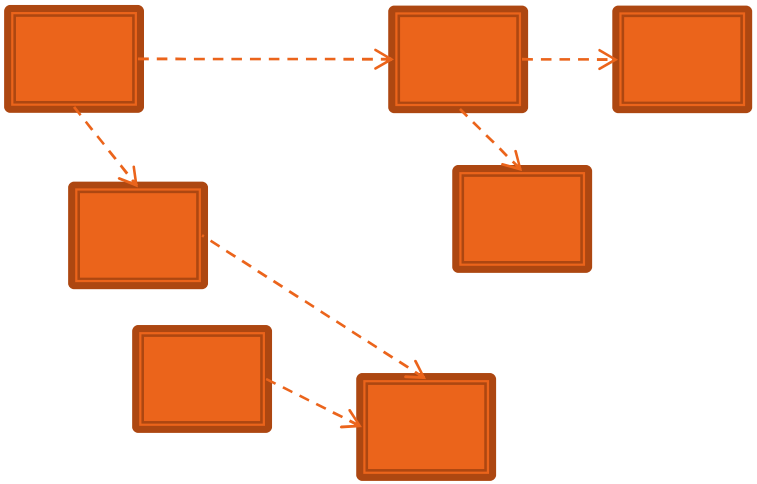


Q5

Coupling

- Lots of dependencies == high coupling
- Few dependencies == low coupling





- Which is better? Why?

Q6

Quality Class Designs

- High cohesion
- Low coupling
- Immutable where practical, document where not
- Inheritance for code reuse
- Interfaces to allow others to interact with your code

Object-Oriented Design

- A practical technique

Object-Oriented Design

- We won't use full-scale, formal methodologies
  - Those are in later SE courses
- We will practice a common object-oriented design technique using CRC Cards
- Like any design technique,
  the key to success is practice

Key Steps in Our Design Process

1. Discover classes based on requirements

2. Determine responsibilities of each class

3. Describe relationships between classes

Q2

Discover Classes Based on Requirements

- Brainstorm a list of possible classes
  - Anything that might work
  - No squashing
- Prompts:
  - Look for nouns
  - Multiple objects are often created from each class → so look for plural concepts
  - Consider how much detail a concept requires:
    - A lot? Probably a class
    - Not much? Perhaps a primitive type
- Don't expect to find them all → add as needed

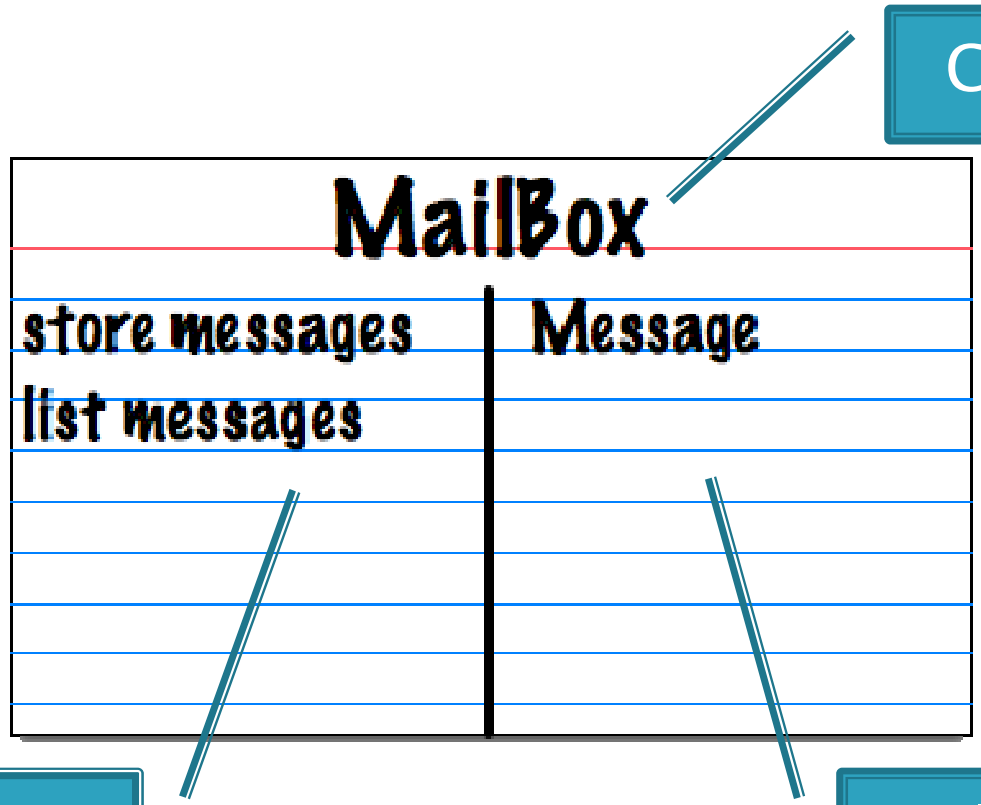
Tired of hearing this yet?

Determine Responsibilities

- Look for verbs in the requirements to identify responsibilities of your system
- Which class handles the responsibility?
- Can use CRC Cards to discover this:
  - Classes
  - Responsibilities
  - Collaborators

CRC Cards

- Use one index card per class



Class name

Responsibilities

Collaborators

CRC Card Technique

1. Pick a responsibility of the program
2. Pick a class to carry out that responsibility
   - Add that responsibility to the class's card
3. Can that class carry out the responsibility by itself?
   - Yes → Return to step 1
   - No →
     - Decide which classes should help
     - List them as collaborators on the first card
     - Add additional responsibilities to the collaborators' cards

CRC Card Tips

- Spread the cards out on a table
  - Or sticky notes on a whiteboard instead of cards
- Use a "token" to keep your place
  - A quarter or a magnet
- Focus on high-level responsibilities
  - Some say < 3 per card
- Keep it informal
  - Rewrite cards if they get to sloppy
  - Tear up mistakes
  - Shuffle cards around to keep "friends" together

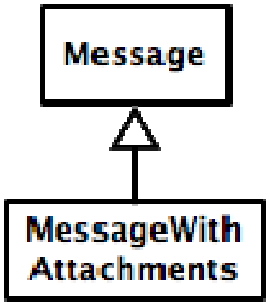
Describe the Relationships

- Classes usually are related to their collaborators
- Draw a UML class diagram showing how
- Common relationships:
  - Inheritance: only when subclass is a special case
  - Aggregation: when one class has a field that references another class
  - Dependency: like aggregation but transient, usually for method parameters, "has a" temporarily
  - Association: any other relationship, can label the arrow, e.g., constructs

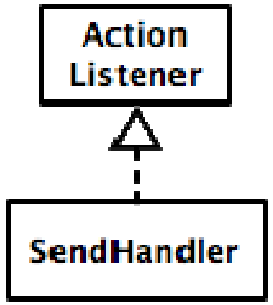
NEW!

Summary of UML Class Diagram Arrows

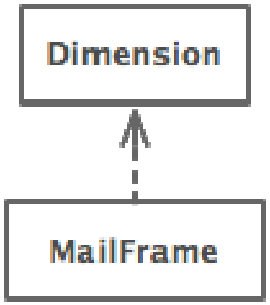
Inheritance (is a)



Interface Implementation (is a)



Dependency (depends on)



Aggregation (has a)

Association

Q4