codemanship

Intensive S.O.L.I.D.

Jason Gorman

codemanship

Today…

- Before We Begin - Refactoring Foundations
- Goals of OO Design
- Dependencies
  - Efferent & Afferent Couplings
  - Dependency Analysis
  - 4 Rules of Dependency Management
- Basic Design Principles
  - Simple Design
  - Don't Repeat Yourself
  - Tell, Don't Ask
- Class Design Principles
  - Single Responsibility
  - Open-Closed
  - Liskov Substitution
  - Interface Segregation
  - Dependency Inversion
- Complimentary Design Principles
  - Law of Demeter

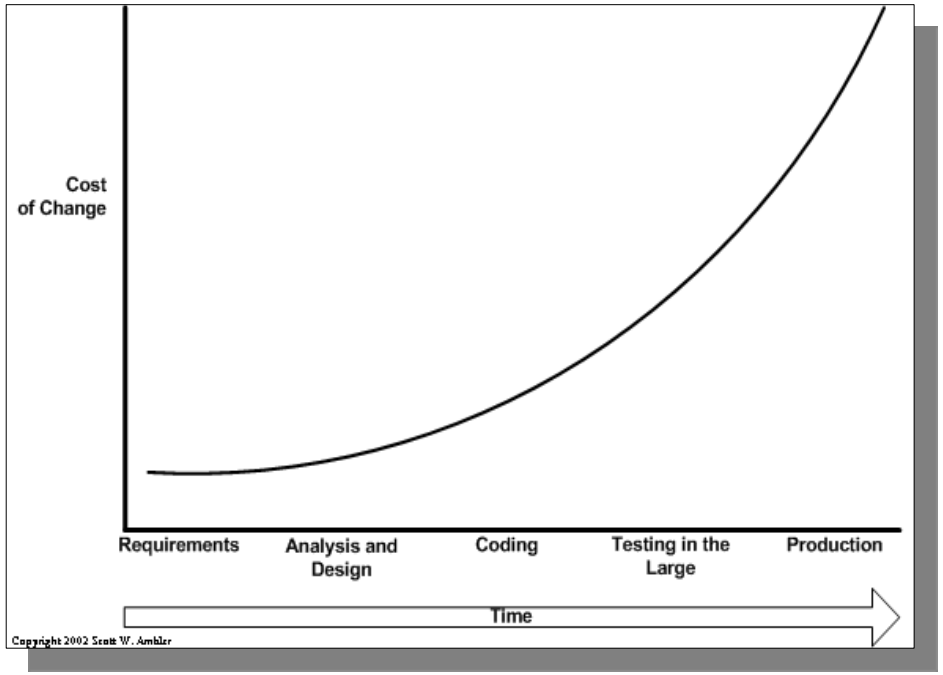
codemanship

Before We Begin…

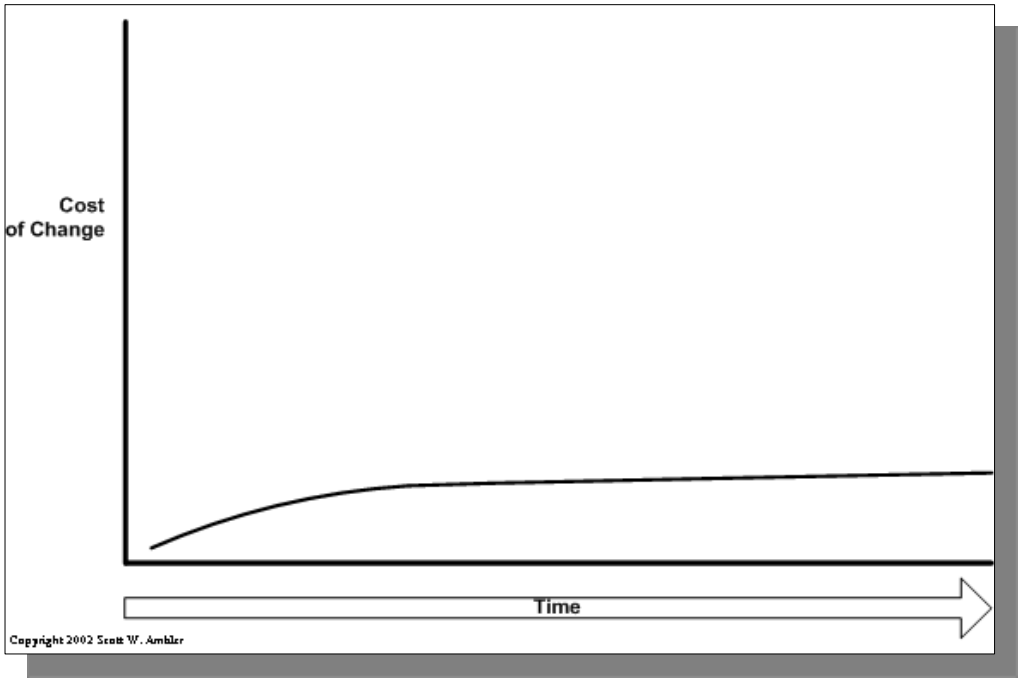
REFACTORING FOUNDATIONS

codemanship

Cost Of Change



Source:Scott W. Ambler, agilemodeling.com



Source: Kent Beck, Extreme Programming Explained

What Makes Code Harder To Change?

codemanship

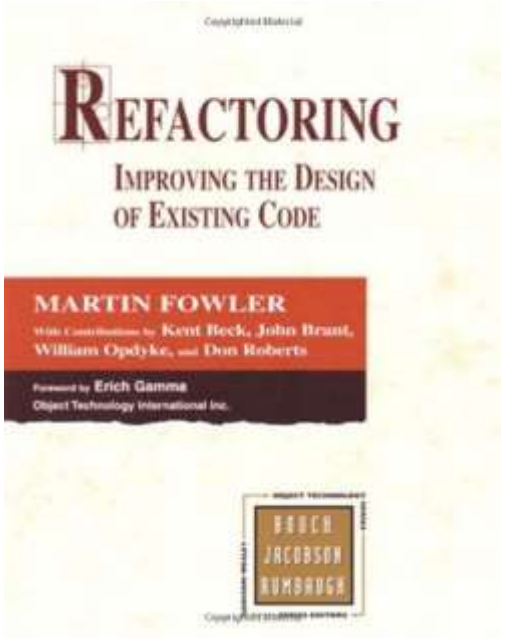
Readability

Complexity

Duplication

Dependencies & The “Ripple Effect”

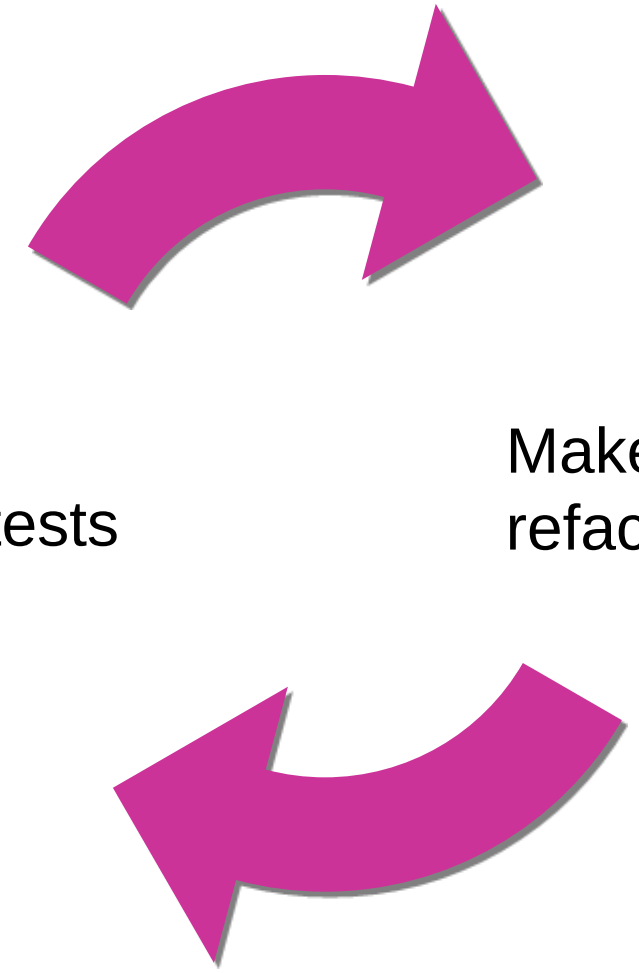
Refactoring is…



…improving the design of existing code
without changing what it does

codemanship

Refactoring Process



codemanship

Code Smells

…are indications of *increasing entropy* in your code

- Increasing complexity
- Increasing duplication
- Increasing dependency issues
- Decreasing comprehensibility

- As time goes on, code can become *rigid* and *brittle*

Classes Of Code Smell

Complexity	Responsibility Problems	Couplers
•**Long Method** •**Large Class** •Primative Obsession •**Data Clumps** •**Long Parameter Lists**	•**Divergent Change** •Shotgun Surgery •**Data Class**	•**Feature Envy** •**Message Chains** •**Inappropriate Intimacy** •Middle Man

OO Abuses	Redundancy
•**Switch Statements** •Temporary Field •Refused Bequest •Alternative Classes With Different Interfaces •Parallel Inheritance Heirarchies	•Lazy Class •**Duplicate Code** •Dead Code •Speculative Generality

codemanship

Long Methods

Extract Method

Decompose Conditional

Duplicate Code (Simple)

Extract Method

Extract Class

Good Refactoring Habits

- Run the tests after every refactoring
- If refactoring fails, undo/roll back
- Identify code smells and use appropriate refactorings
- Refactor directly to well-defined goals
- Check-in after every refactoring goal is met
- Never refactor on a red light
- Use automated refactorings whenever possible
- Do one refactoring at a time

Then let’s begin…

OBJECT ORIENTED DESIGN PRINCIPLES

codemanship

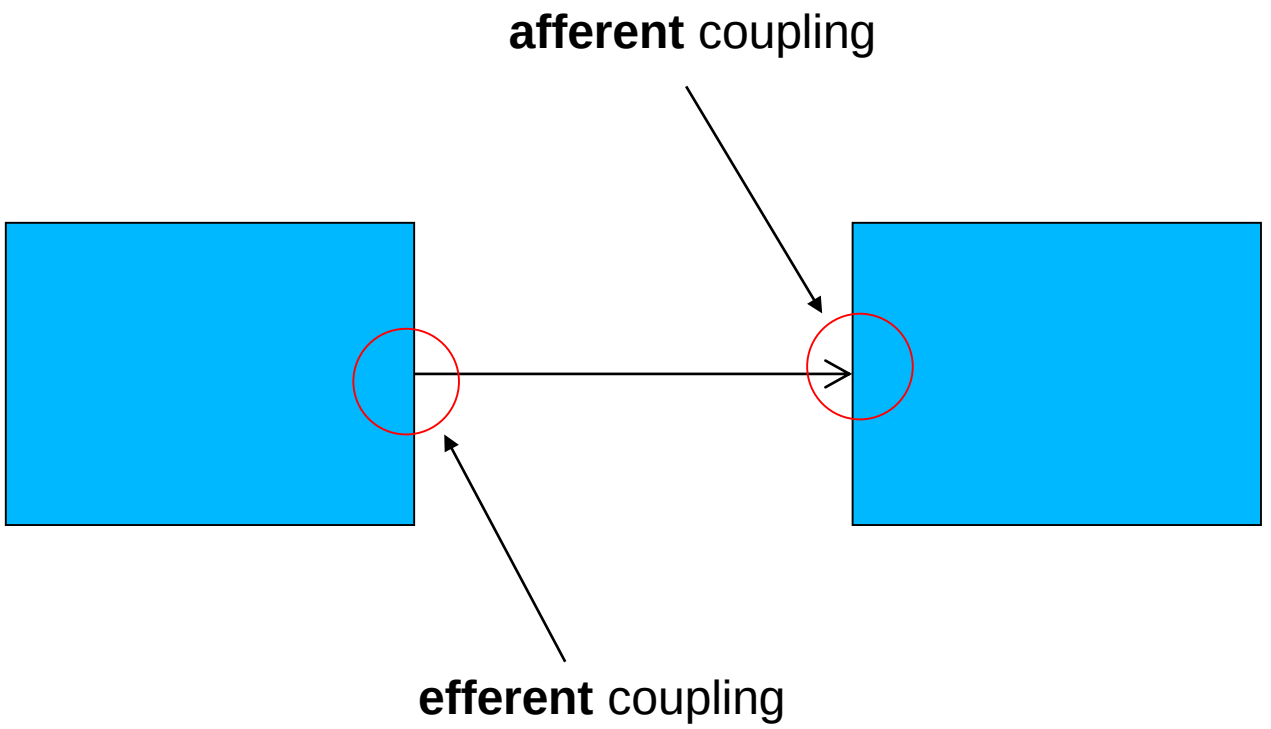
http://www.codemanship.co.uk/files/solid.zip

codemanship

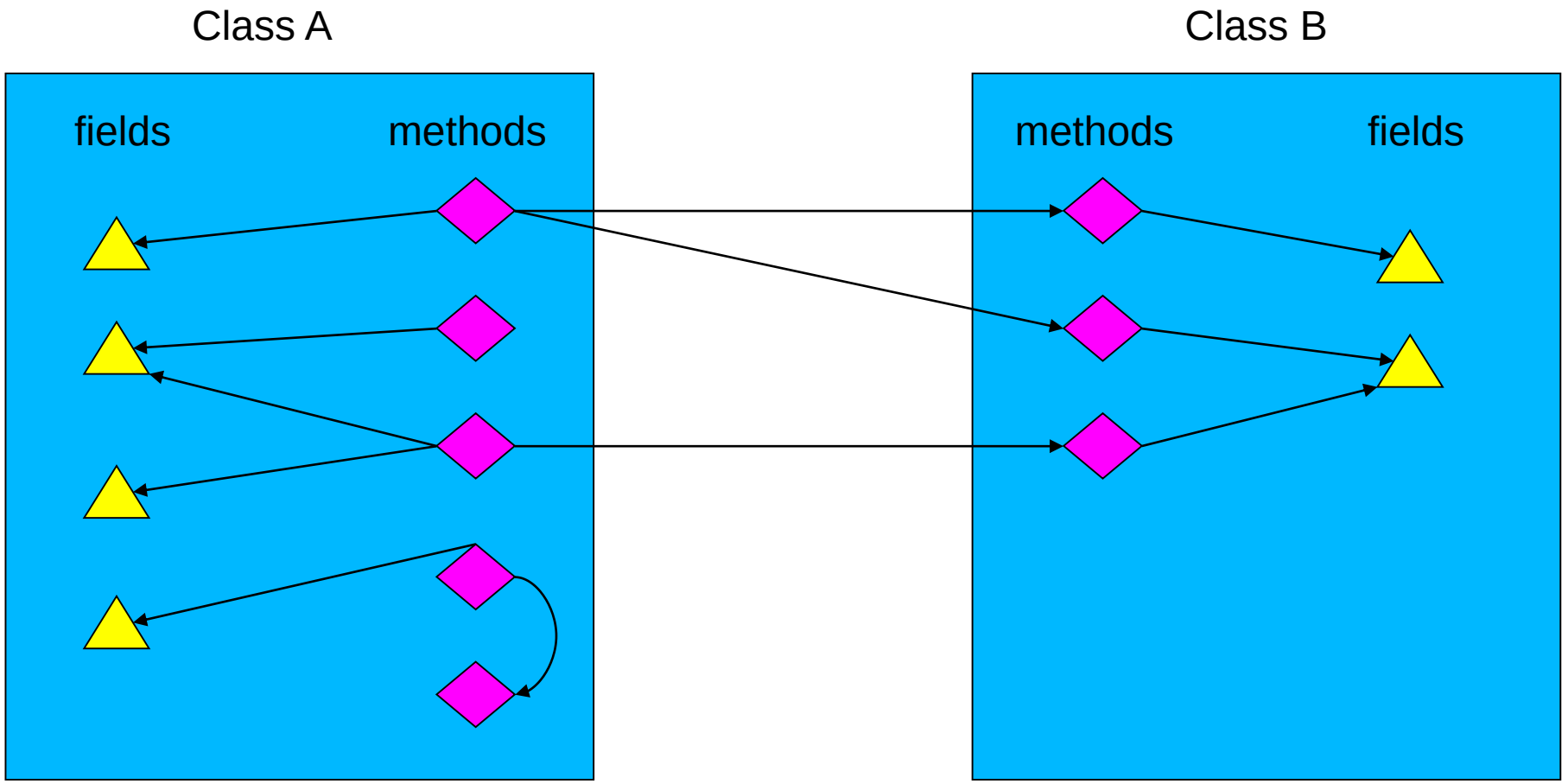
Goals Of OO Design

By packaging data, function and interfaces together, what do we hope to achieve?

Dependencies

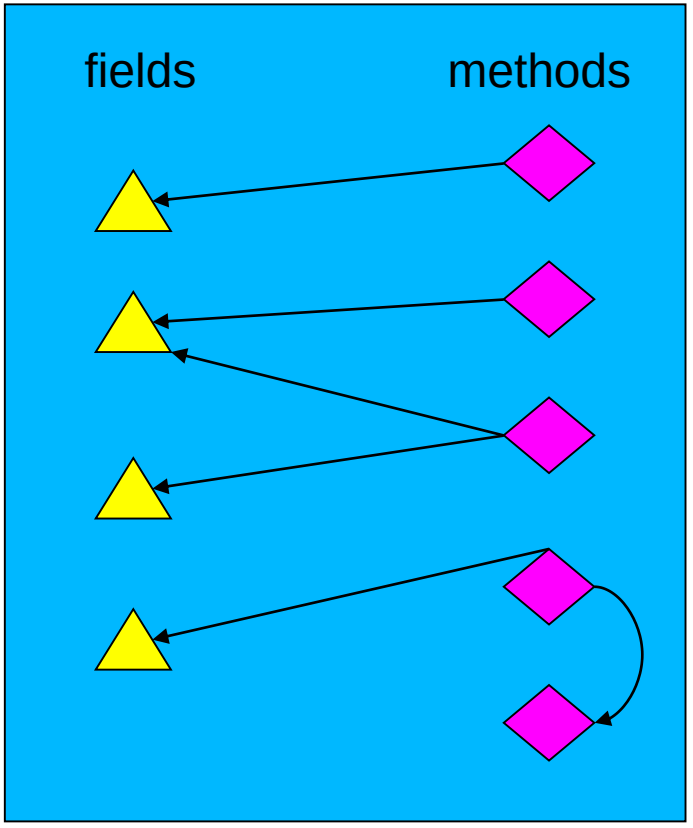


Dependency Analysis



codemanship

Class Cohesion



No. of internal relationships = 6

No. of methods* = 5

Average relationships per method = 1.2

** Including explicitly-declared and implied constructors*

codemanship

Jason's 4 Rules Of Dependency Management

1. **Minimise** dependencies
    - Introduce less dependencies overall
2. **Localise** dependencies
    - Package dependent things together (methods, fields, classes)
3. **Stabilise** dependencies
    - Depend on things that are less likely to change
4. **Abstract** dependencies
    - Depend on things that are easier to substitute

Basic Design Principles

codemanship

Simple Design

Less Code means Less Dependencies

Don’t Repeat Yourself

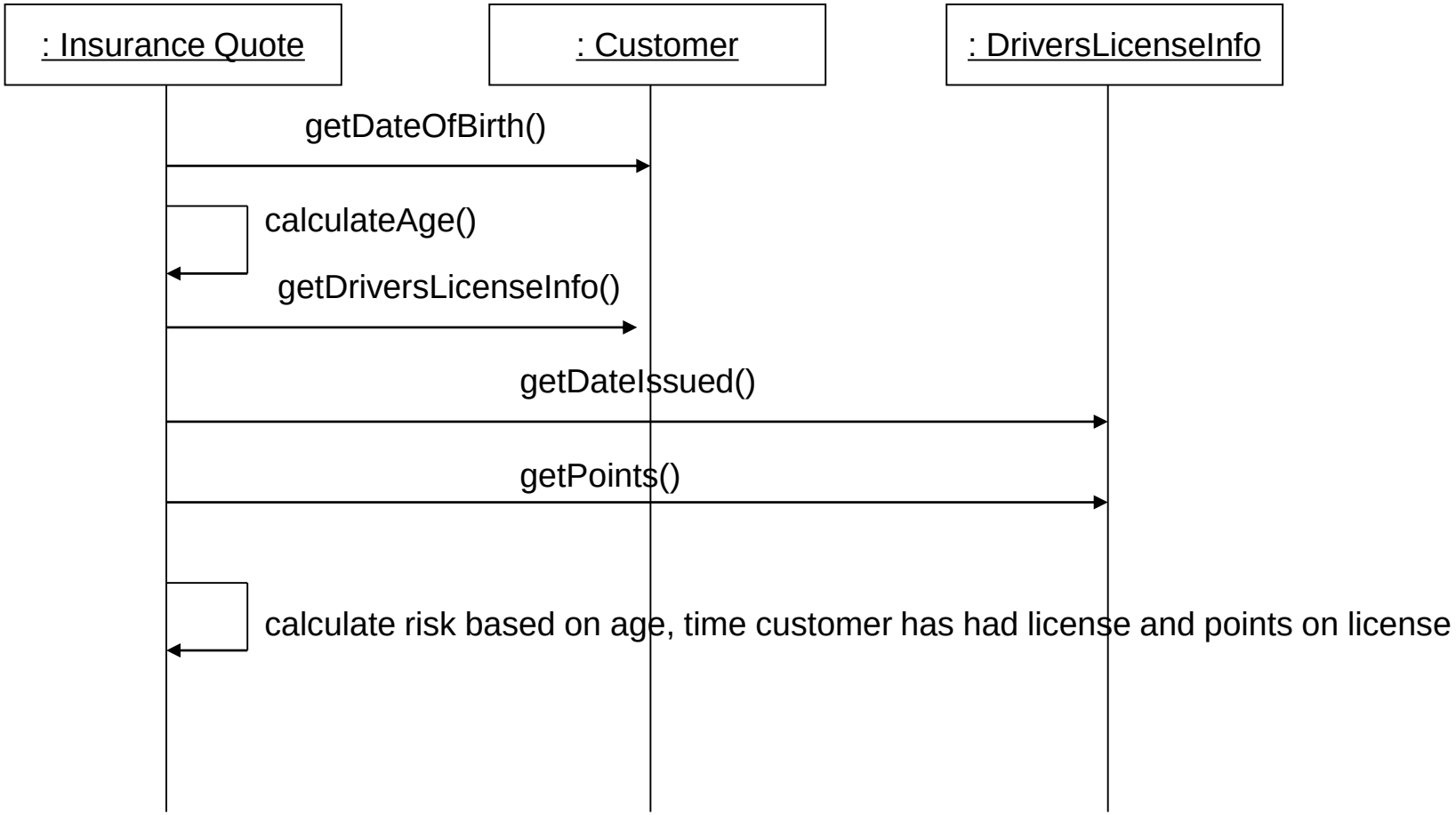
Duplicated Code means Duplicated Dependencies

codemanship

Tell, Don’t Ask

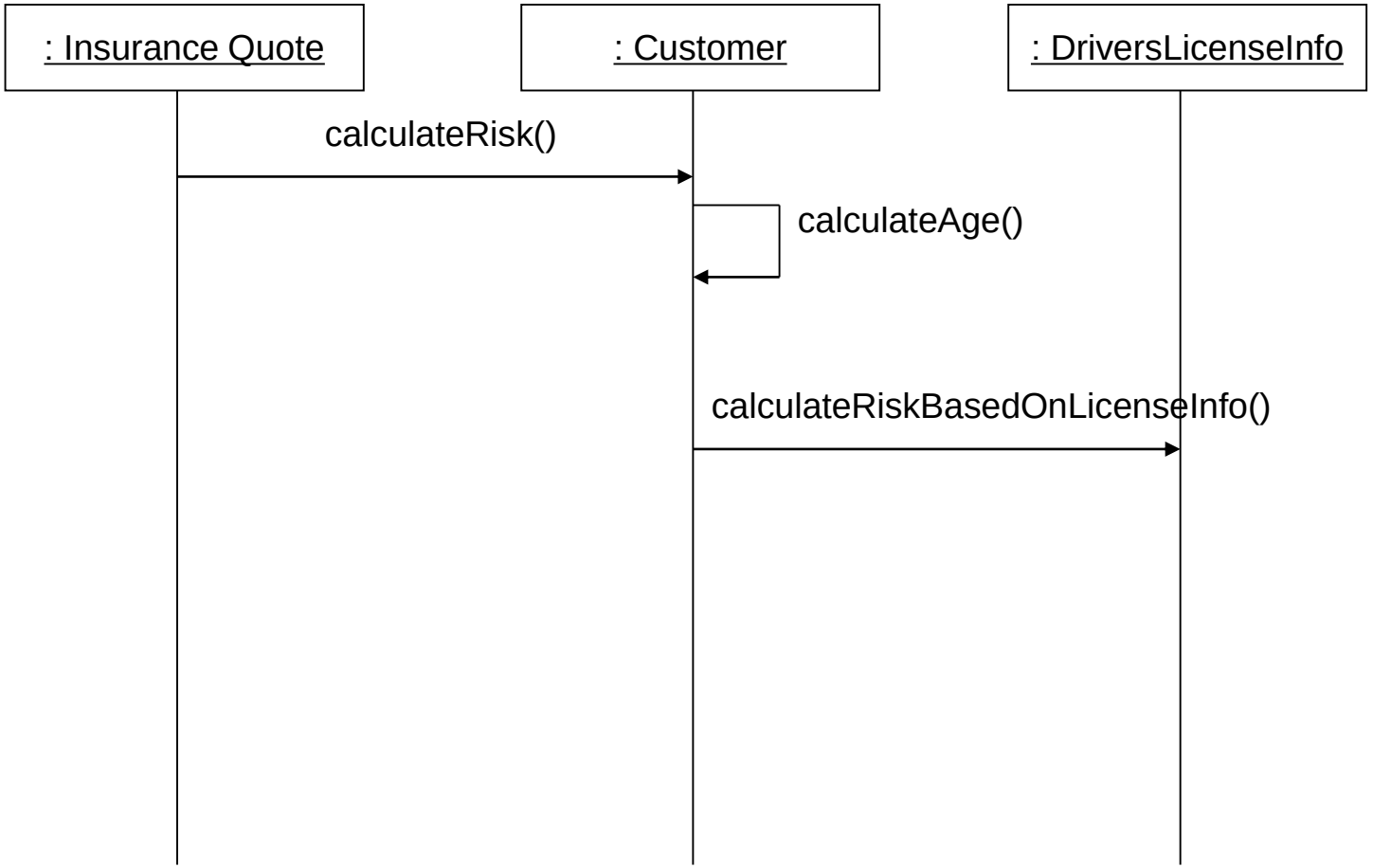
codemanship

Data-driven Design



codemanship

Tell, Don’t Ask



Ex 1 - Tell, Don’t Ask

- Analyse the dependencies in the source folder/project for Tell, Don’t Ask
  - What is the average class cohesion?
  - What are the average efferent and afferent couplings per class?
- Refactor the code to reduce couplings and improve class cohesion

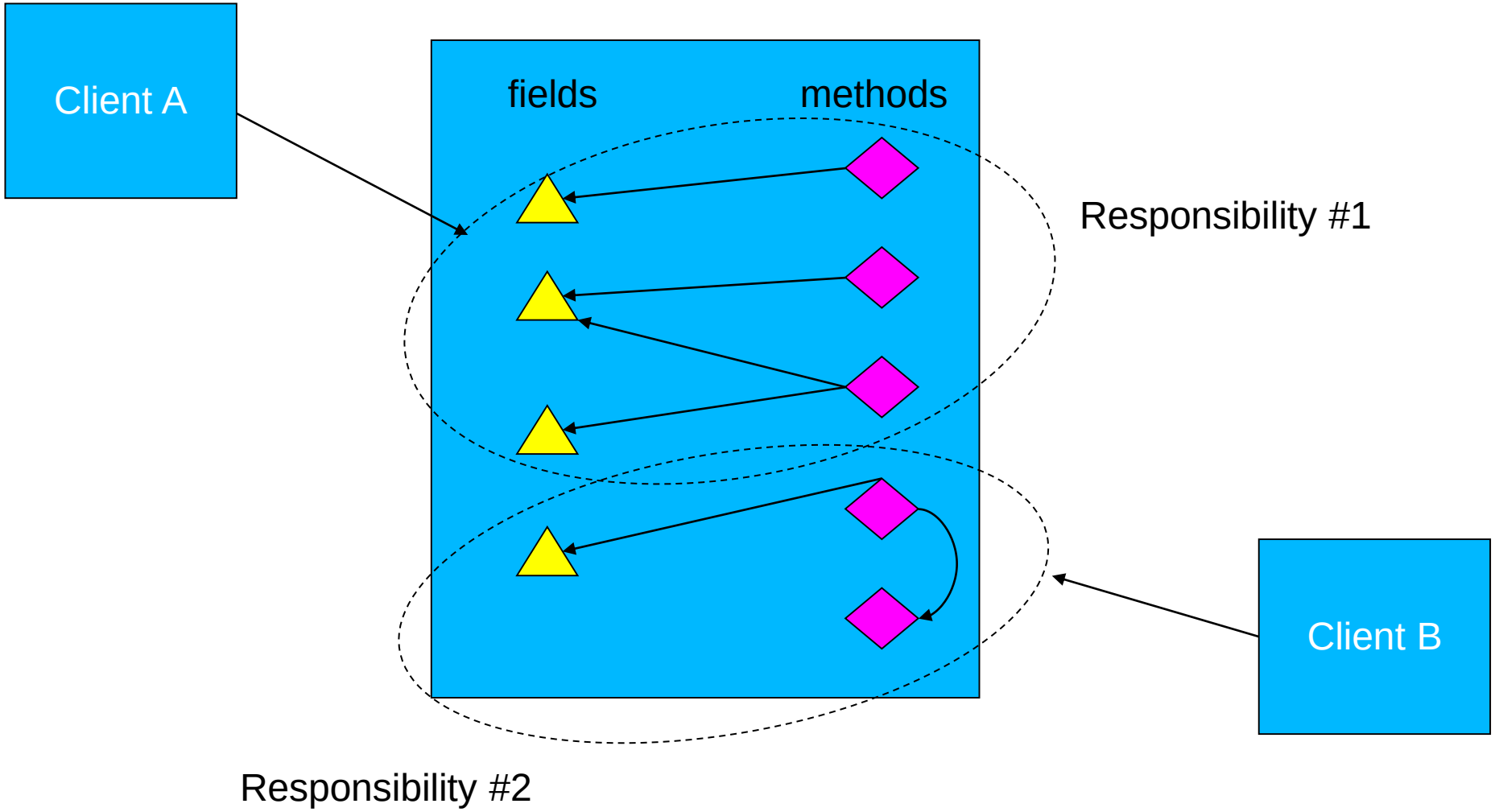
Class Design Principles

codemanship

Single Responsibility

Classes Should Have Only One Reason To Change

Why? - #1



codemanship

Why? - #2

AB
CD
AB.CD
CD.AB

A
B
C
D
A.B
A.C
A.D
A.B.C
… etc

Ex 2 – Single Responsibility

- Identify the responsibilities of classes in the Single Responsibility source folder/project
- Refactor the code so that classes have only on responsibility

Open-Closed

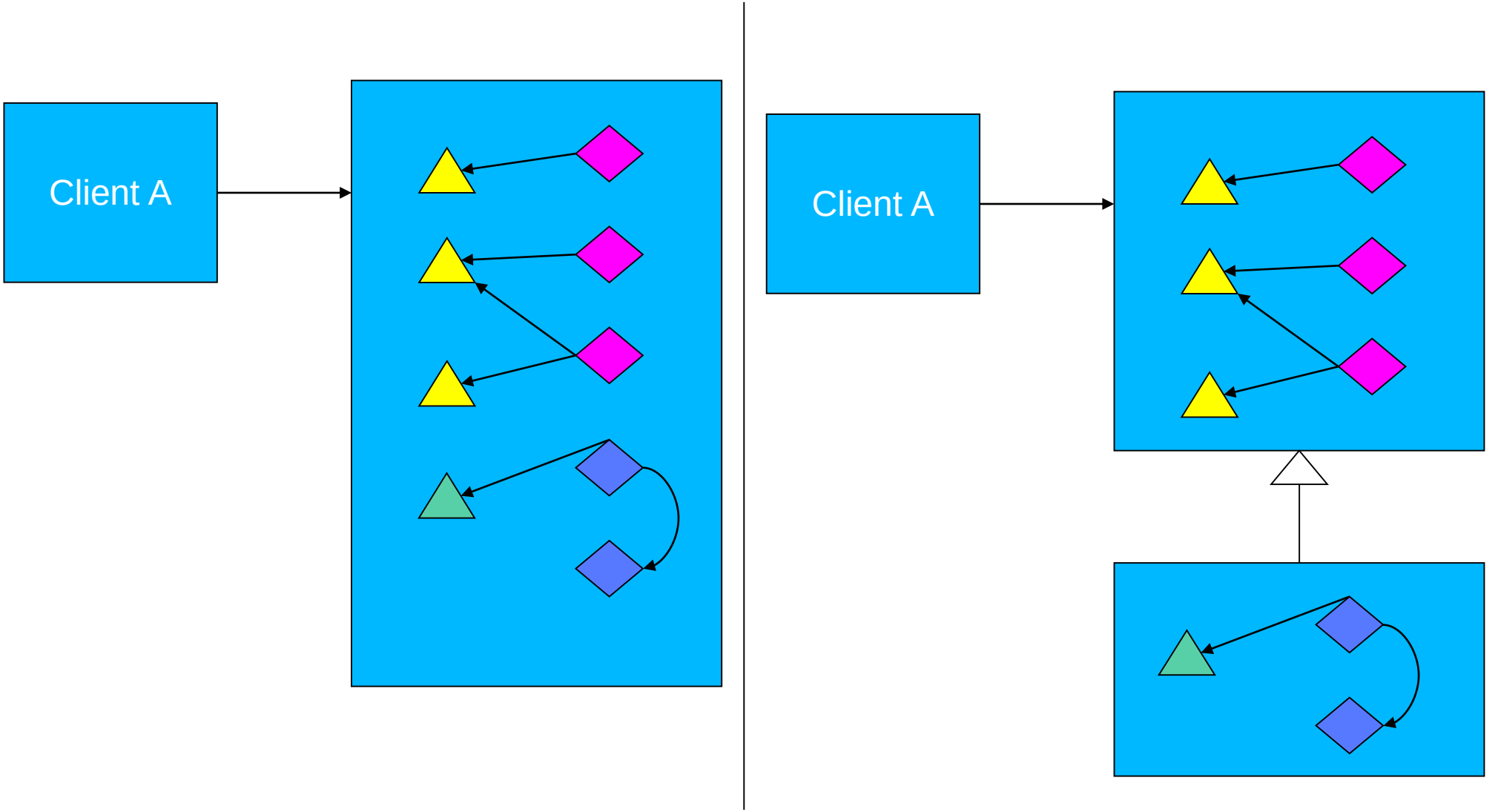
Classes should be…

Open to Extension

Closed to Modification

codemanship

Why?



codemanship

Ex 3 – Open-Closed

- Add functionality to the code in the Open-Closed source project/folder to ensure customers are of legal age to rent videos with the following classifications:
  - 18
  - 15
  - 12
  - U

Liskov Substitution

An instance of a class can be substituted with an instance of any of its subclasses

codemanship

Ex 4 – Liskov Substitution

- Extend BankAccount in the Liskov Substitution source folder/project to allow customers to draw against an agreed overdraft limit
- Modify the tests to ensure that the subclass of BankAccount passes the tests for a BankAccount as well as tests for an account with an overdraft limit

Interface Segregation

Captain
- toss coin
- rally team
- say “at the end of the day” in post-match interviews

Employee
- correct boss’s mistakes
- attend appraisal
- request raise

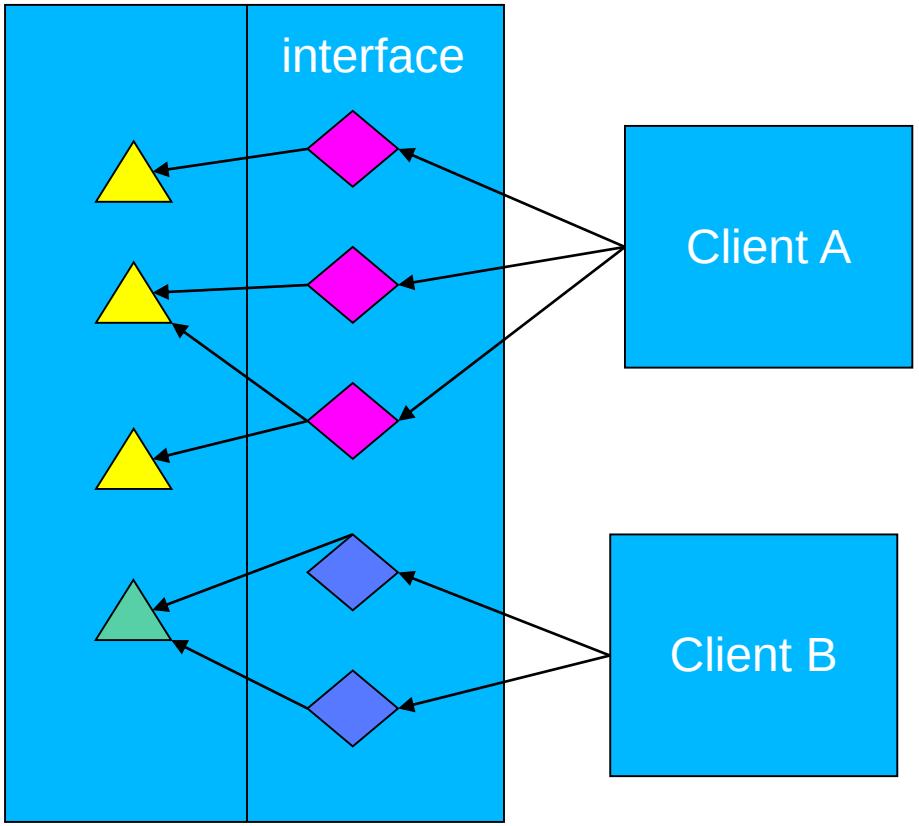
Mother
- feed
- bath
- clothe
- do school run
- make nativity costume
- take to doctor
- etc etc etc

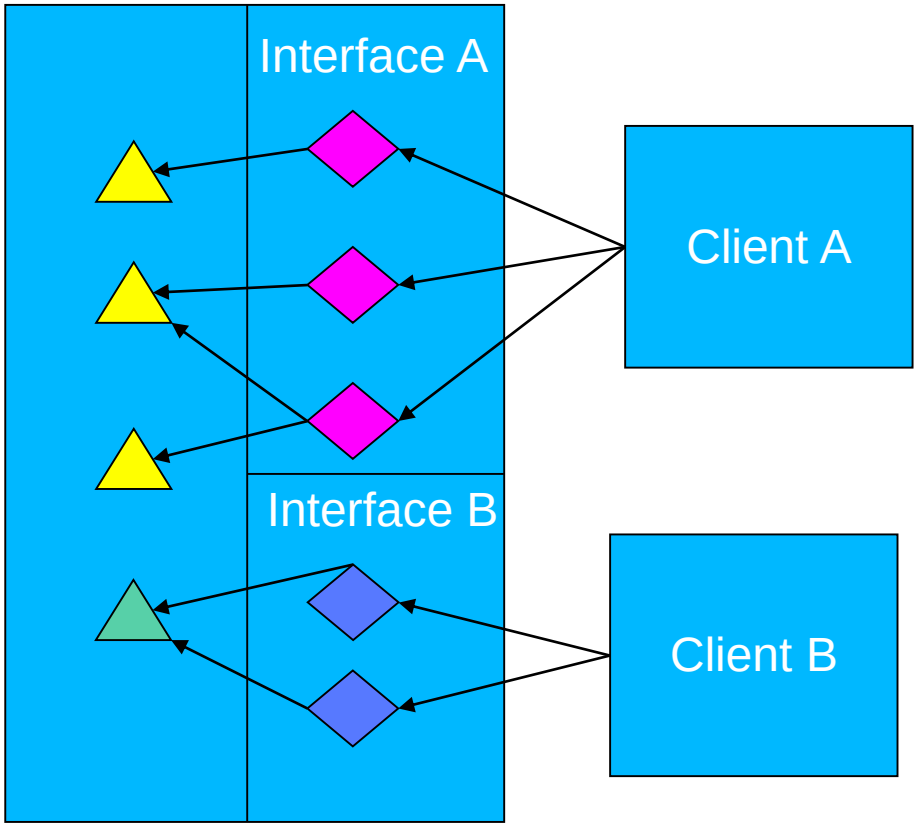
Wife
- put up with Xbox sessions
- invite fat mother for weekend

Classes should present client-specific interfaces

codemanship

Why?





codemanship

Ex 5- Interface Segregation

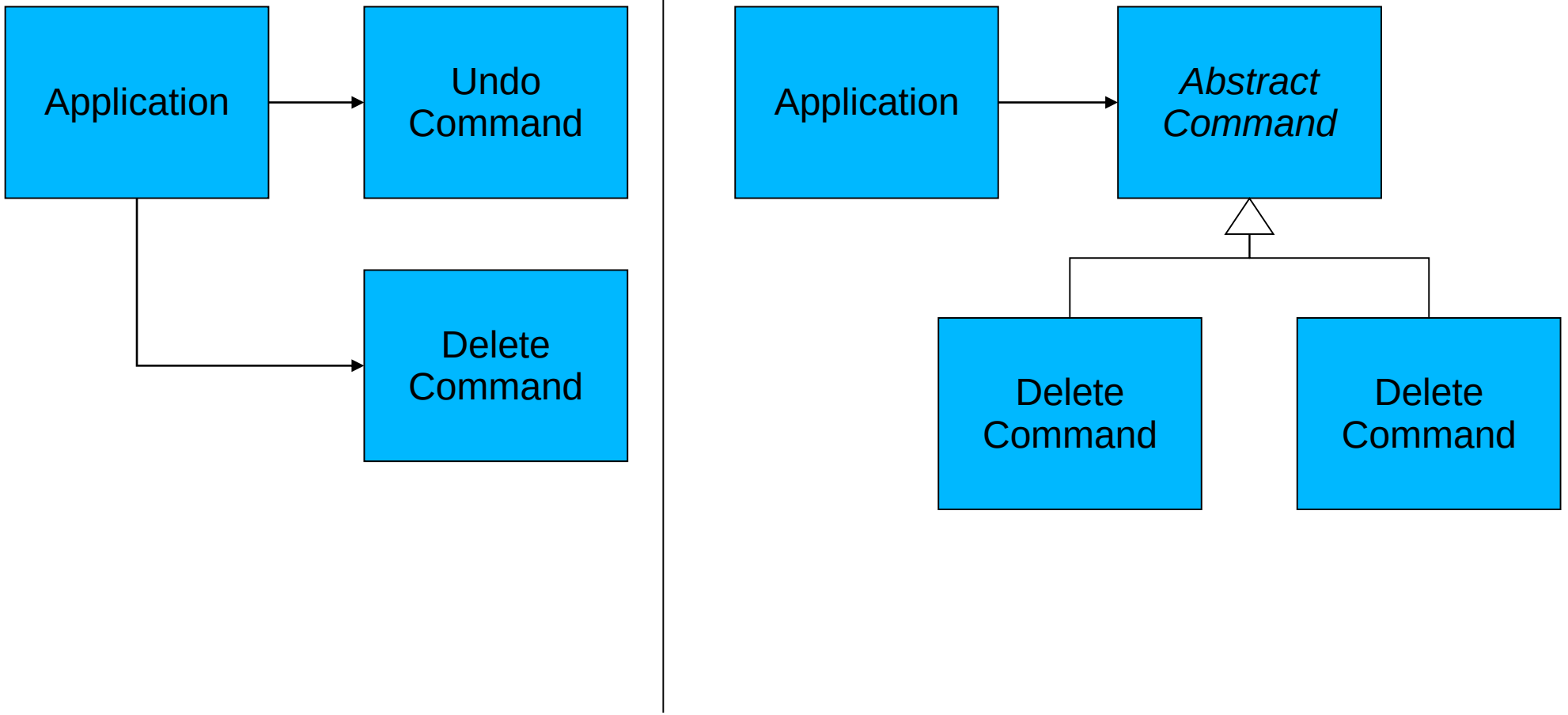
- Refactor the classes in the Interface Segregation source folder/project so they present client-specific interfaces

Dependency Inversion

A. High-level modules should not depend on low-level modules. Both should depend on abstractions.
B. Abstractions should not depend upon details. Details should depend upon abstractions.

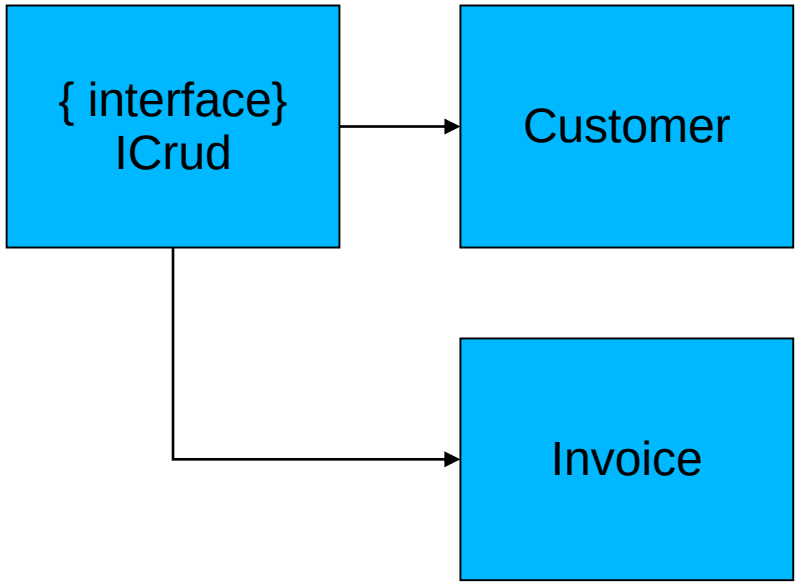
codemanship

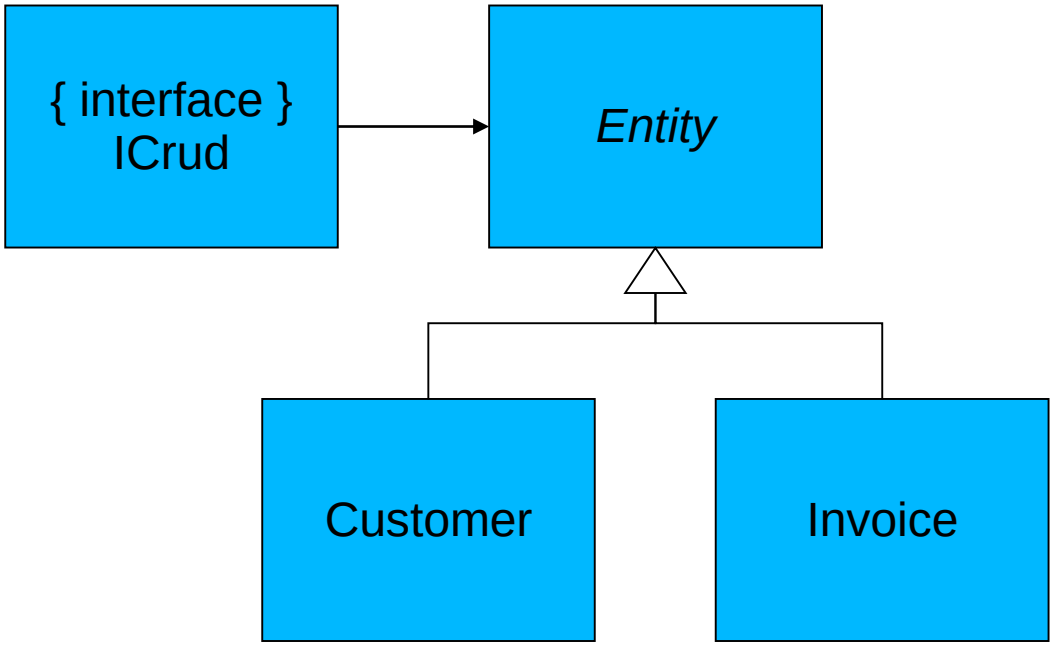
E.g. App class depends on concrete commands



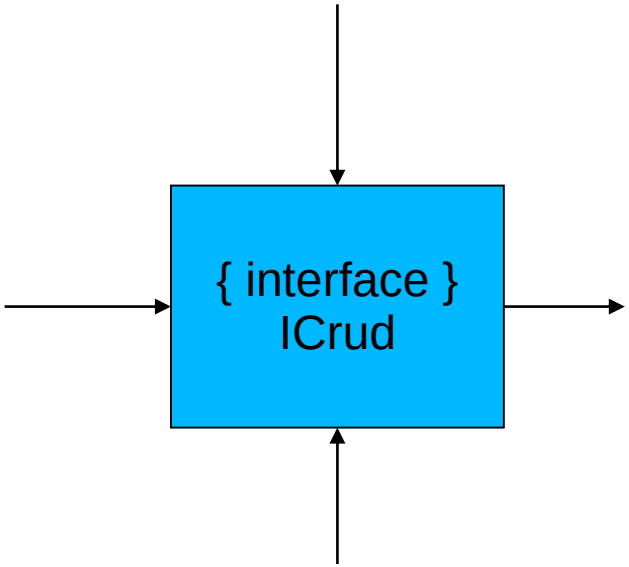
codemanship

E.g. Interface depends on concrete classes

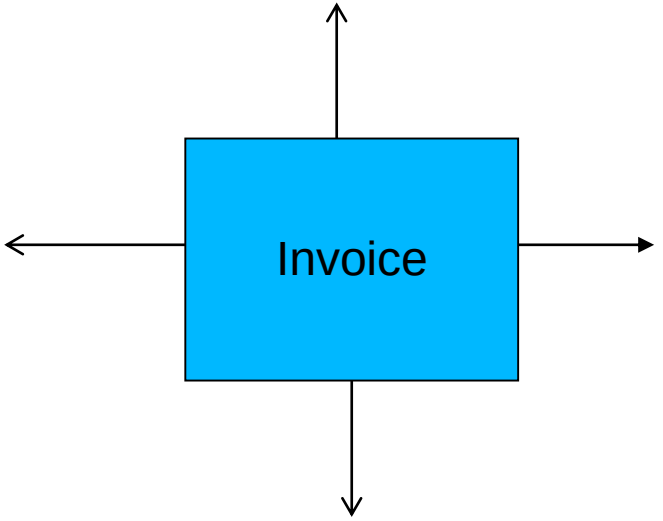




Dependency Inversion Analysis



Abstractions should have higher afferent/efferent couplings



Concrete classes should have higher efferent/afferent couplings

Ex 6 – Dependency Inversion

- Refactor the classes in the Dependency Inversion source folder/project so that higher-level classes don’t depend on lower-level classes, and abstractions don’t depend on details

More Class Design Principles

codemanship

Law of Demeter

Classes should only know about their nearest neighbours

Ex 7 – Law of Demeter

- Refactor the classes in the Law Of Demeter source folder/project so that classes only interact with their direct collaborators