Solid Principles

A (partial) blueprint for dealing with change

Agenda

- Software Facts of Life
- Good Software Design
- SOLID Principles
  - Single Responsibility Principle
  - Open/Closed Principle
  - Liskov Substitution Principle
  - Interface Segregation Principle
  - Dependency Inversion Principle

80:20

Software Facts of Life

- Code has ZERO intrinsic value
  - Developers are engineers, not artists

80:20

Software Facts of Life

- Code has ZERO intrinsic value
  - Developers are engineers, not artists
- High maintenance costs – 50 – 80% of total cost of ownership

80:20

Software Facts of Life

- Code has ZERO intrinsic value
  - Developers are engineers, not artists
- High maintenance costs – 50 – 80% of total cost of ownership
- It is harder to modify code than to write it

80:20

Characteristics of Good Software Design

- Maintainability – minimal effort spent finding and fixing issues; easy to update

80:20

Characteristics of Good Software Design

- Maintainability – minimal effort spent finding and fixing issues; easy to update
- Legibility – easy for new (or forgetful) developers to jump in

80:20

Characteristics of Good Software Design

- Maintainability – minimal effort spent finding and fixing issues; easy to update
- Legibility – easy for new (or forgetful) developers to jump in
- Extensibility – easy to add new, and expand existing, features

80:20

Characteristics of Good Software Design

- Maintainability – minimal effort spent finding and fixing issues; easy to update
- Legibility – easy for new (or forgetful) developers to jump in
- Extensibility – easy to add new, and expand existing, features
- Reusability – easy to reuse low level components

80:20

Characteristics of Good Software Design

- Maintainability – minimal effort spent finding and fixing issues; easy to update
- Legibility – easy for new (or forgetful) developers to jump in
- Extensibility – easy to add new, and expand existing, features
- Reusability – easy to reuse low level components

In Short: **Adaptability**

"Intelligence is the ability to adapt to change" – Stephen Hawking

Single Responsibility (SRP)

- Each component should have one and only one **reason to change**
- Helps limit the scope of **changes**

Open/Closed (OCP)

- Components should be open for extension and closed for modification
- Helps provide control and assurance of what is **changing**
- Heavy reliance on compliance with the SRP

Liskov Substitution (LSP)

- Derived classes must be substitutable for their base classes (or any of their siblings)
  - Also applies to concrete implementations of abstractions
- Prevents dependents from having to know about all implementations of an abstraction
- Makes it possible for external contract implementations to **change**

Interface Segregation (ISP)

- Interfaces should be minimalistic and client specific
- Provides for minimal **change** impact

Goldilocks Conundrum

- Cohesion versus Coupling
- How do we reduce coupling while maintaining cohesion?

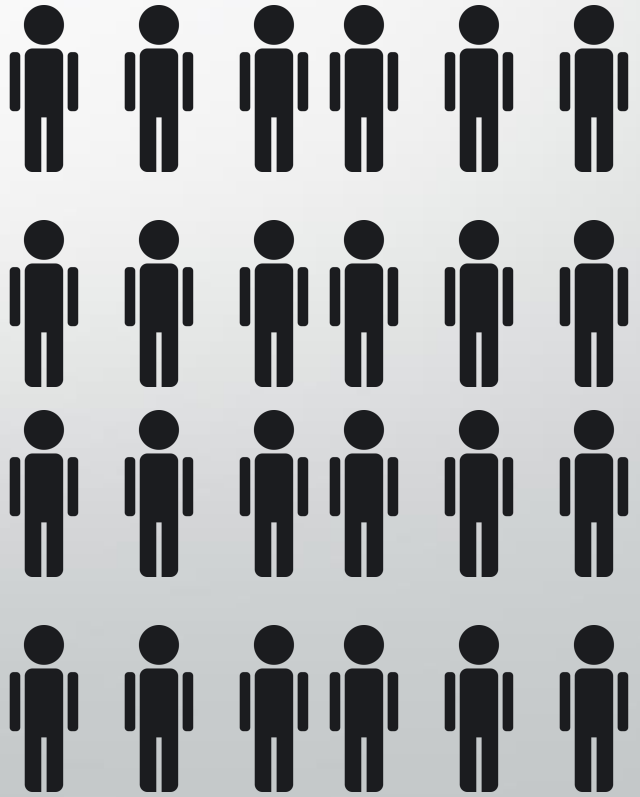
80:20

Too Few

- High Coupling
- Low Cohesion
- I always know who to go to for questions
- Every change affects this single person, and this person has to know about and remember every single change

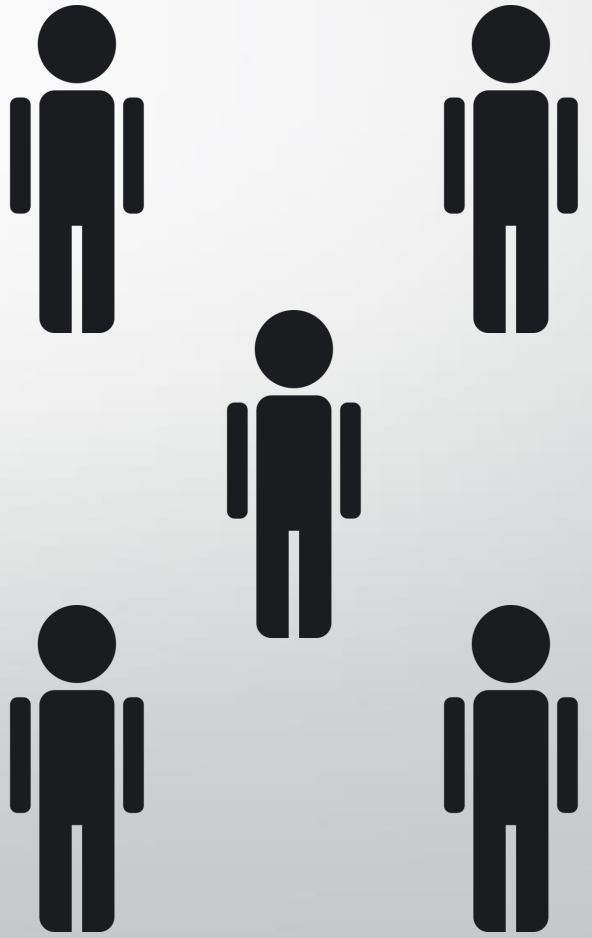
Too Many

- Low Coupling
- High Cohesion
- Each person is responsible for very little, so training them and keeping them updated is extremely easy
- When I have a question, it is very difficult to figure out who I should ask



Just Right

- Moderate Coupling
- Moderate Cohesion
- When I have a question, I probably know who to ask.
- Even if I don't know who to ask, I can very quickly check with each person involved.



Dependency Inversion

- Modules should depend upon abstract concepts, not concrete implementations
- Allows all levels of the application to be insulated from **change**

Resources

- http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod
- http://galorath.com/wp-content/uploads/2014/08/software_total_ownership_costs-development_is_only_job_one.pdf

80:20