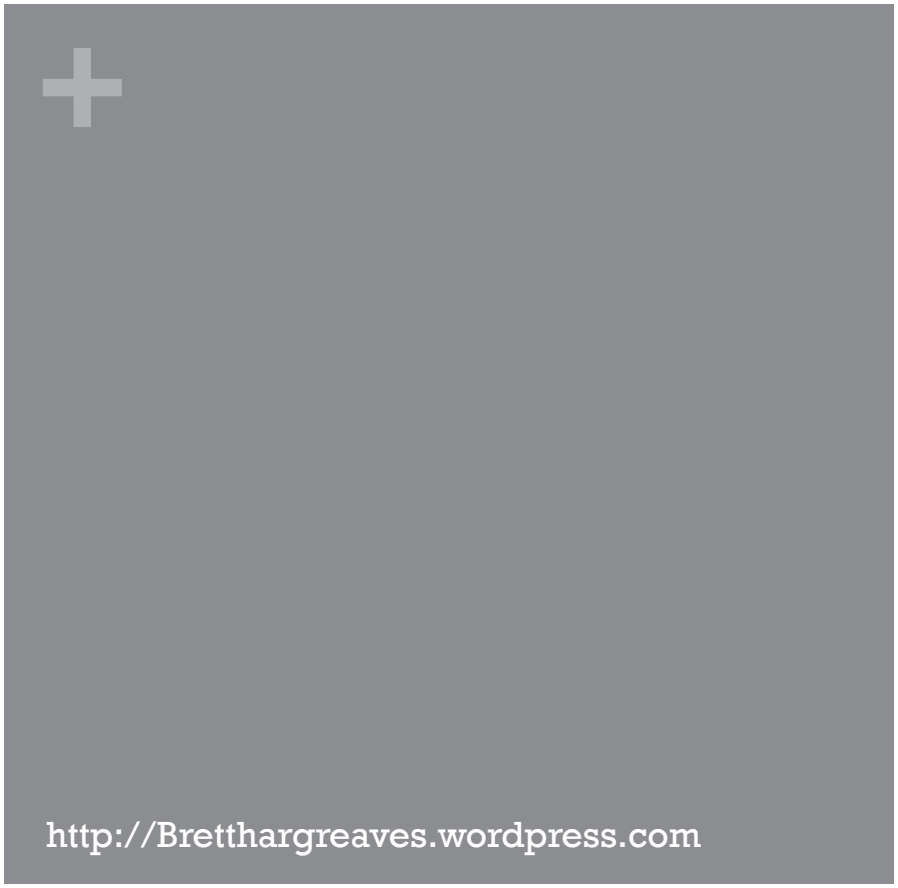

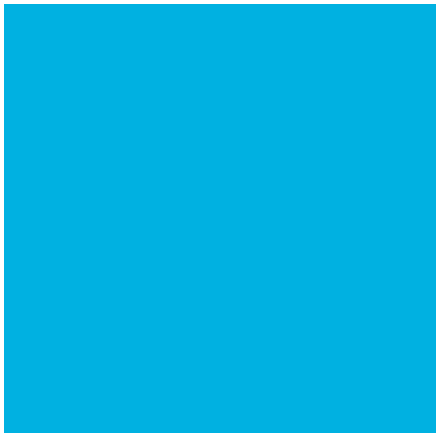

Solid Principals

Introduction

Introduction

- Code can become rigid and fragile
- Code becomes difficult to reuse
- Code becomes difficult to modify/extend
- Changes to classes causes problems in other classes
- Change introduces many bugs that are difficult to track down

S.O.L.I.D.

- Single Responsibility
- Open / Closed Principal
- Liskov Substitution Principal
- Interface Segregation Principal
- Dependency Inversion Principal

Single Responsibility Principal

- There should never be more than one reason for a class change
- A class should only have one job to perform
- Chance of any one class containing bugs is lower, therefore change is less fragile
- Classes become smaller and cleaner
- Simpler to understand and maintain

Open / Closed Principal

- Open to Extension
- Closed for Modification
- Use interfaces
- Limits the need to change code once it has been tested and debugged
- Reduces risks of introducing new bugs
- Reduced coupling

Liskov Substitution Principal

- Use Derived classes without knowing it
- Subclasses must operate in the same manner as their base class

Interface Segregation Principal

- Classes may have interfaces that are not cohesive
- Clients should not be forced to depend upon interfaces they do not use.
- ISP recommends multiple, smaller, cohesive interfaces
- Interfaces become tightly focused
- Easier to implement

Dependency Inversion Principal

- High level modules should not depend on low level modules, both should depend on abstractions
- Clients should not be forced to depend upon interfaces they do not use.
- DIP removes direct dependencies between classes
- Classes become loosely coupled, making it easier to substitute alternative implementations