The SOLID Design Principles – A primer

1. What are the SOLID Principles?
   1. These are 5 principle for O-O class design created to manage class dependencies and avoid what he called “rotting design”
   2. Rotting Design-Software so poorly designed that it becomes impossible to extend or maintain, requiring eventual redesign
   3. S: Single Responsibility Principle (SIP)
   4. O: Open-closed principle (OCP)
   5. L: Liskov substitution principle (LSP)
   6. I: Interface Segregation principle (ISP)
   7. D: dependency Inversion principle (DIP)
2. Why use the SOLID Principles?
   1. Allow us to write easily extensible and modifiable code
   2. Create code both high in cohesion and low in coupling
   3. Reduce bugs, avoid bug smelling code, make code easy to understand for others to work on
   4. Reduce debug time and chance of major effects due to small changes
   5. Never bad to adopt or apply, and apply to any Object Oriented Language and works with design patterns and architecture patterns (MVC)
3. Single Responsibility Principle
   1. SRP, states that each class should have only one purpose
   2. Purpose and tasks are different
      1. Class can do multiple tasks and only serve one purpose as long as all those tasks very closely relate to each other
      2. Even still, it might be better to separate those tasks, depending on design
   3. Creates classes with low coupling design, as well as high cohesion between the classes
4. Open-Close Principle
   1. OCP, states that classes should be able to be extended, but unable to be reasonably modified
      1. Behavior allowed to be extended, but in such a way original class or function does not need to be rewritten! Generally done using polymorphism
   2. Allow extensibility by having not adding more if statement checks to add to original functionality; instead use classes that do what you need
   3. Makes testing much quicker; instead of having to test the original class with all the nested if statement check; just test the new class
   4. Input is handled to the new class to deal with; original class does nothing with it.
5. Liskov Substitution Principle
   1. Originally defined by Barabar Liskov
   2. States that any class created through inheritance must be able to be used in a superclass call without changing overall functionality of system
      1. Expect a predictable result when we use the sub object or base object to generalize all type of that objects!
   3. A different way of thinking about how to build object oriented classes:
      1. Instead of modeling by object properties, model by object behaviour
   4. Breaking LSP implies breaking OCP
6. Interface segregation principle
   1. States that clients should not be required to depend on interface functionality they do not use
   2. Essentially the same idea as the SRP, but for interface
   3. Each interface should have one responsibility if possible
   4. High cohesion, low coupling
   5. This makes finding bugs easy and makes classes easy to understand
7. Dependency Inversion principle
   1. DIP
      1. Higher level modules should not need lower level modules; both should use abstraction
      2. Details of the functionality should depend upon abstraction, not the other way around
   2. More simply, want to have a general common high level interface, and how the class uses the abstract idea of the interface will change depending on the class
   3. Akin to electricity in a home: …
   4. Loosely coupled