Object-Oriented Design

Great Software in 3 easy steps:
1. Make sure your software does what the customer wants it to do.
2. Apply basic OO principals to add flexibility
3. Strive for maintainable, reusable design

Requirement
It is a specific thing your system has to do to work correctly. It is a singular need detailing what a particular product or service should be or do. It is most commonly used in a formal sense in systems engineering or software engineering.

Use Case
It consists of the steps that a system takes to make something happen. It describes what the system does to accomplish a particular customer goal. It is a technique for capturing the potential requirements of a new system or software change. Each use case provides one or more scenarios that convey how they system should interact with the end user or another system to achieve a specific goal.

In a well-designed OO program, objects are very particular about their jobs. Each object is interested in doing its job, and only its job, to the best of its ability. There is nothing a well-designed object hates more than begin used to do something that really isn't its true purpose.

Textual Analysis
- Looking at the nouns and verbs in your use case to figure out classes and methods
- The nouns are usually the classes that are needed in the system
- Use common sense to decide which nouns need classes. Classes are only for the parts of the system that needs to be represented.
- The verbs are usually the methods of the objects in the system

OO Concepts
- Encapsulation – keeps parts of the code that stay the same separate from the parts that change; then it is easy to make changes to the code without breaking everything.
- Flexibility – ensure software can change and grow without constant rework. It keeps application from being fragile.
- Reusability – Objects should be created in such as way that it can be reused in other software
- Inheritance
- Polymorphism

OO Design Principles

- Encapsulate what varies
- Each class in the application should have only one reason to change
- Classes are about behaviour and functionality
- The *Open-Closed Principle* (OCP): Classes should be open for extension, and closed for modification
- The *Don't Repeat Yourself Principle* (DRY): Avoid duplicate code by abstracting out things that are common and placing those things in a single location
- The *Single Responsibility Principle* (SRP): Every object in your system should have a single responsibility, and all the object's services should be focused on carrying out that single responsibility.
- The *Liskov Substitution Principle* (LSP): Subtypes must be substitutable for their base types.