**Design Patterns:**

Domain event

Composite

Facade

Flyweight

Strategy - Interchangeable algorithms

Decorator

Singleton

Null Object pattern

Factory

Repository

State (closely related to strategy) - objects for states

Virtual proxy

Parameter objects (fowler)

Object pooling

Circuit breaker

Dependency injection

Humble object

N-teir

Active record

Query by example

Data Transfer Object (DTO)

Arrange act assert (unit testing)

Design by contract

**Anti patterns:**

God class

Magnet class

Poorly named methods/classes

Arrow code

Magic numbers

Constructor over injection

Bastard injection

Service locator

Exceptions as control flow

Coding by accident

Side Effects (calling method causes other unrelated consequences)

Spaghetti code

**Code smells:**

Out parameters

Comments

Zombie Code

Long Parameter list

Conditional complexity

Bad name

Dead code

And/or in name

Cyclomatic complexity

Duplicated Code

Primitive Obsession

Long Methods

Switch Statements

Large Class

**Principles:**

SOLID

Single responsibility

Open/closed

Liskov substitution principle

Interface segregation

Dependency inversion.

Tell don't ask - tell an object what you want to know rather than asking for properties and determining that yourself

Hollywood principle

Reused abstraction principle

Dry (once and only once)

Loose coupling

Code to an interface, not an implementation

Abstract what changes

Encapsulation (information hiding)

Separation of concerns

Programming style as documentation

Turn implicit concepts into explicit (evans ddd)

Guard clause

Law of Simplicity - The ease of maintenance of any piece of software is proportional to the simplicity of its individual pieces.

Mayfly variables - Minimize variable lifetime

Return early