Introduction to Software Design Part 2

Object Oriented Software Engineering
Lecture 7

Dr. Graham McDonald
graham.mcdonald@glasgow.ac.uk

SOLID principles that help software developers design maintainable and extendable classes.

- Single responsibility
- Open-closed,
- Liskov substitution
- Interface segregation and
- Dependency inversion.

Single Responsibility Principle:

- A class should have one, and only one, reason to change.

Open-Closed Principle:

- You should be able to extend a class's behavior, without modifying it.

Liskov Substitution Principle:

- Derived classes must be substitutable for their base classes.

Interface Segregation Principle:

- Make fine grained interfaces that are client specific.

Dependency Inversion Principle:

- Depend on abstractions, not on concrete implementations.

Design Patterns helps designers apply best practices on how to realise software design principles.

Recurring design problems

- Most of the problem concern looking for optimal suitable arrangement of classes to provide a particular feature, or ensure the system exhibits a required non-functional property.
- Optimal - ensures good quality design that eases maintenance and future evolution of the software.

Recurring design problems

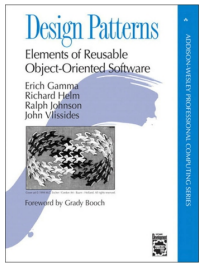
- Can we add a new feature easily?
- Can we change how part of the system is implemented without breaking the rest of the system?
- Can we adapt the system with a new behaviour (i.e at runtime) without having to rebuild everything else from scratch?

Definition:

Design patterns represent the best practices used by experienced object-oriented software developers to solve a problem.

- When a solution to a re-occurring design problem becomes well established, i.e. it is an accepted solution amongst professional software engineers for that problem, it can be documented as a design pattern.

GOF proposed 23 patterns are generally considered the foundation for all other patterns.



- Published 1994 and initiated the concept of Design Pattern in Software development

Creational Patterns:

- These are design patterns that provide a way to create objects while hiding the creation logic, rather than instantiating objects directly using new operator.
  - This gives a program more flexibility in deciding which objects need to be created for a given use case.

Creational Patterns:

	Pattern	Description
1.	Factory Method	Creates an instance of several derived classes
2.	Abstract Factory	Creates an instance of several families of classes
3.	Builder	Separates object construction from its representation
4.	Prototype	A fully initialized instance to be copied or cloned
5.	Singleton	A class of which only a single instance can exist

Structural Patterns

- These are design patterns that describe how objects and classes can be combined to form larger structures.
  - **Class patterns** describe abstraction with the help of inheritance and how it can be used to provide more useful program interface.
  - **Object patterns** describe how objects can be associated and composed to form larger, more complex structures.

Structural Patterns

	Pattern	Description
1.	Adapter	Match interfaces of different classes
2.	Bridge	Separates an object's interface from its implementation
3.	Composite	A tree structure of simple and composite objects
4.	Decorator	Add responsibilities to objects dynamically
5.	Facade	A single class that represents an entire subsystem
6.	Flyweight	A fine-grained instance used for efficient sharing
7.	Proxy	An object representing another object

Gang of Four (GOF) Types of Design Patterns

Behavioral Patterns

- These design patterns are specifically concerned with communication between objects.

Gang of Four (GOF) Types of Design Patterns

Behavioral Patterns

	Pattern	Description
1.	Chain of Resp.	A way of passing a request between a chain of objects
2.	Command	Encapsulate a command request as an object
3.	Interpreter	A way to include language elements in a program
4.	Iterator	Sequentially access the elements of a collection
5.	Mediator	Defines simplified communication between classes
6.	Memento	Capture and restore an object's internal state
7.	Observer	A way of notifying change to a number of classes
8.	State	Alter an object's behavior when its state changes
9.	Strategy	Encapsulates an algorithm inside a class
10.	Template Method	Defer the exact steps of an algorithm to a subclass
11.	Visitor	Defines a new operation to a class without change

Compound Patterns

These are patterns build from two or more design patterns:

- The most well known of these is the MVC or model view controller pattern.
- The MVC combines strategy, observer, and a composite design pattern to supply the user interface, core logic, and data model for GUI-based applications.

Gang of Four (GOF)

Anti-patterns

Definition:

Anti-patterns are common recurrences of poor solutions to problems addressed by real design patterns.

- Repeatable ways of solving problems but in a non-optimal and ineffective ways.

Gang of Four (GOF)

Anti-patterns

Some examples of anti-patterns:

- the inner platform effect;
- excessive sub-typing based on variable value partitioning;
- database as messaging platform;
- abstraction inversion;
- magic push button;
- object orgy; and
- the god object.

Summary

Software design principles are fundamental to realising high quality software.

A pattern is the outline of a reusable solution to a general problem encountered in a particular context.

Three main types**: creational, structural or behavioural**.

MVC combination of the three main types.

It is important to Identify and understand the problem before applying a design pattern.

Using design patterns is just a good design principle!