INTRODUCTION TO SOFTWARE ENGINEERING

Lesson 04 – Software Design Principles

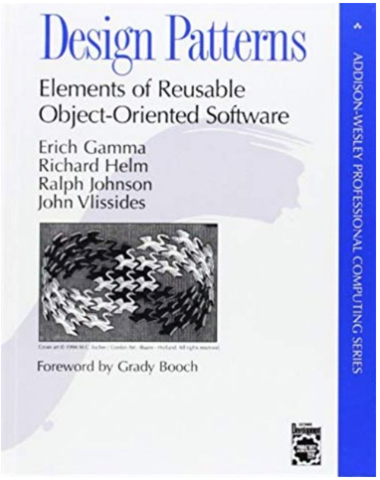
OUTLINE

1. Design patterns and principles
2. Architectural styles
3. Design documentation

1. DESIGN PATTERNS AND PRINCIPLES

The term "pattern" is to be understood as a "sample". It is often replaced with the term "template". As Christopher Alexander said: "…each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way…"[GoF95].

This definition of a pattern exists in an architecture (i.e. construction), but it is also very suitable for determination of a design pattern.



This book was first published in 1994 and it's one of the most popular books to learn design patterns. The book was authored by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. It got nicknamed as Gangs of Four design patterns because of four authors. Furthermore, it got a shorter name as “GoF Design Patterns”. [GoF95]

1.1. WHY LEARN DESIGN PATTERNS?

Code that is simpler to comprehend, update, and expand is produced with the help of design patterns.

They offer solutions that have been tried and tested as well as best practices.

Learning this enables them to quickly and effectively address similar challenges in various projects.

Developers can produce reusable components that can be utilized in a variety of applications by implementing design patterns.

This reduces redundancy and saves development time.

1.2. INDICATORS OF POOR SYSTEM DESIGN

- **Duplicate code**: identical (or very similar code) in different parts of a project
- **Long methods**: subprograms with a large number of code lines;
- **Large classes**: one class incorporates too many functions, which are often inconsistent;
- **Envy**: a class overuses methods of another class;
- **Inappropriate intimacy**: a class that is overly dependent on implementation details of another class;
- **Refused bequest**: a class overrides the base class method in such a way that the base class contract is broken by the derived class (i.e. original purpose);
- **Lazy class**: a class that does too little, i.e. its functionality is not sufficient or integral;
- **Contrived complexity**: a forced usage of complicated solutions, where a simpler design should be sufficient;
- **Excessively long identifiers**: in particular, a use of long names for the dependencies identification, which should be implicit.

1.3. GOF DESIGN PATTERN TYPES

Creational: The design patterns that deal with the creation of an object.

Structural: The design patterns in this category deals with the class structure such as Inheritance and Composition.

Behavioral: This type of design patterns provide solution for the better interaction between objects, how to provide lose coupling, and flexibility to extend easily in future.

C Creational S Structural B Behavioral

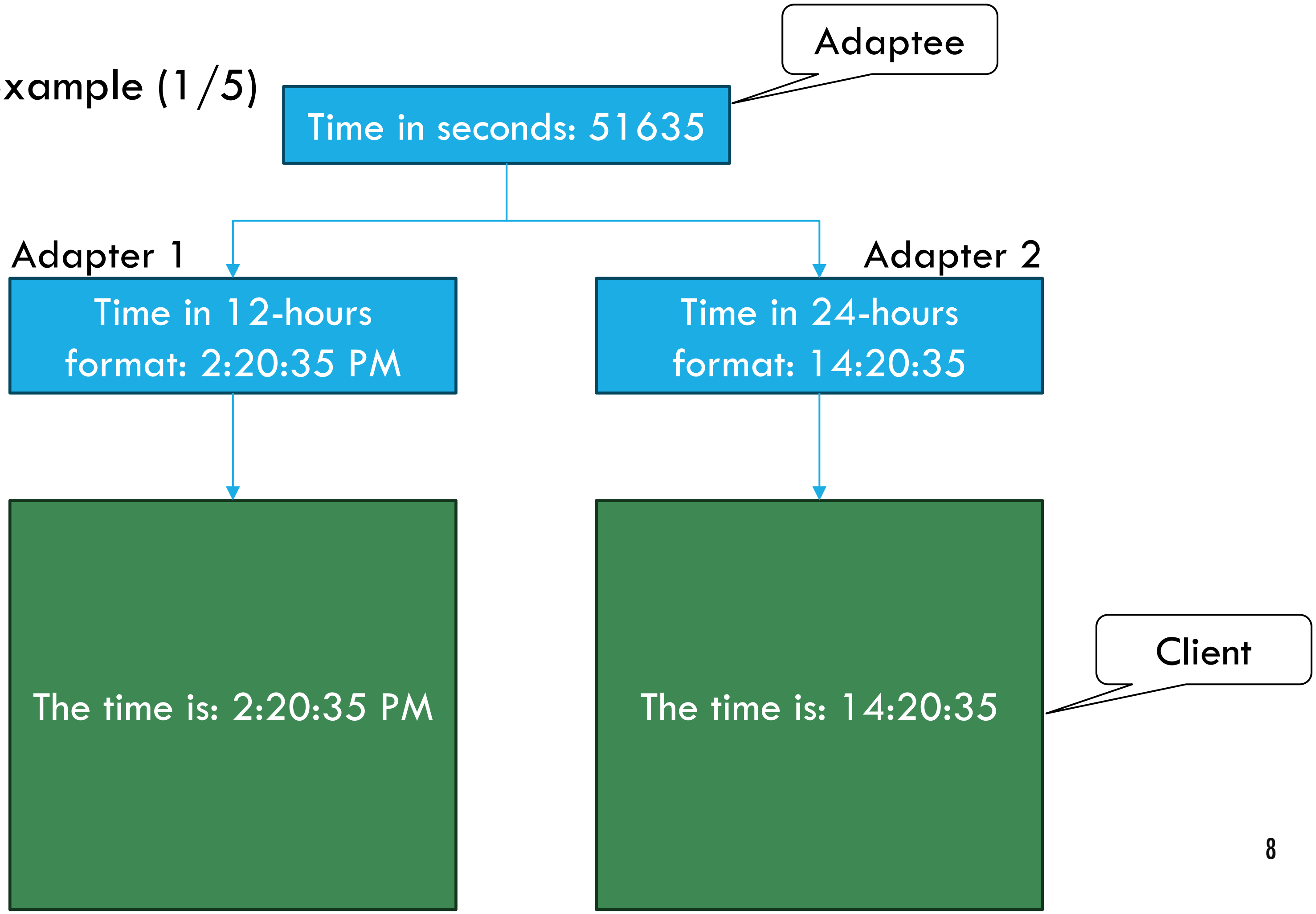
1.4. GOF'S 23 DESIGN PATTERNS

C Abstract factory	C Factory method	C Singleton
S Adapter	S Flyweight	B State
S Bridges	B Interpreter	B Strategy
C Builder	B Iterator	B Template Method
B Chain of responsibility	B Mediator	B Visitor
B Command	B Memento	
S Composite	C Prototype	
S Decorator	S Proxy	
S Façade	B Observer	

C Creational S Structural B Behavioral

1.4. GOF'S DESIGN PATTERNS

- Adapter example (1/5)



1.4. GOF'S DESIGN PATTERNS

S Adapter example (2/5)

```
// adaptee
long time_in_seconds = 51635;
```

```
// helper converts total seconds into hours, minutes, and seconds
int *toHoursMinutesSeconds()
{
    int hours = time_in_seconds / 3600;
    int minutes = (time_in_seconds - (hours * 3600)) / 60;
    int seconds = time_in_seconds % 60;
    hours %= 24;
    return new int[]{hours, minutes, seconds};
}
```

Return as array containing

0	1	2
hours	minutes	seconds

1.4. GOF'S DESIGN PATTERNS

S Adapter example (3/5)

```
// adapters
std::string adapter1()
{
    int *hms = toHoursMinutesSeconds();
    std::string ampm = hms[0] > 12 ? "PM" : "AM";
    hms[0] %= 12;
    if (hms[0] == 0)
    {
        hms[0]++;
    }
    std::stringstream sout;
    sout << hms[0] << ":" << hms[1] << ":" << hms[2] << " " << ampm;
    std::string tmp;
    getline(sout,tmp);
    return tmp;
}
```

2:20:35 PM

1.4. GOF'S DESIGN PATTERNS

S Adapter example (4/5)

```
// adapters
std::string adapter2()
{
    int *hms = toHoursMinutesSeconds();
    std::stringstream sout;
    sout << hms[0] << ":" << hms[1] << ":" << hms[2];
    std::string tmp;
    sout >> tmp;
    return tmp;
}
```

14:20:35

1.4. GOF'S DESIGN PATTERNS

S Adapter example (5/5)

Pointer to function, it can be any adapter.

```
// client
void showTime(std::string (*adapter)())
{
    std::cout << "The time is: " << adapter() << std::endl;
}
```

```
int main()
{
    std::string (*p)() = adapter1;
    showTime(p);

    p = adapter2;
    showTime(p);
    return 0;
}
```

The time is: 2:20:35 PM

The time is: 14:20:35

!!! Adapters help us update client's view without touching its code !!!

1.5. DESIGN PRINCIPLES

Design principles represent high-level guidelines or best practices that software developers should consider while designing system architecture.

The SOLID principles:

- **Single Responsibility Principle (SRP)**: A class should have only one reason to change.
- **Open/Closed Principle (OCP)**: Entities should be open for extension but closed for modification.
- **Liskov Substitution Principle (LSP)**: Superclass objects should be replaced with objects of a subclass without affecting correctness.
- **Interface Segregation Principle (ISP)**: Clients should not be forced to implement interfaces they don't use.
- **Dependency Inversion Principle (DIP)**: High-level modules shouldn't depend on low-level modules; both should depend on abstractions.

2. ARCHITECTURAL STYLES

An architectural pattern is a general, reusable resolution to a commonly occurring problem in software architecture within a given context.

The architectural patterns address various issues in software engineering, such as computer hardware performance limitations, high availability and minimization of a business risk. Some architectural patterns have been implemented within software frameworks.

There are two main categories of architectural patterns: monolithic and distributed.

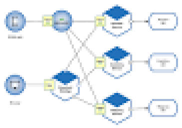
2.1. ARCHITECTURAL STYLE VS DESIGN PATTERN

Architectural style shows the system design at the highest level of abstraction. It also shows the high-level module of the application and how these modules are interacting.

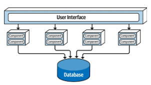
On the other hand, architectural patterns have a huge impact on system implementation horizontally and vertically.

Finally, the **design patterns** are used to solve localized issues during the implementation of the software. Also, it has a lower impact on the code than the architectural patterns since the design pattern is more concerned with a specific portion of code implementation such as initializing objects and communication between objects.

2.2. TYPES OF ARCHITECTURAL STYLE

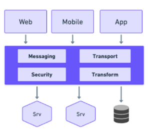


Microservices Architecture



Component-based Architecture

Event-Driven Architecture (EDA)

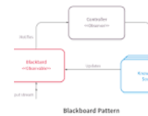


Service-Oriented Architecture (SOA)

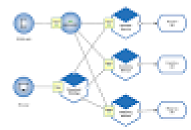
Layered Architecture (N-Tier Architecture)

Hexagonal Architecture (Ports and Adapters)

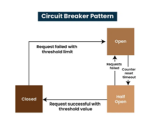
CQRS (Command Query Responsibility Segregation)



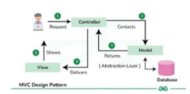
Blackboard Architecture



Serverless architecture

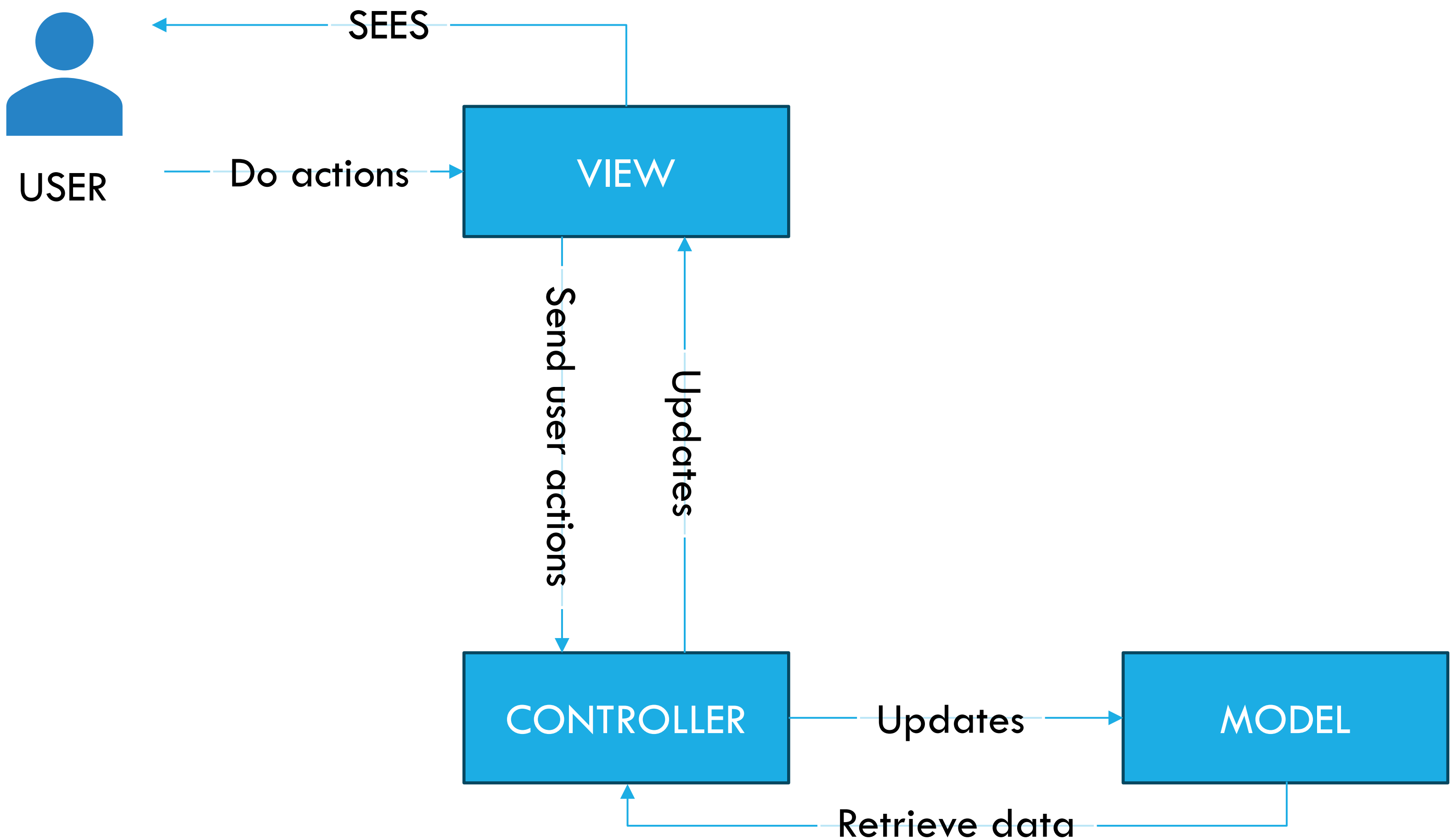


Circuit Breaker Pattern



Model-view-controller pattern

2.3. MODEL-VIEW-CONTROLLER PATTERN



3. DESIGN DOCUMENTATION

Software Design Documentation (SDD) can include:

- Feature outlines
- Meeting minutes
- Persona profiles
- Screenshots
- Diagrams
- Information about target users
- Project and product goals
- The structure of the design
- Key design decisions
- Project timelines

REFERENCES

https://www.digitalocean.com/community/tutorials/gangs-of-four-gof-design-patterns

https://www.geeksforgeeks.org/gang-of-four-gof-design-patterns/

https://www.geeksforgeeks.org/difference-between-architectural-style-architectural-patterns-and-design-patterns/

https://en.wikipedia.org/wiki/Architectural_pattern

https://en.wikipedia.org/wiki/Software_design_pattern

"Design patterns elements of reusable object-oriented software" by Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, 1995.