UTT

UNIVERSIDAD TECNOLÓGICA DE TIJUANA

GOBIERNO DE BAJA CALIFORNIA

TOPIC:

Selection of Design Patterns

STUDENT:

Maciel Leyva Ariana Lizeth

GROUP:

10A

SUBJECT:

Comprehensive Mobile Development

TEACHER:

Ray Brunett Parra Galaviz

Tijuana, Baja California, January 8th of 2025

“Selection of Design Patterns”

What are Design Patterns?

Design patterns are reusable solutions to common problems encountered during software design and development. They represent established best practices for structuring code to address specific challenges in a standardized and efficient manner. Design patterns are typical solutions to commonly occurring problems in software design. They are like pre-made blueprints that you can customize to solve a recurring design problem in your code.

What does the pattern consist of?

Most patterns are described very formally so people can reproduce them in many contexts. Here are the sections that are usually present in a pattern description:

- Intent of the pattern briefly describes both the problem and the solution.
- Motivation further explains the problem and the solution the pattern makes possible.
- Structure of classes shows each part of the pattern and how they are related.
- Code example in one of the popular programming languages makes it easier to grasp the idea behind the pattern.

Some pattern catalogs list other useful details, such as applicability of the pattern, implementation steps and relations with other patterns.

Classification of patterns

Design patterns differ by their complexity, level of detail and scale of applicability to the entire system being designed. I like the analogy to road construction: you can make an intersection safer by either installing some traffic lights or building an entire multi-level interchange with underground passages for pedestrians.

The most basic and low-level patterns are often called idioms. They usually apply only to a single programming language.

The most universal and high-level patterns are architectural patterns. Developers can implement these patterns in virtually any language. Unlike other patterns, they can be used to design the architecture of an entire application.

In addition, all patterns can be categorized by their intent, or purpose. This book covers three main groups of patterns:

- Creational patterns provide object creation mechanisms that increase flexibility and reuse of existing code.
- Structural patterns explain how to assemble objects and classes into larger structures, while keeping these structures flexible and efficient.
- Behavioral patterns take care of effective communication and the assignment of responsibilities between objects.

Creational design patterns

These design patterns are all about class instantiation. This pattern can be further divided into class-creation patterns and object-creational patterns. While class-creation patterns use inheritance effectively in the instantiation process, object-creation patterns use delegation effectively to get the job done.

- **Abstract Factory**

Creates an instance of several families of classes

- **Builder**

Separates object construction from its representation

- **Factory Method**

Creates an instance of several derived classes

- **Object Pool**

Avoid expensive acquisition and release of resources by recycling objects that are no longer in use

- **Prototype**

A fully initialized instance to be copied or cloned

- **Singleton**

A class of which only a single instance can exist

Structural design patterns

These design patterns are all about class and object composition. Structural class-creation patterns use inheritance to compose interfaces. Structural object-patterns define ways to compose objects to obtain new functionality.

- **Adapter**

Match interfaces of different classes

- **Bridge**

Separates an object’s interface from its implementation

- **Composite**

A tree structure of simple and composite objects

- **Decorator**

Add responsibilities to objects dynamically

- **Facade**

A single class that represents an entire subsystem

- **Flyweight**

A fine-grained instance used for efficient sharing

- **Private Class Data**

Restricts accessor/mutator access

- **Proxy**

An object representing another object

Behavioral design patterns

These design patterns are all about class objects communication. Behavioral patterns are those patterns that are most specifically concerned with communication between objects.

- **Chain of responsibility**

A way of passing a request between a chain of objects

- **Command**

Encapsulate a command request as an object

- **Interpreter**

A way to include language elements in a program

- **Iterator**

Sequentially access the elements of a collection

- **Mediator**

Defines simplified communication between classes

- **Memento**

Capture and restore an object's internal state

- **Null Object**

Designed to act as a default value of an object

- **Observer**

A way of notifying change to a number of classes

- **State**

Alter an object's behavior when its state changes

- **Strategy**

Encapsulates an algorithm inside a class

- **Template method**

Defer the exact steps of an algorithm to a subclass

- **Visitor**

Defines a new operation to a class without change

Selection of Design Patterns

Design patterns are reusable solutions to common problems in software design. Choosing the appropriate design pattern depends on the specific problem you are trying to solve, the context of your application, and the desired outcomes. Below are some key considerations and steps to guide the selection of the right design pattern:

1. Understand the Problem Domain

- Clearly define the problem you need to solve.
- Identify recurring issues or challenges in the design.
- Determine whether the problem is structural, behavioral, or creational.

2. Analyze Requirements

- Review the functional and non-functional requirements.
- Consider scalability, performance, maintainability, and flexibility needs.
- Evaluate how objects and classes will interact within the system.

3. Categorize the Problem

Design patterns are generally classified into three categories:

- **Creational Patterns:** For creating objects systematically (e.g., Singleton, Factory Method, Abstract Factory).
- **Structural Patterns:** For organizing relationships between classes and objects (e.g., Adapter, Composite, Proxy).
- **Behavioral Patterns:** For managing communication between objects (e.g., Observer, Strategy, Command).

4. Match Patterns to Requirements

- Compare your problem with the intent and structure of existing design patterns.
- Select a pattern that aligns closely with your design goals and constraints.

5. Consider Trade-offs

- Evaluate the trade-offs of the chosen pattern (e.g., complexity vs. simplicity).
- Consider the learning curve and implementation effort required.

6. Prototype and Test

- Implement the design pattern in a small part of the system to test its feasibility.
- Validate that it solves the problem effectively and integrates well with the overall design.

7. Iterate and Refine

- If the selected pattern does not meet expectations, refine or combine it with other patterns.
- Adapt patterns to fit your specific application needs.

Selecting the right design pattern requires understanding both the problem and the context in which it occurs. Mastery of design patterns comes with experience and practice in recognizing patterns in various scenarios. Always remember, design patterns are tools not strict rules to aid in creating robust, maintainable, and scalable software.

Sources

GeeksforGeeks. (2025, January 2nd). *Software Design Patterns tutorial*. GeeksforGeeks. Retrieved January 7th, 2025, of https://www.geeksforgeeks.org/software-design-patterns/

Refactoring.Guru. (s. f.). *Design patterns*. Retrieved January 7th, 2025, of https://refactoring.guru/design-patterns

Design Patterns and Refactoring. (s. f.). Retrieved January 7th, 2025, of https://sourcemaking.com/design_patterns

Design Gurus. (s. f.). *How to select a design pattern?* Design Gurus: One-Stop Portal For Tech Interviews. Retrieved January 7th, 2025, of https://www.designgurus.io/answers/detail/how-to-select-a-design-pattern