*Review of catalogue of GoF software design patterns*

*Preeti Gahlyan*

*Dept. of Computer Science & Engineering*

*Amity School of Engineering & Technology*

*Amity University, Noida (U. P.)*

*preeti.gahlyan@gmail.com*

*Dr. Shailendra Narayan Singh*

*Dept. of Computer Science & Engineering*

*Amity School of Engineering & Technology*

*Amity University, Noida (U. P.)*

snsingh36@amity.edu

*Abstract*— Software design patterns define the repeatable solution to common design problems. They don’t provide a solution that can be directly used in code but rather expresses an approach to follow while designing a system. Knowledge of these patterns helps a developer to create more flexible and scalable system that not only fulfills today’s requirement but can cater to the future goals as well. This paper lists over 23 GoF software design patterns. The type of problem these patterns address is the guiding principle for this catalogue.

*Index Terms*—Software designs patterns, Gang of Four, GoF, Design patterns.

# Introduction

Designing an Object oriented software is a not an easy job and designing reusable and flexible object oriented system is even harder. We need to take care of the granularity level while creating the objects and their relationship with each other. A good design should be such that it not only solve the specific problem at hand but also take care of the future requirements. The objective of a good designer is not to fix every design issue rather utilize the solutions tried earlier. Each design pattern strategically names, describes and analyzes a reusable design in object-oriented systems.

Christopher Alexander says “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 even doing it the same way twice”

A design pattern has four important properties:

1. Name
2. Problem Statement
3. Consequences
4. Resolution

There are different factors to classify the design patterns. Gamma et al. talked about two factors. The first factor is, reason, distinguishes creational, structural, and behavioral patterns. The second factor, horizon, concerns if the patterns is relevant to classes or objects. The distinction between creational or structural pattern is hard to identify, because if a pattern has some property or not is unclear to point out when identify the pattern itself is a challenge. Horizon suffers from the same issue and is not fit for non-object oriented patterns.

We are considering the patterns based on their use. There was a book (Design patterns: Elements of reusable object oriented software) published in 1994 by four authors Erich Gamma, Richard Helm, Ralph Johnson and John that provided insight into Design patterns that are used in software industry. Gang of Four is the name given to these four authors. Due to the way they put those 23 design patterns to the audience that now these patterns are called as GoF software patterns. We will be discussing here the most widely used design patterns.

# Definition of design pattern

A software design pattern is a tested solution to a recurring design problem. A design pattern is not a “direct code it” type of solution but an approach for how to solve a problem. A same pattern can be used in many different situations. Object oriented design patterns exhibits the relationship between different class and objects without explicitly telling the final classes or objects that are being used.

At higher level design patterns are relationship between objects and classes and works at module level. We can represent an entire subsystem by a single pattern which falls under architectural pattern.

There are several types of design patterns: Structural patterns deal with issues related to the high level structure of the system being developed. Implementation patterns are more concerned about the implementation details at lower level like how the system is actually coded and which data structures are being used that support even the multithreaded environment. Computational patterns deal with high level strategies that help us utilize application properties on computational system. Execution patterns however emphasizes over the execution bit, like how the different tasks will be synchronized at run time and what factors can affect the application performance.

# Classification of design patterns

As per GoF there are mainly three classifications of design patterns template:

1. Creational patterns
2. Structural patterns
3. Behavioral patterns

**Creational Patterns** : Creational design patterns help us in object creation process. They abstract the process of creating, composing and defining the object. There are two types of creational patterns. Class creational patterns consider inheritance for selecting the class to instantiate at runtime while the object creational patterns delegate object instantiation to some other object.

There are two recurring themes in these patterns. First, they hide the details about which concrete class the system is made of. Second, they encapsulate how the instances of these classes are put together. Sometimes, creational patterns are like opponents. For example, there might be scenarios when one pattern can used over the other pattern. At other times one can be used by one other. Factory can use builder inside it. Prototype can use flyweight in its implementation. Singleton can use Lazy initialization.

The most commonly used creational patterns are:

1. *Factory method* : It is one of the most used design patterns. We use a factory pattern when we want a method to return one of the several possible classes that inherits a common base class. The class will be selected at runtime based on the type of object passed to factory class.

Applicability : When we don't know ahead of time what object we need. When all of the potential classes are in same subclass hierarchy. To provide generic centralize code for class selection at runtime. To encapsulate object creation

1. *Abstract factory* : It is like the factory pattern except that everything is encapsulated. The object that creates the object, the factories that build the object and the final objects. This is also called factory of factories.

Applicability : It allows us to create families of related objects without specifying a concrete class. We can use it when we a list of objects that can be changed at run time. When we want only to expose the interface not its internal details

Challenges : It can get very complicated if not implemented property

1. *Singletons* : This pattern creates only a single instance of a class and provides the public access to that instance.

Applicability : When we need only a single instance of a class throughout the application. For example, database connection, file connection, logging, etc.

Challenge : We can end with multiple instances of class by different threads if we don't apply locking on the instance before creating the object

1. *Lazy initialization* : It allows us to delay the process of object creation until the object is accessed first time. It can be applied with singleton
2. *Builder* : This is the pattern used to create an object made up from bunch of other objects. It segregates the fabrication process of a complex object from its depiction enabling us to use the same fabrication process to create different representations.

Applicability : The formation algorithm of a complex object is independent from the parts that actually compose the object. The application requires allowing various representations for the objects that are built. We should use pass the builder in class constructor

1. *Prototype* : It suggests creating new objects by using the existing object as a prototype. It uses the clone concept and help use cut the cost by not creating an object in a usual way.

Applicability : When creating an object is very costly and a clone of existing object can serve the purpose of another object

Challenges : Class needs to inherit from cloneable interface and the object which we are copying should provide the clone feature. It should not be done by some other class.

**Structural Patterns** : Structural patterns concerns with composition of object structures. We can introduce new functionality in the application by composing objects using Structural object patterns whereas using inheritance to organize interfaces or implementations falls under Structural class patterns. It provides the flexibility to change the composition dynamically, which is not likely to happen with static class composition.

There are various structural design patterns. Like composite, it allows us to treat objects and their composition uniformly. It describes a hierarchical structure of simple and composite objects. In proxy pattern, a proxy acts as a mediator or placeholder for another object. It may act as a representative of a remote object or can represent object which may require on-demand loading. So it can be used to enhance, restrict or modify properties. The flyweight pattern outlines a paradigm for sharing the objects. Efficiency and Consistency are the two reasons for sharing an Object, and also improves space efficiency.

Facade shows how to depict the whole subsystem with a single object. Also, a set of objects is depicted by a facade. An object’s abstraction is segregated by Bridge pattern from its implementation such that alterations can be made autonomously. Decorator narrates how to assign responsibility to an object dynamically. Adapter pattern makes one interface conform to another.

The most commonly used structural design patterns are :

1. *Composite* : It enables us to consider individual object and composition of objects uniformly. A composite can contain other composites. Allows recursive composition. Applicability : When we need to represent “whole-part” hierarchical relationship. If we want to perform an operation an leaf node then same operation should be performed on the composite object also
2. *Adapter* : Adapter pattern helps when we want to interact between two incompatible interfaces. It is also known as Wrapper. Applicability : When we need to communicate between two independent or incompatible interfaces. Allows a system to use classes of another system that is incompatible with it.
3. *Bridge* : Progressively adding functionality while separating out major differences using abstract classes. It is also known as Handle/Body. Applicability : When we want to change the abstract classes as well as concrete classes completely independently. When we alter the implementation of an abstraction without breaking the current functionality. When we intend to avoid tight coupling between abstraction and its implementation
4. *Decorator:* Allows to dynamically adding responsibilities to an Object. Flexible substitution to sub-classing for functionality extension is also provided by Decorator. Applicability: Adding responsibilities dynamically to a single object with no affect to other objects. To add withdraw-able responsibilities. When extending classes using inheritance is impractical
5. *Facade* : It yields a single interface to a group of interfaces in the sub system. Facade makes the system easy to use by providing an abstract level interface. Applicability : When we need to provide a unified interface to a complex subsystem. We can use facade as an entry point to layer our subsystem.
6. *Flyweight :* Maximum data sharing between similar objects to reduce the memory storage cost. It helps in space efficiency. Applicability : When the number of objects is quite high in application. Memory storage cost is high due to large no of objects. The state of most of objects can be external.
7. *Proxy :* A remote object or an object part of another interface can be represented by proxy. This patter is also called surrogate. Applicability : When access control to original object is required. When object needs to be created on demand during run time.

**Behavioral Patterns:** Communication, relationship and responsibilities between objects are represented by Behavioral patterns. Responsibility assignment among objects and algorithms are a matter of concern for these patterns. Inheritance is used in class patterns whereas Object composition is used in behavioral object patterns in lieu of inheritance.

The most commonly used behavioral patterns are as follows:

**1.** *Chain of responsibility*: Multiple objects can handle the same request. One object can delegate the request to another thus making a chain. Applicability : Scenarios where the request need not to be handled by a particular object and can be delegated.

**2.** *Command:* This pattern provides the flexibility to wrap our request as an object and enables us to use different parameters depending upon the request. Applicability : Need to implement callbacks. Need to support redo and undo functionality of commands.

**3.** *Interpreter* : This pattern interprets the instruction written in a language grammar. It is used in compilers or parsers or macro expansions. Applicability : When language grammar is elementary. When efficiency is least concern.

**4.** *Iterator* : Empowers us to iterate through the members of an object collection in sequential manner without disclosing internal details. Applicability : When multiple iterations are required on over collection elements. Provides the uniform interface for traversing different aggregate structures.

**5.** *Mediator* : Promote loose coupling by allowing multiple objects to communicate with each other without knowing each other's structure. Applicability : When reusability of object is not easy as it is too tightly coupled with other objects.

**6.** *Memento* : Capture the current state of an object and store it; permitting us to retrieve this state in future without breaking the rules of encapsulation. Applicability : When state of the object need to be saved and restored later. When state of object cannot be exposed directly through an interface without exposing underlying implementation details.

**7.** *Observer :* Notifies and updates the state of all dependent objects(observer) when the state of parent object(subject) changes. Also known as Publish-Subscribe. Applicability : When one object cannot be modified without affecting other in turn forcing the developer to modify it as well.

**8.** *State :* Alters the behavior of an object when it's internal state changes. Applicability : When behavior is dependent on object state and requires to be changed dynamically whenever object state changes.

**9.** *Strategy :* Allows a client to choose an algorithm from a family of algorithms at runtime and gives it a simple way to access it. Applicability : Keeps class changes from forcing other class changes. Many related classes only differ in their behaviors.

**10.** *Visitor:* This pattern is used to create and perform new operations onto a set of objects without changing the object structure or classes. This pattern enables loose coupling and addition of new operations without changing its structure. Applicability: When we need to perform operation on the concrete classes of object structure. When we need to extend functionality of an object without changing its structure.

**11.** *Template Method:* Algorithm skeleton is described by it. It allows the implementation of the individual steps to be changed by the subclasses without changing the algorithm structure. Applicability : To implement the static part of algorithm and letting the subclass to implement the parts that can vary.

1. HOW TO CHOOSE A DESIGN PATTERN

Learning about design patterns seems appealing to use them right away in your code. But we need to be really cautious before selecting a design pattern. For eg. Choosing abstract factory can overcomplicate your design if not implemented properly. Or you may end up using prototype when singleton pattern will suit best. So use a pattern only when it really helps improving the software. We should understand the problem statement first and then go for a type of pattern. For eg. We need to interact with two classes or interfaces and they are not compatible with each other so we can use the adaptor pattern to serve as a bridge and make our life easy. But we need to make sure the benefit of using patterns overpowers the liabilities it will bring to the system.

Design patterns

Benefits the software design

Yes No

Apply the pattern

Look for the alternatives

1. APPROACHES TO IDENITFY PATTERN

There are different approaches tried by researchers to identify the pattern used in the system and they name it as per suitability of the device. One of them is using the PAT system which will check the type of pattern used. Another researcher suggest using empirical study of patterns instead of just codify expert knowledge. Some suggest that a verification approach could be run time measurements for design patterns. The device used is called “Patternometer”. Researchers has shown that standard state probabilities have a stable asymptotic behavior and that allow us for safe identification of standard patterns. Another approach tried is constraint satisfaction optimization problem (CSOP). Automating pattern based design technique can be a practical solution has been suggested by CSOP.

# Conclusion

Software patterns help us to solving a problem statement by applying the approach they present to us. Learning about software design patterns not only helps experienced professionals to design a robust system but also educates the new developer to avoid these pitfalls which otherwise are learnt by costly experience. But just knowing the patterns is not enough a developer must consider which pattern to use in a scenario and whether using it at all is required or not. Also some studies show that pattern classes are more likely to have errors than non-pattern classes because we change pattern classes more often than we should as per the basic idea behind using design patterns.

# Acknowledgment

I express my sincere gratitude towards my guide and co-author Dr. Shailendra Narayan Singh for his invaluable assistance, motivation, guidance and encouragement. I would also like to extend my thanks to entire faculty of the Amity School of Engineering & Technology (ASET) who have encouraged us to complete this review paper.

Last but not the least I am greatly indebted to all my friends who have encouraged directly or indirectly for successful completion of this review paper.

# References

1. E. Gamma et al.,Addison-Wesley,Reading,Mass.,1995 “Design Patterns : Elements of Reusable Object-oriented Design”
2. R OBERT T. M ONROE, ANDREW K OMPANEK, RALPH MELTON, DAVID GARLAN : Architectural Styles, Design Patterns and objects, Carnegie Mellon University
3. Cheng Zhang,David Budgen “What do we know about the effectiveness of software design patterns”
4. Marat Teplitsky, Iaakov Exman “Measuring Behavioral Software design patterns”
5. Jafferey Heer,Maneesh Agrawala “Software Design Patterns for Information Visualization”
6. A. Lau, R.E.Seviora “Design patterns for Software Health Monitoring”
7. Jing Wang,Yeong-Tae Song,L. Chung “From software acrchitecture to design patterns: a case study of an NFR approach”
8. Derek Reimanis,Clemente Izurieta “Towards Assessing the Technical Debt of Undesired Software Behaviors in Design Patterns”
9. M.Duell “Looking beyond software to understand software design patterns”
10. J. McC Smith, D. Stotts “Elemental design patterns: a formal semantics for composition of OO software architecture”
11. Adil Waheed,Ghulam Rasool,Saqib Ubaid,Faisal Ghaffar “Discovery of design patterns variants for quality software development”
12. P. Stevens “Software design patterns”
13. Sriharsha Vathsavayi,Outi Sievi-Korte,Kai Koskimies,Kari Systa “Using Constraint Satisfaction and Optimization for Pattern-Based Software Design”
14. K.Araujo,J.B.Bowles “Design patterns as components of functional models for analyzing the reliability of software systems”
15. Sukant kumar shaoo “Social object –a software design pattern”
16. <http://www.dotnettricks.com/learn/designpatterns/gang-of-four-gof-design-patterns-in-net>
17. Dariusz Rogowski “Software implementation of common criteria related design patterns”
18. Walter F. Tichy, University of Karlsruhe, Karlsruhe,Germany : A Catalogue of General-Purpose Software Design Patterns
19. <https://www.tutorialspoint.com/design_pattern/design_pattern_overview.htm>
20. Chuanjun Li,Qing Wang,Wenwen Cai,Jun He “An efficacious software design method based on pattern and its application”
21. M.Gatrell,S.Counsell “Design patterns and fault-proneness a study of commercial C# software”