MVC Software Design Pattern in Web Application Development

Madiha Hameed, Muhammad Abrar, Ahmer Siddiq, Tahir Javeed

Abstract - Web applications that used design pattern have becomes popular because of their reusability and consistency and flexibility, Developer uses design pattern to increase flexibility and due to its popularity as it proved as a best practice for developer to solve many problem occurs during software/web application, but when some un-experience uses it, unfortunately they does not find desired result because of lack of experience in applying them. This paper is intent to write to understand their proper use and will help to understand the design patterns, their categories, usage, and the situation in which developer has to make right decision to select appropriate design pattern, key elements of design pattern, comparison of design pattern and brief description of all design pattern, this paper also tell the mostly used design pattern during web application.

Index Terms – MVC (Model View Control), Design pattern, web application

---

1 INTRODUCTION

In computer science design pattern is a proper approach of providing a key of answers to a design problem in a particular area in proficiency. [1]

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 ever doing it the same way twice". The elements of this language are entities called patterns. Each pattern describes a problem that 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 twice. [1]

1.1 Design patterns in software engineering

It is reusable solution in commonly occurring problem during software development under certain circumstances, it is basically descriptions or guidelines on how to solve a problems and can be used in different situation according to your need or context, they are known as best practice and the programmers must implement in applications [2]

1.2 Design patterns in development

Today design pattern is the need of almost every web application or software because if programmers not follow the design patterns then we can say that the application structure is not reliable because web application developer need to overcome many challenges during web development in order
to get the quality of service which including speed, scalability and security.
Basically it provides tested, proven development paradigms that can speed up development process [2] Design pattern help us to resolve or fix issues that causes major problem in future or in implementation and it improves code readability for experienced developers.
It is seen that numerous non web application have restructured as a web based. Most common challenges are structuring of web application, organizing and data handling. [2]

2 REASONS OF APPLYING DESIGN PATTERNS

- Reusability
- Best practices for developers as they well known of terminologies
- Give problem oriented solution

3 DESIGN PATTERN CATEGORIES

There are 23 design pattern and these patterns are classified into categories .They are divided into three parts **Creational, Structural and Behavioral** patterns.

S#	Category	Description
1	Creational Patterns	These design patterns hide the creation logic while creation of object, rather than ins tainting object using NEW operator. This gives program more flexibility in deciding which objects need to be created for a given use case.
2	Structural Pattern	These design patterns concern with the class and object composition. Inheritance is used in it in order to compose interfaces and define ways to create object in order to obtain new functionalities.
3	Behavioral Patterns	These design patterns are specifically concerned with communication between objects.

tbl # 1

3.1 Scope of Categories

The term SCOPE specifies whether the pattern applies to class or object. The pattern under the class label is those that focus on class relationship and that are in object label are focus on the object relationship. Class patterns deal with the relationship between classes and their child classes and these relationships are made by inheritance and almost all patterns use inheritance to some extent. Object patterns deal with object relationships, which can be changed at run-time and are more dynamic.[3]
Creational class patterns defer some part of object creation to subclasses, while Creational object patterns defer it to another object. The Structural class patterns use inheritance to compose classes, while the Structural object patterns describe ways to assemble objects. The Behavioral class patterns use inheritance to describe algorithms and flow of control, whereas the Behavioral object patterns describe how a group of objects cooperate to perform a task that no single object can carry out alone.[4]

		Creational	Structural	Behavioral
Scope	**Class**	Factory Method	Adapter	Interpreter Template Method
	Object	Abstract Factory Builder Prototype Singleton	Adaptor Bridge Composite Decorator Façade Flyweight Proxy	Chain of responsibility Command iterator Mediator Memento Observer State Strategy Visitor

tbl # 2[2]

4 APPROPRIATE DESIGN PATTERN SELECTIONS

The harder thing is to understand what you exactly need and what is your problem and which design pattern can fit in your needs to solve your problem, actually it is possible to have more than one design pattern for one problem, developer of the application has to select the appropriate one. [5]]
There is also some problems faced by developers who uses wrong design pattern and thus causes problem.[5]
There are some steps also to select any design pattern for your problem

1. The first thing is that you understand the pattern and make sure you have completely understand it that what the specific deign pattern does and what you what it to do and then finally implement it in your code to solve problem
2. **Get Specific:** Now you have to map the generic pattern into your specific problem, this need to update your participants in the pattern to things that make sense in the context of your application.[11]
3. **Reorganize:** When you adopt specific pattern it may need to add, remove or reorganize classes and objects as well, and if you are documenting your design using UML or other related techniques then you will want to update the stuff for the pattern reflection.
4. **Implementation:** This part is very easy because programming with patterns becomes the easiest way, Now you have to implement the pattern in the code, it becomes very easy to write your code when you have clear model of how you relate and interact objects and classes to each other.[6]

For understanding table is given with some problems with their solution and recommended design pattern

Problem	Solution	Patterns
Your code depends on the names of classes. Changing the class of an object is burdensome because the name of the class is hard-coded in the client program through constructor invocations.	Do not use constructors directly in your client classes. Provide an extra level of indirection to the code that invokes the constructor.	Abstract Factory Factory Method Prototype
Your code depends on platform idiosyncrasies. #ifdef and conditional compilation only takes you so far. Even Java's vaunted platform independence has a few weak spots.	Isolate the platform-dependent parts of your program from the platform-independent parts.	Abstract Factory Bridge
Your code depends on specific methods in specific classes.	Separate the request itself from the object and/or method that handles the request.	Chain of Responsibility Command
Your code depends too closely on exactly how an object is implemented. Changing the implementation of the class forces change on the client class. Encapsulation isn't airtight.	This is more often a problem in C++ with its pointers, pointer arithmetic, and relatively close access to the machine than it is in Java. Often the solution is to wrap an additional layer of interface around the implementation.	Abstract Factory Bridge Memento Proxy
Changing an algorithm requires too many changes in the classes that use it, especially changes that affect the class's interface as well as its implementation.	The algorithm should be separated from the class, and moved into a class of its own.	Builder Iterator Strategy Template Method Visitor
Classes are excessively dependent on each other. It's difficult to change one class without changing most or all other classes.	Separate classes with additional levels of indirection.	Bridge Chain of Responsibility Mediator Observer Command Façade
Sub classing is too difficult.	Use object composition and delegation instead.	Bridge Chain of Responsibility Observer Decorator Composite, Strategy
A class can't be modified, either because you don't have its source code or because too many other classes depend on it.	Use object composition to embed an instance of the class inside another class that provides a new interface. Delegate requests to the embedded object.	Adaptor Decorator Visitor

5 DESIGN PATTERN CONVENTION

Design Patterns have two main conventions in software development.

5.1Common platform for developers

Suppose developer writes program using some techniques prior to design pattern so how the other developer came to know which approach is used by developer to write code and how, so there was also need for standard terminology that can be adopted by others developers too, so design pattern becomes a common platform that others developers also uses to solve their problems so Design patterns become a standard terminology for developers because they are specific for different circumstances.[10]

For example a singleton design pattern describes the use of single object so all the developers know that singleton design pattern use the single object (Not more than 1 object) and they can tell each other that their program is following singleton pattern.[7]

5.2 Best Practices

Design patterns provide best solutions to certain problems faced during software development. Design patterns have been used over a long period of time and it also helps un-experienced programmers to learn software or web application in an easy and faster way [8]

Creational Design Patterns	
Singleton	Ensure that only one instance of a class is created and Provide a global access point to the object.
Factory	Creates objects without exposing the instantiation logic to the client and Refers to the newly created

	object through a common interface.
Factory Method	Defines an interface for creating objects, but let subclasses to decide which class to instantiate and Refers to the newly created object through a common interface.
Abstract Factory	Offers the interface for creating a family of related objects, without explicitly specifying their classes.
Builder	Define an instance for creating an object and allow child class to decide which class should be instantiate
Prototype	Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.
Object Pool	Creation of object is costly steps, to increase the performance object pool reuses the object that are expensive to create
Behavioral Design Patterns	
Chain of Responsibility	The objects become parts of a chain and the request is sent from one object to another across the chain until one of the objects will handle it.
Command	Encapsulate a request as an object, It allows saving the requests in a queue.
Interpreter	Specifies how to evaluate sentences in a language. It defines representation of language that is used to evaluate(interpret) sentence
Iterator	Easily manipulates collections of objects and keep the track of objects that is iterated and to be iterated
Mediator	Define an object that encapsulates how a set of objects interact, It promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently. Note: the Front Controller pattern is a specialized kind of Mediator pattern that handles all requests for a web application.
Observer	It has one to many dependencies , if any object changes then its related dependencies also changes with change of state of object
Strategy	It enables an algorithm's behavior to be selected at runtime. It defines a family of algorithms, encapsulates each algorithm, and makes the algorithms interchangeable within that family.
Template Method	Define Algorithm Skelton in an operation and allow its child classes to change the behavior without changes algorithm structure.
Visitor	Defines an algorithm as an object that "visits" each member of a aggregate performing an operation.
Null Object	The Null Object Pattern provides intelligent do nothing behavior, hiding the details from its collaborators. Or Null Object is to encapsulate the absence of an object by providing a substitutable alternative that offers suitable default do nothing behavior. In short, a design where "nothing will come of nothing"
Memento	It captures internal state of an object and later on restore the object to that state
Structural Design Patterns:	
Adapter	Allows classes to support a familiar interface so you can use new classes without refactoring old code.
Bridge	"Decouple an abstraction from its implementation so that the two can vary independently".[5]
Composite	Compose objects into tree structures to represent part-whole hierarchies. / Composite lets clients treat individual objects and compositions of objects uniformly.
Decorator	It add additional responsibilities dynamically to an object and can simplify class hierarchies by replacing subclasses.
Flyweight	It use sharing to support a large number of objects that have part of their internal state in common where the other part of state can vary./ A flyweight is an object that minimizes memory use by sharing as much data as possible with other similar objects
Proxy	It provides a "Placeholder" for an object to control references to it. Or "Provides access to an object through a surrogate object to allow for delayed instantiation or protection of subject methods."

Tbl #3 [7]

5 DESIGN PATTERNS IN WEB APPLICATIONS DEVELOPMENT

Web application development is a vast field these days and growing rapidly in business community regarding to its flexibility and availability to everywhere at every time [14]

In design pattern introduced approaches there are three approaches are the popular approaches to define web application structure

1. MVC
2. Factory
3. Abstract

As we already discuss the above mention three approaches

MVC (Model View Control) is the most popular approach in web based application development it is flexible and provide the privilege of word division in small chunks to handle them easily and manage the work load in a team .It encourage the team work which is becoming the need of this world as much as web development field is growing among the non- IT people it become the necessary that software are user-friendly and strong in security point

The Model view control approach provide the flexibility and reusability of application structure

More rapidly growing mobile development is also use MVC for its different display manner.

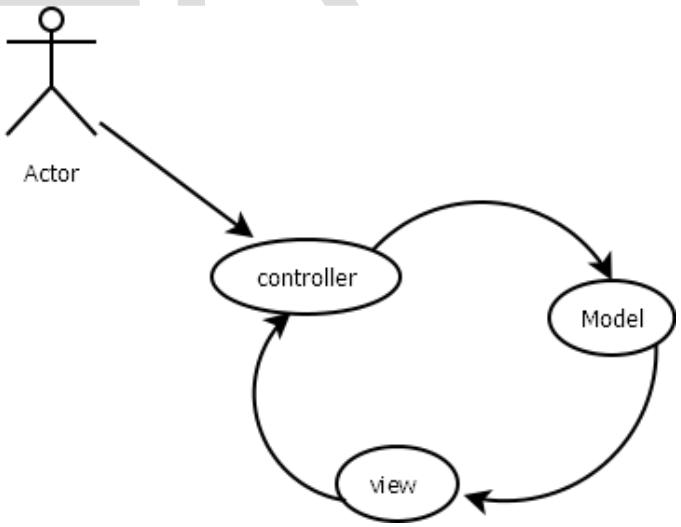

Fig # 1

CONCLUSION

Creational Patterns offer great flexibility in how your software's objects are created. It is of great help to clearly understand these simple starting point patterns with their pros and cons to efficiently extend and maintain an application.

MVC provide the user friendly design pattern and reusability of application through its model base control and view to user. Its successful response from user helpful for the web application developers.

REFERENCES

1. C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King and S. Angel. *A Pattern Language.* Oxford University Press, New York, 1977.
2. E. Gamma, R. Helm, R. Johnson and J. Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1996.
3. G. Rossi,D.Schwabe andF.Lyardet,"Improving Web Information Systemswith Design Patterns". In *Proc. of the 8th International World Wide Web Conference,* Toronto (CA), May 1999, Elsevier Science, 1999, pp. 589-600.
4. D.C. Schmidt, R. E. Johnson and M. Fayad. "Software Patterns". *Communications of the ACM, Special Issue on Patterns and Pattern Languages,* Vol. 39, No. 10,
5. M.P. Cline, "Using Design Patterns to Develop Reusable Object-Oriented Communication Software", *Communication of ACM,* 38 (10), October 1995, pp. 65-74.
6. Singh Sandhu, P.; Pal Singh, P.; Kumar Verma, A. "Evaluating Quality of Software Systems by Design Patterns Detection", Advanced Computer Theory and Engineering, 2008. ICACTE '08. International Conference on, On page(s): 3 – 7
7. Krein, Jonathan L.; Pratt, L.J.; Swenson, A.B.; MacLean, A.C.; Knutson, Charles D.; Eggett, D.L. "Design Patterns in Software Maintenance: An Experiment Replication at Brigham Young University", Replication in Empirical Software Engineering Research (RESER), 2011 Second International Workshop on, On page(s): 25 – 34
8. Ohtsuki, M.; Yoshida, N.; Makinouchi, A. "A source code generation support system using design pattern documents based on SGML", Software Engineering Conference, 1999. (APSEC '99) Proceedings. Sixth Asia Pacific, On page(s): 292 – 299
9. M. Fayad, W. Tsai, and M. Fulghum, "Transition to Object-Oriented Software Development," Communications of the ACM, Jan. 1996.
10. F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture - A System of Patterns.* Wileys and Sons, to appear 1996.
11. G. Booch, *Object Oriented Analysis and Design with Applications* (2 *Edition).* Redwood City, California: Ben-jamin/Cummings, 1993.
12. D. C. Schmidt and P. Stephenson, "Experiences Using Design Patterns to Evolve System Software Across Diverse OS Platforms," in*Proceedings ofthe* 9 *European Conference on Object-Oriented Programming,* (Aarhus, Denmark), ACM, August1995.
13. R. Johnson, "Documenting Frameworks Using Patterns," in *OOPLSA '92,* (Vancouver, British Columbia), pp. 63-76, ACM, October 1992.
14. A. Hussey, D.Carrington, "Comparing the MVC and PAC architectures: a formal perspective", IEE Proceedings Software Engineering, Vol 144, Issue 4, Page 224-36