.NET 7 Design Patterns In-Depth

Enhance code efficiency and maintainability with .NET Design Patterns

Vahid Farahmandian

www.bpbonline.com

First published: 2023

Published by BPB Online
WeWork
119 Marylebone Road
London NW1 5PU

UK | UAE | INDIA | SINGAPORE

ISBN 978-93-55518-729

www.bpbonline.com

Dedicated to

My beloved Parents

&

My dear wife ***Mojgan***

About the Author

Vahid Farahmandian, who currently works as the CEO of Spoota company, was born in Urmia, Iran, in 1989. He got a BSc in Computer Software Engineering from Urmia University and an MSc degree in Medical Informatics from Tarbiat Modares University. He has more than 17 years of experience in the information and communication technology field and more than a decade of experience in teaching different courses of DevOps, programming languages, and databases in various universities, institutions, and organizations in Iran. Vahid also is an active speaker in international shows and conferences, including Microsoft .NET Live TV, Azure, .NET, and SQL Server conferences. The content published by Vahid was available through YouTube and Medium and had thousands of viewers and audiences.

About the Reviewers

- **Kratika Jain** is a senior software developer specializing in .NET technologies. She has a strong understanding of C#, ASP.NET, MVC, .NET Core, SQL, and Entity Framework. She has participated in agile project management, employs continuous integration/deployment (CI/CD) using Azure DevOps, and delivered robust and scalable software solutions. As a meticulous technical reviewer, she ensures accuracy and quality in technical content. Her attention to detail allows her to identify potential pitfalls and offer valuable insights for improvement. With her expertise in .NET development and dedication to enhancing technical content, she contributes to empowering developers and enabling their success in mastering the .NET ecosystem. She is a natural problem solver, team player, adaptable, and always seeking new challenges. You can connect with her on LinkedIn at www.linkedin.com/in/kratikajain29/ or on Twitter via @_KratikaJain.

- **Gourav Garg** is a Senior Software Engineer from India who has been helping companies to build scalable products. He holds a bachelor's degree in software engineering and has been programming for 11 years. He is proficient in .net, C#, and Entity Framework. He has experience in delivering several products and many features at his work.

  Gourav has also experience with JavaScript-related tech stacks like Angular and React. He has developed quite a few open-source libraries using ES6 and Angular.

Acknowledgement

There are a few people I want to thank for the continued and ongoing support they have given me during the writing of this book. First and foremost, I would like to thank my parents for continuously encouraging me to write the book — I could have never completed this book without their support.

I also need to thank my dear wife, who has always supported me. Finally, I would like to thank all my friends and colleagues who have been by my side and supported me during all these years. I really could not stand where I am today without the support of all of them.

My gratitude also goes to the team at BPB Publications, who supported me and allowed me to write and finish this book.

Preface

This book has tried to present important design patterns (including GoF design patterns and Patterns of Enterprise Application Architecture) in software production with a simple approach, along with practical examples using .NET 7.0 and C#.

This book will be useful for software engineers, programmers, and system architects. Readers of this book are expected to have intermediate knowledge of C#.NET programming language, .NET 7.0, and UML.

Practical and concrete examples have been used in writing this book. Each design pattern begins with a short descriptive sentence and is then explained as a concrete scenario. Finally, each design pattern's key points, advantages, disadvantages, applicability, and related patterns are stated.

This book is divided into **15 chapters**, including:

Chapter 1: Introduction to Design Patterns- In this chapter, an attempt has been made to explain why design patterns are important and their role in software architecture, and basically, what is the relationship between design patterns, software design problems, and software architecture? In this chapter, various topics such as Design Principles, including SOLID, KISS, DRY, etc., and Introduction to .NET and UML are covered too.

Chapter 2: Creational Design Patterns- Creative design patterns, as the name suggests, deal with the construction of objects and how to create instances. In C# programming language, wherever an object is needed, the object can be created using the "new" keyword along with the class name. However, there are situations where it is necessary to hide the way the object is made from the user's view. In this case, creative design patterns can be useful. In this chapter, creational design patterns, one of the types of GoF design patterns, have been introduced, and it has been said that these design patterns are useful for what issues.

Chapter 3: Structural Design Patterns- Structural design patterns deal with the relationships between classes in the system. In fact, this category of design patterns determines how different objects can form a more complex structure together. In this chapter, structural design patterns, one of the types of GoF design patterns,

have been introduced, and it has been said that these design patterns are useful for what issues.

Chapter 4: Behavioral Design Patterns - Part I- This category of design patterns deals with the behavior of objects and classes. In fact, the main goal and focal point of this category of design patterns is to perform work between different objects using different methods and different algorithms. In fact, in this category of design patterns, not only objects and classes are discussed, but the relationship between them is also discussed. In this chapter, the most popular and famous behavioral design patterns, one of the types of GoF design patterns, have been introduced, and it has been said that these design patterns are useful for what issues.

Chapter 5: Behavioral Design Patterns - Part II- In continuation of the previous chapter, in this chapter, more complex and less used behavioral design patterns are discussed, and it is shown how these design patterns can be useful in dealing with the behavior of objects and classes. Although these patterns are less known or less used, their use can make much more complex problems be solved in a very simple way. In this chapter, less popular or famous behavioral design patterns, one of the types of GoF design patterns, have been introduced, and it has been said that these design patterns are useful for what issues.

Chapter 6: Domain Logic Design Patterns- To organize domain logic, Domain Logic design patterns can be used. The choice of which design pattern to use depends on the level of logical complexity that we want to implement. The important thing here is to understand when logic is complex and when it is not! Understanding this point is not an easy task, but by using domain experts, or more experienced people, it is possible to obtain a better approximation. In this chapter, it is said how to organize the logic of the domain. And in this way, what are the design patterns that help us have a more appropriate and better design? These design patterns are among the PoEAA design patterns.

Chapter 7: Data Source Architectural Design Patterns- One of the challenges of designing the data access layer is to implement how to communicate with the data source. In this implementation, it is necessary to address issues such as how to categorize SQL codes, how to manage the complexities of communicating with the data of each domain, and the mismatch between the database structure and the domain model. In this chapter, it has been said that in software architecture, communication with data sources can be considered and implemented in a suitable way. These design patterns are among the PoEAA design patterns.

Chapter 8: Object-Relational Behaviors Design Patterns- Among the other challenges that exist when communicating with the database is paying attention to behaviors. What is meant by behaviors is how the data should be fetched from the database or how it should be stored in it. For example, suppose a lot of data is fetched from the database, and some of them have changed. It will be very important to answer the question of which of the data has changed or how to store the changes again in the database, provided that the data consistency is not disturbed. Another challenge is that when the Domain Model is used, most of the models have relationships with other models, and reading a model will lead to fetching all its relationships, which will again jeopardize the efficiency. In this chapter, an attempt has been made to explain how to connect business to data sources in a proper way. These design patterns are among the PoEAA design patterns.

Chapter 9: Object-Relational Structures Design Patterns- Another challenge in mapping the domain to the database is how to map a record in the database to an object. The next challenge is how to implement all types of relationships, including one-to-one, one-to-many and many-to-many relationships. In the meantime, we may face some data that cannot and should not be mapped to any table, and we should think about this problem in our design. Finally, to implement the structure of the database, relationships such as inheritance may be used. In this case, it should be determined how this type of implementation should be mapped to the tables in the database. In this chapter, an attempt has been made to explain how to implement the data source structure in the software. These design patterns are among the PoEAA design patterns.

Chapter 10: Object-Relational Metadata Mapping Design Patterns- When we are producing software, we need to implement the mapping between tables and classes. For the software production process, this will be a process that contains a significant amount of repetitive code, and this will increase the production time. So, it will be necessary to stop writing duplicate codes and extract relationships from metadata. When this challenge can be solved, then it will be possible to generate queries automatically. Finally, when it is possible to automatically extract queries, the database can be hidden from the rest of the program. This chapter describes how to store object metadata in the data source, as well as how to create and manage queries to the data source. These design patterns are among the PoEAA design patterns.

Chapter 11: Web Presentation Design Patterns- One of the most important changes in applications in recent years is the penetration of web-based user interfaces. These types of interfaces come with various advantages, including that the client often does not need to install a special program to use them. The creation of web applications is often accompanied by the generation of server-side codes. The request is entered into the web server, and then the web server delivers the request based on the content of the request to the web application or the corresponding website. To separate the details related to the view from the data structure and logic, you can benefit from the design patterns presented in this chapter. In this chapter, the creation and handling of user interface requests are discussed, and it is stated how you can prepare and implement the view and how you can manage the requests in a suitable way. These design patterns are among the PoEAA design patterns.

Chapter 12: Distribution Design Patterns- One of the problems of implementing communication between systems is observing the level of coarseness and fineness of communication. This level should be such that both the effectiveness and efficiency during the network are not disturbed, and the data structure delivered to the client is the structure that is expected and suitable for the client. In this chapter, design patterns that can be useful in building distributed software are discussed. These design patterns are among the PoEAA design patterns.

Chapter 13: Offline Concurrency Design Patterns- One of the most complicated parts of software production is dealing with topics related to concurrency. Whenever several threads or processes have access to the same data, there is a possibility of problems related to concurrency, so one should think about concurrency in software production. Of course, there are different solutions at different levels for working and managing concurrency in enterprise software applications. For example, you can use transactions, internal features of relational databases, etc., for this purpose. Of course, this reason is not proof of the claim that concurrency management can basically be blamed on these methods and tools. In this chapter, design patterns that can be useful in solving these problems have been introduced. These design patterns are among the PoEAA design patterns.

Chapter 14: Session State Design Patterns- When we talk about transactions, we often talk about system transactions and business transactions. This discussion continues to the discussion of stateless or stateless sessions. Obviously, first, it should be determined what is meant by Stateful or Stateless. When we look at an object, this object consists of a series of data (status) and a series of behaviors. If

we assume that the object does not contain any data, then we have accepted that the object in question does not have any data with it. If we bring this discussion to enterprise software, the meaning of Stateless will be a state in which the server does not keep any data of the request between two requests. If the server needs to store data between two requests, then we will face stateful mode. This chapter talks about how to manage user sessions. Some points have been raised regarding stateless and stateful sessions. These design patterns are among the PoEAA design patterns.

Chapter 15: Base Design Patterns- When we are designing software, we need to use different design patterns. To use these patterns, it is also necessary to use a series of basic design patterns to finally provide a suitable and better design. In fact, basic design patterns provide the foundation for designing and using other patterns. In this chapter, a series of basic design patterns have been introduced, and it has been shown how the use of these design patterns can be effective on the use of other design patterns. These design patterns are among the PoEAA design patterns.

Code Bundle and Coloured Images

Please follow the link to download the
Code Bundle and the ***Coloured Images*** of the book:

https://rebrand.ly/g3mn07e

The code bundle for the book is also hosted on GitHub at **https://github.com/bpbpublications/.NET-7-Design-Patterns-In-Depth**. In case there's an update to the code, it will be updated on the existing GitHub repository.

We have code bundles from our rich catalogue of books and videos available at **https://github.com/bpbpublications**. Check them out!

Errata

We take immense pride in our work at BPB Publications and follow best practices to ensure the accuracy of our content to provide with an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors, if any, that may have occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at :

errata@bpbonline.com

Your support, suggestions and feedbacks are highly appreciated by the BPB Publications' Family.

Piracy

If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at **business@bpbonline.com** with a link to the material.

If you are interested in becoming an author

If there is a topic that you have expertise in, and you are interested in either writing or contributing to a book, please visit **www.bpbonline.com**. We have worked with thousands of developers and tech professionals, just like you, to help them share their insights with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions. We at BPB can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about BPB, please visit **www.bpbonline.com**.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

Table of Contents

Chapter 1
Introduction to Design Patterns

Introduction

One of the problems in understanding and using design patterns is the need for proper insight into software architecture and the reason for using design patterns. When this insight does not exist, design patterns will increase complexity. As they are not used in their proper place, the use of design patterns will be considered a waste of work. The reason for this is that the design patterns will not be able to have a good impact on quality because they need to be placed in the right place.

In this chapter, an attempt has been made to briefly examine the software architecture and design patterns. The enterprise applications architecture has been introduced, and the relationship between software design problems and design patterns has been clarified. In the rest of the chapter, a brief look at .NET, some object-oriented principles, and the UML is given because, throughout the book, UML is used for modeling, and the .NET framework and C# language are used for sample codes.

Structure

In this chapter, we will cover the following topics:

- What is software architecture
- What are design patterns

- GoF design patterns
- Enterprise application and its design patterns
    - Different types of enterprise applications
- Design patterns and software design problems
    - Effective factors in choosing a design pattern
- .NET
    - Introduction to object orientation in .NET
- Object orientation SOLID principles
- UML class diagram
- Conclusion

Objectives

By the end of this chapter, you will be able to understand the role and place of design patterns in software design, be familiar with software architecture, and evaluate software design problems from different aspects. You are also expected to have a good view of SOLID design principles at the end of this chapter and get to know .NET and UML.

What is software architecture

Today, there are various definitions for software architecture. The system's basic structure, related to design decisions, must be made in the initial steps of software production. The common feature in all these definitions is their importance. Regardless of our attitude towards software architecture, we must always consider that suitable architecture can be developed and maintained. Also, when we want to look at the software from an architectural point of view, we must know what elements and items are of great importance and always try to keep those important elements and items in the best condition.

Consider software that needs to be better designed, and its essential elements must be identified. During the production and maintenance of this software, we will need help with various problems, including implementing changes, which will reduce the speed of providing new features and increase the volume of software errors and bugs. For example, pay attention to the following figure:

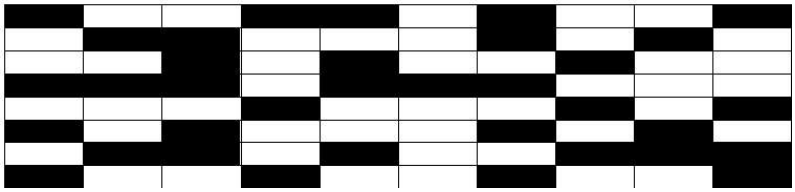

***Figure 1.1**: An example of software without proper architecture*

In the preceding figure, full cells are the new features provided, and empty cells are the design and architectural problems and defects.

If we consider one row of *Figure 1.1*, the following figure will be seen:

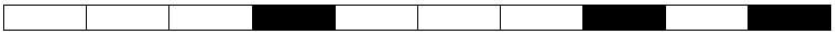

***Figure 1.2**: Sample feature delivery in software without proper architecture*

We see how much time it takes to provide three different features. If the correct design and architecture were adopted, new features would be delivered more quickly. The same row could be presented as the following figure:

***Figure 1.3**: Sample feature delivery in software WITH proper architecture*

The difference in length in the preceding two forms (*Figure 1.2 and Figure 1.3*) is significant. This shows the importance of the right design and architecture in the software. A high-quality infrastructure in the short term may indicate that production speed decreases. This natural and high-quality infrastructure will show its effect in the long run.

The following figure shows the relationship between Time and Output:

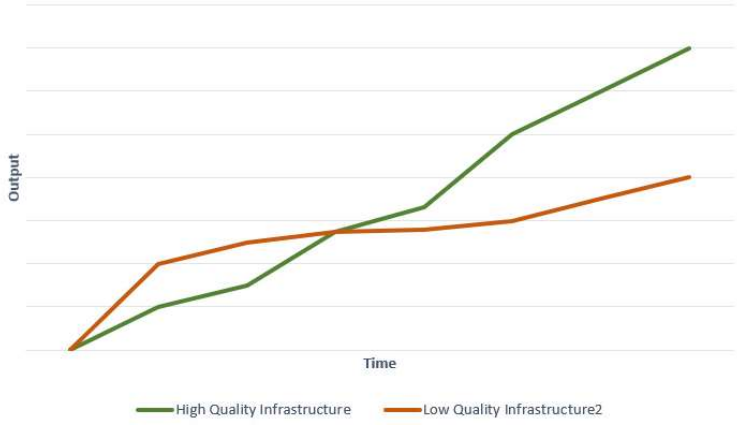

***Figure 1.4**: Time-Output Relation in Software Delivery*

In *Figure 1.4*, at the beginning of the work, reaching the output with a low-quality Infrastructure is faster than with a high-quality Infrastructure. However, with the passage of time and the increase in the capabilities and complexity of the software, the ability to maintain and apply software change is accelerated with better quality infrastructure. This will reduce costs, increase user satisfaction, and improve maintenance.

In this regard, *Gerald Weinberg*, the late American computer science scientist, has a quote that says,

"If builders-built buildings the way programmers wrote programs, then the first woodpecker that came along would destroy civilization."

Weinberg tried to express the importance of infrastructure and software architecture. According to Weinberg's quote, paying attention to maintainability in the design and implementation of software solutions is important. Today, various principles can be useful in reaching a suitable infrastructure.

Some of these principles are as follows:

- **Separation of concerns**: Different software parts should be separated from each other according to their work.
- **Encapsulation**: This is a way to restrict the direct access to some components of an object, so users cannot access state values for all the variables of a particular object. Encapsulation can hide data members, functions, or methods associated with an instantiated class or object. Users will have no idea how classes are implemented or stored, and the users will only know that the values are being passed and initialized (Data Hiding). Also, it would be easy to change and adapt to new requirements (ease of use) using Encapsulation.
- **Dependency inversion**: High-level modules should not depend on low-level modules, and the dependence between these two should only happen through abstractions. To clarify the issue, consider the following example:

  We have two different times in software production: compile and run time. Suppose that in a dependency graph at compile-time, the following relationship exists between classes **A**, **B**, and **C**:

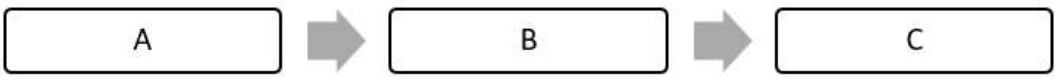

***Figure 1.5**: Relationship between A, B, and C in compile-time*

  As you can see, at compile-time, **A** is directly connected to **B** to call a method in **B**, and the exact relationship is true for the relationship between **B** and