Kjell Coppin

Kjellcoppin@gmail.com

Ted talk script

Real world use of design patterns in .Net applications

Table of Contents

[Introduction 2](#_Toc162816689)

[Goal 2](#_Toc162816690)

[Gang of four 2](#_Toc162816691)

[Explain a few in detail with example code 2](#_Toc162816692)

[Cases from bestmix 2](#_Toc162816693)

# Introduction

# Ladies and gentlemen, imagine a world where every architectural marvel, from the towering skyscrapers of New York to the serene temples of Kyoto, follows a set of universal principles. These principles, known as design patterns, provide architects with a language of solutions to recurring challenges in their craft. But what if I told you that this concept didn't just stay within the realm of buildings and bridges? What if it transcended into the very codes and algorithms that power our digital world?

# In 1977, Christopher Alexander, an architect with a keen eye for patterns, introduced a revolutionary idea in his book 'A Pattern Language.' He proposed a language centered around entities called patterns—timeless solutions to architectural problems found across different cultures. These patterns weren't just bricks and mortar; they were the building blocks of universal design thinking.

# Fast forward to 1995, a landmark year for software engineering. Four visionaries—Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, Also known as ‘the Gang of Four’ —crafted a masterpiece: 'Design Patterns: Elements of Reusable Object-Oriented Software.' This book brought the concept of design patterns into the digital age. Drawing inspiration from Alexander's work in architecture, they observed how software development teams faced similar challenges. Just as a blueprint guides an architect, these design patterns became the blueprint for software engineers, guiding them through the complexities of code.

# But what exactly is a design pattern? Imagine it as a roadmap, not for physical structures, but for the structures of code itself. These are tried-and-tested solutions to common programming problems, offering a clear path through the maze of software development. From object composition to class structure, design patterns in languages like C# provide a framework for developers to create more readable, maintainable, and efficient code.

# Goal

Hello, I'm Kjell Coppin, a passionate software engineer with a focus on .NET development. Today, in this TED talk, I aim to empower fellow developers by sharing insights on writing cleaner, more efficient code.

# Gang of four

In the famous book written by the Gang of four, 3 types of design patterns are talked about.

**Creational Patterns**: These patterns focus on the process of object creation, providing mechanisms for creating objects in a manner suitable for a given situation. Examples include Factory Method, Abstract Factory, Singleton, Builder, and Prototype patterns.

**Structural Patterns**: Structural patterns deal with the composition of classes or objects to form larger structures. They help ensure that if one part of a system changes, the entire system doesn't need to do so. Examples include Adapter, Decorator, Proxy, Composite, and Facade patterns.

**Behavioral Patterns**: Behavioral patterns are concerned with communication between objects, focusing on how objects distribute responsibilities and duties among themselves. Examples include Observer, Strategy, Command, Iterator, and State patterns.

For each type of pattern, use one to clarify the pattern (talking about creational patterns e.g. ‘For example with the factory pattern we create a factory class which is used to create...’)

# Explain a few in detail with example code

* Explain pattern
* Use cases, benefits and cons
* Example code

# Cases from Bestmix

Talk about some cases of design pattern use in the Bestmix codebase (idk yet how many, got to see how big they are).

For every case:

* Give some context about code I am talking about
* Show snippets
* Explain what design pattern we are dealing with
* Why is this a good/bad choice
* What are possible improvements?
* Show improved snippets