Kjell Coppin

Kjellcoppin@gmail.com

Ted talk script

Real world use of design patterns in .Net applications

Table of Contents

[Introduction 2](#_Toc166338290)

[Goal 2](#_Toc166338291)

[Gang of four 2](#_Toc166338292)

[Cases from Bestmix 2](#_Toc166338293)

[Observer pattern: validator 2](#_Toc166338294)

# 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.

**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.

**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.

# Cases from Bestmix

Since i did my internship at Bestmix i will use their codebase to show you how some of these design patterns are used in real production software.

Bestmix recipe management is a software that allows users to accurately manage recipes and generate product specifications.

## Observer pattern: validator

An interesting behavioural pattern is the Observer pattern.

Observer lets you define a subscription mechanism to notify multiple objects about any events that happen to the object they’re observing.

In Bestmix there are many entities such as recipes, products, parameters and much more. These entities can be created and edited, which means they need to be validated as well. An entity with invalid data, such as empty mandatory fields should not be saved to the database. For this reason there is a validator class.

The observer pattern is implemented in 2 ways here, the validator subscribes to an entity, so that the entity is validated the moment it changes. But an entity can also subscribe to the validator, so that it knows and can react to it being validated.

To do this, the validator class has an event and a delegate. So that entities can subscribe to it.

One place where this validator is used is in the ChangeMultipleRecipesViewmodel. This viewmodel shows a grid with multiple entities, which you can edit in the grid itself.

In this viewmodel, a validator gets added to each row (with each row representing an entity)

In the 2 highlighted lines you can see that first the row subscribes to its validator and assigns UpdateValidity() as event handler. Then the validator subscribes to the row’s entity using its own Subscribe method. This subscribe method looks like this:

Entity.PropertyChanged is an event that variables have by default in C#, and the validator subscribes to it, using propertyChangedHandler as event handler. The main thing that propertyChangedHanlder does is trigger the validate method:

The validate method then triggers the EntityValidated, so now the row knows it has been validated and it’s handler method can set the isValid parameter and potentially block it from being saved to the database.