Refactoring to SOLID C# Code

Assessing Legacy Code

Steve “ardalis” Smith

Founder and Principal Architect, NimblePros

@ardalis | ardalis.com

Refactoring to SOLID C# Code

Version Check

Version Check

This course was created by using:

- C# 12
- .NET 8
- Visual Studio 2022

Version Check

This course is 100% applicable to:

- C# 10+
- .NET 6+

Overview

Introduce a Legacy Code Example

- A complex pricing service

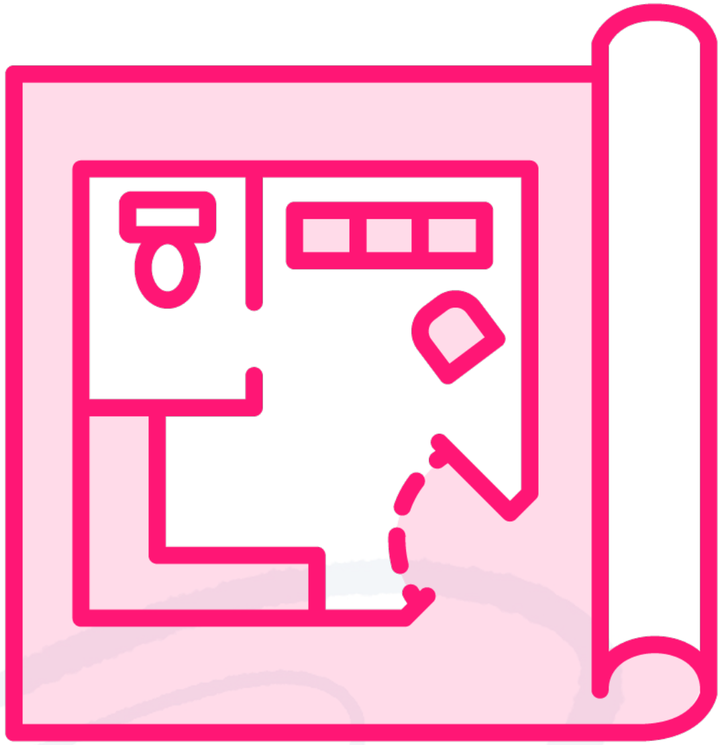
Review the Code and Identify Problems

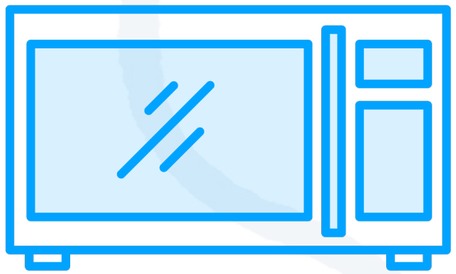
- Leverage principles
- Identify known code smells

Make a Plan to Address Issues

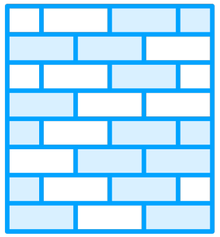
- Consider various planning tools
- Prioritize and scope fixes

Introduction

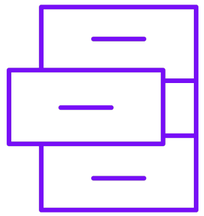




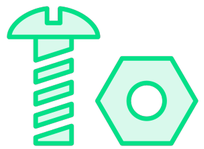
CalculateTotalPrice()



Per Wall



Per Cabinet



Per Feature

CalculateTotalPrice()

Realtime Quote

Place an Order

Generate Report

Demo

Review the Initial State of the Sample Code

Review – What did we find?

Reviewing the Legacy Code

Code Metrics Results

Filter: None Min: Max:

Hierarchy	Maintainability Index	Cyclomatic Complexity	Depth of Inheritance	Class Coupling	Lines of Source code
MegaPricer (Debug)	72	422	6	189	6,842
Program	39	6	1	48	61
SeedData	59	8	1	19	184
AspNetCore	96	12	4	10	61
MegaPricer.Data	95	155	6	33	229
MegaPricer.Data.Migrations	37	25	2	46	4,753
MegaPricer.Pages	81	132	4	45	722
MegaPricer.Pages.Shared	75	42	4	31	420
MegaPricer.Services	45	42	1	28	412
GlobalHelpers	76	2	1	5	18
PricingService	14	40	1	24	390
CalculatePrice(int, int, string, string) : string	14	40		24	387

High Complexity

Too Many Responsibilities

Tight Coupling

Demo

Identifying and Recording Problems with the Sample Code

Make a Plan to Address Problems

Planning

Spreadsh

Tracking
ftware

ardalis commented 11 minutes ago

Convert to issue

- [ ] Static method
- [ ] string return type - replace with a return object
- [ ] refType should be replaced with different objects / strategies
- [ ] Refactor checking session for PricingOff
- [ ] Remove unused locals
- [ ] Refactor prices to use decimal
- [ ] Declare locals where they're used
- [ ] Remove DataTables - use strongly typed types
- [ ] Refactor writing to Session - CAREFULLY
- [ ] Move try-catch to a wrapper / decorator
- [ ] Simplify validation/guards
- [ ] Return a Result type with support for listing validation problems
- [ ] Refactor Kitchen to use a Repository or similar to fetch from data source
- [ ] Refactor all ADO.NET Connection blocks to use Dapper and/or helper methods
- [ ] Replace `if refType` checks with polymorphism / strategy pattern / DI
- [ ] Group locals into types or pull up fields/properties on class
- [ ] Focus on the price calculation logic - pull in data as parameters or pass in ways to get data needed
- [ ] Delete empty else block
- [ ] Load actual Feature objects instead of using DataReader/DataTables
- [ ] Make references to Color ID consistent everywhere (rename Color -> ColorId, etc.)
- [ ] Refactor !IsIland logic into its own special method
- [ ] Move String Format logic to an extension method (if still needed)
- [ ] Use using for StringWriter to obviate finally block

Task Considerations

Rough Effort

Risk

Value

Summary

Exploring the Sample

Reviewing Legacy Code

- What to look for
- How to recognize common problems

Documenting Potential Problem Areas

Making a (Tentative) Plan to Address Issues

Up Next:

Updating Legacy Code