Strengthening Our Code's Defenses

Deborah Kurata
CONSULTANT | SPEAKER | AUTHOR | MVP | GDE

@deborahkurata | blogs.msmvps.com/deborahk/

Update the Price for a Product

Update Price for: Hammer

Cost — Current cost (required)

Price — Suggested price (required)

Category — Category (required)

Reason — Reason for the price change

Effective Date — mm/dd/yyyy

Calculate Margin | Cancel

Update the Price for a Product

$$\text{Profit Margin} = \frac{\text{Suggested Price - Cost}}{\text{Suggested Price}}$$

Profit Margin >= Corporate Minimum

Update the Price for a Product

Update Price for: Hammer

Cost: 100

Price: 199.99

Category: TBX

Reason: The costs rose significantly.

Effective Date: 05/20/2020

Calculate Margin | Cancel

Minimim Required Profit Margin	Calculated Profit Margin
40%	50%

Confirm Price Change | Cancel

Update the Price for a Product

Update Price for: Hammer

Success
Product Price Changed

Cost: 100

Price: 199.99

Category: TBX

Reason: The costs rose significantly.

Effective Date: 05/20/2020

Calculate Margin | Cancel

Minimim Required Profit Margin	Calculated Profit Margin
40%	50%

Confirm Price Change | Cancel

Prototypes

```
private void Calculate_Click(object sender, RoutedEventArgs e)
{
    // Calculate and check the profit margin
    var price = priceText.Text;
    var cost = costText.Text;
    calculatedMargin = ((price - cost) / price) * 100;

    isAcceptable = calculatedMargin >= 40;

    // Display the results
    marginText.Text = calculatedMargin;
}
```

```
private void Save_Click(object sender, RoutedEventArgs e)
{
    // Save the new price information
    // ...

    // Display success or fail
    messageText.Text = "Success";
}
```

Prototypes

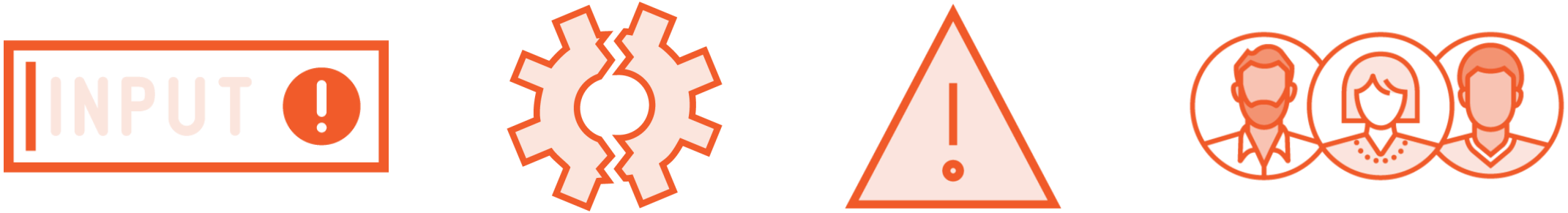
```
private void Calculate_Click(object sender, RoutedEventArgs e)
{
    // Calculate and check the profit margin
    var price = priceText.Text;
    var cost = costText.Text;
    calculatedMargin = ((price - cost) / price) * 100;

    isAcceptable = calculatedMargin >= 40;

    // Display the results
    marginText.Text = calculatedMargin;
}
```

```
public IActionResult Calculate(ProductViewModel vm)
{
    // Calculate and check the profit margin
    var price = vm.Price;
    var cost = vm.Cost;
    calculatedMargin = ((price - cost) / price) * 100;

    isAcceptable = calculatedMargin >= 40;

    // Display the results
    ViewData["ProfitMargin"] = calculatedMargin;
    return View();
}
```

Defensive Coding

```
private void Calculate_Click(object sender, RoutedEventArgs e)
{
    // Calculate and check the profit margin
    var price = priceText.Text;
    var cost = costText.Text;
    calculatedMargin = ((price - cost) / price) * 100;

    isAcceptable = calculatedMargin >= 40;

    // Display the results
    marginText.Text = calculatedMargin;
}
```

```
public IActionResult Calculate(ProductViewModel vm)
{
    // Calculate and check the profit margin
    var price = vm.Price;
    var cost = vm.Cost;
    calculatedMargin = ((price - cost) / price) * 100;

    isAcceptable = calculatedMargin >= 40;

    // Display the results
    ViewData["ProfitMargin"] = calculatedMargin;
    return View();
}
```



Identify Potential Weaknesses

```
private void Calculate_Click(object sender, RoutedEventArgs e)
{
    // Calculate and check the profit margin
    var price = priceText.Text;
    var cost = costText.Text;
    calculatedMargin = ((price - cost) / price) * 100;

    isAcceptable = calculatedMargin >= 40;

    // Display the results
    marginText.Text = calculatedMargin;
}
```

```
public IActionResult Calculate(ProductViewModel vm)
{
    // Calculate and check the profit margin
    var price = vm.Price;
    var cost = vm.Cost;
    calculatedMargin = ((price - cost) / price) * 100;

    isAcceptable = calculatedMargin >= 40;

    // Display the results
    ViewData["ProfitMargin"] = calculatedMargin;
    return View();
}
```

- Copy/paste
- Presentation logic mixed with business logic
- Difficult to unit test
- No validation
- No exception handling

Defensive Coding

Code comprehension

Code quality

Code predictability

Code Comprehension

Easy to read

Have a clear intent

Simple

Thoughtful

Refactoring is the process of
restructuring code,
altering its organization
without changing its behavior.

Refactor

```
private void Calculate_Click(object sender, RoutedEventArgs e)
{
    // Calculate and check the profit margin
    var price = priceText.Text;
    var cost = costText.Text;
    calculatedMargin = ((price - cost) / price) * 100;

    isAcceptable = calculatedMargin >= 40;

    // Display the results
    marginText.Text = calculatedMargin;
}
```

```
public IActionResult Calculate(ProductViewModel vm)
{
    // Calculate and check the profit margin
    var price = vm.Price;
    var cost = vm.Cost;
    calculatedMargin = ((price - cost) / price) * 100;

    isAcceptable = calculatedMargin >= 40;

    // Display the results
    ViewData["ProfitMargin"] = calculatedMargin;
    return View();
}
```

Improving Code Comprehension

Single responsibility principle

Separation of concerns

Don't repeat yourself (DRY)

Single Responsibility Principle

Multi-tasker

Single focused

Windows Code

```
private void Calculate_Click(object sender, RoutedEventArgs e)
{
    // Calculate and check the profit margin
    var price = priceText.Text;
    var cost = costText.Text;
    calculatedMargin = ((price - cost) / price) * 100;

    isAcceptable = calculatedMargin >= 40;

    // Display the results
    marginText.Text = calculatedMargin;
}
```

```
private void Save_Click(object sender, RoutedEventArgs e)
{
    // Save the new price information
    // ...

    // Display success or fail
    messageText.Text = "Success";
}
```

Single Responsibility Principle

```
private void Calculate_Click(object sender, RoutedEventArgs e)
{
    // Calculate and check the profit margin
    var price = priceText.Text;
    var cost = costText.Text;
    calculatedMargin = ((price - cost) / price) * 100;

    isAcceptable = calculatedMargin >= 40;

    // Display the results
    marginText.Text = calculatedMargin;
}
```

```
private void Save_Click(object sender, RoutedEventArgs e)
{
    // Save the new price information
    // ...

    // Display success or fail
    messageText.Text = "Success";
}
```

```
public ??? ???(???)
{
    // Calculate profit margin
    calculatedMargin = ((price - cost) / price) * 100;
}
```

```
public ??? ???(???)
{
    // Save the new price information
    // ...
}
```

```
public ??? ???(???)
{
    // Check the profit margin
    isAcceptable = calculatedMargin >= 40;
}
```

Single Responsibility Principle

```
public ??? CalculateMargin(???)
{
    // Calculate profit margin
    calculatedMargin = ((price - cost) / price) * 100;
}
```

```
public ??? CheckMinimumMargin(???)
{
    // Check the profit margin
    isAcceptable = calculatedMargin >= 40;
}
```

```
public ??? SavePrice(???)
{
    // Save the new price information
    // ...
}
```

Single Responsibility Principle

```
public ??? CalculateMargin(string cost, string price)
{
    // Calculate profit margin
    calculatedMargin = ((price - cost) / price) * 100;
}
```

```
public ??? CheckMinimumMargin(decimal calculatedMargin)
{
    // Check the profit margin
    isAcceptable = calculatedMargin >= 40;
}
```

```
public ??? SavePrice(string cost, string price, string category,
                     string reason, string effectiveDate)
{
    // Save the new price information
    // ...
}
```

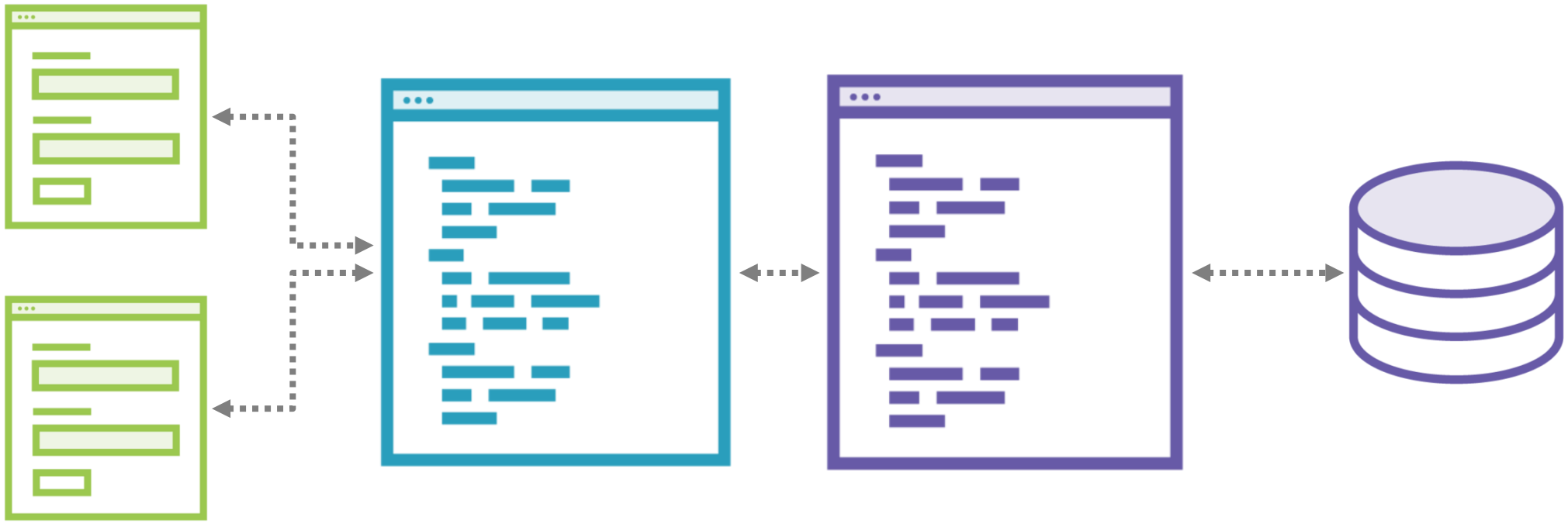
Single Responsibility Principle

```
public decimal CalculateMargin(string cost, string price)
{
    // Calculate profit margin
    return ((price - cost) / price) * 100;
}
```

```
public boolean CheckMinimumMargin(decimal calculatedMargin)
{
    // Check the profit margin
    return calculatedMargin >= 40;
}
```

```
public boolean SavePrice(string cost, string price, string category,
                         string reason, string effectiveDate)
{
    // Save the new price information
    // ...
    return success;
}
```

Separation of Concerns

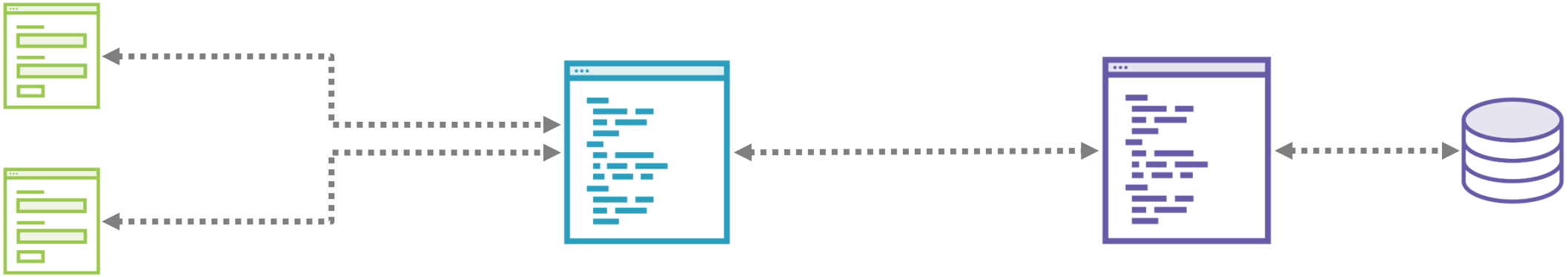


Presentation Layer

Business logic or Services Layer

Data Layer

Separation of Concerns



```
private void Calculate_Click(object sender,
                             RoutedEventArgs e)
{ // ... }
```

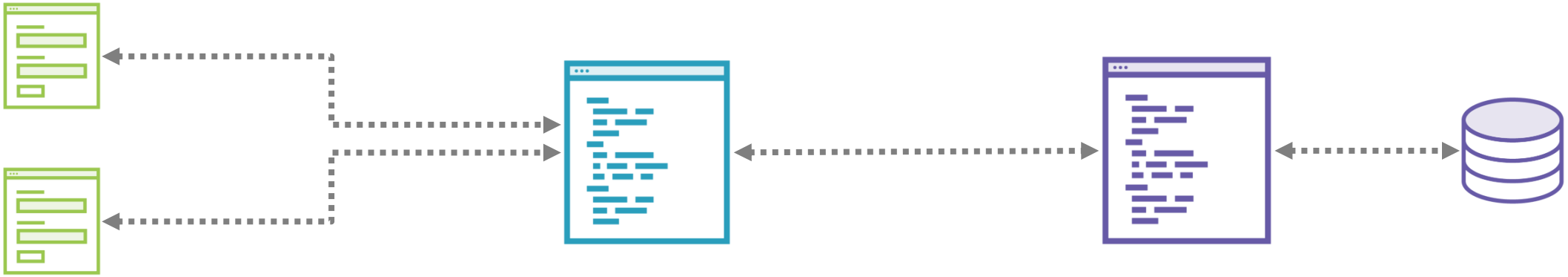
```
private void Save_Click(object sender,
                        RoutedEventArgs e)
{ // ... }
```

```
public decimal CalculateMargin(string cost, string price)
{ // ... }
```

```
public boolean CheckMinimumMargin(decimal calculatedMargin)
{ // ...}
```

```
public boolean SavePrice(string cost, string price,
                         string category, string reason,
                         string effectiveDate)
{ // ... }
```

Separation of Concerns



```
private void Calculate_Click(object sender,
                             RoutedEventArgs e)
{ // ... }
```

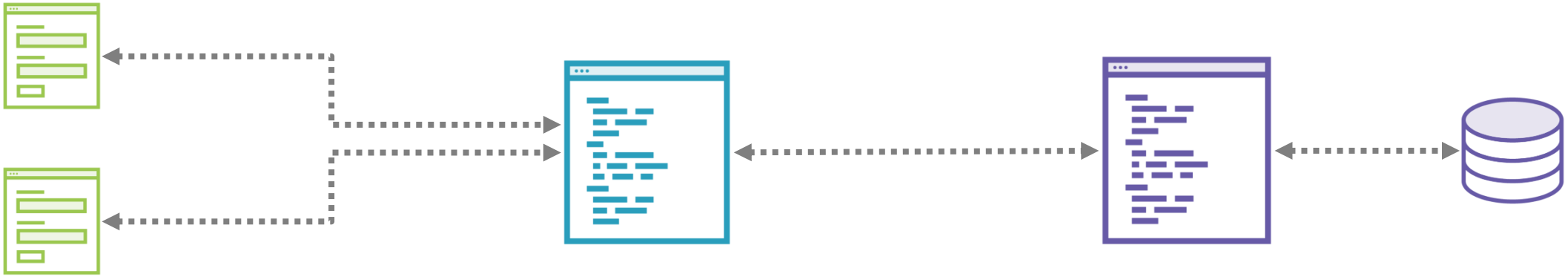
```
private void Save_Click(object sender,
                        RoutedEventArgs e)
{ // ... }
```

```
public decimal CalculateMargin(string cost, string price)
{ // ... }
```

```
public boolean CheckMinimumMargin(decimal calculatedMargin)
{ // ...}
```

```
public boolean SavePrice(string cost, string price,
                         string category, string reason,
                         string effectiveDate)
{ // ... }
```

Separation of Concerns



```
private void Calculate_Click(object sender,
                             RoutedEventArgs e)
{ // ... }
```

```
private void Save_Click(object sender,
                        RoutedEventArgs e)
{ // ... }
```

```
public decimal CalculateMargin(string cost, string price)
{ // ... }
```

```
public boolean CheckMinimumMargin(decimal calculatedMargin)
{ // ...}
```

```
public boolean SavePrice(string cost, string price,
                         string category, string reason,
                         string effectiveDate)
{ // ... }
```

Windows UI

```
private void Calculate_Click(object sender, RoutedEventArgs e)
{
    // Calculate and check the profit margin
    var price = priceText.Text;
    var cost = costText.Text;
    calculatedMargin = CalculateMargin(cost, price);

    isAcceptable = CheckMinimumMargin(calculatedMargin);

    // Display the results
    marginText.Text = calculatedMargin;
}
```

```
public decimal CalculateMargin(string cost, string price)
{
    // Calculate profit margin
    return ((price - cost) / price) * 100;
}
```

```
public boolean CheckMinimumMargin(decimal calculatedMargin)
{
    // Check the profit margin
    return calculatedMargin >= 40;
}
```

Web UI

```
public IActionResult Calculate(ProductViewModel vm)
{
    // Calculate and check the profit margin
    var price = vm.Price;
    var cost = vm.Cost;
    calculatedMargin = ((price - cost) / price) * 100;

    isAcceptable = calculatedMargin >= 40;

    // Display the results
    ViewData["ProfitMargin"] = calculatedMargin;
    return View();
}
```

Don't
Repeat
Yourself

Don't
Repeat
Yourself

Don't
Repeat
Yourself

Windows UI

```
private void Calculate_Click(object sender, RoutedEventArgs e)
{
    // Calculate and check the profit margin
    var price = priceText.Text;
    var cost = costText.Text;
    calculatedMargin = CalculateMargin(cost, price);

    isAcceptable = CheckMinimumMargin(calculatedMargin);

    // Display the results
    marginText.Text = calculatedMargin;
}
```

```
public decimal CalculateMargin(string cost, string price)
{
    // Calculate profit margin
    return ((price - cost) / price) * 100;
}
```

```
public boolean CheckMinimumMargin(decimal calculatedMargin)
{
    // Check the profit margin
    return calculatedMargin >= 40;
}
```

Web UI

```
public IActionResult Calculate(ProductViewModel vm)
{
    // Calculate and check the profit margin
    var price = vm.Price;
    var cost = vm.Cost;
    calculatedMargin = ((price - cost) / price) * 100;

    isAcceptable = calculatedMargin >= 40;

    // Display the results
    ViewData["ProfitMargin"] = calculatedMargin;
    return View();
}
```

Refactored Code

```
private void Calculate_Click(object sender,
                             RoutedEventArgs e)
{ // ... }
```

```
private void Save_Click(object sender,
                        RoutedEventArgs e)
{ // ... }
```

```
public IActionResult Calculate(ProductViewModel vm)
{ // ... }
```

```
public IActionResult Save(string cost, string price,
                          string category, string reason,
                          string effectiveDate)
{ // ... }
```

```
public decimal CalculateMargin(string cost, string price)
{
    // Calculate profit margin
    return ((price - cost) / price) * 100;
}
```

```
public boolean CheckMinimumMargin(decimal calculatedMargin)
{
    // Check the profit margin
    return calculatedMargin >= 40;
}
```

```
public boolean SavePrice(string cost, string price,
                         string category, string reason,
                         string effectiveDate)
{ // ... }
```

Defensive Coding

Code comprehension

Code quality

Code Quality

Is it a thing of beauty?

Does it work?

Does it meet requirements?

Improving Code Quality

Code reviews

Code execution

Unit testing

Automated Code Testing

Test Explorer

15 11 4

Search Test Explorer

Test	Duration	Traits	Error Message
APM.SL.Test (15)	16 ms		
APM.SL.Test (15)	16 ms		
ProductTest (15)	16 ms		
CalculateMargin_WhenInvalidCostContainsDollar_ShouldError	1 ms		
CalculateMargin_WhenInvalidCostIsEmpty_ShouldGenerateError	< 1 ms		
CalculateMargin_WhenInvalidCostIsNotANumber_ShouldGenerateError	< 1 ms		
CalculateMargin_WhenInvalidPriceIs0_ShouldGenerateError	< 1 ms		
CalculateMargin_WhenInvalidPriceIsEmpty_ShouldGenerateError	< 1 ms		
CalculateMargin_WhenInvalidPriceIsNotANumber_ShouldGenerateError	< 1 ms		
CalculateMargin_WhenValidCost50PercentOfPrice_ShouldReturn50	< 1 ms		
CalculateMargin_WhenValidCostCloseToPrice_ShouldReturn1	< 1 ms		Assert.Equal() Failure Expected: 1 Actual: 0.99
CalculateMargin_WhenValidCostContainsDecimal50PercentOfPrice_ShouldReturn50	12 ms		Assert.Equal() Failure Expected: 50 Actual: 50.45
CalculateMargin_WhenValidCostEqualPrice_ShouldReturn0	< 1 ms		
CalculateMargin_WhenValidCostIsMoreThanPrice_ShouldReturnNegative	< 1 ms		
CalculateMargin_WhenValidCostIsZero_ShouldReturn100	3 ms		
CalculateMargin_WhenValidCostLessThan1_ShouldReturn100	< 1 ms		Assert.Equal() Failure Expected: 100 Actual: 99.99
CalculateMargin_WhenValidCostOneThirdOfPrice_ShouldRoundTo33	< 1 ms		Assert.Equal() Failure Expected: 33 Actual: 33.33
CalculateMargin_WhenValidSmallValues50PercentOfPrice_ShouldReturn50	< 1 ms		

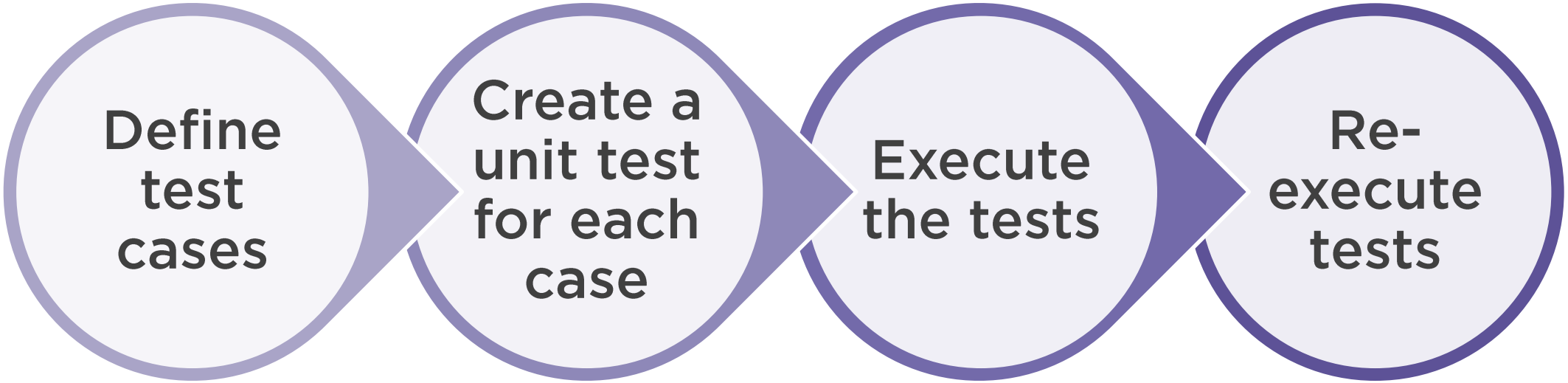
Visual Studio Unit Testing Options

MS Test

NUnit

XUnit

Unit Testing Steps



Defining Test Cases

```
public decimal CalculateMargin(string cost, string price)
{
    // Calculate profit margin
    return ((price - cost) / price) * 100;
}
```

Defining Test Cases

Valid inputs

Data entry rules

Edge cases

Invalid inputs

Possible exceptions

Empty and null

Demo

Examine and execute unit tests

Defensive Coding

Code comprehension

Code quality

Code predictability

Code predictability:

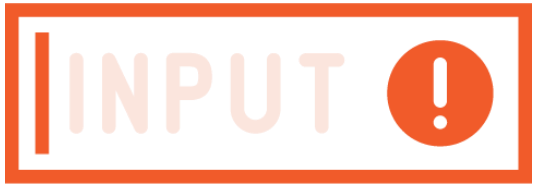
Principle of Least Surprise

The principle of least surprise suggests that each feature of our application "should behave in a way that most users will expect it to behave".

Wikipedia

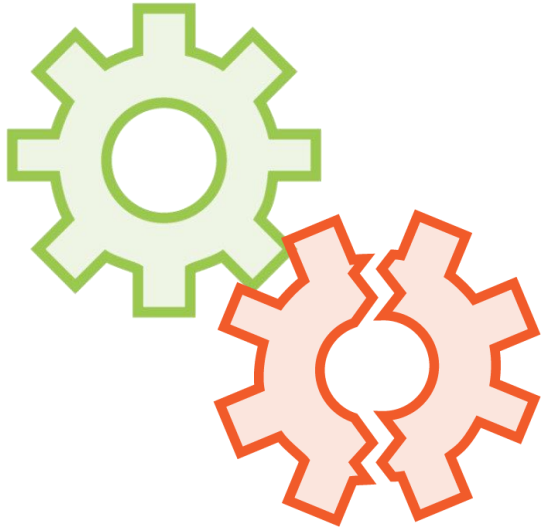
What should happen if...?

What Should Happen If...?



The user enters invalid data?

The user breaks a business rule?



An operation succeeds, or fails?

Something goes wrong?

Line of Defense: User Interface

Update Price for: Hammer

Cost — Current cost (required)

Price — Suggested price (required)

Category — Category (required)

Reason — Reason for the price change

Effective Date — mm/dd/yyyy

Calculate Margin | Cancel

Update Price for: Hammer

Cost — 100

Price — Suggested price (required)

Category — Select a Category...

Reason — Reason for the price change

Effective Date — mm/dd/yyyy

Calculate Margin | Cancel

Line of Defense: User Interface

Update Price for: Hammer

Cost — Current cost (required)

Price — Suggested price (required)

Category — Category (required)

Reason — Reason for the price change

Effective Date — mm/dd/yyyy

Calculate Margin | Cancel

Update Price for: Hammer

Cost — 100

Price — 200

Category — Select a Category...

- Select a Category...
- Garden
- Toolbox
- Gaming

Reason

Effective Date — mm/dd/yyyy

Calculate Margin | Cancel

Line of Defense: User Interface

Update Price for: Hammer

Cost — Current cost (required)

Price — Suggested price (required)

Category — Category (required)

Reason — Reason for the price change

Effective Date — mm/dd/yyyy

Calculate Margin | Cancel

Update Price for: Hammer

Cost — 100

Price — 200

Category — Garden

Reason — Overstocked

Effective Date — 04/14/2020

April 2020

Sun	Mon	Tue	Wed	Thu	Fri	Sat
29	30	31	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	1	2

Line of Defense: User Interface

Update Price for: Hammer

Cost: Current cost (required)

Price: Suggested price (required)

Category: Category (required)

Reason: Reason for the price change

Effective Date: mm/dd/yyyy

Calculate Margin | Cancel

Update Price for: Hammer

Cost: 100

Price: Suggested price (required)

Price is required.

Category: Garden

Reason: Overstocked

Effective Date: 04/14/2020

Calculate Margin | Cancel

Line of Defense: Our Methods

```
public decimal CalculateMargin(string cost, string price)
{
    // Calculate profit margin
    return ((price - cost) / price) * 100;
}
```

Guard against invalid arguments

Return predictable results

Only throw expected exceptions

Guidelines and Summary

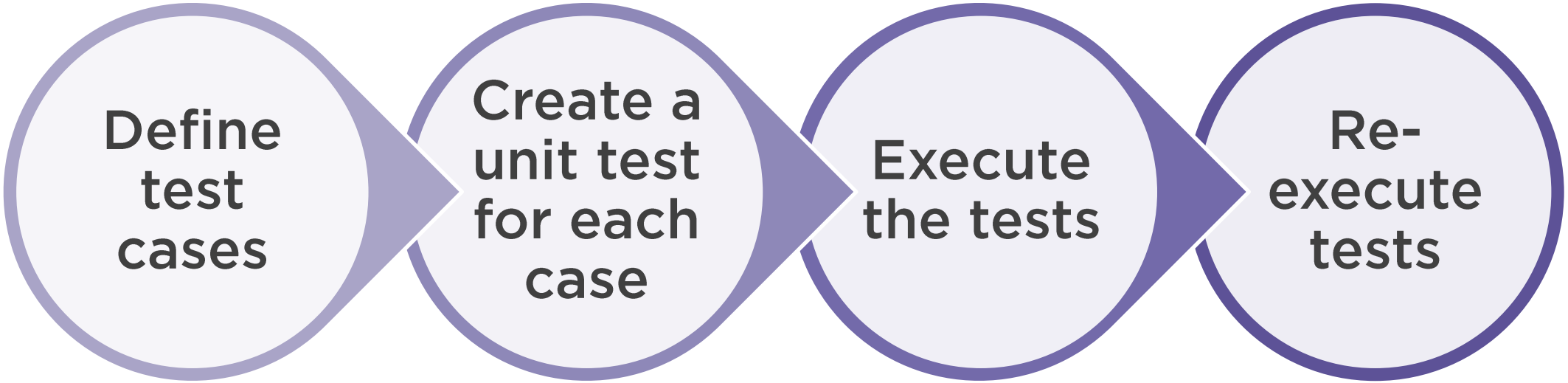
Improve Code Comprehension

Single responsibility principle

Separation of concerns

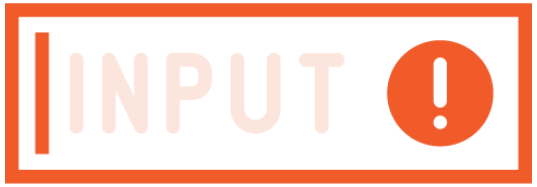
Don't repeat yourself (DRY)

Improve Code Quality



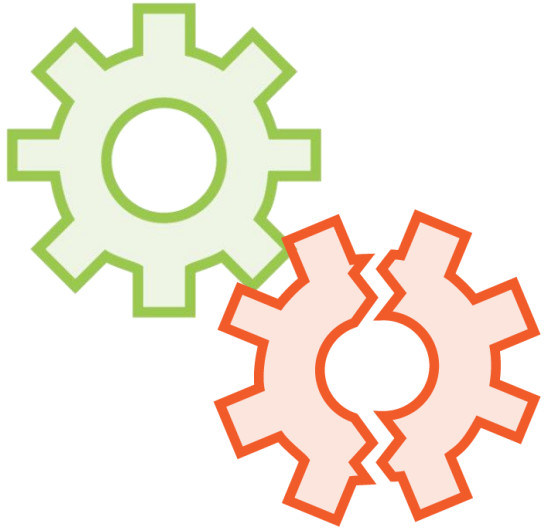
- Valid inputs
- Data entry business rules
- Edge cases
- Invalid inputs
- Generated exceptions
- Empty and null values

Improve Predictability



Incorrect entry

If a business rule is broken



Invalid operations

When something goes wrong