

APPLYING AOP TO INCREASE SOLUTION DEVELOPMENT VELOCITY



Sean P. McDonough
Senior Solution Architect and Consultant
Akumina, Inc.

Cincy Deliver 2024 Sponsors

Diamond



zenhub

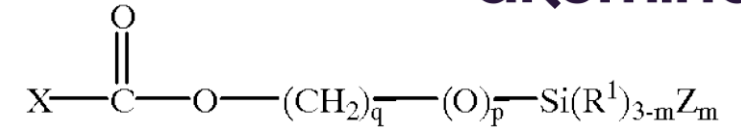
Gold



Silver



A BIT ABOUT ME ...



Started professional career as a polymer chemist for Procter & Gamble

- Transitioned within P&G to Information Systems

Developing software professionally since mid '90s

Focus has been primarily on SharePoint since 2004

Became a Microsoft MVP in 2016 (Office Apps & Services)



Nowadays, I work for Akumina, Inc.

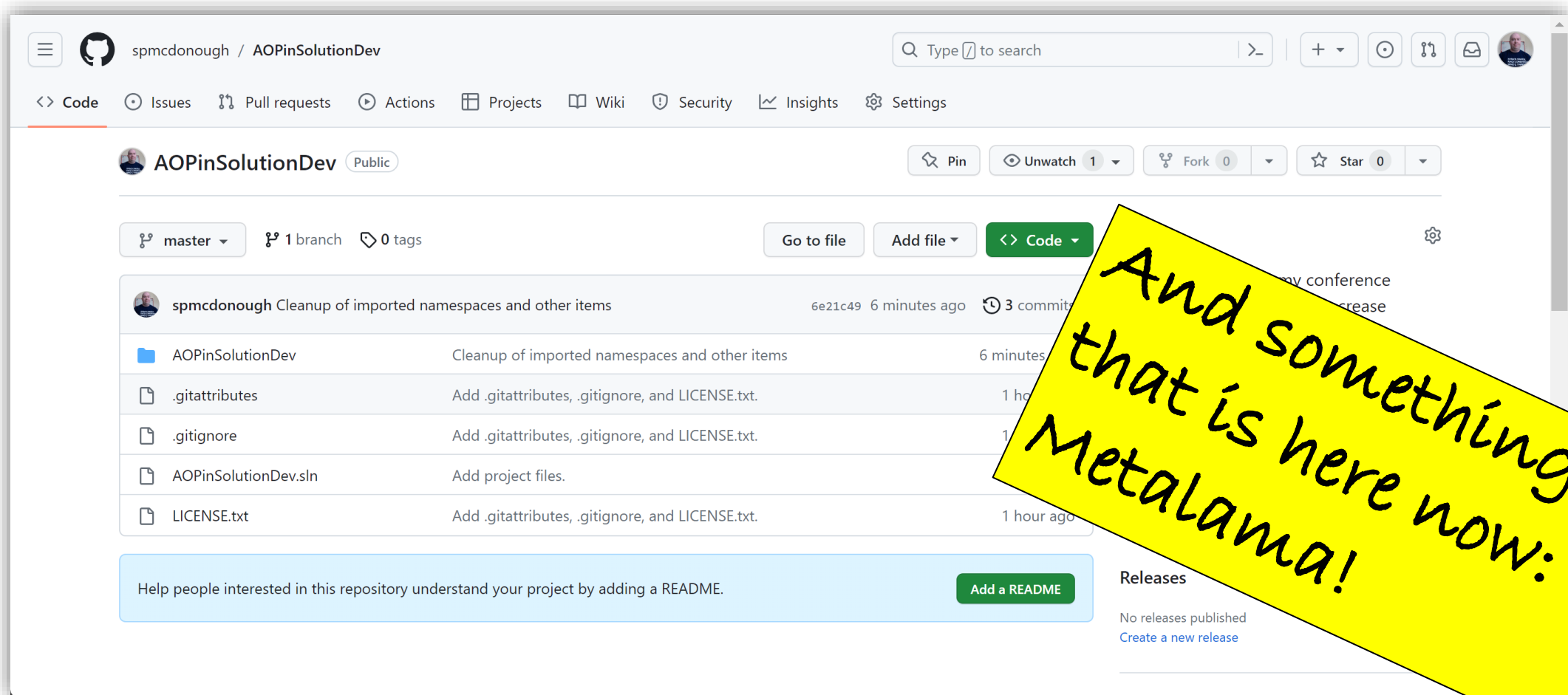
- Senior Solution Architect and Consultant

Still have Bitstream Foundry going

- Good way to organize my professional activities
- Educational and non-profit technical services



WANT THE CODE?



GitHub: <https://github.com/spmcdonough/AOPinSolutionDev>

PROBLEMS SOLVED WITH AOP

What sort of “problems are we actually talking about?”

Let's illustrate with an

example



THE EXAMPLE SCENARIO



You're an architect/developer for a large organization, and you've been tasked with building a new enterprise-class, full-trust SharePoint solution, provider-hosted add-in, an Azure Logic App, or some other .NET application.

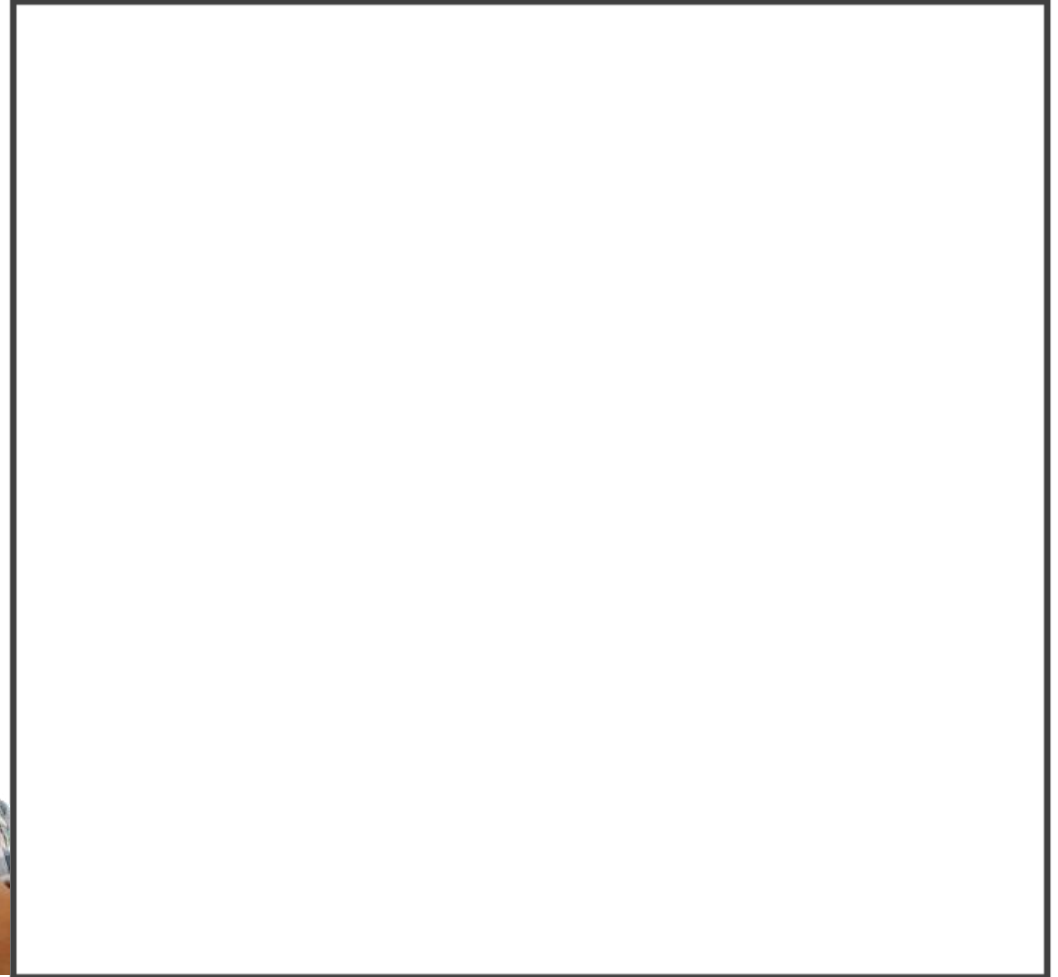
There are a wide-array of functional and non-functional requirements you need to address with your design.

THE EXAMPLE SCENARIO

This box represents your application



- You need to fill it with the functionality your users need
- And the best part ...

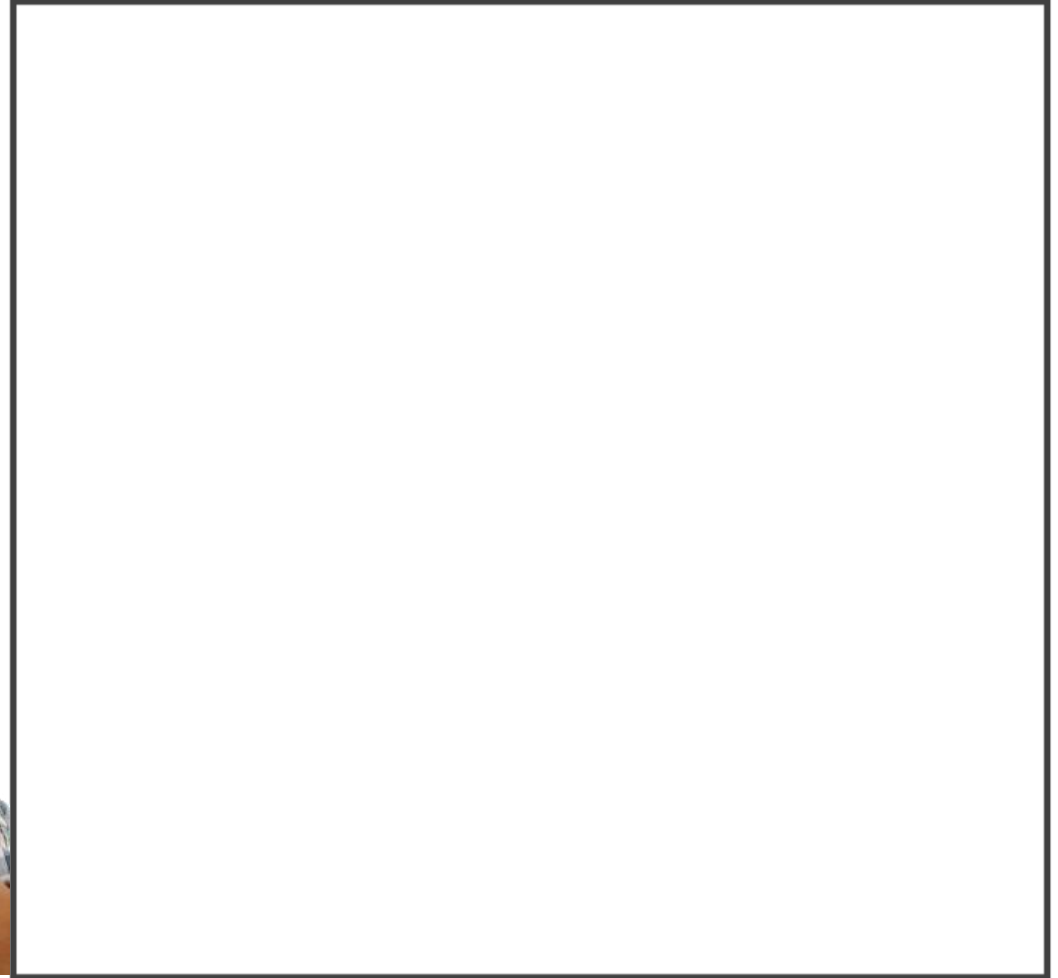


THE EXAMPLE SCENARIO

This box represents your application



- You need to fill it with the functionality your users need
- And the best part ...
- Greenfield development!



THE EXAMPLE SCENARIO

First: functional requirements

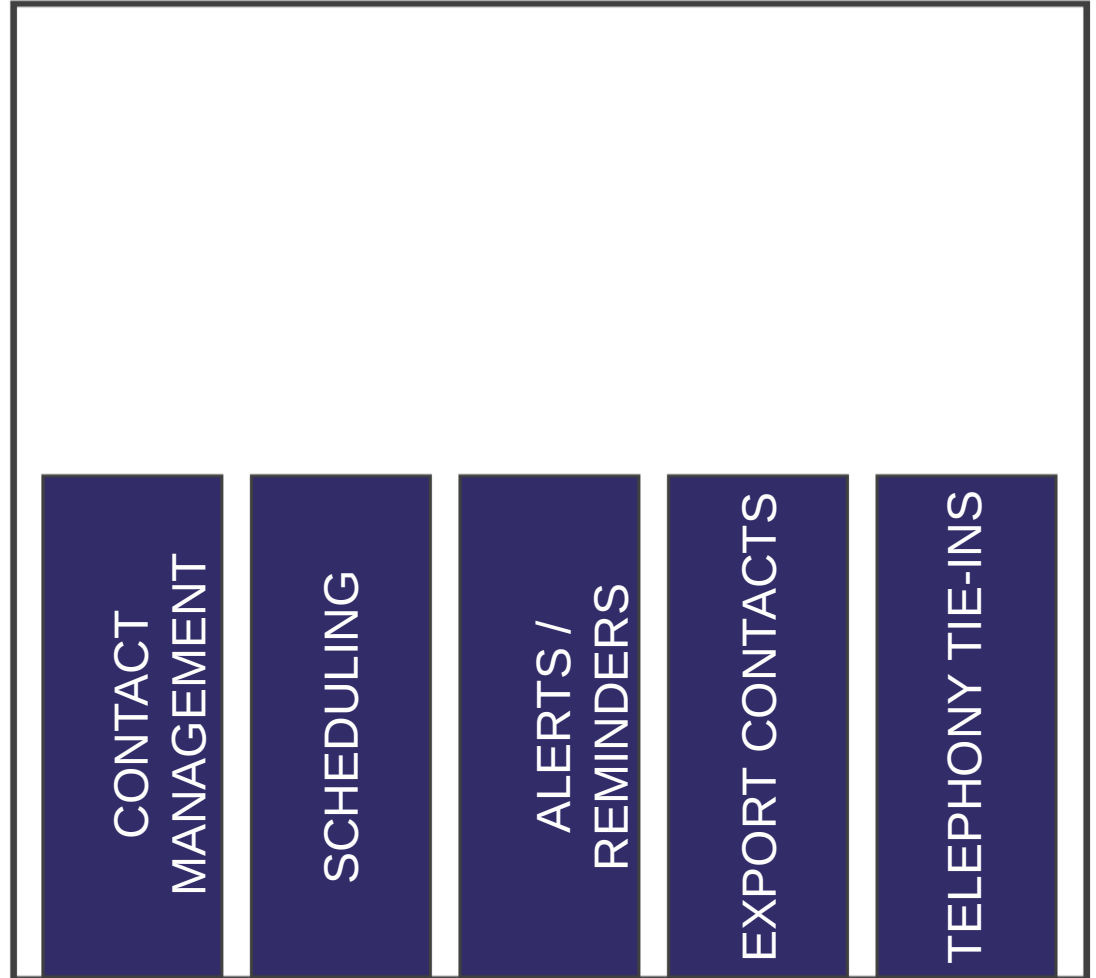
- The “business end” of what the application does.
- Ideally driven by stated user requirements and a functional specification



THE EXAMPLE SCENARIO

First: functional requirements

- The “business end” of what the application does.
- Ideally driven by stated user requirements and a functional specification

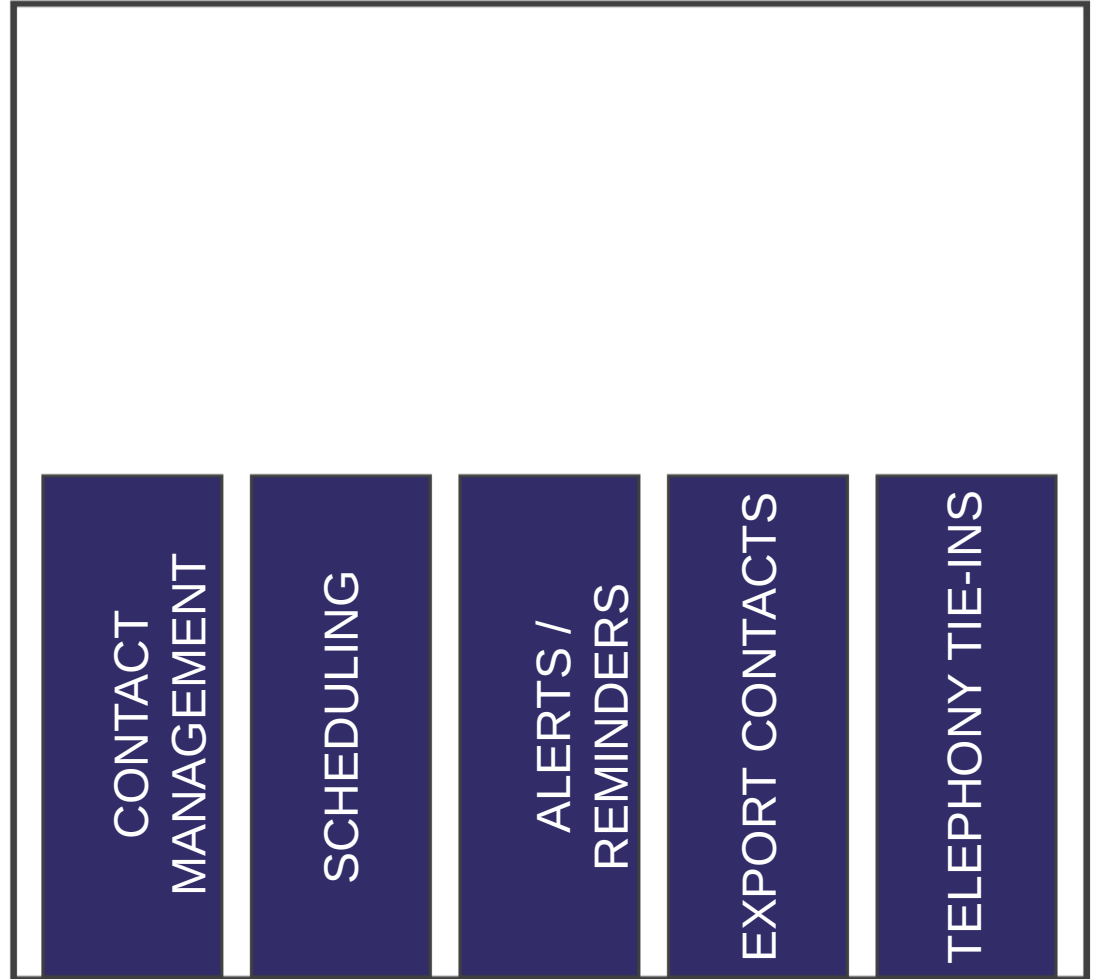


EXAMPLE 1



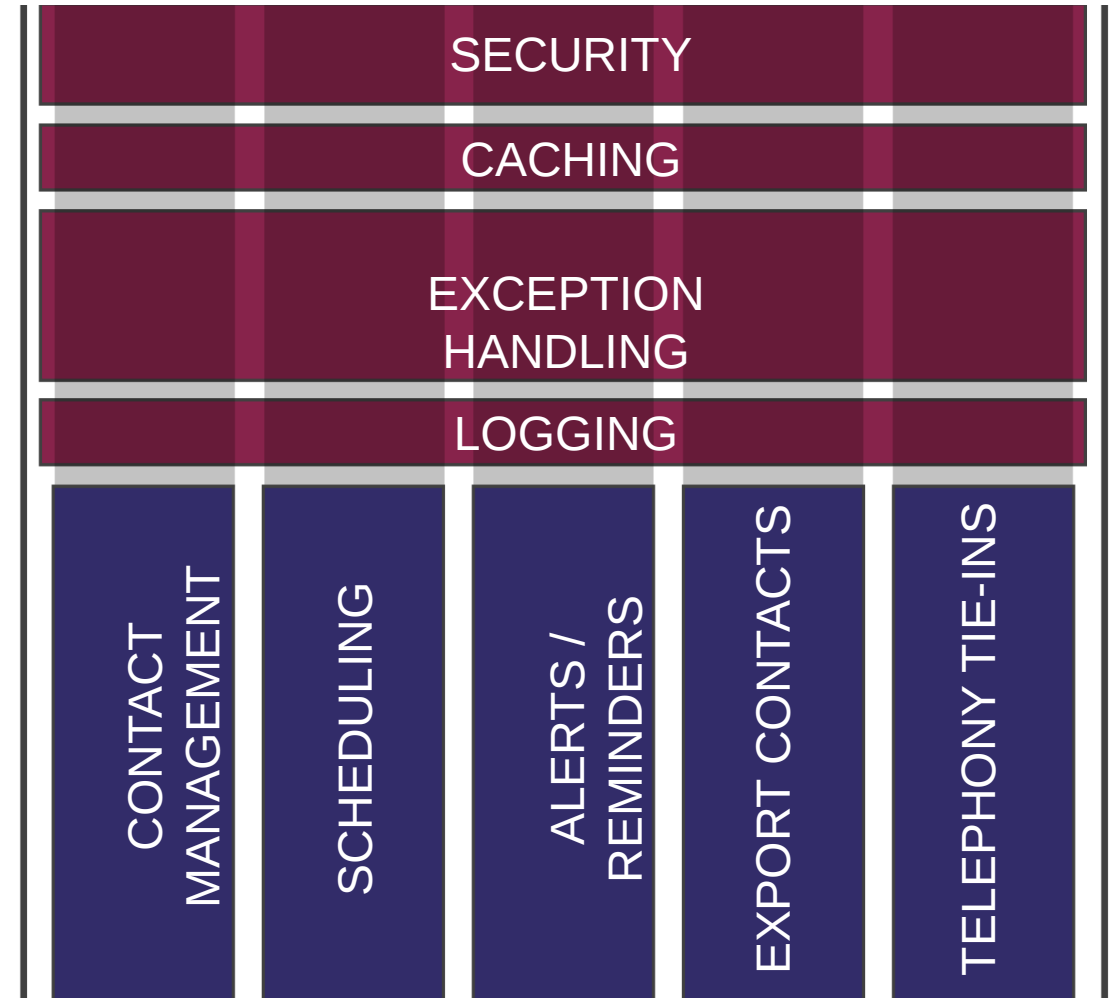
THE EXAMPLE SCENARIO

- If we could stop here, we'd be happy and could finish the project feeling like rock stars.



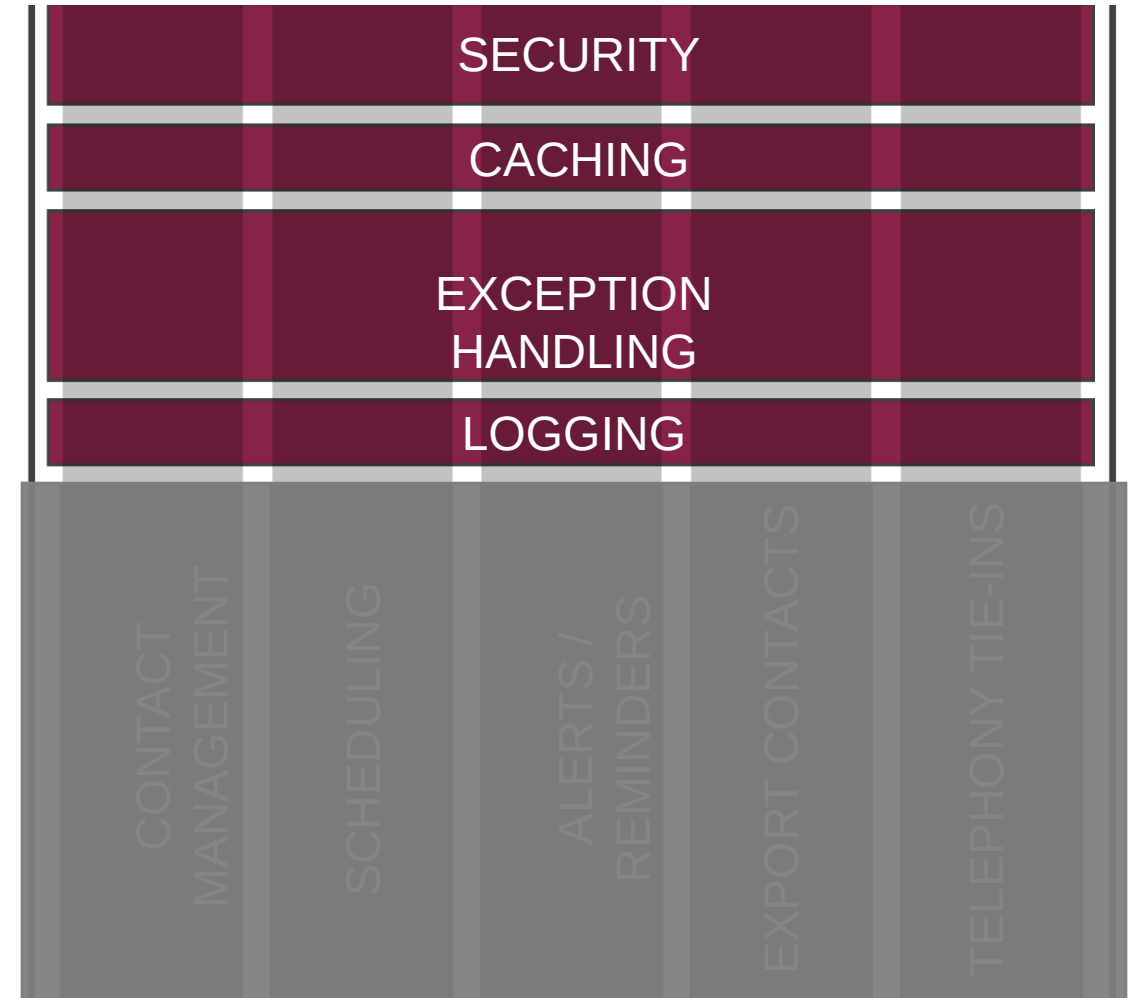
THE EXAMPLE SCENARIO

- In reality, we have a lot more code to write.
- It's not really fun code.
- It's generally not glorious.
- It's typically not at all interesting or glamorous.
- But it's got to be done for enterprise applications.



THE EXAMPLE SCENARIO

- Cross-cutting concerns like
 - Security
 - Exception handling
 - Logging
 - Caching
 - Performance counters
 - Thread management
- Dealing with these concerns traditionally entails a lot of cutting and pasting
- It's highly repetitive in nature

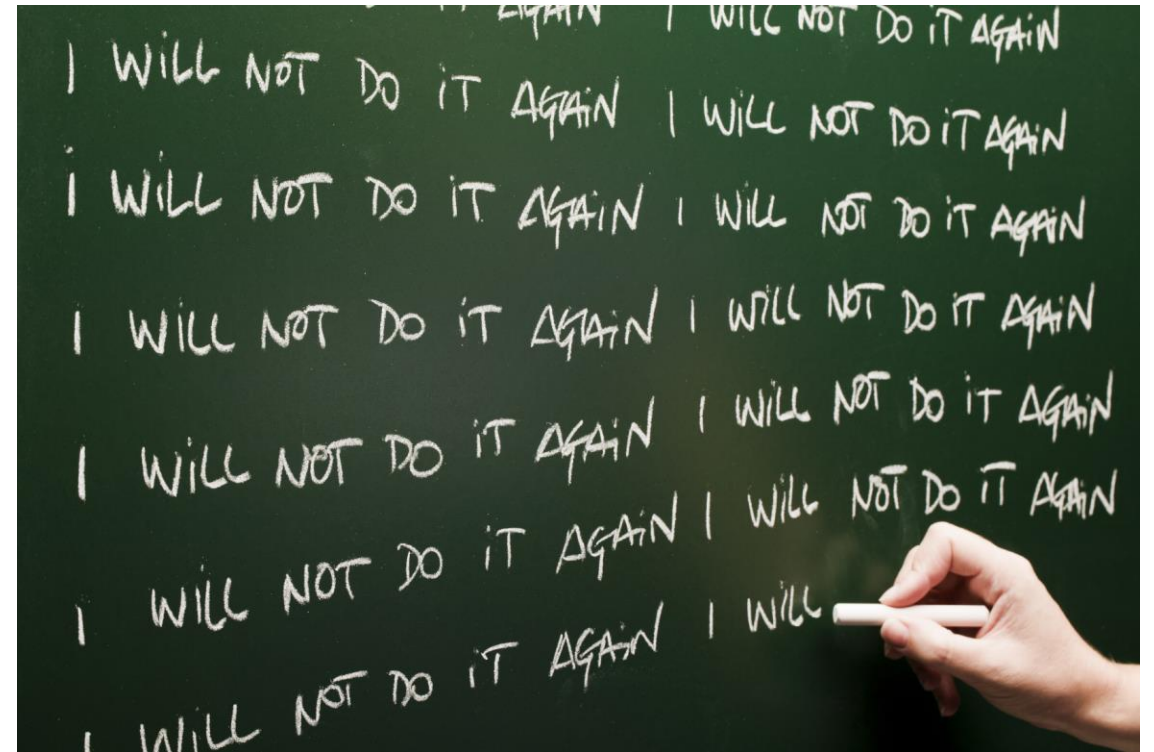


THE EXAMPLE SCENARIO

- This tends to result in some less-than-desirable effects
 - The repetitive nature of the code tends to lead to copy-and-paste “re-use.”
 - As humans, we tend to fatigue as we do all of this repetition and we mentally “check out.”
 - Repetition and fatigue inevitably lead to unexpected and unintentional deviations/errors.



EXAMPLE 2



THE EXAMPLE SCENARIO

Going from this:

```
private void OperationNo1()  
{  
    LoggingSupport.WriteToLog(message: "It is by caffeine alone I set my mind in motion,\r\n");  
}
```

THE EXAMPLE SCENARIO

Going from this:

```
private void OperationNo1()
{
    LoggingSupport.WriteToLog(message: "It is by caffeine alone I set my mind in motion,\r\n");
}
```

... to this:

```
private void OperationNo1()
{
    LoggingSupport.WriteToLog(message: "Entering OperationNo1()\r\n");
    LoggingSupport.WriteToLog(message: "It is by caffeine alone I set my mind in motion,\r\n");
    LoggingSupport.WriteToLog(message: "Exiting OperationNo1()\r\n");
}
```

THE EXAMPLE SCENARIO

Going from this:

```
private void OperationNo1()  
{  
    LoggingSupport.WriteToLog(message: "It is by caffeine alone I set my  
}
```

... to this:

```
private void OperationNo1()  
{  
    LoggingSupport.WriteToLog(message: "Entering OperationNo1()\r\n");  
    LoggingSupport.WriteToLog(message: "It is by caffeine alone I set my mind in motion,\r\n");  
    LoggingSupport.WriteToLog(message: "Exiting OperationNo1()\r\n");  
}
```

... is a substantial code change (i.e., we tripled our lines of code)



not cool

AND IT ONLY GETS WORSE WITH EACH CONCERN

Adding in some exception handling ...

even less cool

```
private void OperationNo1()
{
    LoggingSupport.WriteToLog(message: "Entering OperationNo1()\r\n");
    try
    {
        LoggingSupport.WriteToLog(message: "It is by caffeine alone I set my mind in motion,\r\n");
    }
    catch (Exception ex)
    {
        var exMessage = "Exception encountered in OperationNo1";
        LoggingSupport.WriteToLog(exMessage);
        throw new ApplicationException(exMessage, ex.InnerException);
    }
    finally
    {
        LoggingSupport.WriteToLog(message: "Exiting OperationNo1()\r\n");
    }
}
```



AND IT ONLY GETS WORSE WITH EACH CONCERN

And adding in some caching logic, too ...

```
private void OperationNo1()
{
    LoggingSupport.WriteToLog(message: "Entering OperationNo1()\r\n");
    Object cachedObject = null;
    String operationOutput = null;
    try
    {
        String operation01KeyName = "OPERATION01_RESULT";
        cachedObject = CacheSupport.Get(operation01KeyName);
        if (null == cachedObject) {
            operationOutput = "It is by caffeine alone I set my mind in motion,\r\n";
            CacheSupport.Add(operation01KeyName, operationOutput);
        }
        else
        {
            operationOutput = cachedObject.ToString();
        }
        LoggingSupport.WriteToLog(operationOutput);
    }
    catch (Exception ex)
    {
        var exMessage = "Exception encountered in OperationNo1";
        LoggingSupport.WriteToLog(exMessage);
        throw new ApplicationException(exMessage, ex.InnerException);
    }
    finally
    {
        LoggingSupport.WriteToLog(message: "Exiting OperationNo1()\r\n");
    }
}
```

surely you're kidding



AND IT ONLY GETS WORSE WITH EACH CONCERN

We started with this:

```
private void OperationNo1()
{
    LoggingSupport.WriteLine(message: "It is by caffeine alone I set my mind in motion,\r\n");
}
```



wanna cry yet?

And ended-up with this:

```
private void OperationNo1()
{
    LoggingSupport.WriteLine(message: "Entering OperationNo1()\r\n");
    Object cachedObject = null;
    String operationOutput = null;
    try
    {
        String operation01KeyName = "OPERATION01_RESULT";
        cachedObject = CacheSupport.Get(operation01KeyName);
        if (null == cachedObject) {
            operationOutput = "It is by caffeine alone I set my mind in motion,\r\n";
            CacheSupport.Add(operation01KeyName, operationOutput);
        }
        else
        {
            operationOutput = cachedObject.ToString();
        }
        LoggingSupport.WriteLine(operationOutput);
    }
    catch (Exception ex)
    {
        var exMessage = "Exception encountered in OperationNo1";
        LoggingSupport.WriteLine(exMessage);
        throw new ApplicationException(exMessage, ex.InnerException);
    }
    finally
    {
        LoggingSupport.WriteLine(message: "Exiting OperationNo1()\r\n");
    }
}
```

AND THE WORST PART?

- That was only a single method in one class.
- Typically, we're talking about countless methods and properties and dozens of classes that need this treatment.
- It readily gets out of hand in very short order.



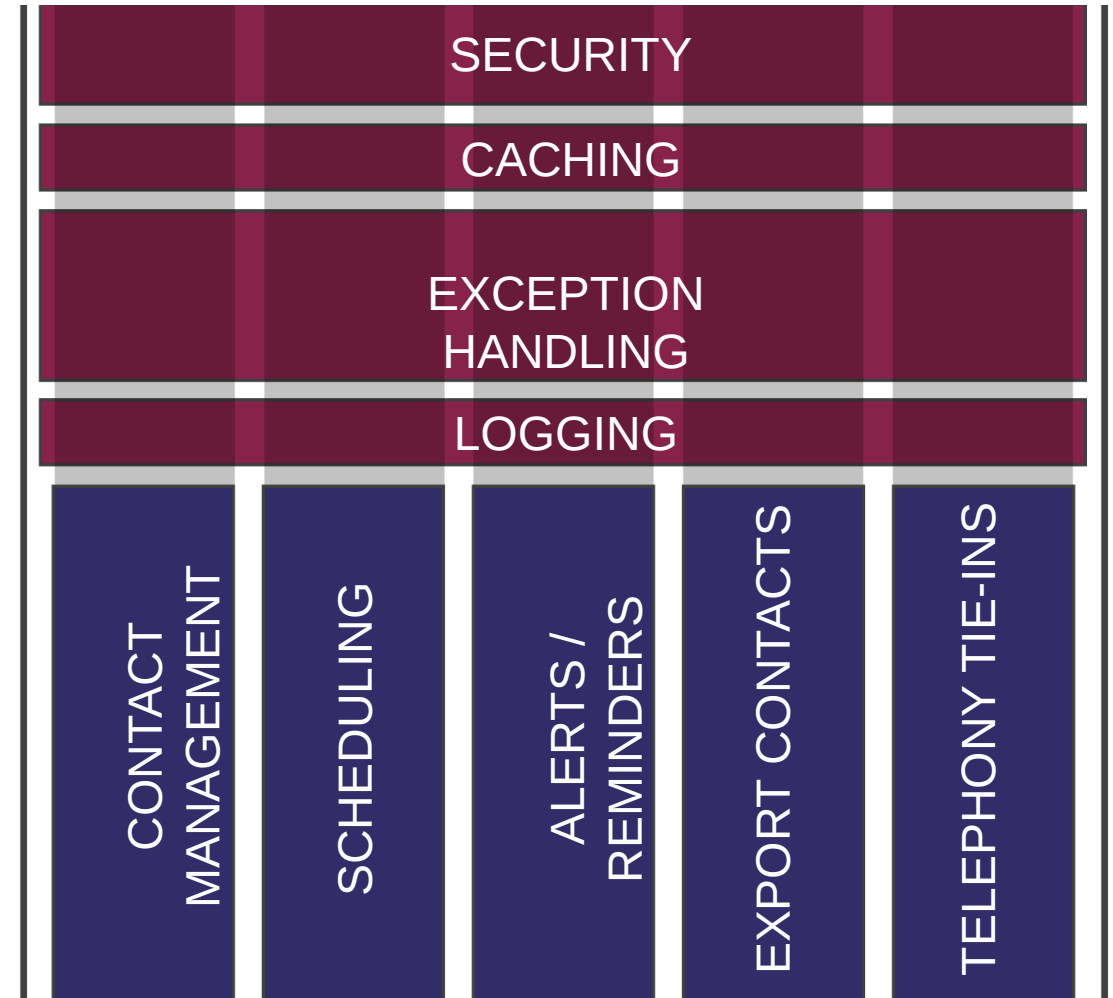
THERE HAS TO BE A BETTER WAY

akumina[®]



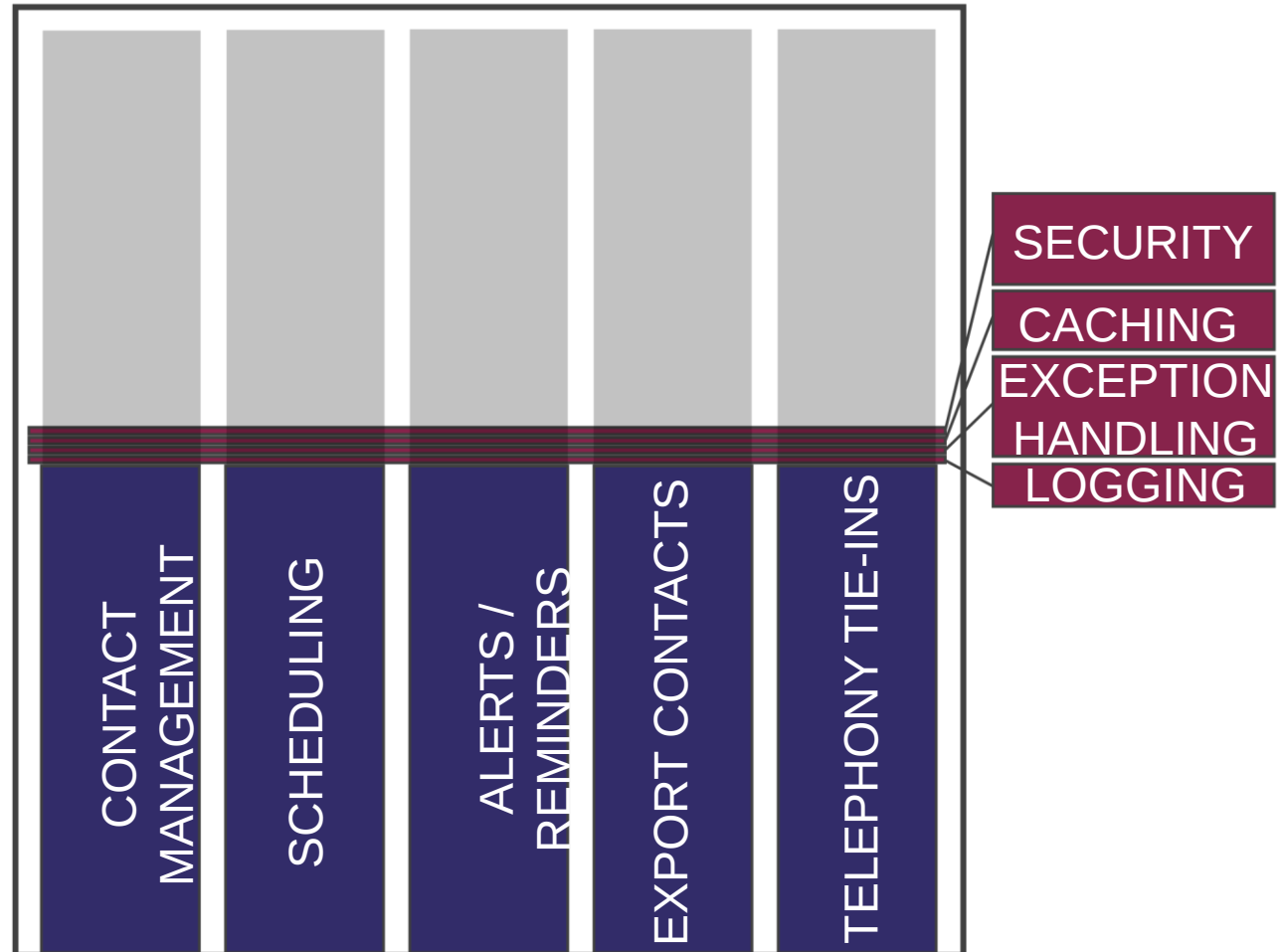
THERE HAS TO BE A BETTER WAY

- Instead of having to repeatedly code all of those cross-cutting concerns ...



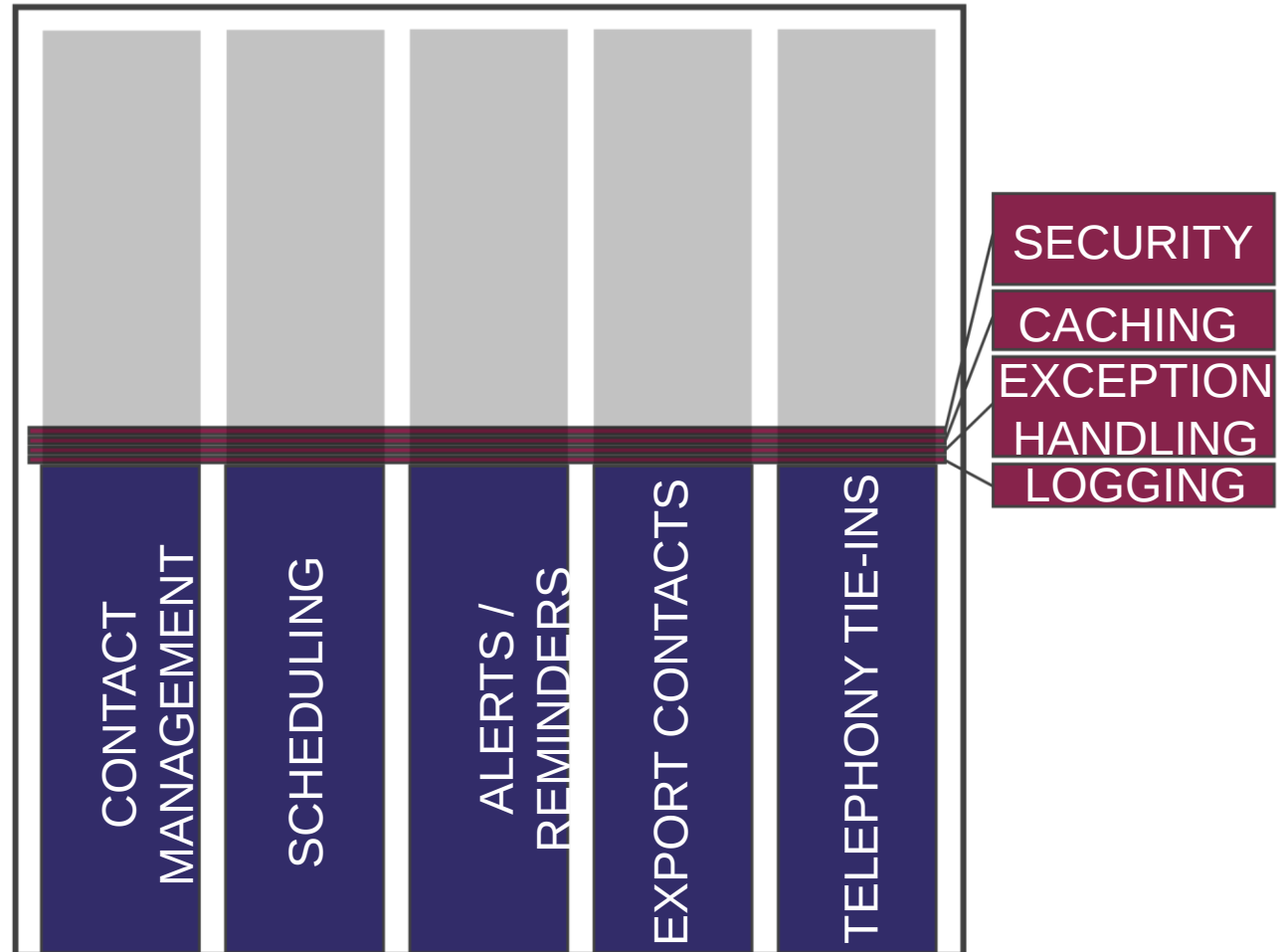
THERE HAS TO BE A BETTER WAY

- ... there should be a way to code them once and re-apply them in pattern or template form.
- You may have tried to tackle this need with specific classes, but that usually drives up complexity and line counts in its own way.
- Guess what?



THAT'S WHAT AOP IS ALL ABOUT

- Cross-cutting concerns are encapsulated in special classes called **aspects**.
- Business logic remains clear of redundant plumbing code.
- This reduces clutter and overall line counts.
- It simplifies maintenance significantly.



EXAMPLE 3



FREQUENTLY ASKED QUESTION

FREQUENTLY
?ASKED
QUESTION

“AOP seems pretty neat, but if it is so useful, how come I haven’t seen it ‘in the wild’ by (or before) now?”



FREQUENTLY ASKED QUESTION

- Aspects come in many forms
- If you weren't explicitly looking, you may have seen them and didn't recognize them.



FREQUENTLY ASKED QUESTION



- Aspects come in many forms
- If you weren't explicitly looking, you may have seen them and didn't recognize them.
- Have you heard of or used HTTP Modules during development?

ASP.NET HTTP modules and HTTP handlers

04/03/2020 • 4 minutes to read •

This article introduces the ASP.NET Hypertext Transfer Protocol (HTTP) modules and HTTP handlers.

Original product version: ASP.NET

Original KB number: 307985

Summary

HTTP modules and HTTP handlers are an integral part of the ASP.NET architecture. While a request is being processed, each request is processed by multiple HTTP modules (for example, the authentication module and the session module) and is then processed by a single HTTP handler. After the handler has processed the request, the request flows back through the HTTP modules.

HTTP modules overview

Modules are called before and after the handler executes. Modules enable developers to intercept, participate in, or modify each individual request. Modules implement the `IHttpModule` interface, which is located in the `System.Web` namespace.

Available events that HTTP modules can synchronize with

An `HttpApplication` class provides a number of events with which modules can synchronize. The following events are available for modules to synchronize with on each request. These events are listed in sequential order:

- **BeginRequest**: Request has been started. If you need to do something at the beginning of a request (for example, display advertisement banners at the top of each page), synchronize this event.
- **AuthenticateRequest**: If you want to plug in your own custom authentication scheme (for example, look up a user against a database to validate the password), build a module that synchronizes this event and authenticates the user how you want to.
- **AuthorizeRequest**: This event is used internally to implement authorization mechanisms (for example, to store your access control lists (ACLs) in a database rather than in the file system). Although you can override this event, there are not many good reasons to do so.
- **ResolveRequestCache**: This event determines if a page can be served from the Output cache. If you want to write your own caching module (for example, build a file-based cache rather than a memory cache), synchronize this event to determine whether to serve the page from the cache.
- **AcquireRequestState**: Session state is retrieved from the state store. If you want to build your own state management module, synchronize this event to grab the session state from your state store.

FREQUENTLY ASKED QUESTION



- Aspects come in many forms
- If you weren't explicitly looking, you may have seen them and didn't recognize them.
- Have you heard of or used HTTP Modules during development?
- How about [ASP.NET MVC Custom Action Filters](#)?
- Both of the above adhere to AOP concepts and patterns.

ASP.NET MVC 4 Custom Action Filters

02/18/2013 • 20 minutes to read • +4

By [Web Camps Team](#)

[Download Web Camps Training Kit](#)

ASP.NET MVC provides Action Filters for executing filtering logic either before or after an action method is called. Action Filters are custom attributes that provide declarative means to add pre-action and post-action behavior to the controller's action methods.

In this Hands-on Lab you will create a custom action filter attribute into MvcMusicStore solution to catch controller's requests and log the activity of a site into a database table. You will be able to add your logging filter by injection to any controller or action. Finally, you will see the log view that shows the list of visitors.

This Hands-on Lab assumes you have basic knowledge of ASP.NET MVC. If you have not used ASP.NET MVC before, we recommend you to go over [ASP.NET MVC 4 Fundamentals Hands-on Lab](#).

Note

All sample code and snippets are included in the Web Camps Training Kit, available from at [Microsoft-Web/WebCampTrainingKit Releases](#). The project specific to this lab is available at [ASP.NET MVC 4 Custom Action Filters](#).

Objectives

In this Hands-On Lab, you will learn how to:

- Create a custom action filter attribute to extend filtering capabilities
- Apply a custom filter attribute by injection to a specific level
- Register a custom action filters globally

Prerequisites

You must have the following items to complete this lab:

- [Microsoft Visual Studio Express 2012 for Web](#) or superior (read [Appendix A](#) for instructions on how to install it).

Setup

Installing Code Snippets

AOP TOOLS AND PRODUCTS

- My AOP tool of choice: [PostSharp](#)*
- Built my SharpCrafters / PostSharp Technologies (Gael Fraiteur, CEO)
- Comes in both commercial (i.e. paid) and community (i.e. free) editions
- Everything I have demonstrated and will show will use capabilities that are part of the FREE (community) version of PostSharp.
- We'll talk about Metalama shortly ...



IMPLEMENTING AND USING AN ASPECT



- Must follow these basic rules (specific to PostSharp)
 - Must be adorned with **[PSerializable]** attribute
 - PostSharp will yell at you if you don't do this
- Must inherit from / implement a PostSharp base class
 - Our **LoggingAspect** implemented **OnMethodBoundaryAspect**
 - Others are available depending on your version of PostSharp
- Your “business logic” must be decorated with an aspect-specific attribute determined by the name of the aspect class you want to use
 - Our **Example 03** class was adorned with the **[LoggingAspect]** attribute
 - For more granular application, method adornment can be used with attributes
 - To implement an aspect across a project, apply attribute at the assembly level

WHAT ABOUT OTHER WAYS TO DO AOP?

- You don't need PostSharp to do AOP with .NET.
- Check out the [Castle Project and its Dynamic Proxy](#) for an alternative ([inversion of control](#)) approach.
- I favor PostSharp for several reasons:
 - Provides cleanest separation of code
 - Professionally maintained
 - Low cost of entry / “it just works”
 - Employs compile-time weaving
 - Castle Project is no longer maintained



Build your .NET projects on a rock solid foundation

DynamicProxy

Castle DynamicProxy is a library for generating lightweight .NET proxies on the fly at runtime. Proxy objects allow calls to members of an object to be intercepted without modifying the code of the class. Both classes and interfaces can be proxied, however only virtual members can be intercepted.

DynamicProxy differs from the proxy implementation built into the CLR which requires the proxied class to extend `MarshalByRefObject`. Extending `MarshalByRefObject` to proxy an object can be too intrusive because it does not allow the class to extend another class and it does not allow transparent proxying of classes.

You can use DynamicProxy to generate lightweight proxies on the fly for one or more interfaces or even concrete classes (but only virtual methods will be intercepted).

Why use proxies?

Proxy objects can assist in building a flexible application architecture because it allows functionality to be transparently added to code without modifying it. For example, a class could be proxied to add logging or security checking without making the code aware this functionality has been added.

For example, NHibernate, an object/relational mapper uses DynamicProxy to provide lazy loading of data without the domain model classes being aware of this functionality.

For more, [check out the documentation](#).

COMPILE TIME *WHAT*?

- Compile-time weaving.
- No, we're not talking about basket weaving and associated container "technologies."
- This is probably a good time to introduce some domain-specific terminology that you'll likely hear if you spend any time with AOP.



AOP DOMAIN CONCEPTS AND TERMS

- Let's start with a snippet of the LoggingAspect we've seen

```
1 #region Namespace Imports
2
3
4 using System;
5 using AOPinSolutionDev.Plumbing;
6 using PostSharp.Aspects;
7
8
9 #endregion Namespace Imports
10
11
12 namespace AOPinSolutionDev.Aspects
13 {
14
15
16     /// <summary>
17     /// This method boundary aspect (created with PostSharp) is responsible
18     /// for handling logging activities for each of the methods with which
19     /// it is associated.
20     /// </summary>
21     [Serializable]
22     1 reference
23     internal class LoggingToTextboxAspect : OnMethodBoundaryAspect
24     {
25
26         #region Overrides: OnMethodBoundaryAspect
27
28
29         /// <summary>
30         /// The OnEntry method fires on the join point that occurs just before
31         /// a method is entered and its first lines of code are executed.
32         /// </summary>
33         0 references
34         public override void OnEntry(MethodExecutionArgs args)
35         {
36             CreateLogEntry(args, action: "Entering Method");
37         }
38
39         /// <summary>
40         /// The OnExit method fires on the join point that occurs just after
```

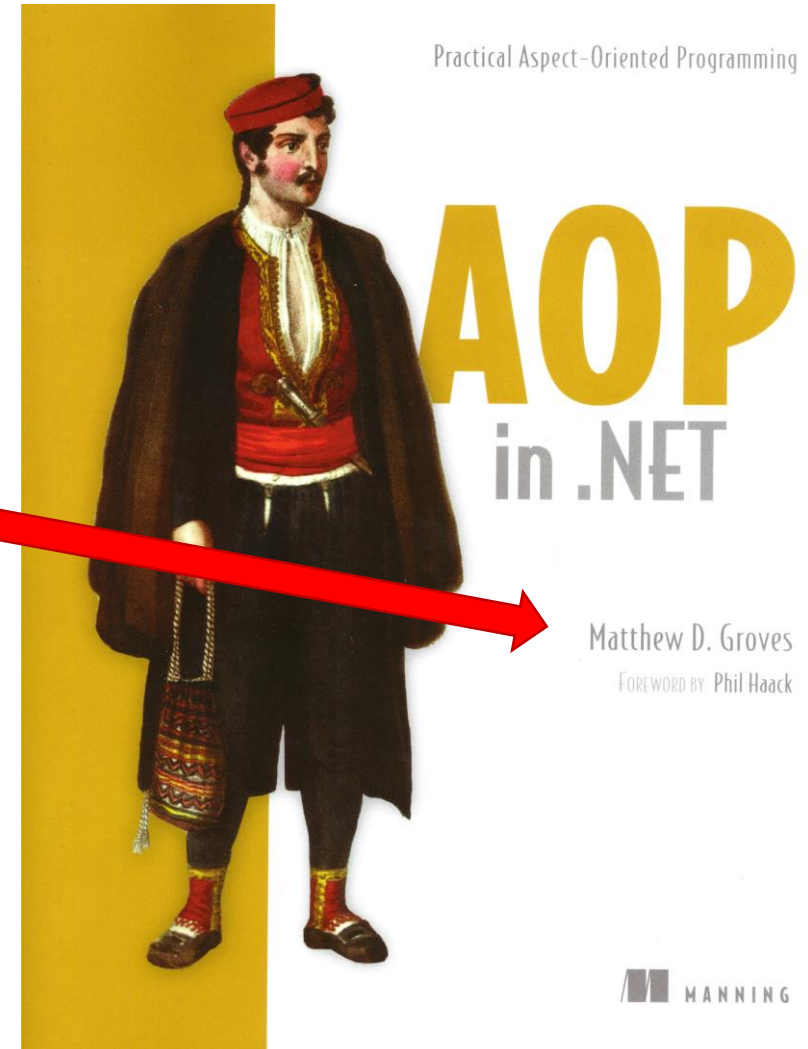
AOP DOMAIN CONCEPTS AND TERMS

- Let's start with a snippet of the LoggingAspect we've seen
- The aspect class and code itself are known as *advice*

```
1  #region Namespace Imports
2
3
4  using System;
5  using AOPinSolutionDev.Plumbing;
6  using PostSharp.Aspects;
7
8
9  #endregion Namespace Imports
10
11
12 namespace AOPinSolutionDev.Aspects
13 {
14
15
16     /// <summary>
17     /// This method boundary aspect (created with PostSharp) is responsible
18     /// for handling logging activities for each of the methods with which
19     /// it is associated.
20     /// </summary>
21     [Serializable]
22     1 reference
23     internal class LoggingToTextboxAspect : OnMethodBoundaryAspect
24     {
25
26         #region Overrides: OnMethodBoundaryAspect
27
28
29         /// <summary>
30         /// The OnEntry method fires on the join point that occurs just before
31         /// a method is entered and its first lines of code are executed.
32         /// </summary>
33         0 references
34         public override void OnEntry(MethodExecutionArgs args)
35         {
36             CreateLogEntry(args, action: "Entering Method");
37         }
38
39         /// <summary>
40         /// The OnExit method fires on the join point that occurs just after
```

AOP DOMAIN CONCEPTS AND TERMS

- Let's start with a snippet of the LoggingAspect we've seen
- The aspect class and code itself are known as *advice*
- Matt Groves (who wrote "AOP in .NET") defines *join points* as places that can be defined between logical steps of the execution of your program"



AOP DOMAIN CONCEPTS AND TERMS

- Let's start with a snippet of the LoggingAspect we've seen
- The aspect class and code itself goes by the term *advice*
- Matt Groves (who wrote "AOP in .NET") defines *join points* as places that can be defined between logical steps of the execution of your program"
- The areas with red arrows would be *join points* for our aspect.

```
/// <summary>
/// The OnEntry method fires on the join point that occurs just before
/// a method is entered and its first lines of code are executed.
/// </summary>
0 references
public override void OnEntry(MethodExecutionArgs args)
{
    CreateLogEntry(args, action: "Entering Method");
}

/// <summary>
/// The OnExit method fires on the join point that occurs just after
/// a method is exited and its execution is complete.
/// </summary>
0 references
public override void OnExit(MethodExecutionArgs args)
{
    CreateLogEntry(args, action: "Exiting Method");
}

#endregion Overrides: OnMethodBoundaryAspect
```

AOP DOMAIN CONCEPTS AND TERMS

- A set of join points is known as a *pointcut*. Pointcuts are points where execution transitions into and out of your advice (aspect).

```
/// <summary>
/// The OnEntry method fires on the join point that occurs just before
/// a method is entered and its first lines of code are executed.
/// </summary>
```

0 references

```
public override void OnEntry(MethodExecutionArgs args)
{
    CreateLogEntry(args, action: "Entering Method");
}
```

```
/// <summary>
```

```
/// The OnExit method fires on the join point that occurs just after
/// a method is exited and its execution is complete.
```

```
/// </summary>
```

0 references

```
public override void OnExit(MethodExecutionArgs args)
{
    CreateLogEntry(args, action: "Exiting Method");
}
```

```
#endregion Overrides: OnMethodBoundaryAspect
```

AOP DOMAIN CONCEPTS AND TERMS

- A set of join points is known as a *pointcut*. Pointcuts are points where execution transitions into and out of your advice (aspect).
- Pointcuts are integrated with your business logic through a process called *weaving*.



AOP DOMAIN CONCEPTS AND TERMS

- A set of join points is known as a *pointcut*. Pointcuts are points where execution transitions into and out of your advice (aspect).
- Pointcuts are integrated with your business logic through a process called *weaving*.
- Fact: I can't talk about this section without imagining baskets and crochet. Sad, but true.



AOP DOMAIN CONCEPTS AND TERMS

- Weaving currently takes place by two different processes depending on the technology you're using.
- *Run-time weaving* happens with IoC containers and systems like Castle Dynamic Proxy.
- No special tools required to use
- Relies on run-time reflection
- For (GoF) software pattern fans, think “decorator” and “proxy” patterns as associated with AOP.



AOP DOMAIN CONCEPTS AND TERMS

- On the other hand, *compile-time weaving* uses .NET intermediate language (IL) integration steps following compilation to more tightly and seamlessly integrate with your business logic.
- PostSharp uses compile-time weaving.
- Allows for some optimization.
- Requires tooling; historically harder to test ...



~~COMING SOON~~ HERE NOW!



- PostSharp Metalama was recently released and is available.
- It can replace the MSIL-based PostSharp we use today.
- It's a bottom-up rewrite of PostSharp concepts based on the [Roslyn framework](#)
- Metalama emits .NET code during compilation that is not opaque and is very debugger friendly, making it easier to test.

	Metalama	Roslyn	Fody	PostSharp
Technology	Roslyn	Roslyn	MSIL	MSIL
> Transforming the compilation using low-level APIs	Yes	No	Yes	Yes
> Adding behaviors to source code <i>simply</i> using aspects.	Yes	No	Very limited	Yes
> Introduce new members or interfaces and reference them in source code.	Yes	Difficult	No	No
> Analyze source code and report warnings and errors.	Yes	Difficult	Requires rebuild.	Requires rebuild.
> Debug and export transformed code.	Yes	N/A	No	No

NOW*
EXCLUSIVE

GO METALAMA?

Should I use PostSharp IL or Metalama?

- I'm still using PostSharp IL for a (little) while longer because it's the more mature product and I'm more familiar with it.
- Going forward, I'm working on learning Metalama. The code transparency benefits and support for automated testing are promising.
- The choice is yours, but Metalama is where SharpCrafters are focused.
- Currently, there are free versions of both PostSharp and Metalama.

```
using Metalama.Framework.Engine.AspectWeavers;
using Microsoft.CodeAnalysis;
using Microsoft.CodeAnalysis.CSharp;
using Microsoft.CodeAnalysis.CSharp.Syntax;
using System;
using System.Linq;
using System.Threading.Tasks;
using static Microsoft.CodeAnalysis.CSharp.SyntaxKind;

namespace Metalama.Community.Virtuosity
{
    [MetalamaPlugIn]
    public sealed class VirtuosityWeaver : IAspectWeaver
    {
        public Task TransformAsync( AspectWeaverContext context )
        {
            return context.RewriteAspectTargetsAsync( new Rewriter() );
        }

        private class Rewriter : CSharpSyntaxRewriter
        {
            private static readonly SyntaxKind[]? _forbiddenModifiers =
                new[] { StaticKeyword, SealedKeyword, VirtualKeyword, OverrideKeyword };

            private static readonly SyntaxKind[]? _requiredModifiers =
                new[] { PublicKeyword, ProtectedKeyword, InternalKeyword };

            private static bool CanTransformType( MemberDeclarationSyntax node )
            {
            }
        }
    }
}
```

Extensible with Roslyn: Overcome the limitations of Metalama and write transformations directly with Roslyn.

ASPECT TYPES

- We've seen one type – the OnMethodBoundaryAspect – in action so far.
- There are many other aspect types, but the one we've seen is joined by one other in the free version of PostSharp: the MethodInterceptionAspect
- You can implement nearly any type of aspect with these two types alone.



ONMETHODOBOUNDARYASPECT

- Primary Methods

- OnEntry

- OnException

- OnExit

- OnResume

- OnSuccess

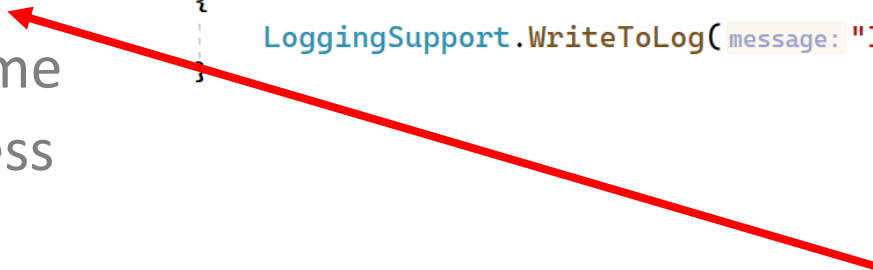
- OnYield

```
private void OperationNo1()  
{  
    LoggingSupport.WriteLine(message: "It is by caffeine alone I set my mind in motion,\r\n");  
}
```

OnEntry



OnExit



ONMETHODOBOUNDARYASPECT

- Parameter type passed:
 - MethodExecutionArgs
- Properties
 - Arguments
 - DeclarationIdentifier
 - Exception
 - FlowBehavior
 - Instance
 - Method
 - MethodExecutionTag
 - ReturnValue
 - YieldValue

```
private void OperationNo1()  
{  
    LoggingSupport.WriteToLog(message: "It is by caffeine alone I set my mind in motion,\r\n");  
}
```

OnEntry



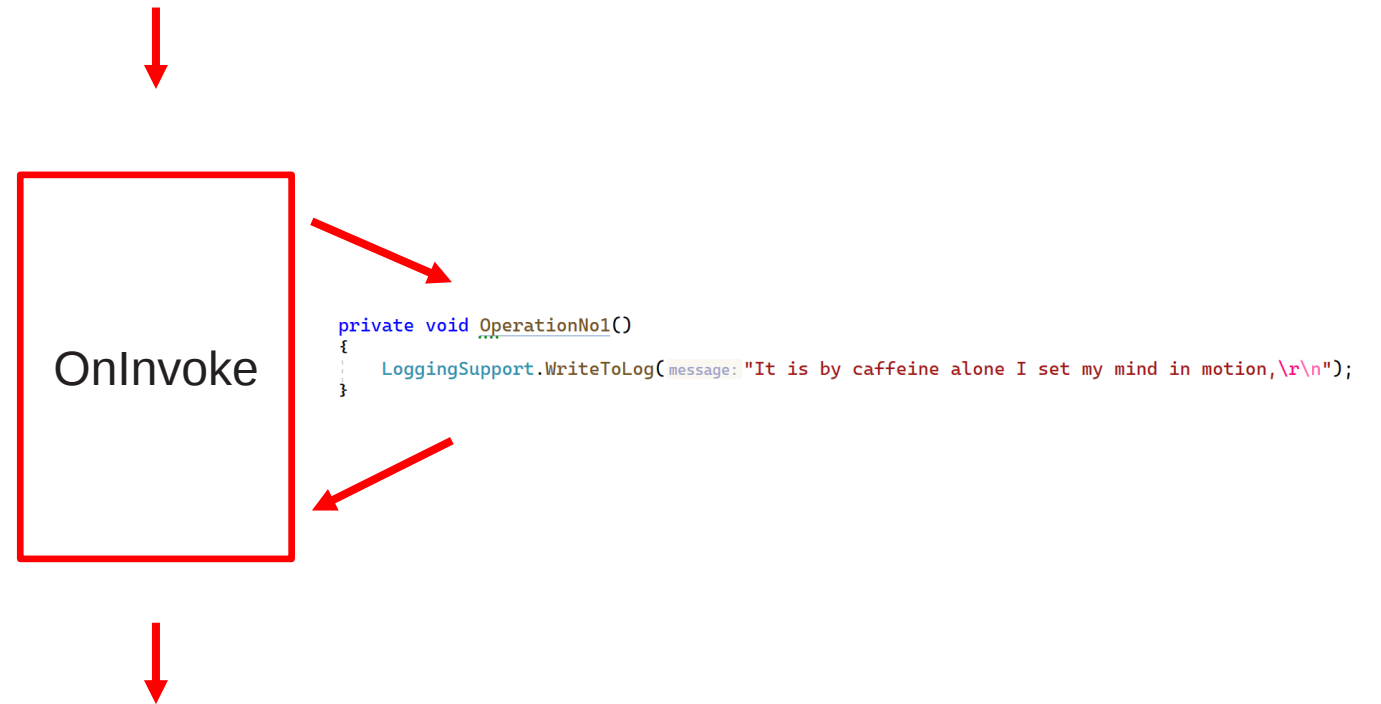
OnExit

ONMETHODOBOUNDARYASPECT

- Notes
 - Statically scoped aspect
 - One instance of aspect used to service all requests going to the aspect.
 - Storing data across method calls isn't inherently (thread-)safe
 - If thread-safety is needed, use the MethodExecutionTag or implement the IInstanceScopedAspect interface in your aspect.
 - IL Optimization
 - Since arguments are boxed/unboxed when sent to the aspect, PostSharp avoids processing unused parameters that aren't actually used.
- Common uses
 - Repetitive tasks
 - Logging, tracing, performance profiling, exception handling

METHODINTERCEPTIONASPECT

- Primary Method
 - OnInvoke



METHODINTERCEPTIONASPECT

- Parameter type passed:
 - MethodInterceptionArgs

- Properties
 - Arguments
 - AsyncBinding
 - Binding
 - DeclarationIdentifier
 - Instance
 - IsAsync
 - Method
 - ReturnValue

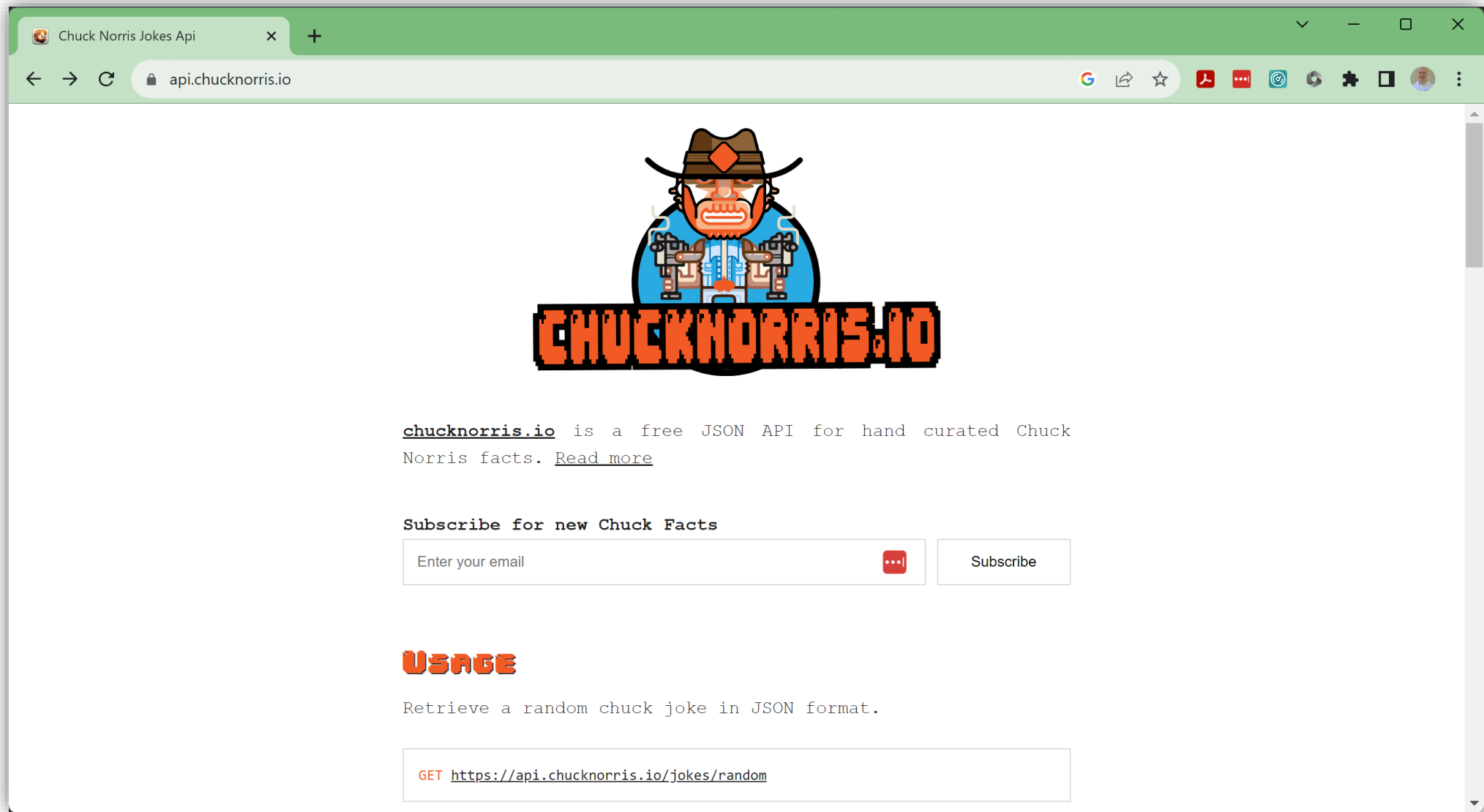


```
private void OperationNo1()
{
    LoggingSupport.WriteToLog(message: "It is by caffeine alone I set my mind in motion,\r\n");
}
```

METHODINTERCEPTIONASPECT

- Notes
 - Running your method code is actually optional
 - You have slightly reduced clarity with everything happening in one method (small downside)
 - IL optimization cannot occur
 - Shared state (such as thread-safe variable values) benefits since atomic call happens with **OnInvoke**)
- Common uses
 - Wrapper nature good for certain code needs
 - Caching, retry support, multi-threading assistance, lazy loading

OUR REMAINING DEMOS WILL USE ...



EXAMPLES 4+



AND ONE FINAL APPEAL

- I'm a huge proponent of Microsoft Philanthropies' TEALS program
 - TEALS stands for "technical education and literacy in schools," and I've been a volunteer for five years.
 - The goal of the TEALS program is to bring computer science education to traditionally underserved and marginalized populations.
- TEALS operates on volunteer basis
 - If you have any technical background, TEALS would love to see you volunteer!
 - I've co-instructed both "An Introduction to Game Design" (block-based programming and Python) and a CompTIA Security+ curriculum over the years.
 - It is exceptionally rewarding on a personal level. I get at least as much out of it as I put into it.
 - TEALS is always welcoming (new) volunteers.
- Visit <https://www.microsoft.com/en-us/teals>



THANK YOU FOR ATTENDING

Obviously not me
(*but the sentiment is*)



Adobe PhotoShop Beta AI Prompt Used to Generate Image: *A thankful balding and middle-aged presenter in polo shirt and shorts expressing his tremendous appreciation and thanks to the gathered audience*

Final
Questions
?

Thank you



In the top-left corner, there are two overlapping geometric shapes: a magenta triangle pointing towards the top-right and a yellow triangle pointing towards the bottom-left.

WRAP-UP AND CONTACT INFO

A thin teal line starts from the right edge of the slide, curves downwards and to the left, and then continues downwards towards the bottom right corner.



Sean P. McDonough

sean@SharePointInterface.com

sean.mcdonough@akumina.com

LinkedIn: <https://www.linkedin.com/in/smcdonough/>

Twitter: [@spmcdonough](https://twitter.com/spmcdonough)

Blog: <https://SharePointInterface.com/>

About: <https://spmcdonough.com/>