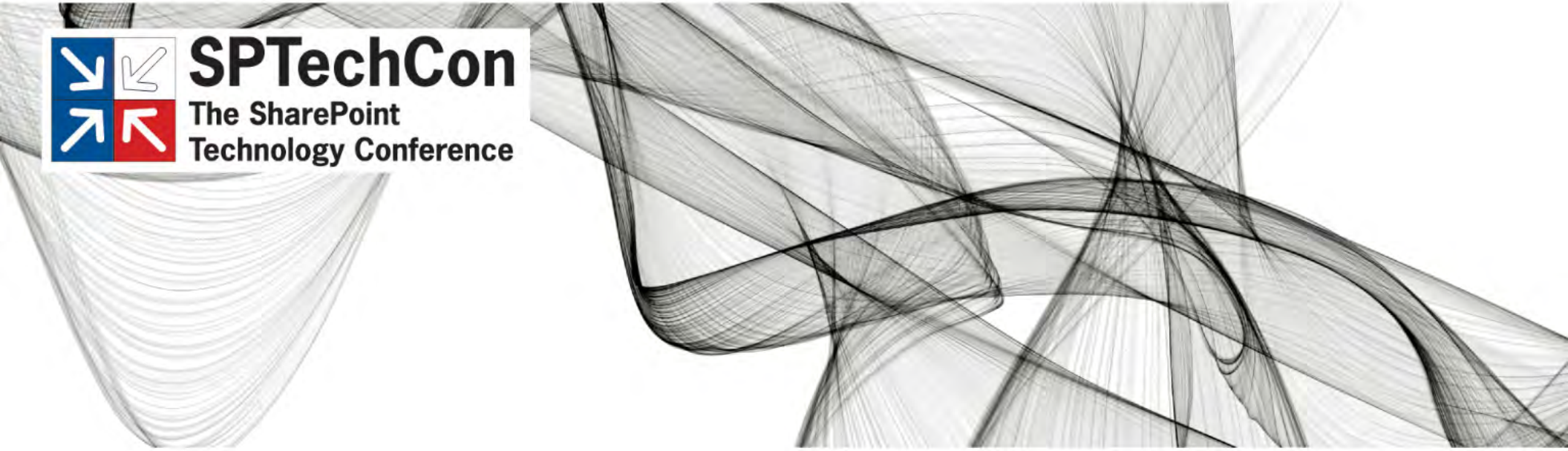
Leveraging AOP in SharePoint Custom Development



Sean P. McDonough
Lead Bitsmith and Owner
Bitstream Foundry LLC

bitstream FOUNDRY

My background

- Developing software since mid '90s

My company

- Started in early 2013

Sean P. McDonough
Lead Bitsmith and Owner
Bitstream Foundry LLC

bitstream FOUNDRY

My background

- Developing software since mid '90s
- Working with SharePoint since 2004
- Many roles: developer, administrator, product manager, evangelist, and more
- Community focus: free solutions, writing, speaking, and mentoring

My company

- Started in early 2013
- Focus on SharePoint
- Deliver apps for SharePoint and the SharePoint ecosystem
- Doing custom development and infrastructure work on a daily basis

Microsoft BizSpark Startup Microsoft

Our Agenda for this Session

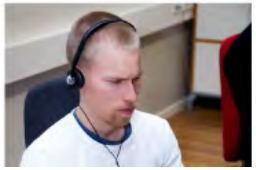
- Problems solved with AOP
- AOP terminology and concepts
- Tools that enable AOP in .NET
- Creating aspects
- Potential watch-outs with AOP
- Q&A throughout!

Our Age

What sort of "problems?"

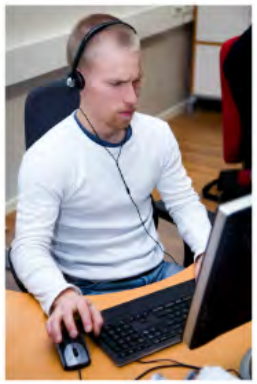
- Problems
- AOP termi
- Tools that
- Creating a

Let's illustrate with an



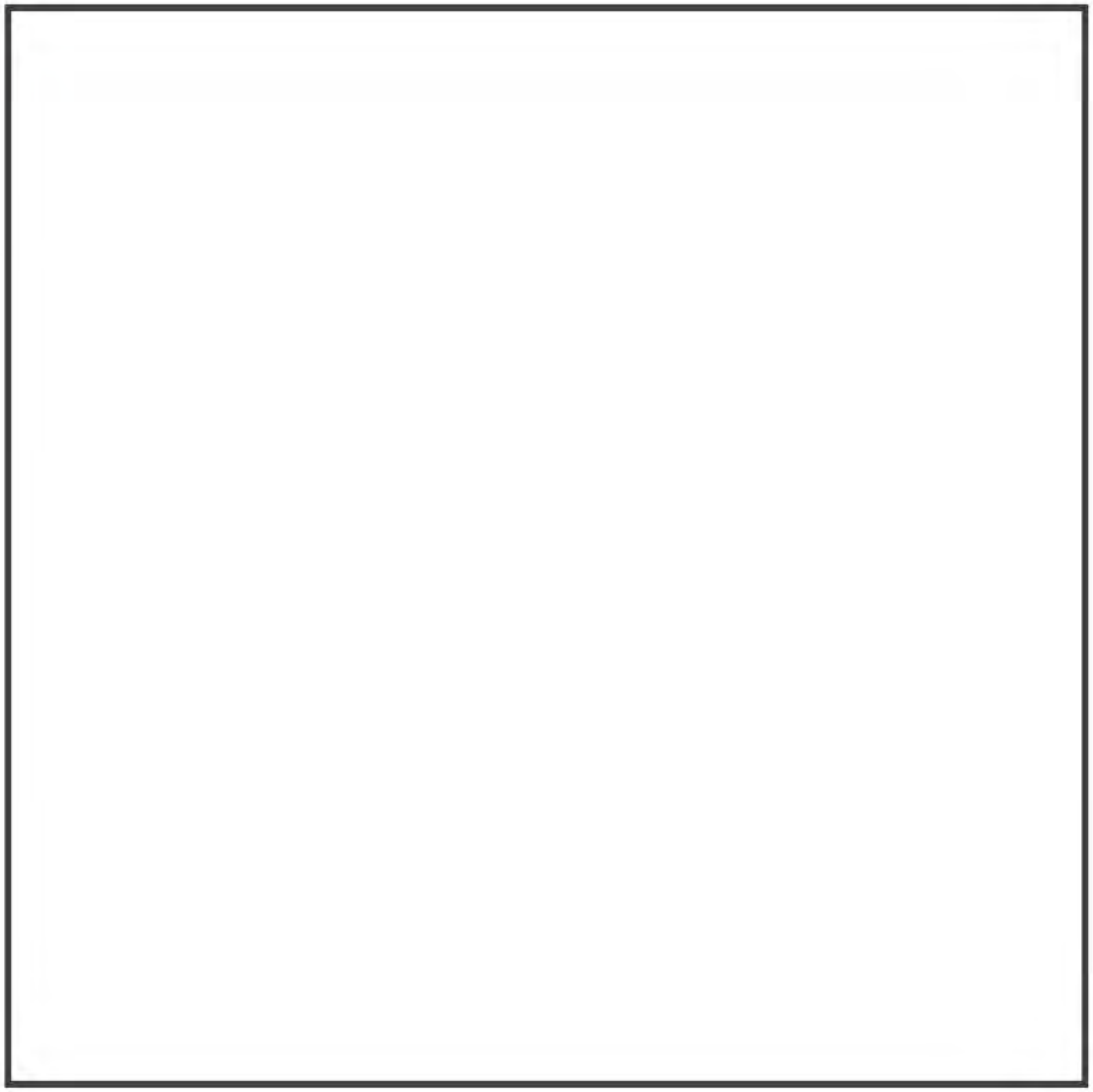
You've been tasked with building a new enterprise-class, full-trust SharePoint

Let's illustrate with an



You've been tasked with building a new enterprise-class, full-trust SharePoint solution - complete with a wide-array of functional and non-functional requirements

This box represents your application



Beautiful, isn't it?

This box represents your application

Beautiful, isn't it?

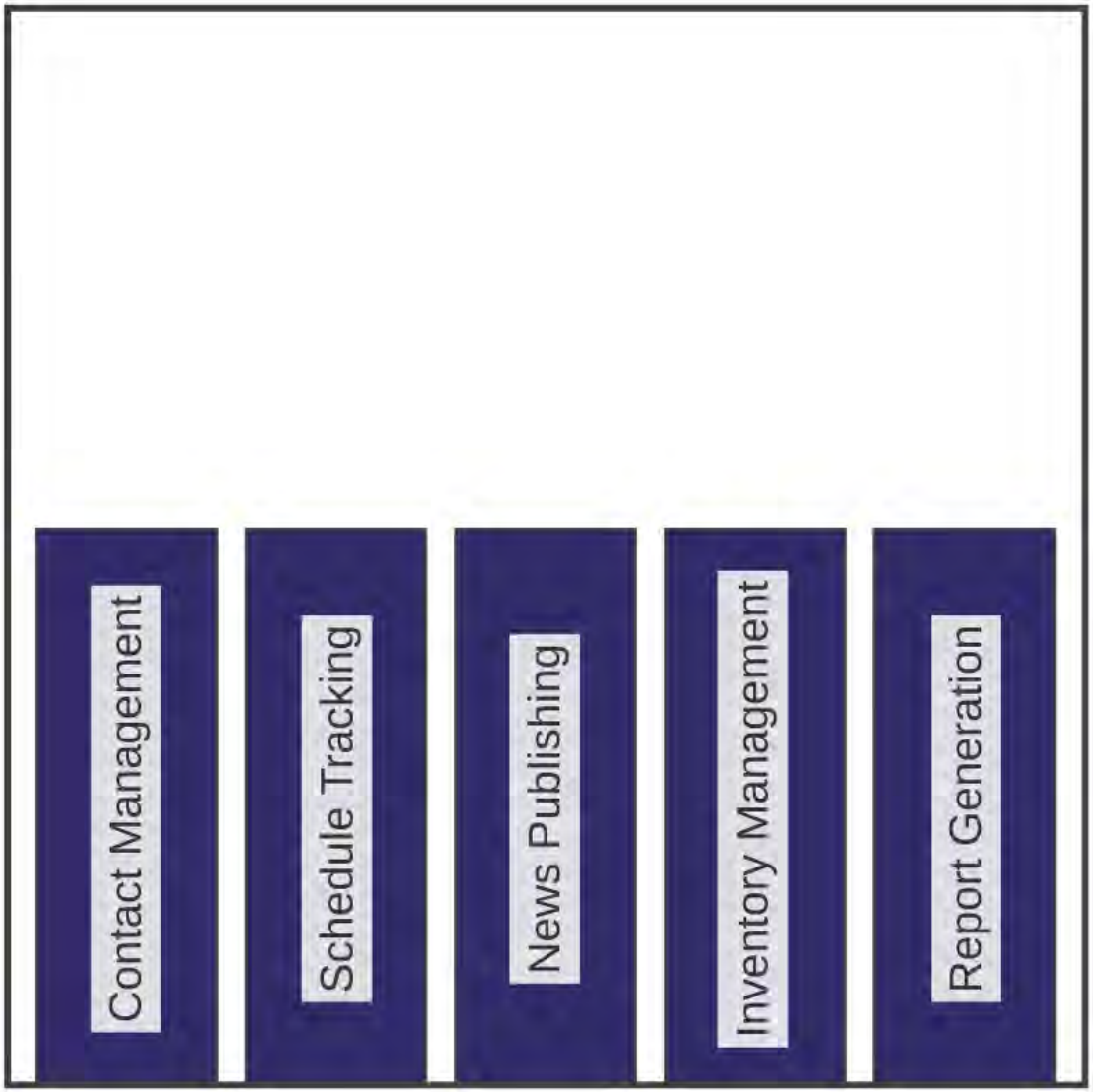
First up: functional requirements

The app needs to do something business-related, so we typically start by adding code based on our solution requirements.

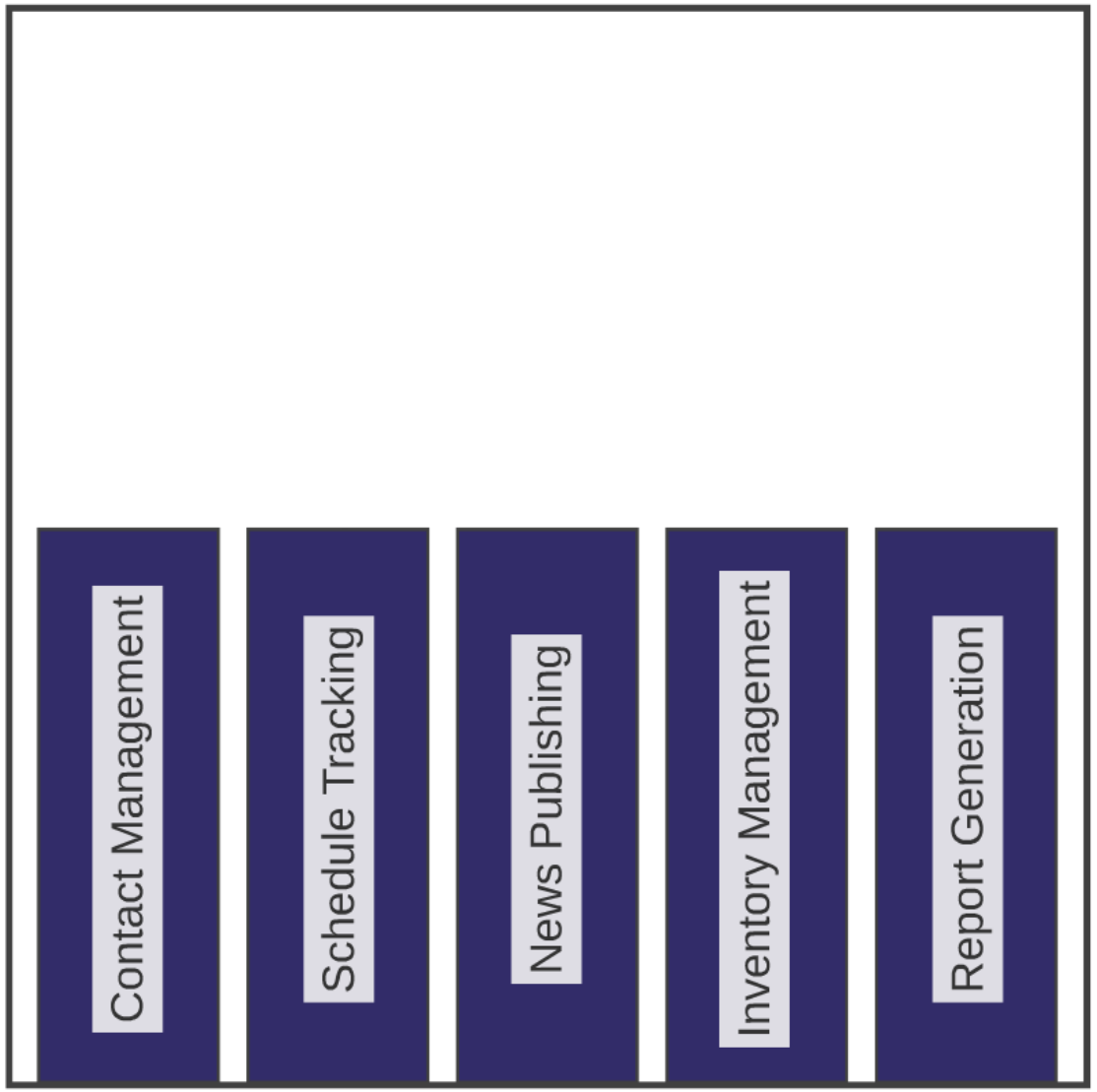
ful, isn't it?

First up: functional requirements

The app needs to do something business-related, so we typically start by adding code based on our solution requirements.



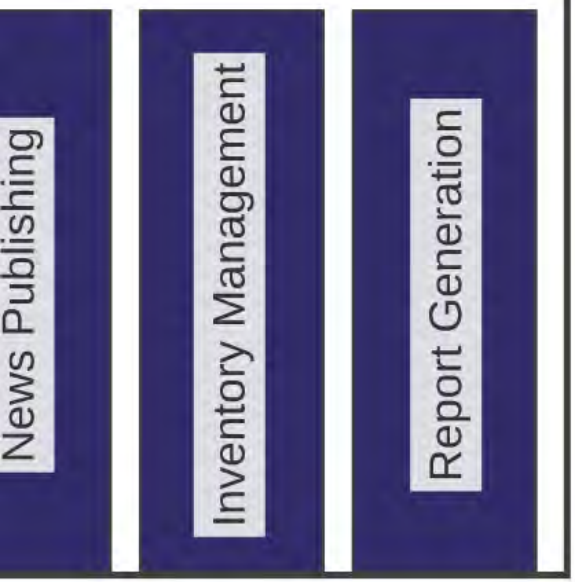
This is th



Chunks of code oriented around delivering desired functionality

- Vertical slices
- When end-users think of apps, this is where their focus usually is
- This code is generally "the fun stuff" to design and write

This is the part of our jo



around delivering desired functionality

- Vertical slices
- When end-users think of apps, this is where their focus usually is
- This code is generally "the fun stuff" to design and write

This is the part of our job where we feel like rock stars ...

coding like

"the fun stuff" to design and write

This is the part of our job where we feel like rock stars ...

... coding like crazy, showing users what we've done, havin' fun!

and then

code is generally
un stuff" to design
write

rt of our job where

ock stars ...

Example #1

and then

we feel like rock stars ...

... coding like crazy, showing users what we've done, havin' fun!

... and then ...

Reality catches up to us.

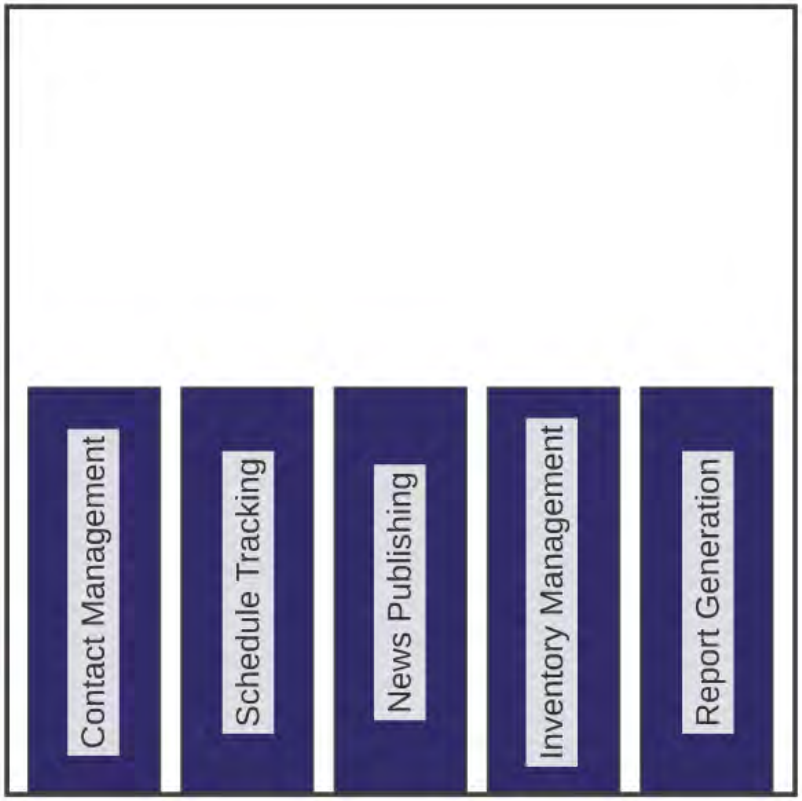
Business functionality is only part of the puzzle. There's a less glorious side to all of this solution development stuff.

In reality, our

glorious side to all of this
tion development stuff.

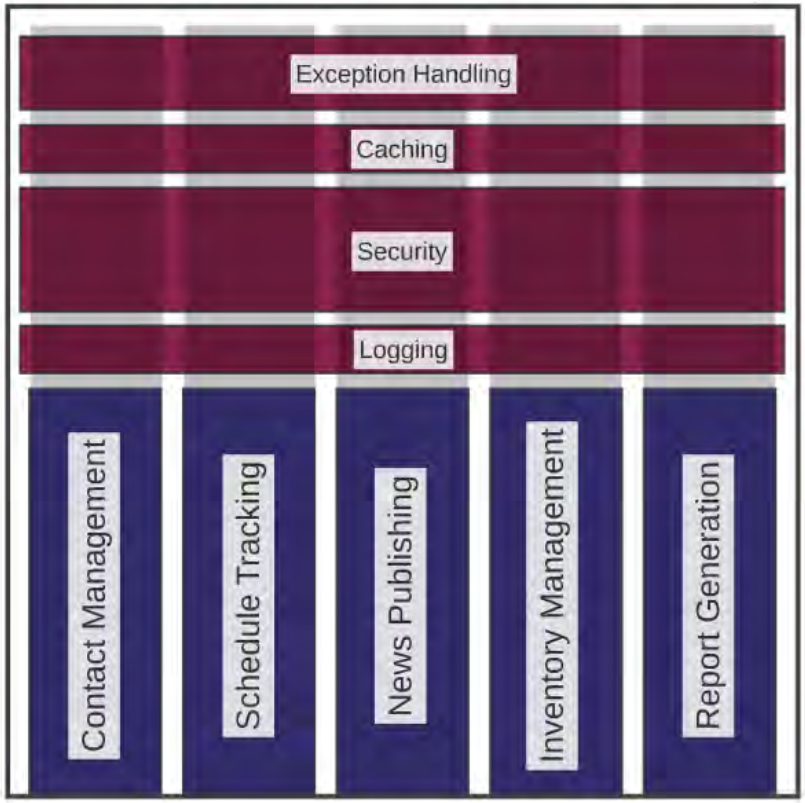
The other part: plumbing and non- functional requirements

In reality, our application doesn't look like this



Cross-cutting concerns

It looks like this

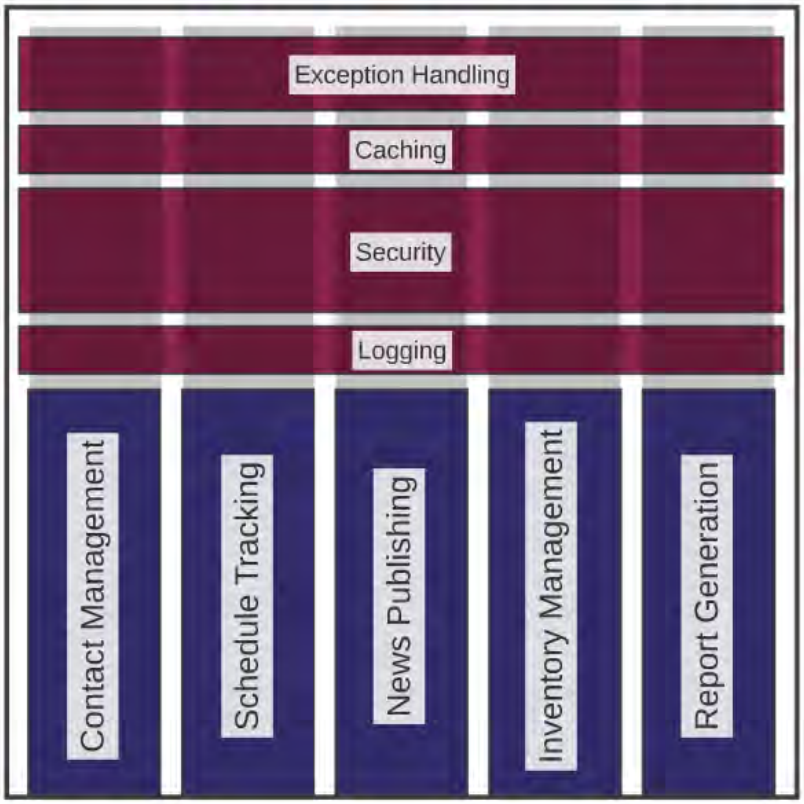


- Needs that "cut across" all of the solution's functional areas
- Plumbing code like security, exception handling, logging, caching, performance monitoring, and more
- Code tends to be highly repetitive in nature

Unfortunately, this tends to lead to a lot of cut-and-paste between classes in the average solution

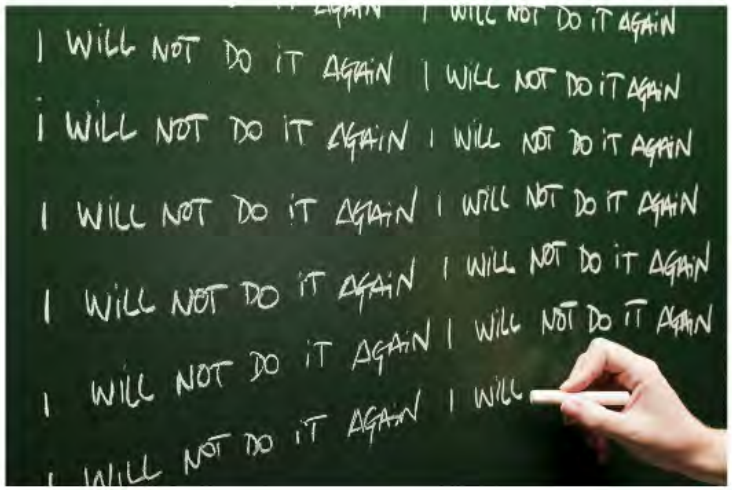
Cross-cutting concerns

It looks like this



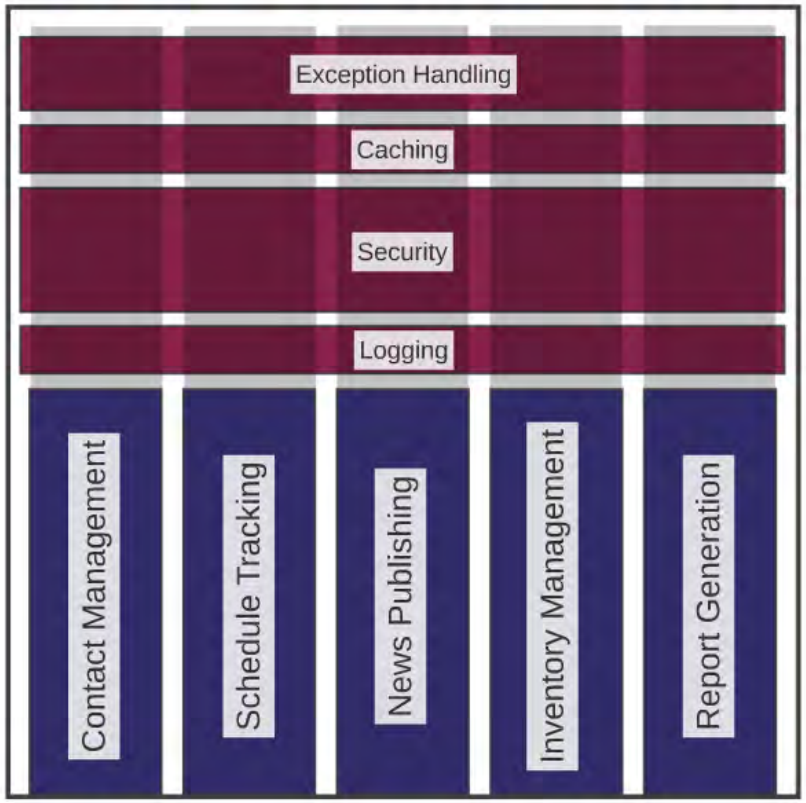
- Needs that "cut across" all of the solution's functional areas
- Plumbing code like security, exception handling, logging, caching, performance monitoring, and more
- Code tends to be highly repetitive in nature

Unfortunately, this tends to lead to a lot of cut-and-paste between classes in the average solution



Example #2

It looks like this



Unfortunately, this tends t
and-paste between classes

Going from this:

```
1 reference | spmcdonough, 14 hours ago | 3 changes
private String GenerateLine1()
{
    return "It is by caffeine alone that I set my mind in motion.\n";
}
```

... to this:

```
1 reference | spmcdonough, 18 hours ago | 2 changes
private String GenerateLine1()
```

Going from this:

```
1 reference | spmcdonough, 14 hours ago | 3 changes
private String GenerateLine1()
{
    return "It is by caffeine alone that I set my mind in motion.\n";
}
```

... to this:

```
1 reference | spmcdonough, 18 hours ago | 2 changes
private String GenerateLine1()
{
    LoggingSupport.WriteToLog("Entering Method GenerateLine1", 2);
    String whatToWrite = "It is by caffeine alone that I set my mind in motion.\n";
    LoggingSupport.WriteToLog("Exiting Method GenerateLine1", 2);
    return whatToWrite;
}
```

... is a substantial code change

Going from this:

```
1 reference | spmcdonough, 14 hours ago | 3 changes
private String GenerateLine1()
{
    return "It is by caffeine alone that I set my mind in motion.\n";
}
```

... to this:

```
1 reference | spmcdonough, 18 hours ago | 2 changes
private String GenerateLine1()
{
    LoggingSupport.WriteToLog("Entering Method GenerateLine1", 2);
    String whatToWrite = "It is by caffeine alone that I set my mind in motion.\n";
    LoggingSupport.WriteToLog("Exiting Method GenerateLine1", 2);
    return whatToWrite;
}
```

... is a substantial code change

Going from this:

```
1 reference | spmcdonough, 14 hours ago | 3 changes
private String GenerateLine1()
{
    return "It is by caffeine alone that I set my mir
}
```

... to this:

```
1 reference | spmcdonough, 18 hours ago | 2 changes
private String GenerateLine1()
{
    LoggingSupport.WriteToLog("Entering Method GenerateLine1",
    String whatToWrite = "It is by caffeine alone that I set m
    LoggingSupport.WriteToLog("Exiting Method GenerateLine1",
    return whatToWrite;
}
```

not cool

... is a substantial code change

And it only gets worse with each additional concern we add ...

Let's add in some exception handling

```
0 references | 0 authors | 0 changes
private String GenerateLine1()
{
    LoggingSupport.WriteToLog("Entering Method GenerateLine1", 2);
    String whatToWrite;

    try
    {
        whatToWrite = "It is by caffeine alone that I set my mind in motion.\n";
    }
    catch (Exception ex)
    {
        var newAppException = new Exception("Unexpected problem generating line 1", ex);
        LoggingSupport.WriteToLog(newAppException.ToString(), 3);
        throw newAppException;
    }

    LoggingSupport.WriteToLog("Exiting Method GenerateLine1", 2);
    return whatToWrite;
}
```

exception handling, and caching brought us to this

And some memory-based caching

```
0 references | 0 authors | 0 changes
private String GenerateLine1()
{
    LoggingSupport.WriteToLog("Entering Method GenerateLine1", 2);
    const String lineCacheKey = "TESTAPP_GenerateLine1_KEY";
    String whatToWrite;

    try
    {
        var aspNetCache = HttpContext.Current.Cache;
        Object targetLineObject = aspNetCache[lineCacheKey];
        if (targetLineObject == null)
        {
            targetLineObject = "It is by caffeine alone that I set my mind in motion.\n";
            aspNetCache.Add(lineCacheKey, targetLineObject, null, Cache.NoAbsoluteExpiration,
                TimeSpan.FromMinutes(15), CacheItemPriority.Default, null);
        }
        whatToWrite = targetLineObject.ToString();
    }
    catch (Exception ex)
    {
        var newAppException = new Exception("Unexpected problem generating line 1", ex);
        LoggingSupport.WriteToLog(newAppException.ToString(), 3);
        LoggingSupport.WriteToLog("Exiting Method GenerateLine1 due to exception", 2);
        throw newAppException;
    }

    LoggingSupport.WriteToLog("Exiting Method GenerateLine1", 2);
    return whatToWrite;
}
```

We started with this:

```
1 reference | spmcdonough, 14 hours ago | 3 changes
private String GenerateLine1()
{
    return "It is by caffeine alone that I set my mind in motion.\n";
}
```

And adding logging, exception handling, and caching brought us to this

```
0 references | 0 authors | 0 changes
private String GenerateLine1()
{
    LoggingSupport.WriteToLog("Entering Method GenerateLine1", 2);
    const String lineCacheKey = "TESTAPP_GenerateLine1_KEY";
    String whatToWrite;

    try
    {
        var aspNetCache = HttpContext.Current.Cache;
        Object targetLineObject = aspNetCache[lineCacheKey];
        if (targetLineObject == null)
        {
            targetLineObject = "It is by caffeine alone that I set my mind in motion.\n";
            aspNetCache.Add(lineCacheKey, targetLineObject, null, Cache.NoAbsoluteExpiration,
                TimeSpan.FromMinutes(15), CacheItemPriority.Default, null);
        }
        whatToWrite = targetLineObject.ToString();
    }
    catch (Exception ex)
    {
        var newAppException = new Exception("Unexpected problem generating line 1", ex);
        LoggingSupport.WriteToLog(newAppException.ToString(), 3);
        LoggingSupport.WriteToLog("Exiting Method GenerateLine1 due to exception", 2);
        throw newAppException;
    }

    LoggingSupport.WriteToLog("Exiting Method GenerateLine1", 2);
    return whatToWrite;
}
```

24 lines

And the worst part

We started with this:

```
1 reference | spmcdonough, 14 hours ago | 3 changes
private String GenerateLine1()
{
    return "It is by caffeine alone that I set my mind in motion.\n";
}
```

1 line

And adding logging, exception handling, and caching brought us to this

```
0 references | 0 authors | 0 changes
private String GenerateLine1()
{
    LoggingSupport.WriteToLog("Entering Method GenerateLine1", 2);
    const String lineCacheKey = "TESTAPP_GenerateLine1_KEY";
    String whatToWrite;

    try
    {
        var aspNetCache = HttpContext.Current.Cache;
        Object targetLineObject = aspNetCache[lineCacheKey];
        if (targetLineObject == null)
        {
            targetLineObject = "It is by caffeine alone that I set my mind in motion.\n";
            aspNetCache.Add(lineCacheKey, targetLineObject, null, Cache.NoAbsoluteExpiration,
                TimeSpan.FromMinutes(15), CacheItemPriority.Default, null);
        }
        whatToWrite = targetLineObject.ToString();
    }
    catch (Exception ex)
    {
        var newAppException = new Exception("Unexpected problem generating line 1", ex);
        LoggingSupport.WriteToLog(newAppException.ToString(), 3);
        LoggingSupport.WriteToLog("Exiting Method GenerateLine1 due to exception", 2);
        throw newAppException;
    }

    LoggingSupport.WriteToLog("Exiting Method GenerateLine1", 2);
    return whatToWrite;
}
```

24 lines

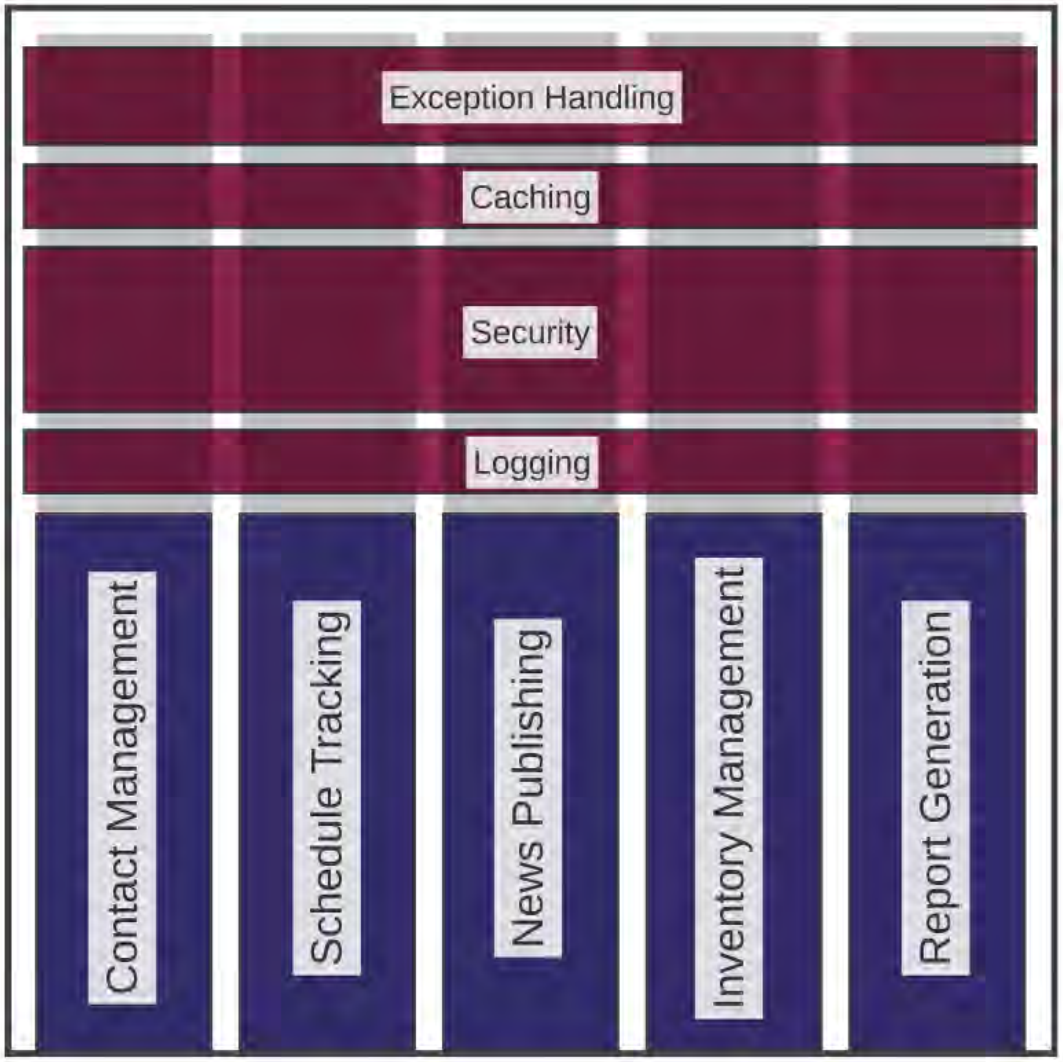
And the worst part of this ...

We end up doing it

And the worst part of this ...

We end up doing it for nearly all methods and properties

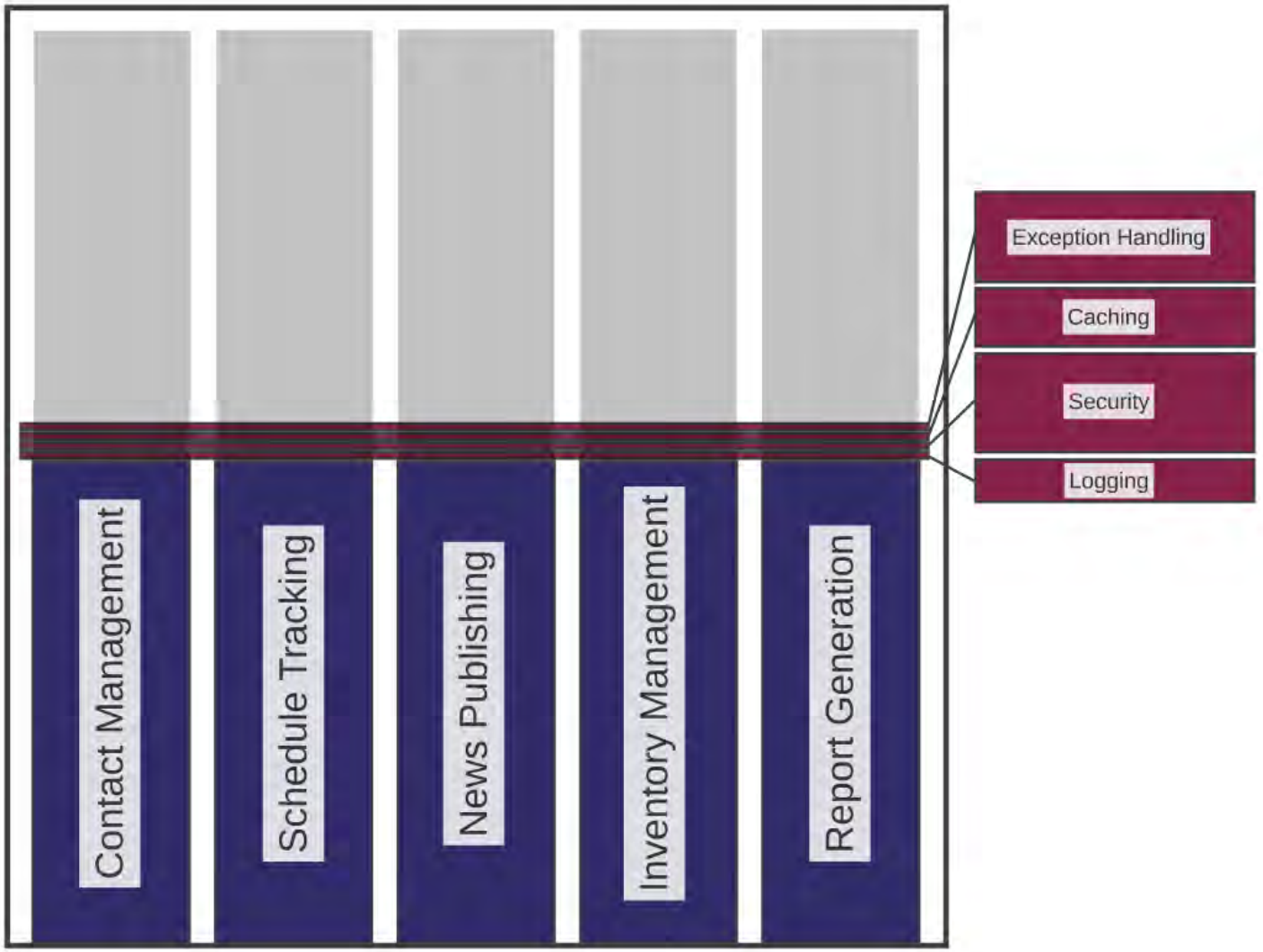
There has to be a better way



Instead of this

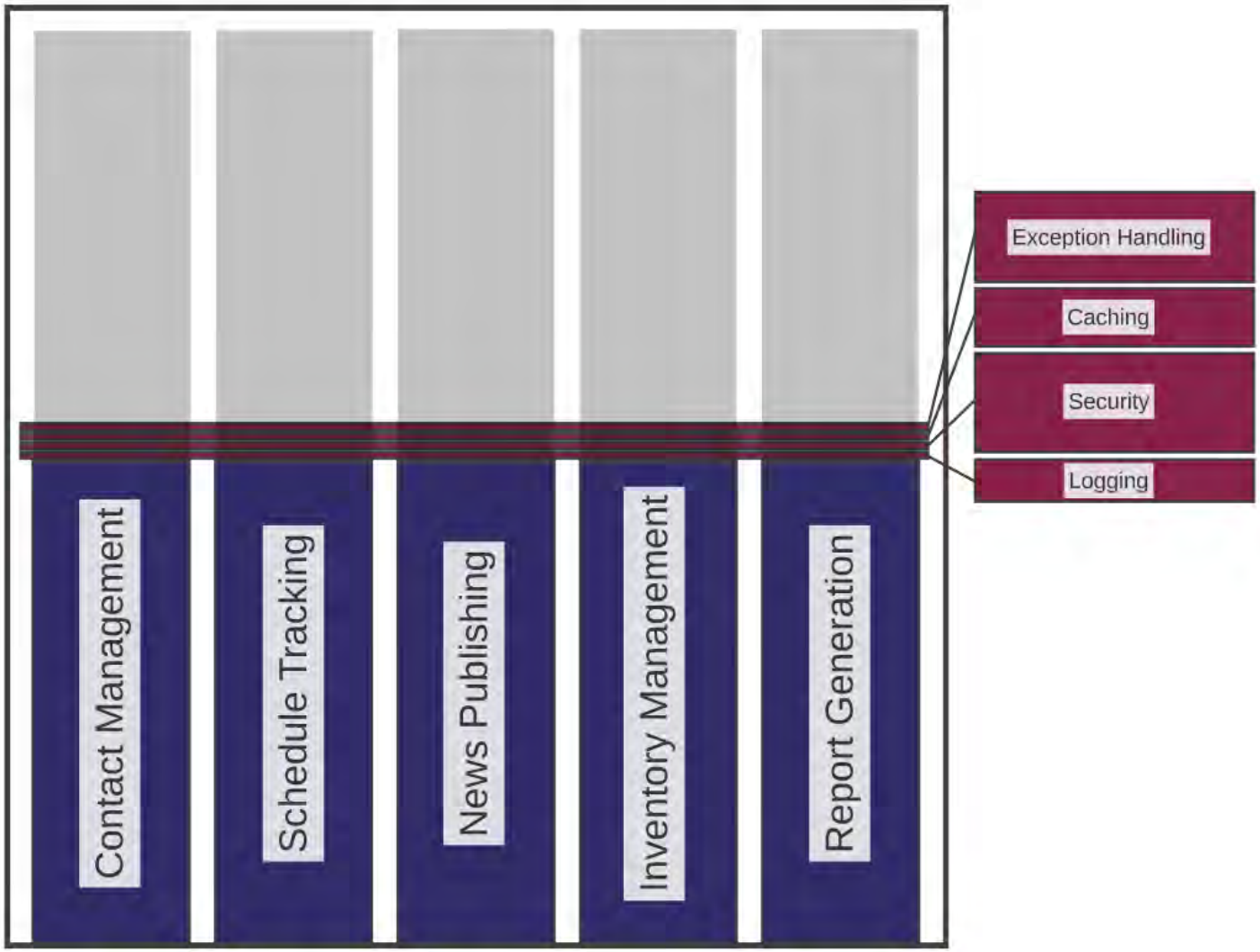
This is exactly what AOP offers

we need this

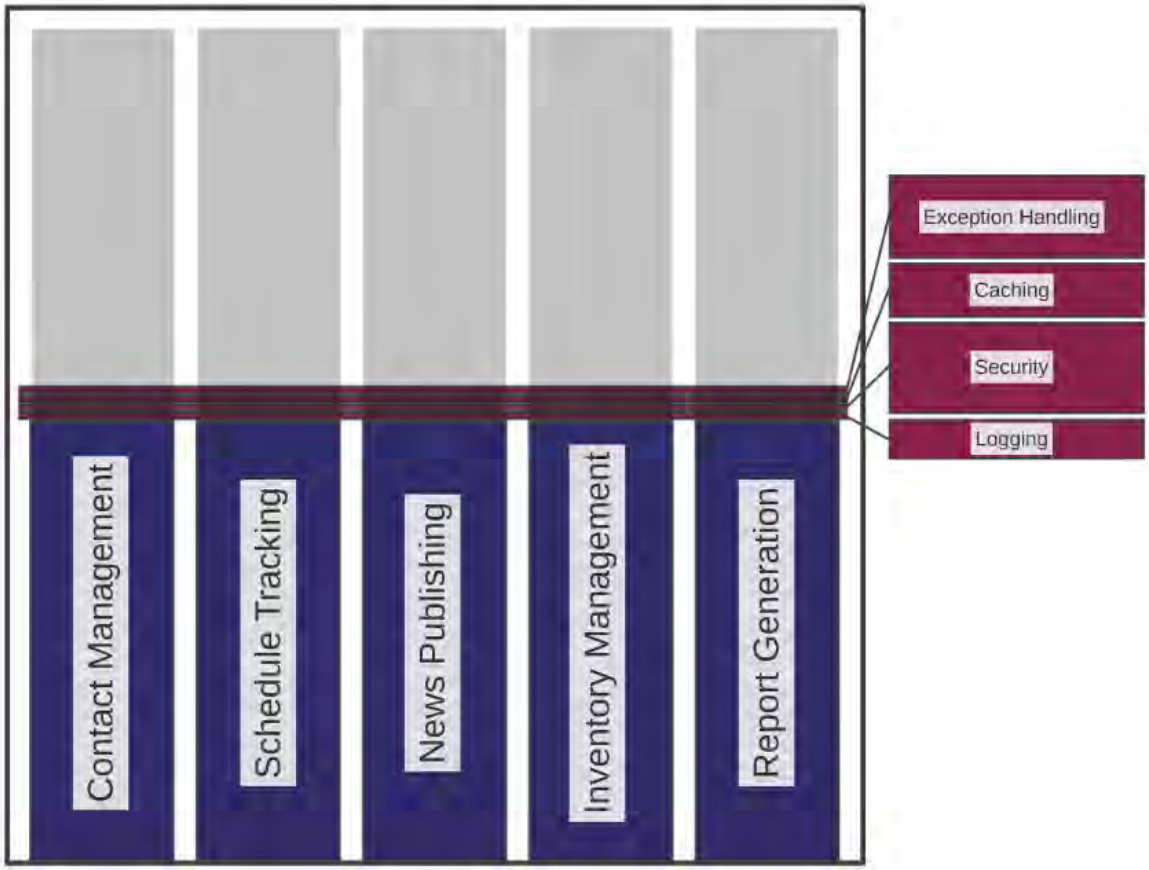


This is exactly what AOP offers

we need this



actly what AOP offers



- Cross-cutting concerns are encapsulated within aspects
- Functional code remains clear of redundant plumbing code
- Reduces clutter and overall line counts
- Simplifies maintenance

- Cross-cutting concerns are encapsulated within aspects
- Functional code remains clear of redundant plumbing code
- Reduces clutter and overall line counts
- Simplifies maintenance

Example #3

FREQUENTLY

- Simplifies maintenance

Seems neat, but if AOP is so useful, how come I haven't seen it "in the wild" by now?

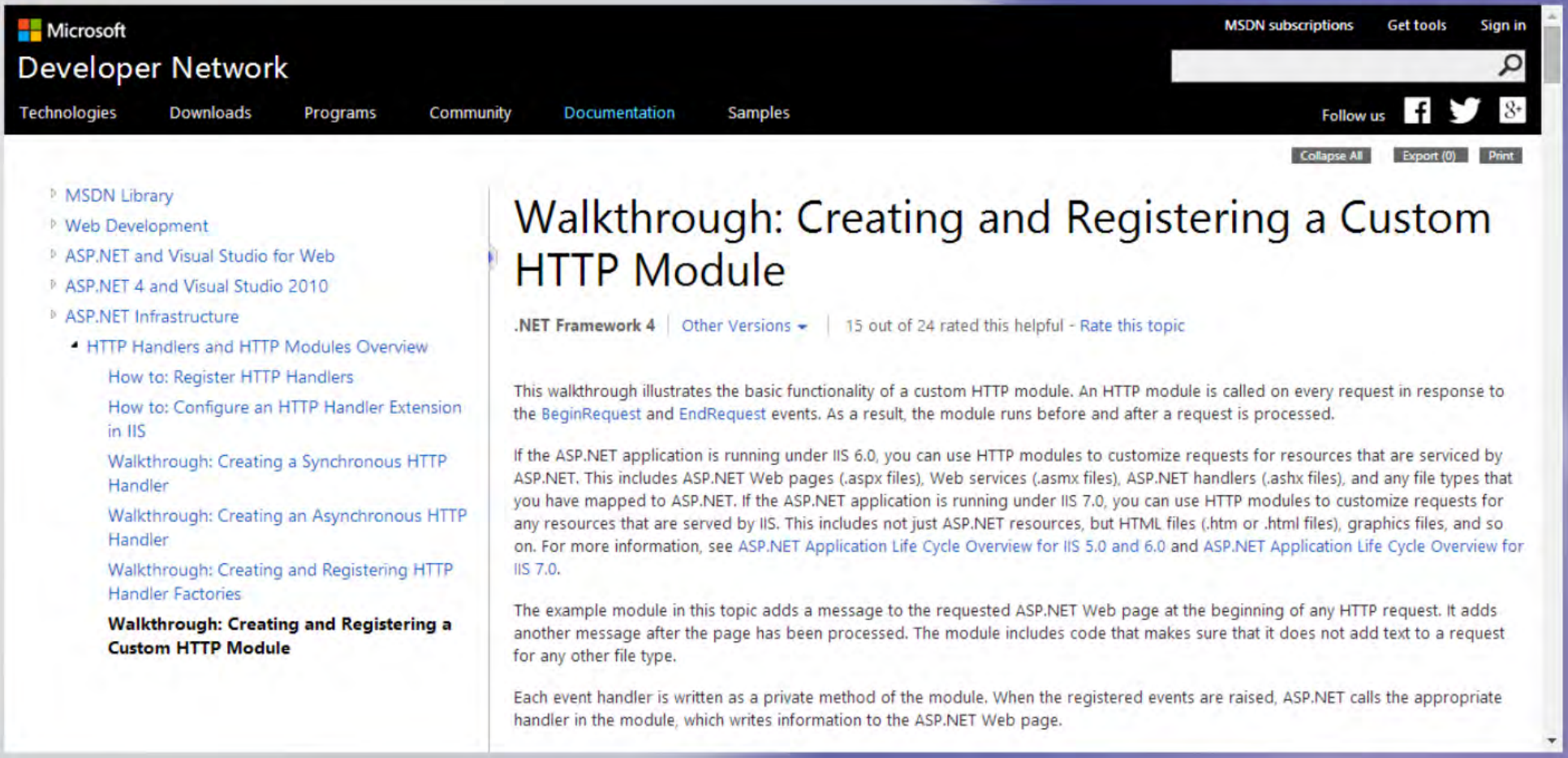
Aspects come in many forms

And if you haven't been looking

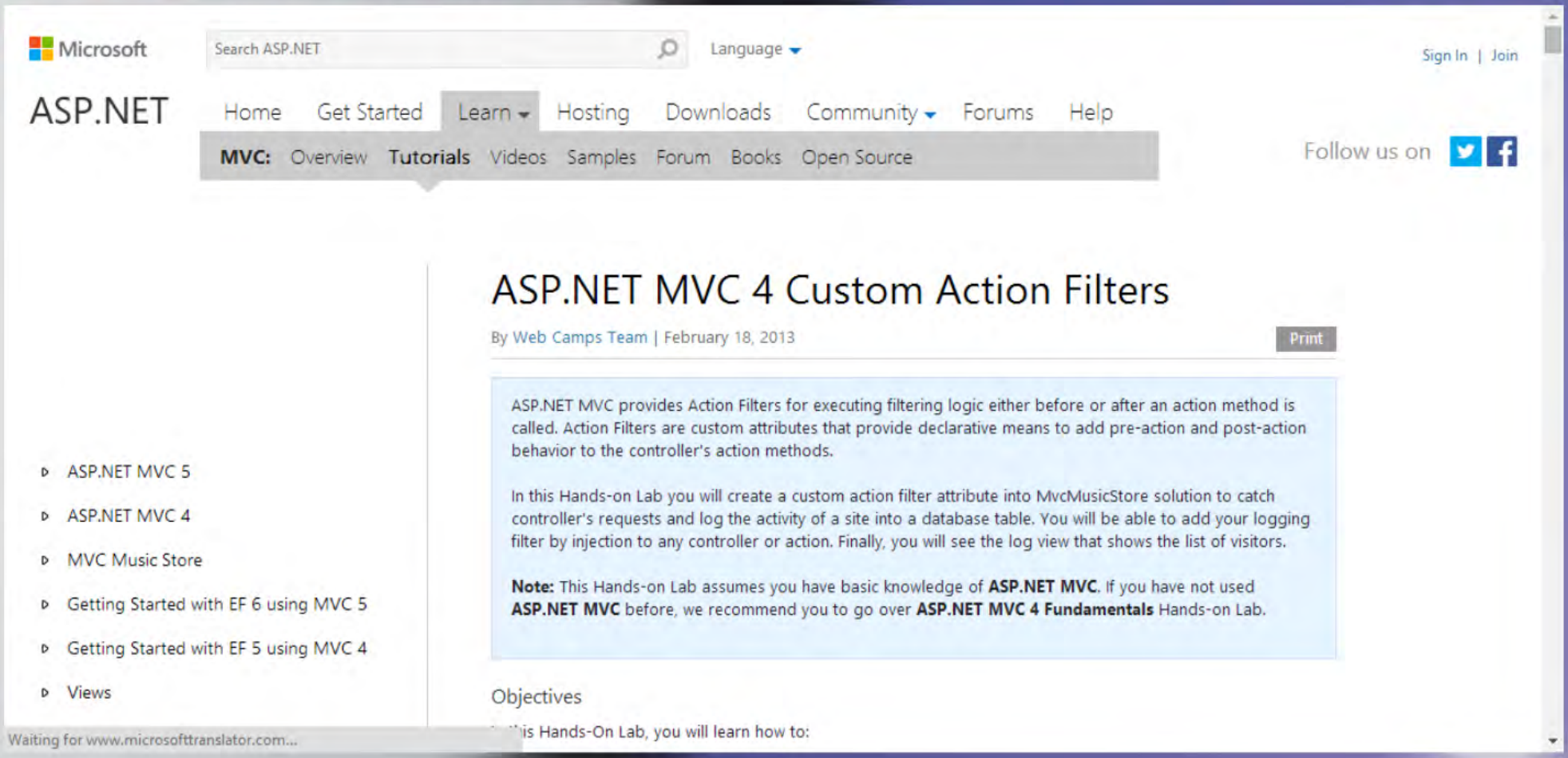
And if you haven't been looking, you may have missed them

HTTP Modules and ASP.NET MVC Action

Have you ever used an HTTP Module?



How about an ASP.NET MVC Action Filter?



you may have missed them

HTTP Modules and ASP.NET MVC Action Filters are just two examples of AOP at work

There are other tools, but this
the best implemented and supp

My AOP Tool of Choice:

PostSharp Ultimate

There are other tools, but this is (in my opinion) the best implemented and supported

PostSharp Ultimate

- Free version available
- Cleanest separation of concerns (no spaghetti code)
- Employs compile-time weaving

Um ...

spaghetti code)

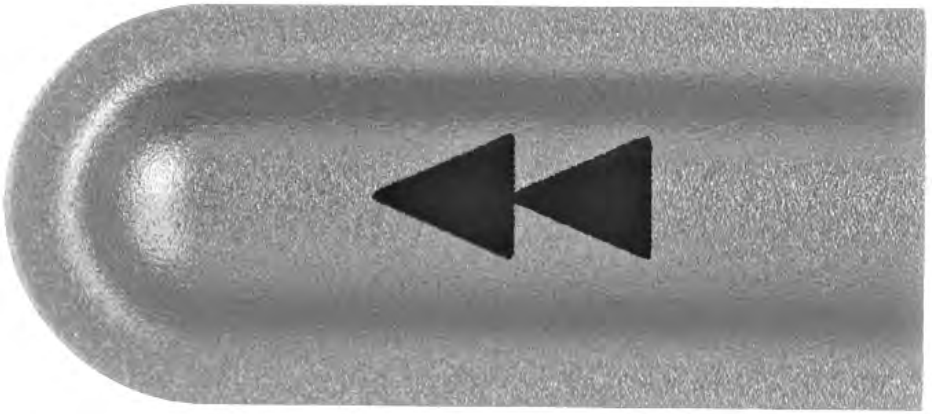
- Employs compile-time weaving

Um ...

compile-time what?

Okay, let's pause for a second

And rewind to cover some basics



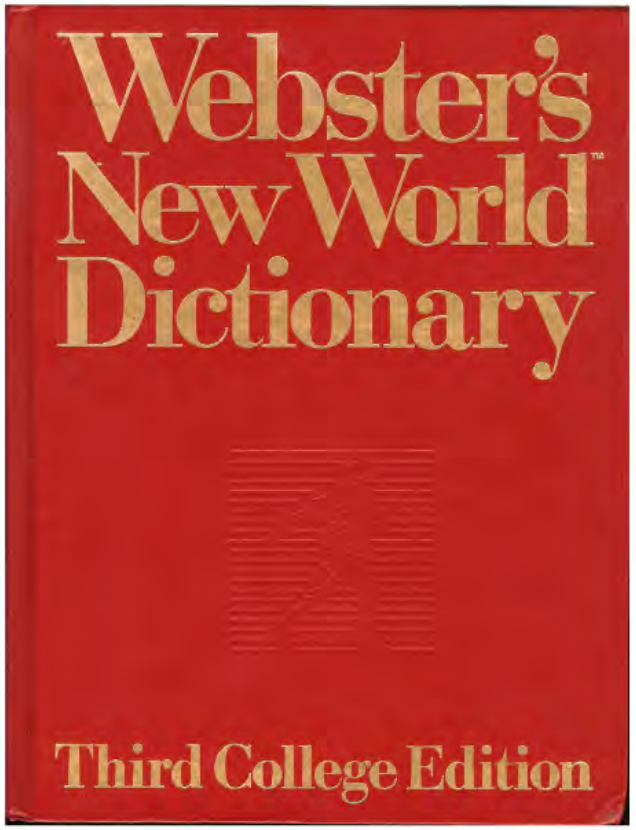
A code snippet from the LoggingToTextboxAspect

```
/// <summary>
/// This method boundary aspect (created with PostSharp) is responsible
/// for handling logging activities for each of the methods with which
/// it is associated.
/// </summary>
[Serializable]
1 reference | spmcdonough, 3 days ago | 1 change
internal class LoggingToTextboxAspect : OnMethodBoundaryAspect
{

    #region Overrides: OnMethodBoundaryAspect

    /// <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 | spmcdonough, 3 days ago | 1 change
    public override void OnEntry(MethodExecutionArgs args)
    {
        CreateLogEntry(args, "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 | spmcdonough, 3 days ago | 1 change
    public override void OnExit(MethodExecutionArgs args)
    {
        CreateLogEntry(args, "Exiting Method");
    }

    #endregion Overrides: OnMethodBoundaryAspect
```

Let's establish some

pet from the LoggingToTextboxAspect

```
/// <summary>
/// This method boundary aspect (created with PostSharp) is responsible
/// for handling logging activities for each of the methods with which
/// it is associated.
/// </summary>
[Serializable]
1 reference | spmcdonough, 3 days ago | 1 change
internal class LoggingToTextboxAspect : OnMethodBoundaryAspect
{

    #region Overrides: OnMethodBoundaryAspect

    /// <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 | spmcdonough, 3 days ago | 1 change
    public override void OnEntry(MethodExecutionArgs args)
    {
        CreateLogEntry(args, "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 | spmcdonough, 3 days ago | 1 change
    public override void OnExit(MethodExecutionArgs args)
    {
        CreateLogEntry(args, "Exiting Method");
    }

    #endregion Overrides: OnMethodBoundaryAspect
```



Let's establish some definitions

A code snippet from the LoggingToTe

The aspect code itself is called ..


```
/// <summary>
/// This method boundary aspect (created with PostSharp) is responsible
/// for handling logging activities for each of the methods with which
/// it is associated.
/// </summary>
[Serializable]
1 reference | spmcdonough, 3 days ago | 1 change
internal class LoggingToTextboxAspect : OnMethodBoundaryAspect
{

    #region Overrides: OnMethodBoundaryAspect


    /// <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 | spmcdonough, 3 days ago | 1 change
    public override void OnEntry(MethodExecutionArgs args)
    {
        CreateLogEntry(args, "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 | spmcdonough, 3 days ago | 1 change
    public override void OnExit(MethodExecutionArgs args)
    {
        CreateLogEntry(args, "Exiting Method");
    }


    #endregion Overrides: OnMethodBoundaryAspect
```

Let's establis

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

The arrows represent

```
1 reference | spmcdonough, 14 hours ago | 3 changes
private String GenerateLine1()
{
    return "It is by caffeine alone that I set my mind in motion.\n";
}
```

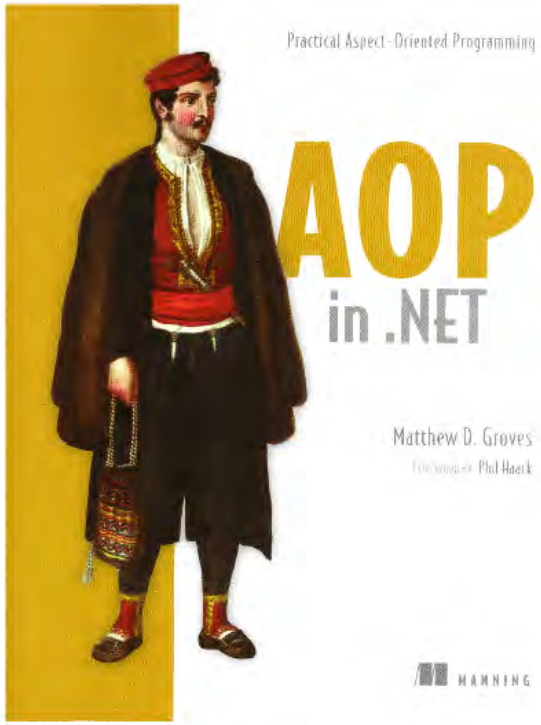
```
/// <summary>
/// This method boundary aspect (created with PostSharp) is responsible
/// for handling logging activities for each of the methods with which
/// it is associated.
/// </summary>
[Serializable]
1 reference | spmcdonough, 3 days ago | 1 change
internal class LoggingToTextboxAspect : OnMethodBoundaryAspect
{

    #region Overrides: OnMethodBoundaryAspect

    /// <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 | spmcdonough, 3 days ago | 1 change
    public override void OnEntry(MethodExecutionArgs args)
    {
        CreateLogEntry(args, "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 | spmcdonough, 3 days ago | 1 change
    public override void OnExit(MethodExecutionArgs args)
    {
        CreateLogEntry(args, "Exiting Method");
    }

    #endregion Overrides: OnMethodBoundaryAspect
```

Join Points

"A join point is a place that can be defined between logical steps

AOP targets

Join Points

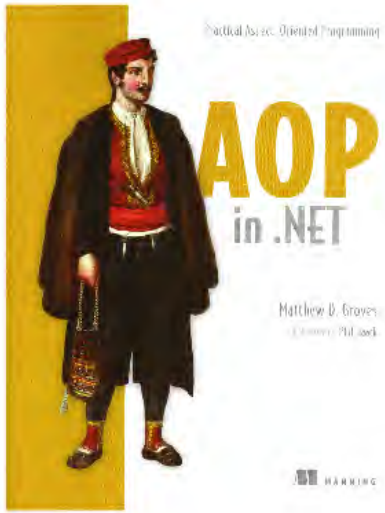


"A join point is a place that can be defined between logical steps of the execution of your program."

- Matthew D. Groves

Join Points

```
/// </summary>
0 references | spmcdonough, 3 days ago |
public override void OnExi
{
    CreateLogEntry(args, "
}


#endregion Overrides: OnMe
```



"A join point is a place that can be defined between logical steps of the execution of your program."

- Matthew D. Groves

AOP targets these

A set of join points is known as a **pointcut**

brinas

roves

pointcut

brings us to

weaving

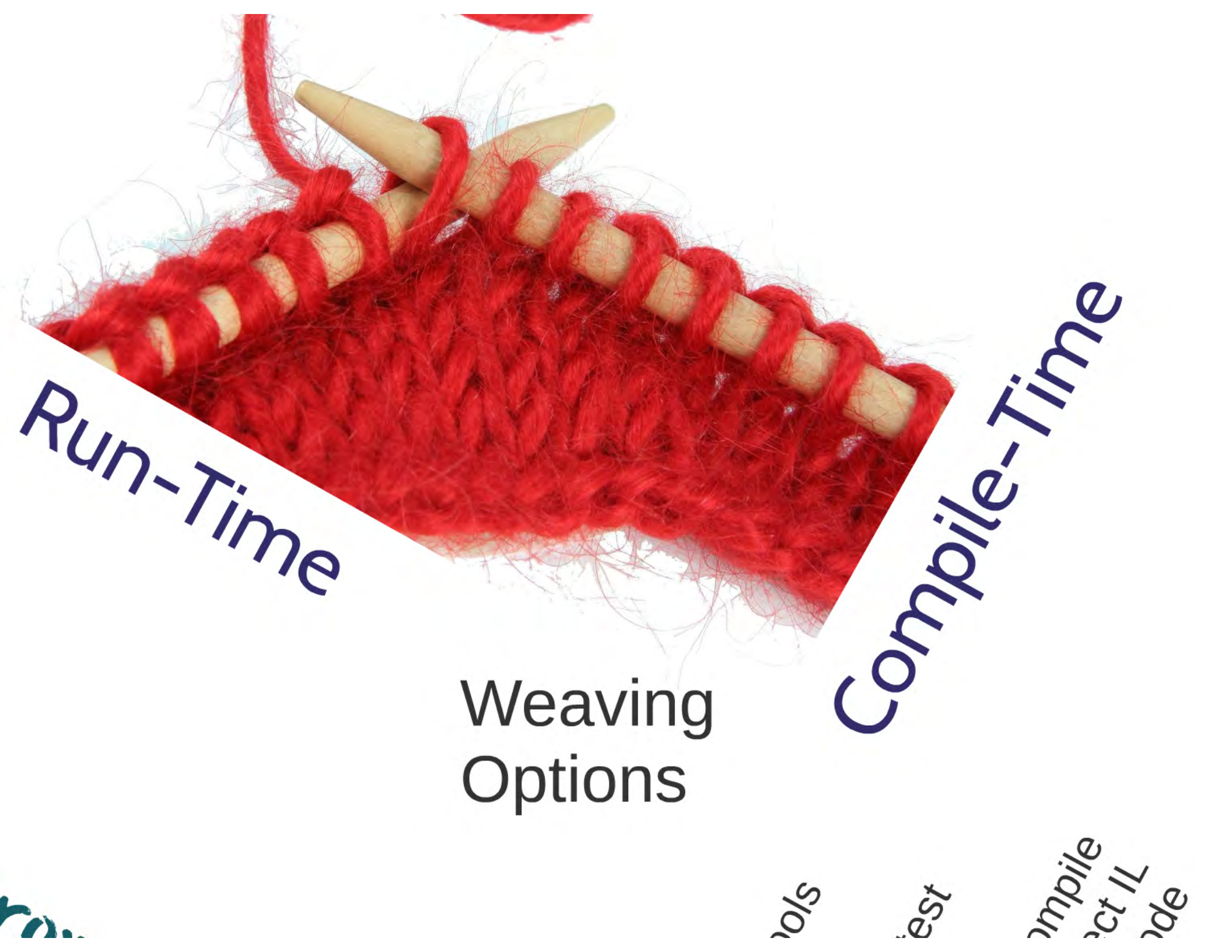
The process by which aspects (advice) are

as to

Compile-Time

weaving

The process by which aspects (advice) are applied to pointcuts for use by and with your code


Run-Time
Compile-Time
Weaving
Options

Run-Time

- Typically relies on reflection
- Doesn't require special tools
- Easier to (unit) test
- Acts similarly to a proxy or decorator

Example: Castle DynamicProxy

Compile-Time

- Requires tools
- Hard to (unit) test
- Involves a post-compile step to weave aspect IL with main solution code
- Allows for optimizations

Example:

PostSharp

We've covered key AOP concepts:

- Advice
- Join Points
- Pointcuts
- Weaving

these apply to aspects

We've covered

Aspects

- Advice
- Join Points
- Pointcuts
- Weaving

And we've talked about how these apply to aspects

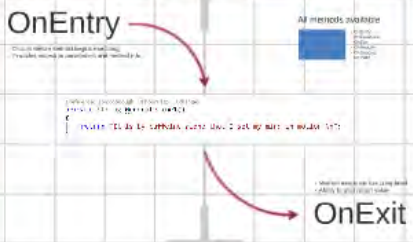
Let's look at some of the common aspect types and how they work

Aspects

Aspect Types We're Going To Examine

How It Works | Potential Uses | Considerations

Method Boundary



Well-suited to repetitive tasks

- (ULS) Logging
- Performance profiling
- Tracing
- Exception handling*

Aspect methods are statically scoped

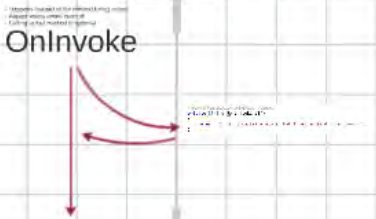
- Only one instance of each method services all requests from implementing types
- Sidestep this with either the MethodExecutionTag or by implementing IInstanceScopeAspect

IL can be optimized by PostSharp*

- Arguments selectively copied boxed/unboxed

Multiple methods = great flexibility

Method Interception



Perfect for tasks that involve selective execution

- Caching
- Retry support
- Threading

Shared state benefits

- All activity happens in OnInvoke, so state is easy to track outside of intercepted method
- No need for MethodExecutionTags and what-not

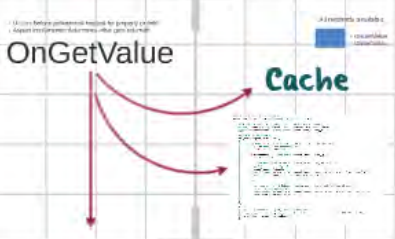
Somewhat reduced clarity

- Downside of everything in one method

IL cannot be optimized

- All arguments are boxed/unboxed per invocation

Location Interception



Similar to method interception, but more granular

- Validation
- Filtering
- Change tracking & notification
- Lazy loading & initialization

Same basic set of considerations as method interception aspects

- Again, similar in operation - just narrower scope

Works for properties and fields

- Including auto-properties

Method

Boundary

Method

OnEntry

- Occurs before method begins executing
- Provides access to parameters and method info

All methods available

- OnEntry
- OnException
- OnExit
- OnResume
- OnSuccess
- OnYield

```
1 reference | spmcdonough, 14 hours ago | 3 changes
private String GenerateLine1()
{
    return "It is by caffeine alone that I set my mind in motion.\n";
}
```

- Method execution has completed
- Ability to alter return value

OnExit

- Happens instead of the method being called
- Aspect wraps entire method
- Calling actual method is optional

OnEntry

- Occurs before method begins executing
- Provides access to parameters and method info

- Method execution has completed
- Ability to alter return value

OnExit

All methods available

- OnEntry
- OnException
- OnExit
- OnResume
- OnSuccess
- OnYield

OnEntry

- Occurs before method begins executing
- Provides access to parameters and method info

All methods available

- OnEntry
- OnException
- OnExit
- OnResume
- OnSuccess
- OnYield

```
1 reference | spmcdonough, 14 hours ago | 3 changes
private String GenerateLine1()
{
    return "It is by caffeine alone that I set my mind in motion.\n";
}
```

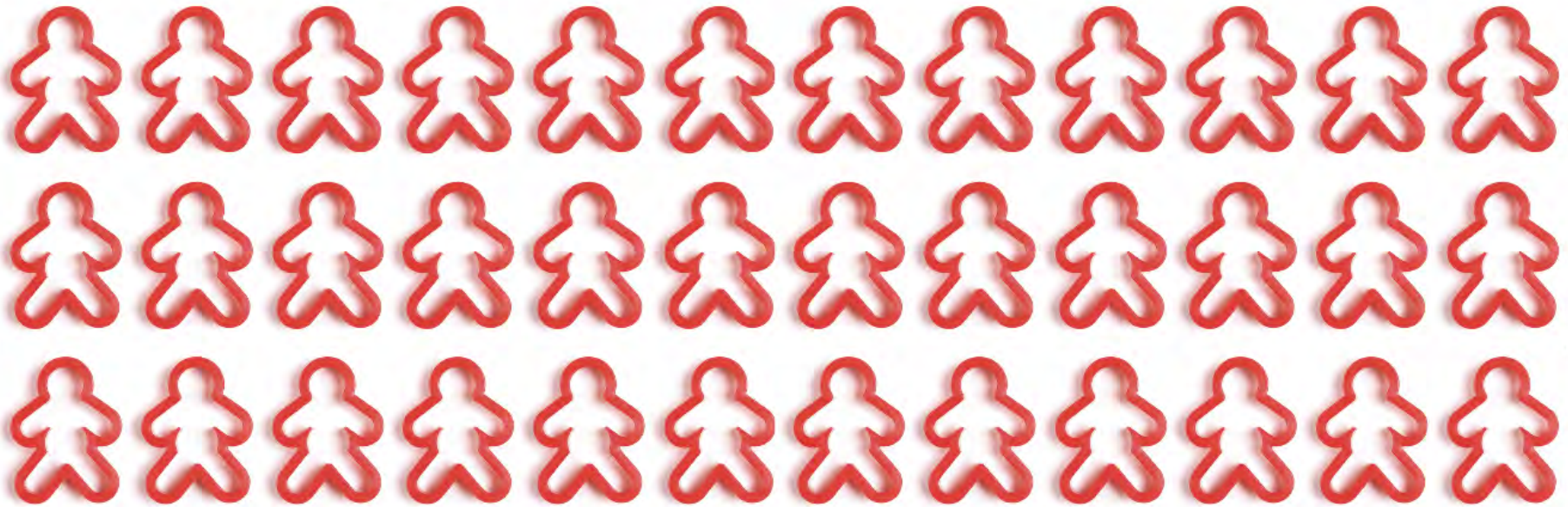
- Method execution has completed
- Ability to alter return value

OnExit

- Happens instead of the method being called
- Aspect wraps entire method
- Calling actual method is optional

Well-suited to repetitive tasks

- (ULS) Logging
- Tracing
- Performance profiling
- Exception handling*



Perfect for tasks that

Aspect methods are statically scoped

- Only one instance of each method services all requests from implementing types
- Sidestep this with either the MethodExecutionTag or by implementing IInstanceScopeAspect

IL can be optimized by PostSharp*

- Arguments selectively copied boxed/unboxed

Multiple methods = great flexibility

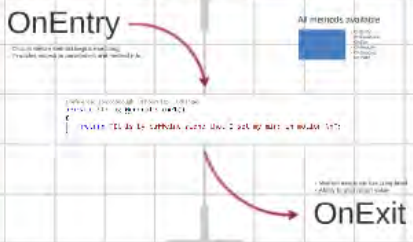
Shared state benefits

Example #4

Aspect Types We're Going To Examine

How It Works | Potential Uses | Considerations

Method Boundary



Well-suited to repetitive tasks

- (ULS) Logging
- Performance profiling
- Tracing
- Exception handling*

Aspect methods are statically scoped

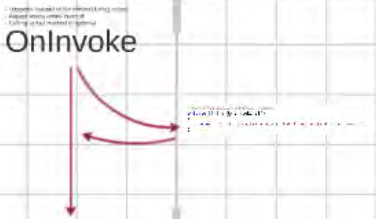
- Only one instance of each method services all requests from implementing types
- Sidestep this with either the MethodExecutionTag or by implementing IInstanceScopeAspect

IL can be optimized by PostSharp*

- Arguments selectively copied boxed/unboxed

Multiple methods = great flexibility

Method Interception



Perfect for tasks that involve selective execution

- Caching
- Retry support
- Threading

Shared state benefits

- All activity happens in OnInvoke, so state is easy to track outside of intercepted method
- No need for MethodExecutionTags and what-not

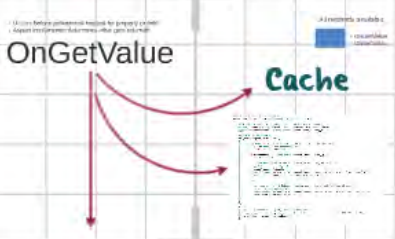
Somewhat reduced clarity

- Downside of everything in one method

IL cannot be optimized

- All arguments are boxed/unboxed per invocation

Location Interception



Similar to method interception, but more granular

- Validation
- Filtering
- Change tracking & notification
- Lazy loading & initialization

Same basic set of considerations as method interception aspects

- Again, similar in operation - just narrower scope

Works for properties and fields

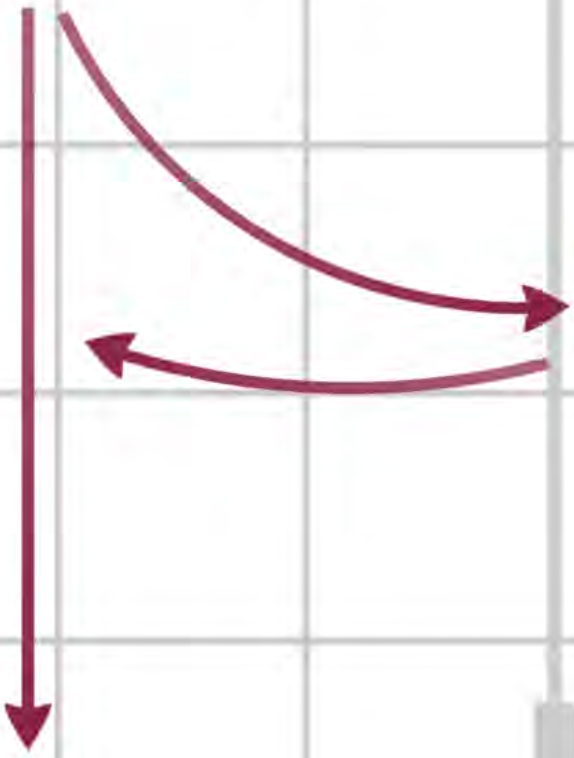
- Including auto-properties

Boundary Method Interception Location

OnExit

- Happens instead of the method being called
- Aspect wraps entire method
- Calling actual method is optional

OnInvoke



```
1 reference | spmcdonough, 14 hours ago | 3 changes
private String GenerateLine1()
{
    return "It is by caffeine alone that I set my mind in motion.\n";
}
```

- Occurs before get/retrieval request for property or field

All methods available

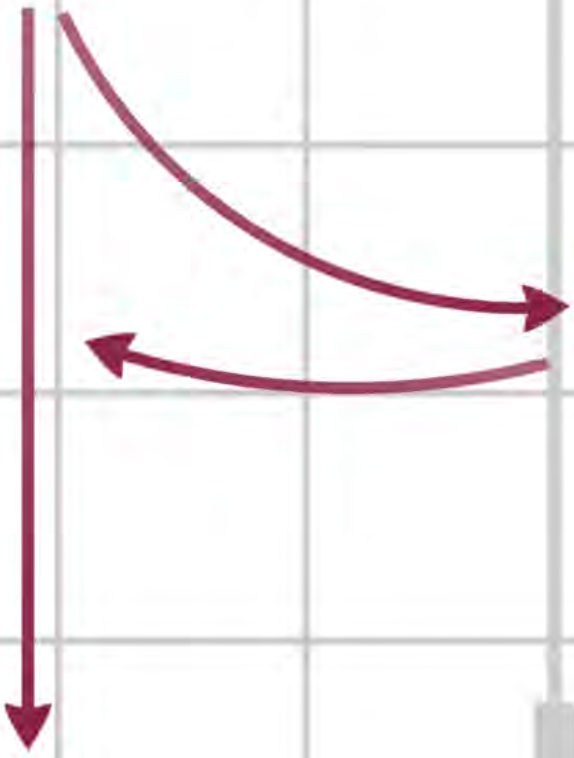
- Happens instead of the method being called
- Aspect wraps entire method
- Calling actual method is optional

OnInvoke

OnExit

- Happens instead of the method being called
- Aspect wraps entire method
- Calling actual method is optional

OnInvoke



```
1 reference | spmcdonough, 14 hours ago | 3 changes
private String GenerateLine1()
{
    return "It is by caffeine alone that I set my mind in motion.\n";
}
```

- Occurs before get/retrieval request for property or field

All methods available

Perfect for tasks that involve selective execution

- Caching
- Retry support
- Threading

Multiple methods = great flexibility

Shared state benefits

- All activity happens in OnInvoke, so state is easy to track outside of intercepted method
- No need for MethodExecutionTags and what-not

Somewhat reduced clarity

- Downside of everything in one method

IL cannot be optimized

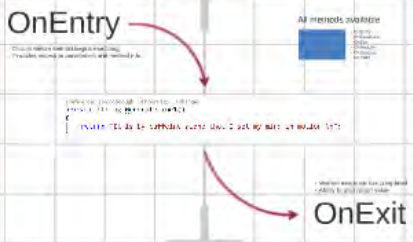
- All arguments are boxed/unboxed per invocation

Example #5

Aspect Types We're Going To Examine

How It Works | Potential Uses | Considerations

Method Boundary



Well-suited to repetitive tasks

- (ULS) Logging
- Performance profiling
- Tracing
- Exception handling*

Aspect methods are statically scoped

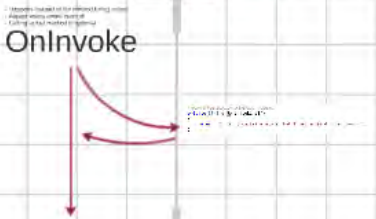
- Only one instance of each method services all requests from implementing types
- Sidestep this with either the MethodExecutionTag or by implementing IInstanceScopeAspect

IL can be optimized by PostSharp*

- Arguments selectively copied boxed/unboxed

Multiple methods = great flexibility

Method Interception



Perfect for tasks that involve selective execution

- Caching
- Retry support
- Threading

Shared state benefits

- All activity happens in OnInvoke, so state is easy to track outside of intercepted method
- No need for MethodExecutionTags and what-not

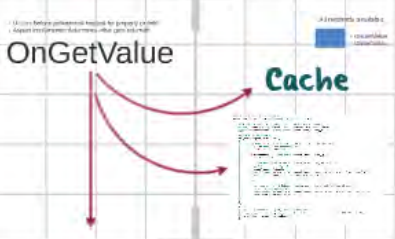
Somewhat reduced clarity

- Downside of everything in one method

IL cannot be optimized

- All arguments are boxed/unboxed per invocation

Location Interception



Similar to method interception, but more granular

- Validation
- Filtering
- Change tracking & notification
- Lazy loading & initialization

Same basic set of considerations as method interception aspects

- Again, similar in operation - just narrower scope

Works for properties and fields

- Including auto-properties

Interception

Location

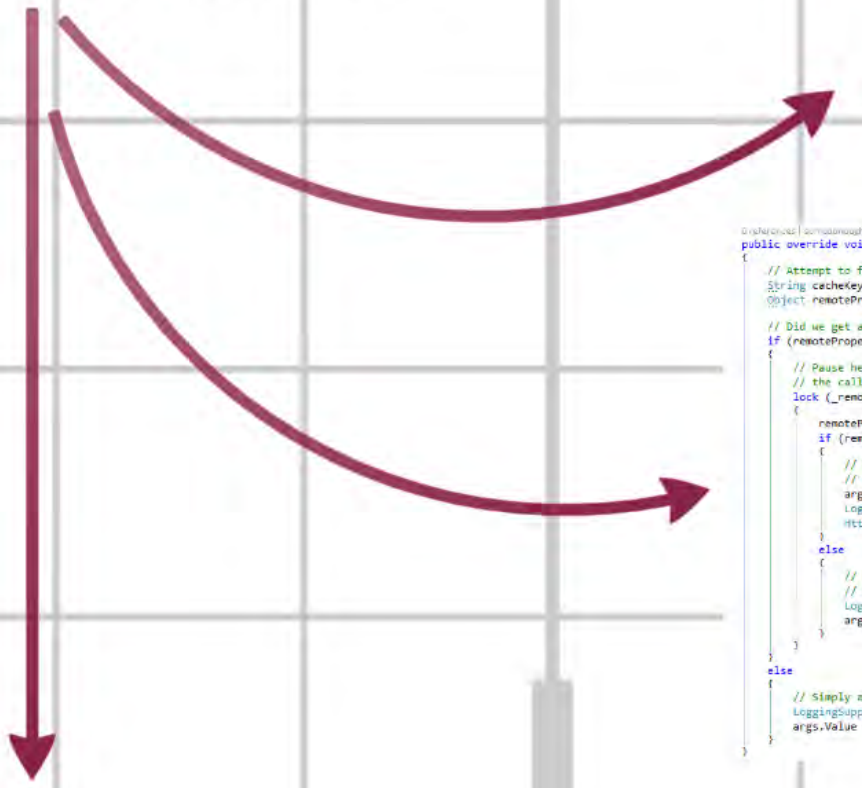
Interception

- Occurs before get/retrieval request for property or field
- Aspect implementer determines what gets returned

OnGetValue

All methods available

- OnGetValue
- OnSetValue



Cache

```
0 references | 0 changes | 0 authors, 11 hours ago | 1 change
public override void OnGetValue(LocationInterceptionArgs args)
{
    // Attempt to fetch the desired property from the ASP.NET Cache.
    String cacheKey = String.Format(CACHE_KEY_TEMPLATE, args.LocationName);
    Object remotePropertyValue = HttpContext.Current.Cache[cacheKey];

    // Did we get anything back?
    if (remotePropertyValue == null)
    {
        // Pause here by locking to ensure that only one caller actually makes
        // the call to retrieve the property value.
        lock (_remotePropertyLockObject)
        {
            remotePropertyValue = HttpContext.Current.Cache[cacheKey];
            if (remotePropertyValue == null)
            {
                // The property value isn't available in the Cache, so we need to
                // fetch it, store it, and pass it back.
                args.ProceedGetValue();
                LoggingSupport.WriteToLog(args.LocationName + " property value fetched from source.");
                HttpContext.Current.Cache.Insert(cacheKey, args.Value);
            }
            else
            {
                // Property wasn't initially in cache, but another thread (in ahead of the
                // current one) populated it.
                LoggingSupport.WriteToLog(args.LocationName + " property value fetched from ASP.NET Cache.");
                args.Value = remotePropertyValue;
            }
        }
    }
    else
    {
        // Simply assign the property value from the Cache.
        LoggingSupport.WriteToLog(args.LocationName + " property value fetched from ASP.NET Cache.");
        args.Value = remotePropertyValue;
    }
}
```

- Occurs before get/retrieval request for property or field
- Aspect implementer determines what gets returned

OnGetValue

All methods available

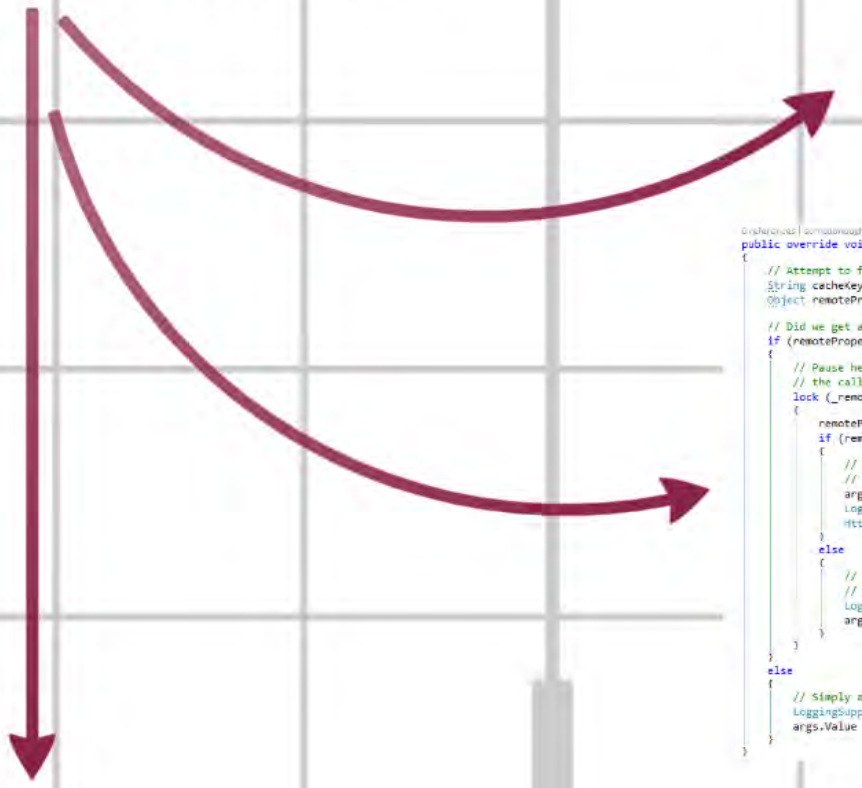
- OnGetValue
- OnSetValue

- Occurs before get/retrieval request for property or field
- Aspect implementer determines what gets returned

OnGetValue

All methods available

- OnGetValue
- OnSetValue



Cache

```
0 references | 0 changes | 0 authors, 11 hours ago | 1 change
public override void OnGetValue(LocationInterceptionArgs args)
{
    // Attempt to fetch the desired property from the ASP.NET Cache.
    String cacheKey = String.Format(CACHE_KEY_TEMPLATE, args.LocationName);
    Object remotePropertyValue = HttpContext.Current.Cache[cacheKey];

    // Did we get anything back?
    if (remotePropertyValue == null)
    {
        // Pause here by locking to ensure that only one caller actually makes
        // the call to retrieve the property value.
        lock (_remotePropertyLockObject)
        {
            remotePropertyValue = HttpContext.Current.Cache[cacheKey];
            if (remotePropertyValue == null)
            {
                // The property value isn't available in the Cache, so we need to
                // fetch it, store it, and pass it back.
                args.ProceedGetValue();
                LoggingSupport.WriteToLog(args.LocationName + " property value fetched from source.");
                HttpContext.Current.Cache.Insert(cacheKey, args.Value);
            }
            else
            {
                // Property wasn't initially in cache, but another thread (in ahead of the
                // current one) populated it.
                LoggingSupport.WriteToLog(args.LocationName + " property value fetched from ASP.NET Cache.");
                args.Value = remotePropertyValue;
            }
        }
    }
    else
    {
        // Simply assign the property value from the Cache.
        LoggingSupport.WriteToLog(args.LocationName + " property value fetched from ASP.NET Cache.");
        args.Value = remotePropertyValue;
    }
}
```

- Threading

Similar to method interception, but more granular

- Validation
- Filtering
- Change tracking & notification
- Lazy loading & initialization

- All arguments are boxed/unboxed per invocation

Same basic set of considerations as method interception aspects

- Again, similar in operation - just narrower scope

Works for properties and fields

- Including auto-properties

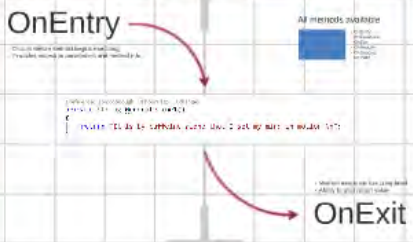
Example #6


bunker tuneage
Search...
home
mixes
about
bunker tuneage 2013 mixes are here!

Aspect Types We're Going To Examine

How It Works | Potential Uses | Considerations

Method Boundary



Well-suited to repetitive tasks

- (ULS) Logging
- Performance profiling
- Tracing
- Exception handling*

Aspect methods are statically scoped

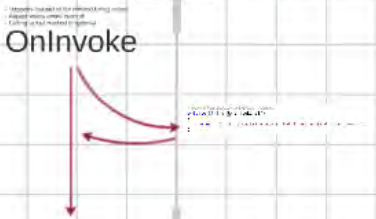
- Only one instance of each method services all requests from implementing types
- Sidestep this with either the MethodExecutionTag or by implementing IInstanceScopeAspect

IL can be optimized by PostSharp*

- Arguments selectively copied boxed/unboxed

Multiple methods = great flexibility

Method Interception



Perfect for tasks that involve selective execution

- Caching
- Retry support
- Threading

Shared state benefits

- All activity happens in OnInvoke, so state is easy to track outside of intercepted method
- No need for MethodExecutionTags and what-not

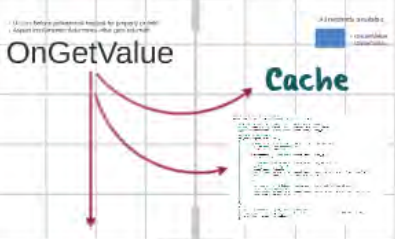
Somewhat reduced clarity

- Downside of everything in one method

IL cannot be optimized

- All arguments are boxed/unboxed per invocation

Location Interception



Similar to method interception, but more granular

- Validation
- Filtering
- Change tracking & notification
- Lazy loading & initialization

Same basic set of considerations as method interception aspects

- Again, similar in operation - just narrower scope

Works for properties and fields

- Including auto-properties

References

Walkthrough: Creating and Registering a Custom HTTP Module
http://msdn.microsoft.com/en-us/library/ms227673(v=vs.110).aspx

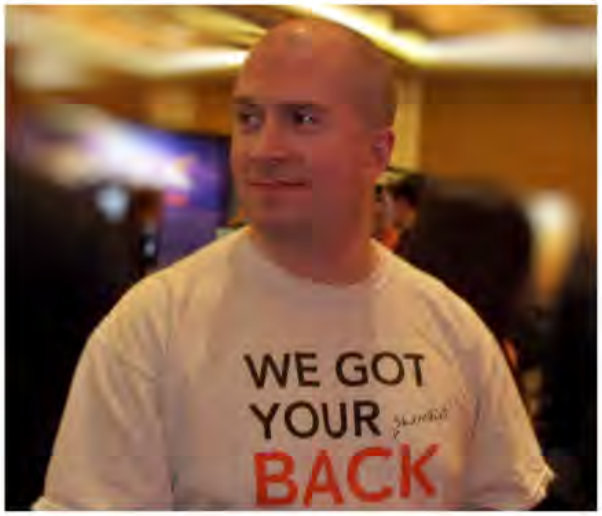
ASP.NET MVC 4 Custom Action Filters
http://www.asp.net/mvc/tutorials/hands-on-labs/aspnet-mvc-4-custom-action-filters

PostSharp in the Visual Studio Gallery
http://visualstudiogallery.msdn.microsoft.com/a058d5d3-e654-43f8-a308-c3bdfdd0be4a

PostSharp
http://www.postsharp.net

AOP in .NET
http://tinyurl.com/AOPinDotNet

Castle DynamicProxy
http://www.castleproject.org/projects/dynamicproxy/



Sean P. McDonough

SharePoint Gearhead, Developer, and Problem-Solver

My Company: bitstream FOUNDRY

Twitter: @spmcdonough

Blog: http://SharePointInterface.com

About: http://about.me/spmcdonough