

A Hitch-Hikers Guide to [ASPeKT]

Based on a true story

Written and Presented by Peter Lorimer



The Journey

Chapter 1: Introduction

Chapter 2: A walk down memory lane

Chapter 3: Trying to simplify, something that isn't simple.

Chapter 4: Words of advice

Chapter 5: Conclusion

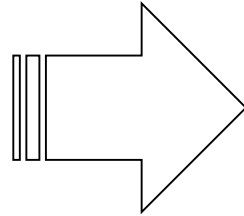


A Brief Introduction

- Peter Lorimer
- I am now 30!
- Software scientist
- Climbing enthusiast
- Outdoor idealist
- Wannabe chef
- Still a Tacoma owner



Recapping AOP



Explicit vs. Implicit Contracts

What is the explicit contract; and what is the implicit contract?

```
// Adds value1 to value2 and returns the result  
public Int32 Add(Int32 value1, Int32 value2);
```



Benefits of AOP

- Modularize functionality
- Declutter code
- Reduce signal-to-noise ratio



A Quick Example

```
static class AccountManager
{
    static List<Account> Accounts { get; } = new List<Account>();

    public static void CreateAccount(string ownerName)
    {
        if (ownerName == null)
            throw new ArgumentNullException("ownerName");

        Console.WriteLine($"Entered CreateAccount({ownerName})");

        if (!AuthorizationManager.IsAuthorized(WindowsIdentity.GetCurrent(), Operation.Create |
Operation.Account))
            throw new NotAuthorizedException("CreateAccount");

        Accounts.Add(new Account(ownerName));

        Console.WriteLine("Exiting CreateAccount()");
    }
}
```



Same Example, On [ASPeKT]

```
static class AccountManager
{
    static List<Account> Accounts { get; } = new List<Account>();

    [NotNull("ownerName")]
    [ConsoleLogged]
    [IsAuthorized(Operation.Create | Operation.Account)]
    public static void CreateAccount(string ownerName)
    {
        Accounts.Add(new Account(ownerName));
    }
}
```



Components of AOP

- Cross Cutting Concerns
 - Logging
 - Auditing
 - Code contracts
 - Error handling
- Advice
- Join Point
- Point-cut
- Aspect



An AOP Model

- As per Wikipedia a Join Point Model is defined by 3 things
 1. When the Advice can run
 2. A way to specify Join Points
 3. A means of specifying code to run at the Join Point



Approaches to AOP

- Source processing i.e. Source weaving
- Runtime interpretation i.e. Run-time weaving
- Post processing i.e. Compile-time weaving
- Combination



In the Beginning

It always starts with beer; because no good story ever started with a salad.



C'Mon Gary...

“To avoid leaking dependencies between layers, it is good practice to wrap an exception thrown at a lower layer in a new exception for this layer.”

```
public void AddTransactionToAccount(string uniqueAccountName, decimal transactionAmount)
{
    var account = repository.GetByName(uniqueAccountName);
    if(account != null)
    {
        try
        {
            account.AddTransaction(transactionAmount);
        }
        catch (DomainException)
        {
            throw new ServiceException();
        }
    }
}
```



C'Mon Gary...

You just told me about this neat thing called AOP!

```
[TranslateException(typeof(DomainException), typeof(ServiceException))]  
public void AddTransactionToAccount(string uniqueAccountName, decimal transactionAmount)  
{  
    var account = repository.GetByName(uniqueAccountName);  
    if(account != null)  
    {  
        account.AddTransaction(transactionAmount);  
    }  
}
```



Where Do I Even Start?



Pfft... I can do that.

Sources of Inspiration:

- PostSharp
- Fody



Setting Goals.

Things I wanted to achieve:

- Compile time IL re-writing
- Function level join-points
- Attribute based means to specify advice execution

```
[TranslateException(typeof(DomainException), typeof(ServiceException))]  
public void AddTransactionToAccount(string uniqueAccountName, decimal transactionAmount)  
{  
    var account = repository.GetByName(uniqueAccountName);  
    if(account != null)  
    {  
        account.AddTransaction(transactionAmount);  
    }  
}
```



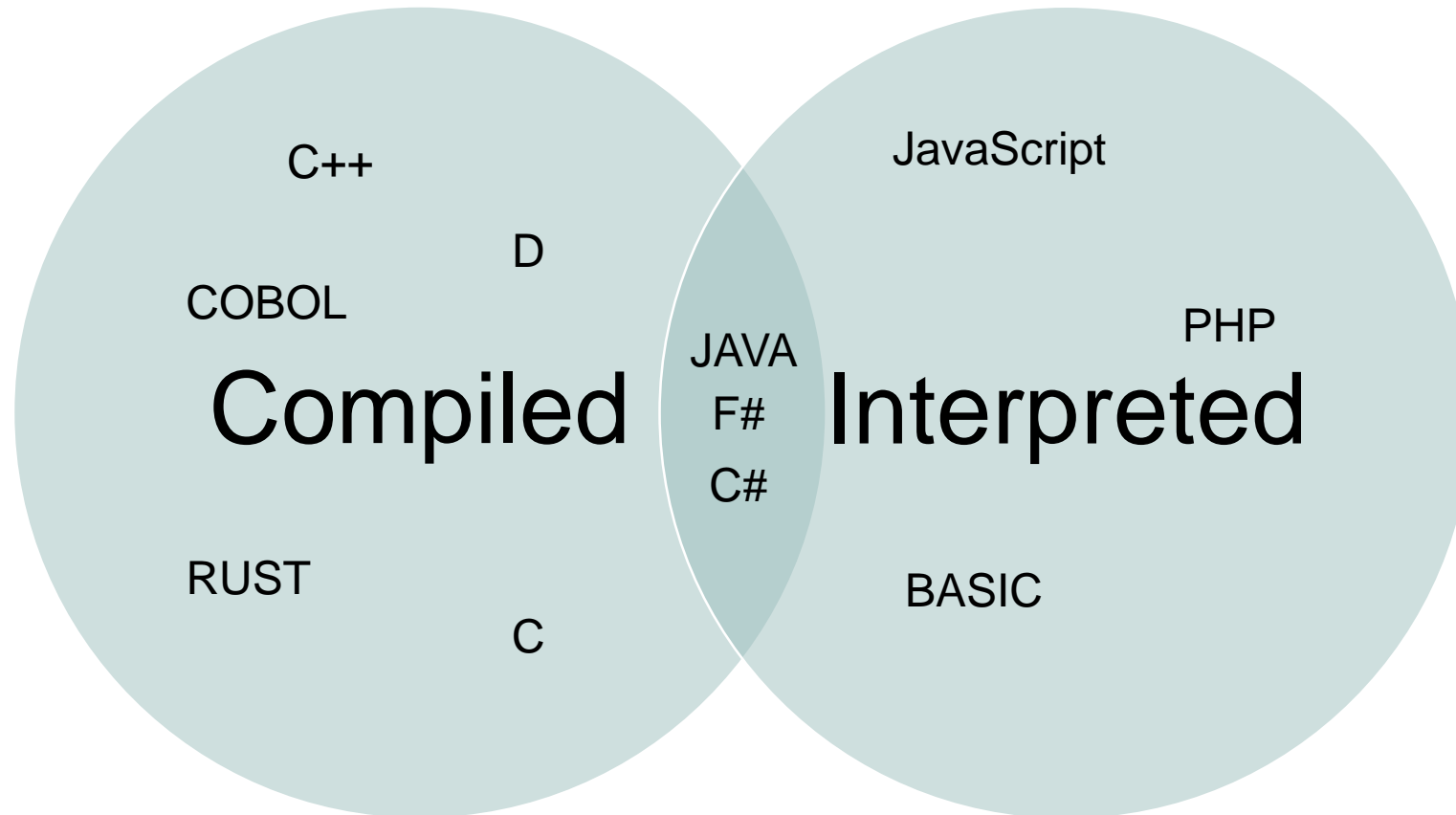
That's so IL, Bro.

To do compile time IL re-writing, I needed to know a couple things first.

- What is IL?
- What actually is IL weaving? And how do I do this?



Compiled vs. Interpreted

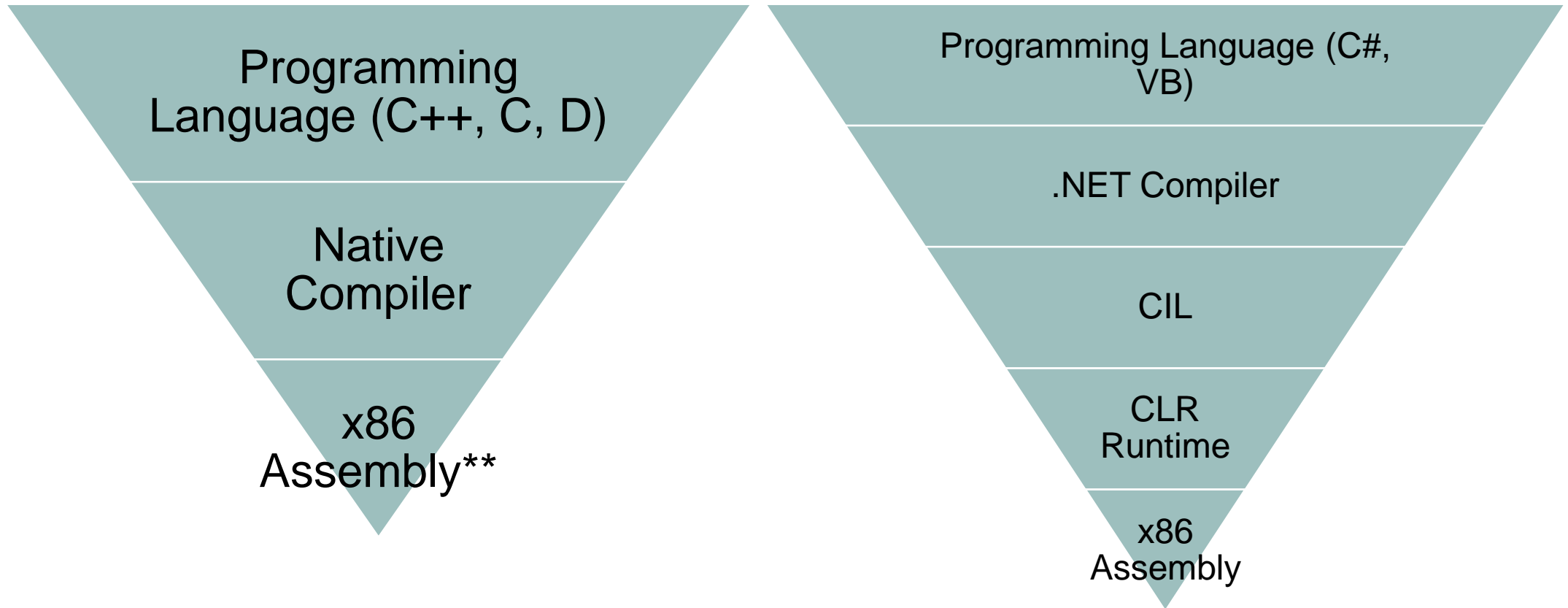


Trivial Pursuit

If Sun Microsystems wouldn't have sued Microsoft, we wouldn't have our beloved C#.



Native vs. Managed Languages



.NET CLR

- Garbage Collected!
- Multi-Language support
- Built-in exception handling & security
- Stack based machine
- Most Important for this talk: Interprets MSIL (CIL)



Simple IL

```
public Int32 Add(Int32 value1, Int32 value2)
{
    return value1 + value2;
}
```

Resulting IL

```
Ldarg.1
Ldarg.2
Add
Ret
```

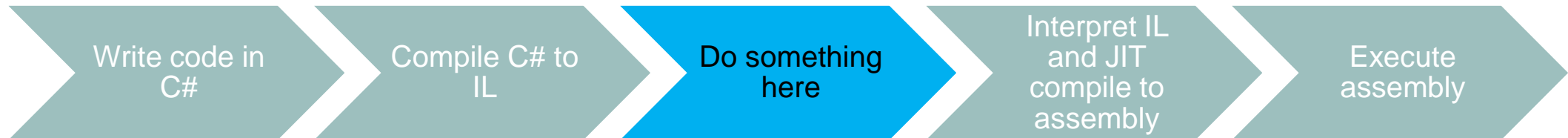


The Workflow

Standard C# .NET Execution Workflow

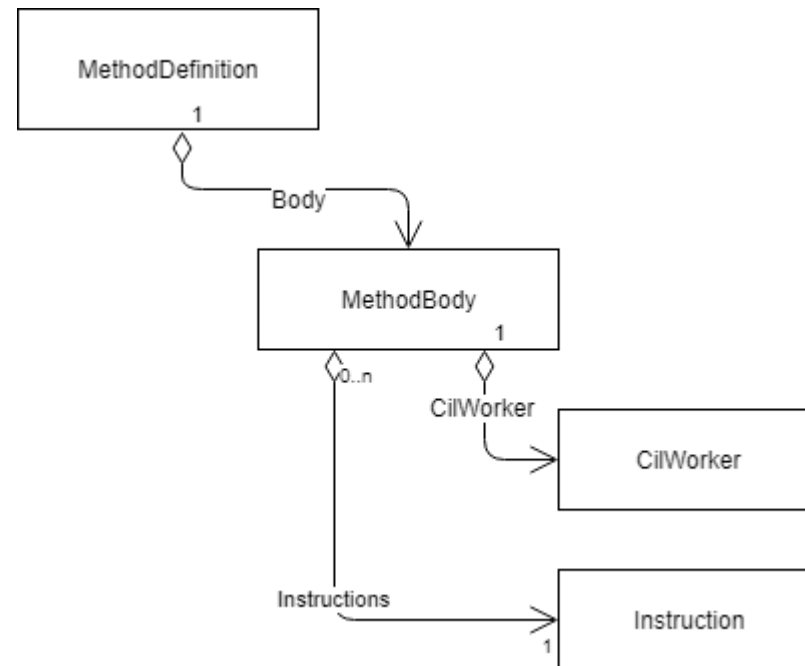


[ASPeKT] C# .NET Execution Workflow



Drawing the Line: Enter Mono.Cecil

I am not a masochist! I don't write everything from scratch.



Reflection:

Just because you can, doesn't mean you should. Just because you shouldn't, doesn't mean you can't.

What is reflection?

- Ability for a program to inspect and introspect its (or other programs) shape, at runtime.
- Ability to do some really, really, really naughty things.



??? Profit!!!

Step 1: Open assembly.

Step 2: Use Mono.Cecil + reflection to find attributes of my Aspect type.

Step 3: ???

Step 4: Profit.



A Mono.Cecil Demo

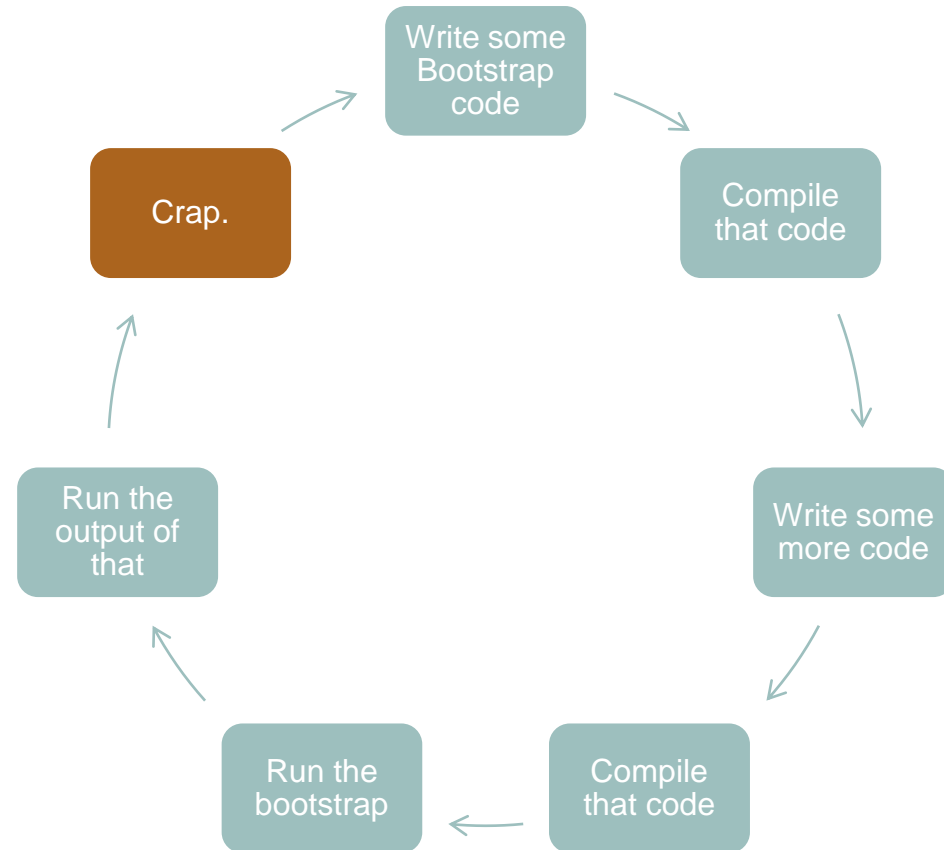
- Explanation of method re-writing



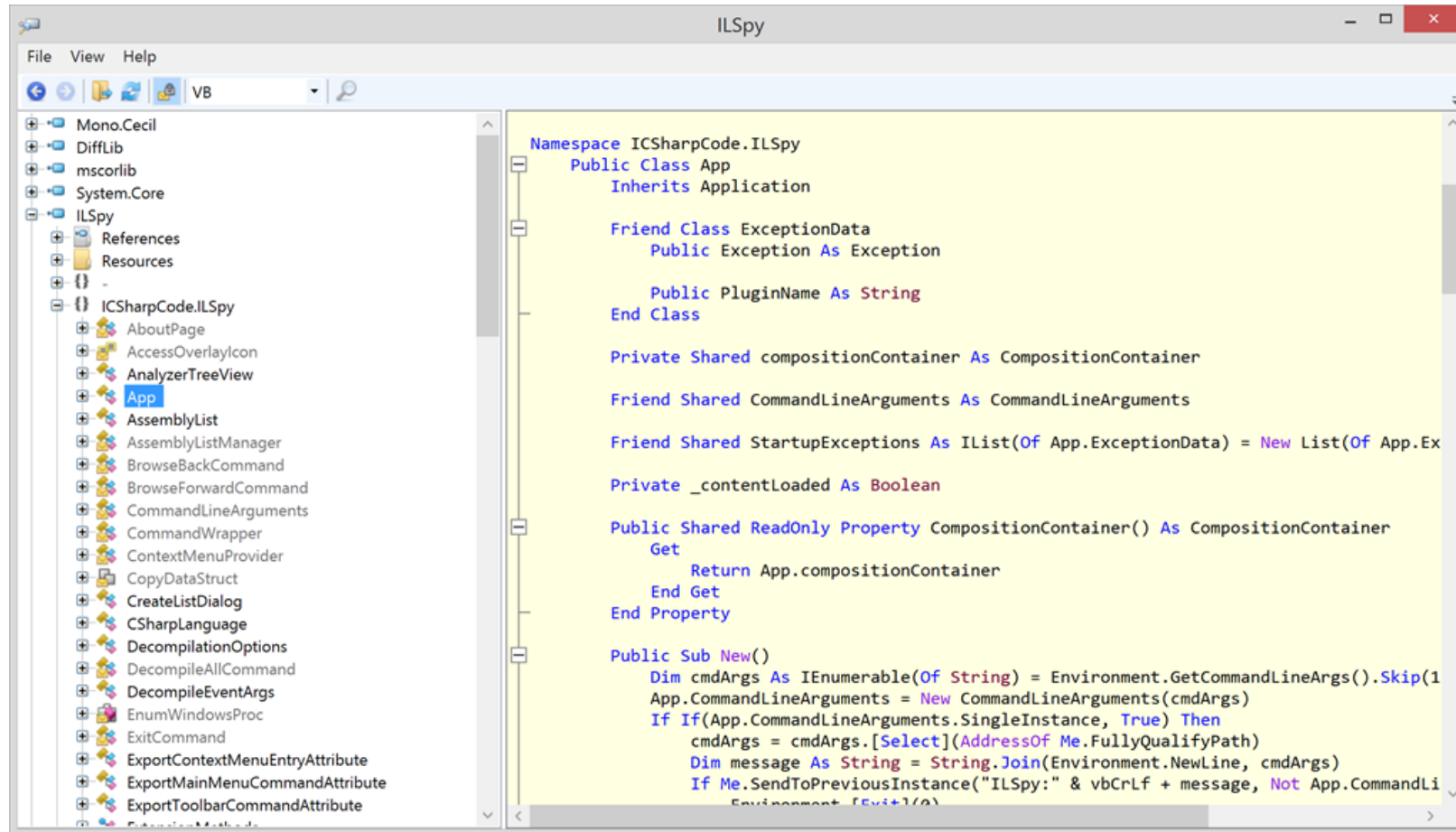
Great. I have an idea of what I want.



An Ineffective Dev Loop



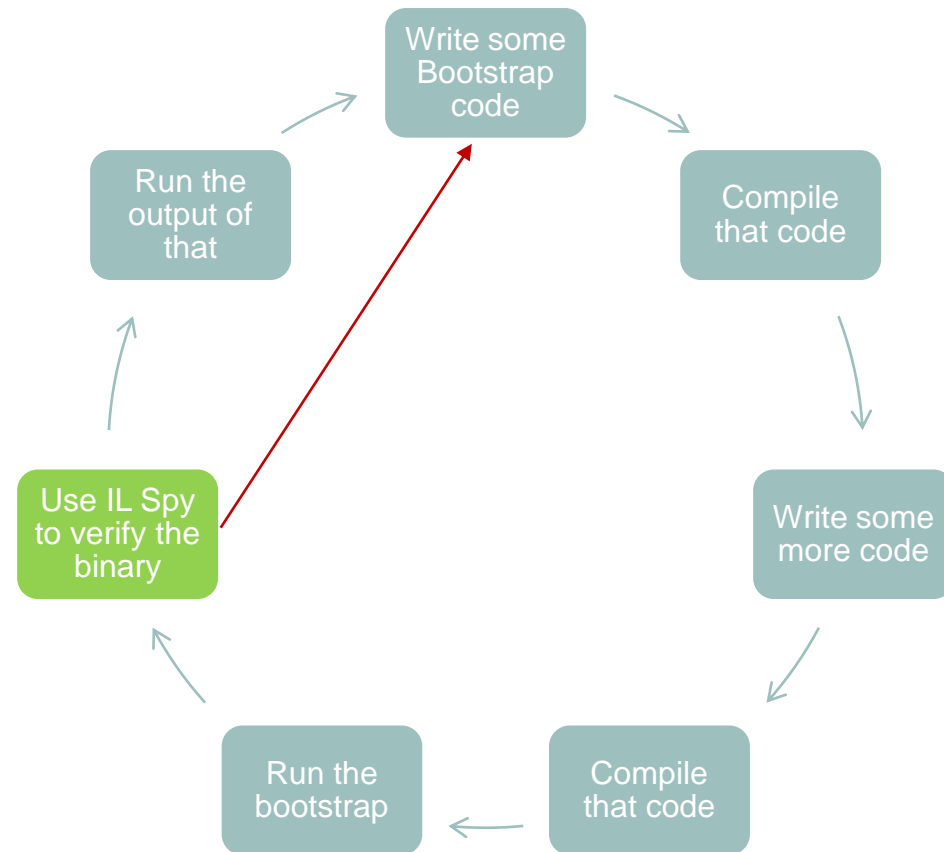
ILSpy – With My Little Eye



IL Spy Demo



An Slightly More Effective Dev Loop

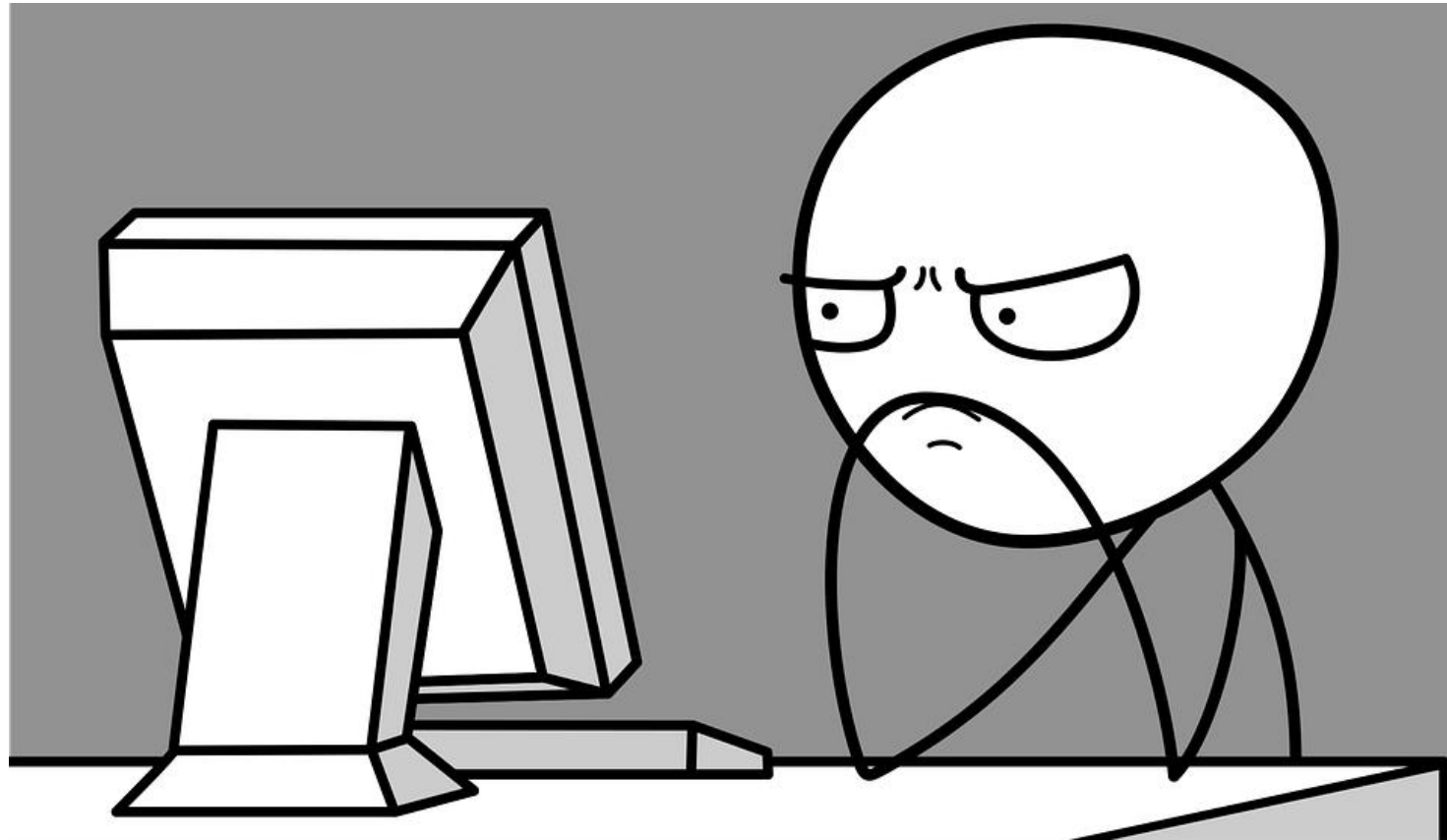


Chasing the Alien

That's awesome! Wait, I **must** be doing something wrong.



Document? Nope.



Creating a Package by Hand

- Lesson learned – It's hard. Don't do it.

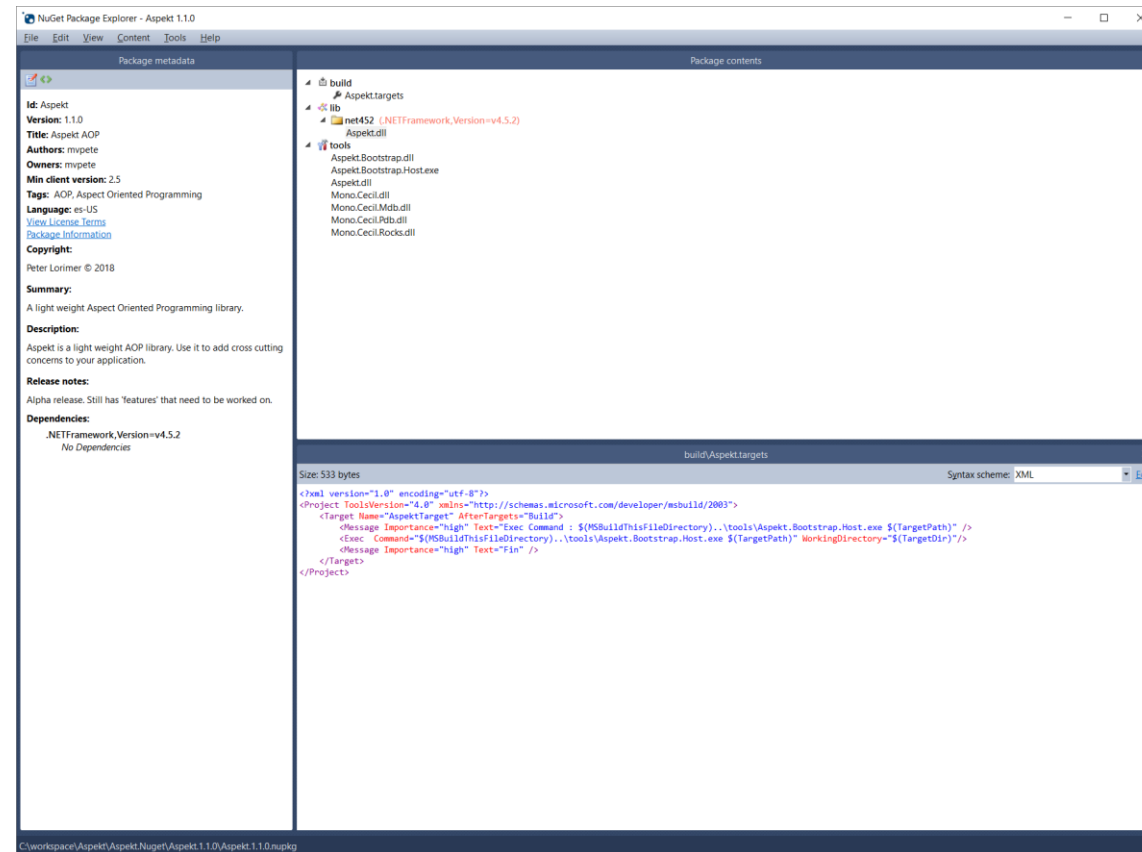


Components of NuGet Package



Folder	Description	Action upon package install
(root)	Location for readme.txt	Visual Studio displays a readme.txt file in the package root when the package is installed.
lib/{tfm}	Assembly (.dll), documentation (.xml), and symbol (.pdb) files for the given Target Framework Moniker (TFM)	Assemblies are added as references; .xml and .pdb copied into project folders. See Supporting multiple target frameworks for creating framework target-specific sub-folders.
runtime	Architecture-specific assembly (.dll), symbol (.pdb), and native resource (.pri) files	Assemblies are added as references; other files are copied into project folders. See Supporting multiple target frameworks .
content	Arbitrary files	Contents are copied to the project root. Think of the content folder as the root of the target application that ultimately consumes the package. To have the package add an image in the application's <i>/images</i> folder, place it in the package's <i>content/images</i> folder.
build	MSBuild .targets and .props files	Automatically inserted into the project file or project.lock.json (NuGet 3.x+).
tools	Powershell scripts and programs accessible from the Package Manager Console	The tools folder is added to the PATH environment variable for the Package Manager Console only (Specifically, <i>not</i> to the PATH as set for MSBuild when building the project).



An Easier Way to Package



Publishing

 [Packages](#) [Upload](#) [Statistics](#) [Documentation](#) [Downloads](#) [Blog](#)  mvpete

[Home](#) > [Packages](#) > Upload

Your package file will be uploaded and hosted on the NuGet Gallery server (<https://www.nuget.org>).

▼ Upload

Browse or Drop files to select a package

Browse...



Semantic Versioning 2.0

v1.0.0



Semantic Versioning 2.0

v1.1.0



Semantic Versioning 2.0

v2.0.0



Semantic Versioning 2.0

v2.0.1



The Elusive DevOps

- AppVeyor vs. Travis CI
- Test Driven Development



Using GitHub.

- Use GitHub to document and manage code. It's good.



Renovations

- What's next for [ASPeKT]?
 - Aspekt.Contracts Invariant
 - Refactor of bootstrap
 - NLog logging target
 - Broader set of join-points
 - More control in Aspects



Lessons Learned

1. Writing a consumable library from scratch is a lot of work.
2. Writing a *stable* consumable library from scratch is substantially more work.
3. I wish I would've written more down.
4. Use version control.
5. Think about how to version your software.
6. Make it easy for people to consume.
 1. Package it.
 2. Document it.
7. Go public with it. People love to learn.



Questions??



Thank you for attending!

Please feel free to ask any more questions you have, and take a card!



Uh oh!

- If you're seeing this slide, I ran out of slides and still had time...
- Here's a list of interesting topics to talk about:
 - Other uses for AOP?
 - Questions about other people's projects?
 - What is everyone's favourite dish?
 - How much wood, could a woodchuck, really chuck, if a woodchuck, could chuck, wood?



References

- Some info on IL
 - https://en.wikipedia.org/wiki/List_of_CIL_instructions
 - <https://www.codeproject.com/Articles/3778/Introduction-to-IL-Assembly-Language>
- The History of .NET
 - https://www.youtube.com/watch?v=FFCn_z7dn_A
- [ASPeKT] Project
 - <https://github.com/mvpete/aspekt>
- Semantic Versioning 2.0
 - <https://semver.org>

