



Produce Cleaner Code with Aspect-Oriented Programming

An Introduction to Aspect-Oriented Programming in Microsoft .NET.

Gaël Fraiteur

gael@sharpcrafters.com

[@gfraiteur](#)

<http://www.sharpcrafters.com/>



Full disclosure: I make a living
from selling PostSharp.

Agenda

1. The Problem with Conventional Programming
2. Defining AOP
3. Demo 1 – Example: Rich Exception Logging
4. Convince Your Boss
5. Demo 2 – Example: Performance Instrumentation and Caching
6. Comparing AOP Frameworks
7. Demo 3 –Advanced Aspects

Part 1

The Problem with Conventional Programming

In the beginning
there was nothing.

```
public class CustomerProcesses  
{  
}
```

Customer said: let there be business value.

```
public class CustomerProcesses
{
    public bool ReserveBook( int bookId, int customerId )
    {
        Book book = Book.GetById(bookId);
        Customer customer = Customer.GetById(customerId);

        if ( book.BorrowedTo != null || book.ReservedTo != null)
        {
            return false;
        }

        book.ReservedTo = customer;
        return true;
    }
}
```

And there was business code.

```

internal class CustomerProcesses
{
    private static readonly TraceSource trace =
        new TraceSource(typeof(CustomerProcesses).FullName);

    public bool ReserveBook(int bookId, int customerId)
    {
        trace.TraceInformation(
            "Entering CustomerProcesses.ReserveBook( bookId = {0}, customerId = {1} )",
            bookId, customerId);

        try
        {
            Book book = Book.GetById(bookId);
            Customer customer = Customer.GetById(customerId);

            bool returnValue;
            if (book.BorrowedTo != null || book.ReservedTo != null)
            {
                returnValue = false;
                goto exit;
            }

            book.ReservedTo = customer;
            returnValue = true;

        exit:
            trace.TraceInformation(
                "Leaving CustomerProcesses.ReserveBook( bookId = {0}, customerId = {1} ) "+
                "with return value {2}", bookId, customerId);

            return returnValue;
        }
        catch (Exception e)
        {
            trace.TraceEvent(TraceEventType.Error, 0,
                "Exception: CustomerProcesses.ReserveBook(
                bookId = {0}, customerId = {1} ) failed : {2}",
                bookId, customerId, e.Message);

            throw;
        }
    }
}

```

Testers said:
Let there be
logging

And there was
logging code.

```

internal class CustomerProcesses
{
    private static readonly TraceSource trace =
        new TraceSource(typeof(CustomerProcesses).FullName);

    public bool ReserveBook(int bookId, int customerId)
    {
        if (bookId <= 0) throw new ArgumentOutOfRangeException("bookId");
        if (customerId <= 0) throw new ArgumentOutOfRangeException("customerId");

        trace.TraceInformation(
            "Entering CustomerProcesses.ReserveBook( bookId = {0}, customerId = {1} )",
            bookId, customerId);

        try
        {
            Book book = Book.GetById(bookId);
            Customer customer = Customer.GetById(customerId);

            bool returnValue;
            if (book.BorrowedTo != null || book.ReservedTo != null)
            {
                returnValue = false;
                goto exit;
            }

            book.BorrowedTo = customer;
            returnValue = true;

        exit:
            trace.TraceInformation(
                "Leaving CustomerProcesses.ReserveBook( bookId = {0}, customerId = {1} ) "+
                "with return value {2}", bookId, customerId);

            return returnValue;
        }
        catch (Exception e)
        {
            trace.TraceEvent(TraceEventType.Error, 0,
                "Exception: CustomerProcesses.ReserveBook(
                bookId = {0}, customerId = {1} ) failed : {2}",
                bookId, customerId, e.Message);

            throw;
        }
    }
}

```

Devs said:
Let there be
defensive
programming

Then
there was
precondition
checking code.

```

internal class CustomerProcesses
{
    private static readonly TraceSource trace =
        new TraceSource(typeof(CustomerProcesses).FullName);

    public bool ReserveBook(int bookId, int customerId)
    {
        if (bookId <= 0) throw new ArgumentOutOfRangeException("bookId");
        if (customerId <= 0) throw new ArgumentOutOfRangeException("customerId");

        trace.TraceInformation(
            "Entering CustomerProcesses.ReserveBook( bookId = {0}, customerId = {1} )",
            bookId, customerId);

        try
        {
            bool returnValue;

            for (int i = 0; ; i++)
            {
                try
                {
                    using ( var ts = new TransactionScope() )
                    {
                        Book book = Book.GetById( bookId );
                        Customer customer = Customer.GetById( customerId );

                        if ( book.BorrowedTo != null || book.ReservedTo != null )
                        {
                            returnValue = false;
                            goto exit;
                        }

                        book.ReservedTo = customer;
                        returnValue = true;
                    }
                }
            }
            catch (TransactionConflictException)
            {
                if (i < 3)
                    continue;
                else
                    throw;
            }
        }
    }
}

}

exit:
trace.TraceInformation(
    "Leaving CustomerProcesses.ReserveBook( bookId = {0}, customerId = {1} )
    with return value {2}",
    bookId, customerId);

return returnValue;
}
catch (Exception e)
{
    trace.TraceEvent(TraceEventType.Error, 0,
        "Exception: CustomerProcesses.ReserveBook(
        bookId = {0}, customerId = {1} ) failed : {2}",
        bookId, customerId, e.Message);

    throw;
}
}
}

```

Let there be safe concurrent execution.
 And there was transaction handling code.



YOHJI YAMAMOTO
customer



AKSEL BACHMEIER
architect

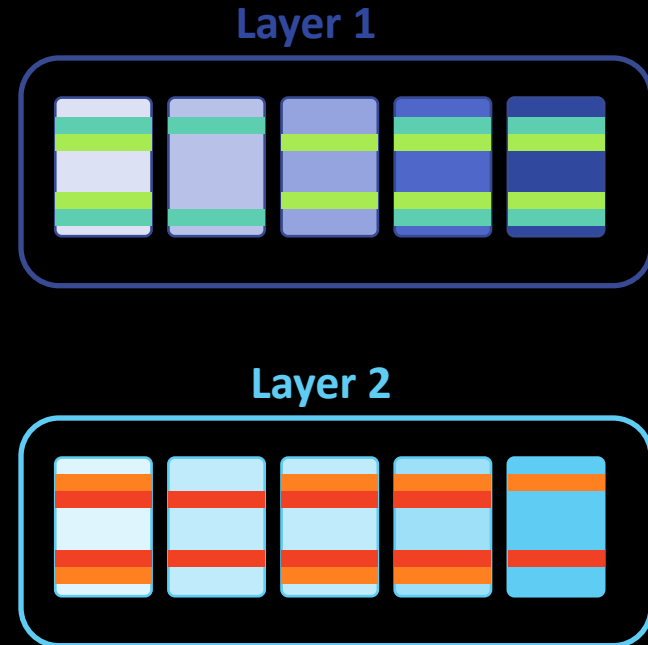
Prototype Code



Production Code

Why do we write ugly code?

- We want a nice separation of concerns
(assembly > namespace > class > method)
- OOP forces us to write crap!
 - **Code Scattering**
 - **Code Tangling**
 - **Code Coupling**



Non-Functional Requirements

- Security
- Exception Handling
- Tracing
- Monitoring
- Transaction
- Data Binding
- Thread Sync
- Caching
- Validation

Cross-Cutting Concerns

Encapsulating Cross-Cutting Concerns?

A woman in a black bikini and a white sash with the text "ASPECT-ORIENTED PROGRAMMING" is standing in a pool setting. The background is a blurred pool area with lights. The word "Aspects!" is written in large white letters over the image.

Aspects!

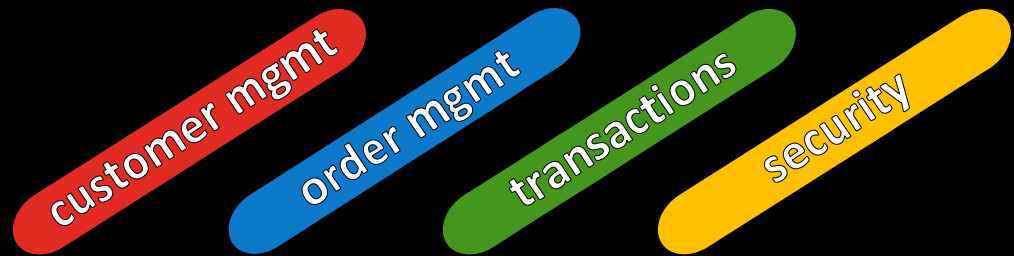
Section 2

Defining AOP

Problem Domain
**Cross-Cutting
Concerns**



Solution Domain
**Separation of
Concerns**

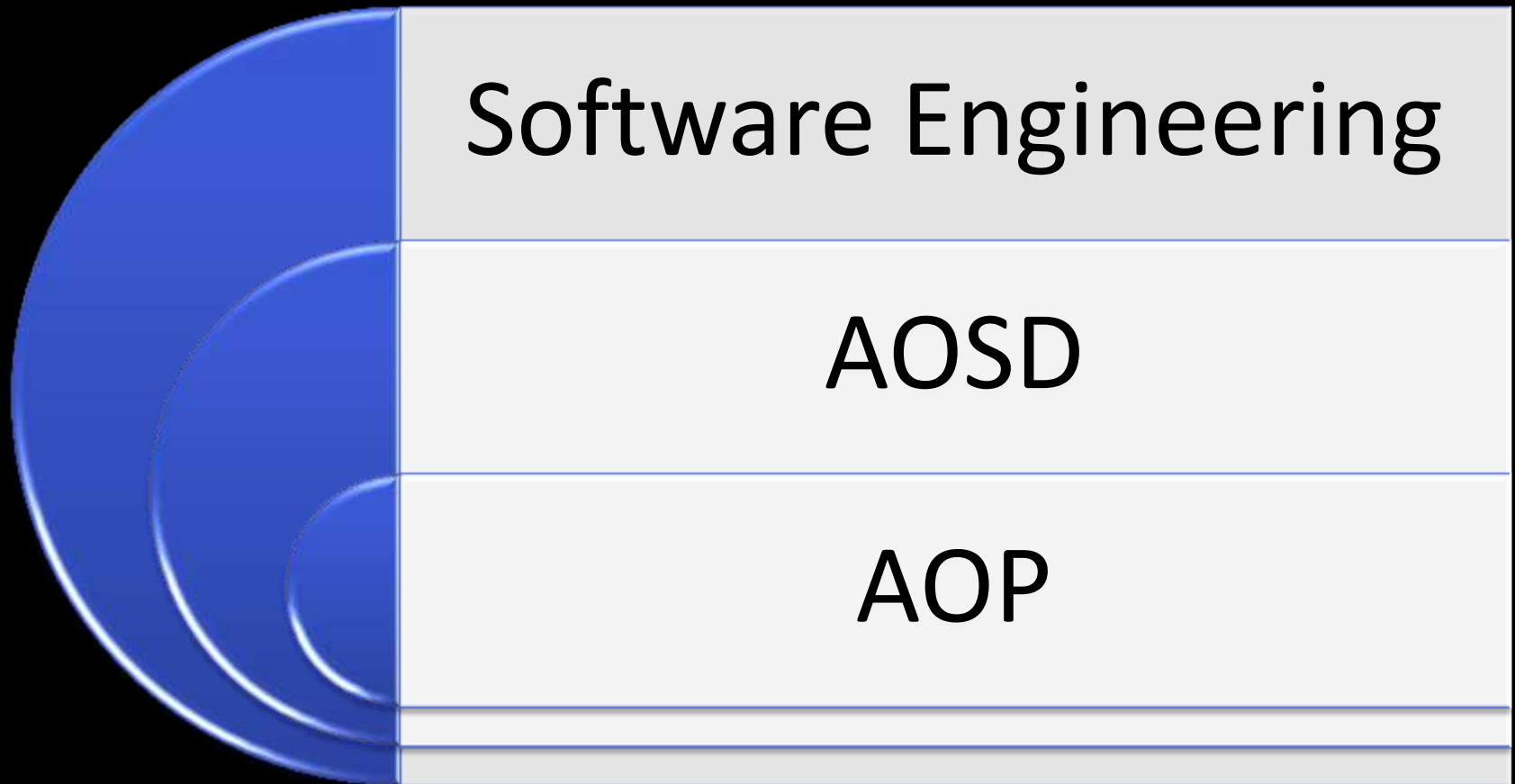


What is AOP?

An extension of (not an alternative to) OOP that addresses the issue of cross-cutting concerns by providing a mean to:

- **Encapsulate** cross-cutting concerns into **Aspects** = collection of transformations of code
- **Apply** aspects to elements of code

An engineering discipline



15 Years of AOP History

Research Years

Hype Years

Productivity Years

1994-1996

1997

2001

2003

2004

2007-2008

2009

2010

- First efforts on program transformation

- AOP coined by Gregor Kiczales (Xerox PARC)

- AspectJ published; First AOSD Conference

- AspectJ released to the Eclipse OSS community
- Spring Framework 1.0
- .NET 1.0

- Build up of Interface21, later SpringSource, around IoC and AOP
- Works Begins on PostSharp
- JBoss AOP
- WebSphere AOP
- AJDT
- SAP Enhancement Framework

- PostSharp 1.0
- PostSharp 1.5
- ALCOB (AOP for COBOL)

- SpringSource acquired by VMWare, \$400M

- PostSharp 2.0



Part 3 – Live Demo

Example:
Rich Exception Logging

How does PostSharp work?



1. Source



2. Compiler



3. PostSharp



4. Run Time



AOP is an *approach* to programming, not a technology.

Part 4

Convince Your Boss

The benefits of aspect-oriented programming


Improve teamwork



I understand provisioning processes better than anyone in this company,

UI/Business Developers
Use Aspects

n



I master multithreading better than anyone on earth

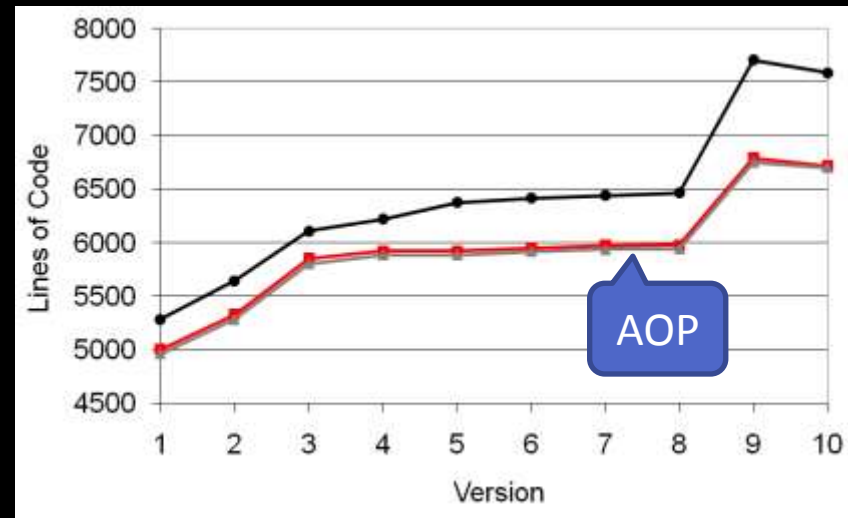
System Developers
Prepare Aspects

1

The benefits of aspect-oriented programming

Reduce development time

- Less code to write = fewer defects
- Less code to read = easier maintenance



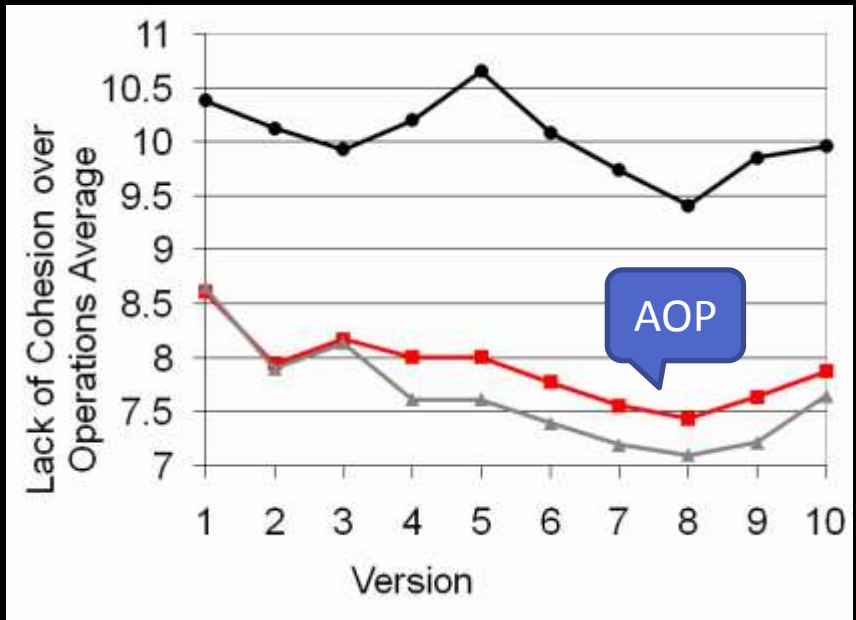
On the Impact of Aspectual Decompositions on Design Stability: An Empirical Study

Phil Greenwood, Thiago Bartolomei, Eduardo Figueiredo, Marcos Dosea, Alessandro Garcia, Nelio Cacho, Cláudio Sant'Anna, Sergio Soares, Paulo Borba, Uirá Kulesza, and Awaís Rashid
ECOOP 2007 – OBJECT-ORIENTED PROGRAMMING (Springer)

The benefits of aspect-oriented programming

Produce more maintainable apps

- Reduce code scattering
- Reduce code tangling
- Improve the open/closed principle



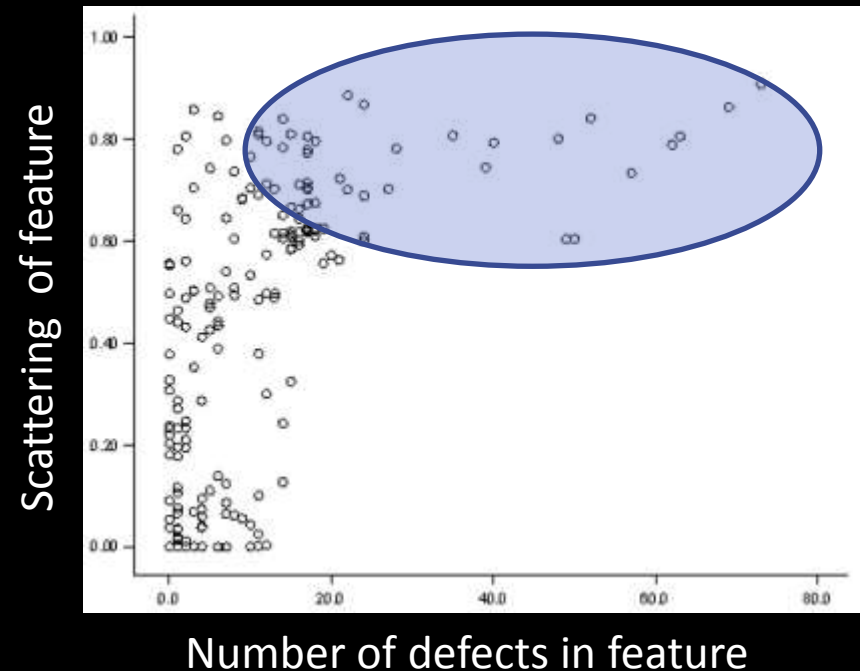
On the Impact of Aspectual Decompositions on Design Stability: An Empirical Study

Phil Greenwood, Thiago Bartolomei, Eduardo Figueiredo, Marcos Dosea, Alessandro Garcia, Nelio Cacho, Cláudio Sant'Anna, Sergio Soares, Paulo Borba, Uirá Kulesza, and Awaís Rashid
ECOOP 2007 – OBJECT-ORIENTED PROGRAMMING (Springer)

The benefits of aspect-oriented programming

Reduce the number of defects

- Scattering produces defects!
- Less code means less defects.
- Better division of labour means less defects



Do Crosscutting Concerns Cause Defects?

Marc Eaddy, Student Member, IEEE, Thomas Zimmermann, Student Member, IEEE, Kaitlin D. Sherwood, Vibhav Garg, Gail C. Murphy, Member, IEEE Computer Society, Nachiappan Nagappan, Member, IEEE, and Alfred V. Aho, Fellow, IEEE
IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 34, NO. 4, JULY/AUGUST 2008

The benefits of aspect-oriented programming

Risks and costs

Risks and Costs

- Staff lacks knowledge and experience
- Vendors are small (except in Java)

Mitigation

- Start with non-essential aspects
- Start with non-critical applications
- After >6 month of experience, scale up



Part 5 – Live Demo

Example:
**Performance Instrumentation
and Caching**

Part 6

Comparing Aspect Frameworks

Comparing Aspect Frameworks

Static vs Dynamic AOP

PostSharp
AspectJ + AJDT
Afterthought
AspectDNG
Aspect.NET,
Compose*,

Gripper-LOOM.NET,
LinFu AOP,
NSurgeon,
Phx.Morph/Wicca,
PostCrap,
SheepAOP,
Snap

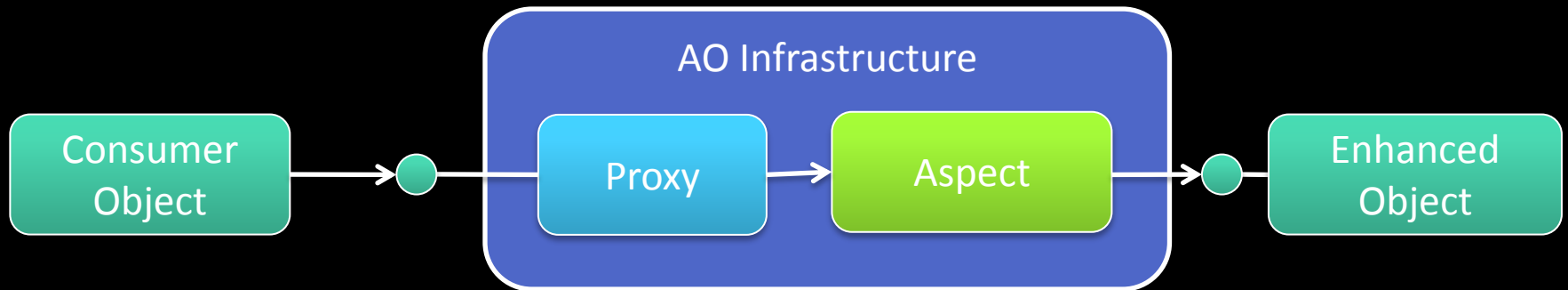
Spring AOP
Spring.NET
Castle
MS Unity/PIAB
LinFu
(MEF)
(WCF)
(ASP.NET MVC)



Build-Time

Run-Time

Proxy-Based AOP



Proxies: JIT Emitted (interface or subclassing), Transparent

Comparing Aspect Frameworks

Static vs Dynamic AOP

PostSharp
AspectJ + AJDT
Afterthought
AspectDNG
Aspect.NET,
Compose*,

Gripper-LOOM.NET,
LinFu AOP,
NSurgeon,
Phx.Morph/Wicca,
PostCrap,
SheepAOP,
Snap

Spring AOP
Spring.NET
Castle
MS Unity/PIAB
LinFu
(MEF)
(WCF)
(ASP.NET MVC)



Build-Time:

Very Expressive
Robust Model
IDE Integration
Not Invasive
Static

Run-Time:

Less Expressive
Brittle Model
No IDE Integration
Invasive
Dynamic

Comparing Aspect Frameworks

My Own Summary

- Aspects on Service Boundaries:
use your favorite application framework.
- Aspects on Ordinary and GUI Objects: use build-time.
- You can mix build-time and run-time frameworks.

Part 7 – Live Demo

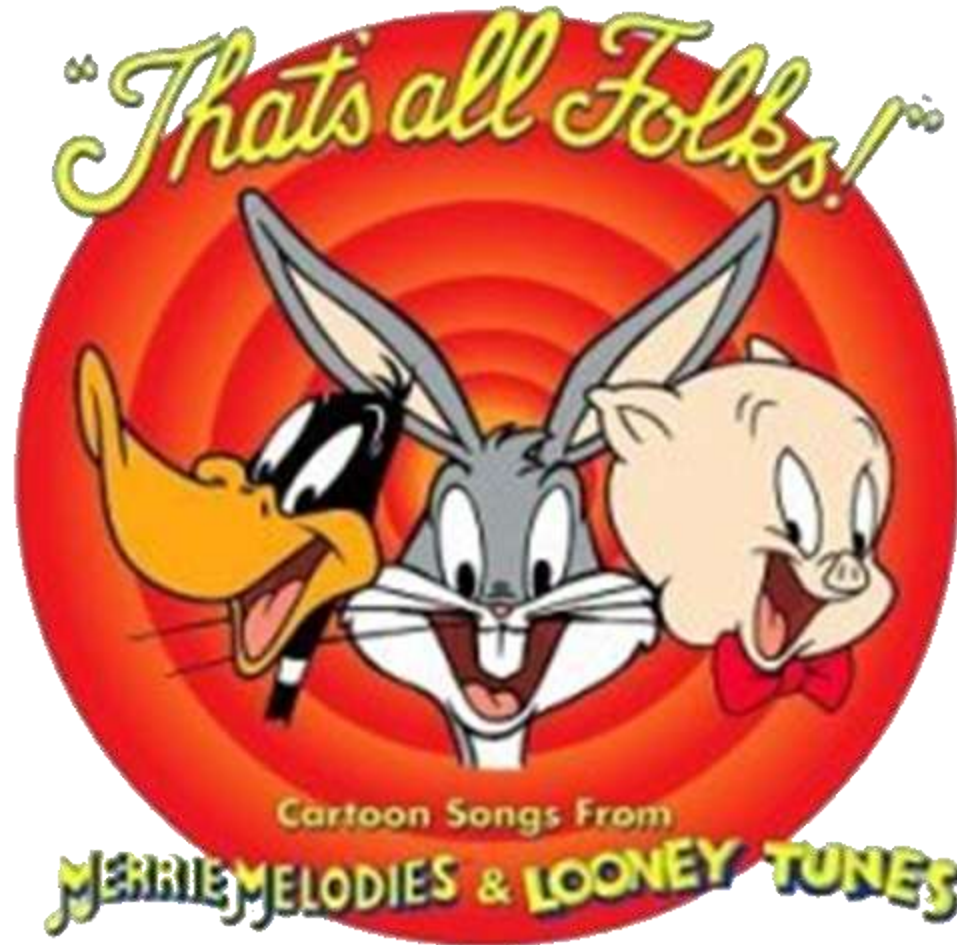
More Aspects

Summary

We need Aspects!

**We have
great frameworks!**

Help spread the word.



Questions

gael@sharpcrafters.com
@gfraiteur
<http://www.sharpcrafters.com/>