AOP

Aspect-Oriented Programming

How to implement in .NET

MuhaMad Fathi
www.MiMFa.net

CONTENT

Introduction

ROP: Reflection-Oriented Programming (1982)

Reflection can be used for observing and modifying program execution at runtime.

SOP: Subject-Oriented Programming (1993)

implementation of viewing subject instead of the viewing object

AOP: Aspect-Oriented Programming (1997)

What is AOP?

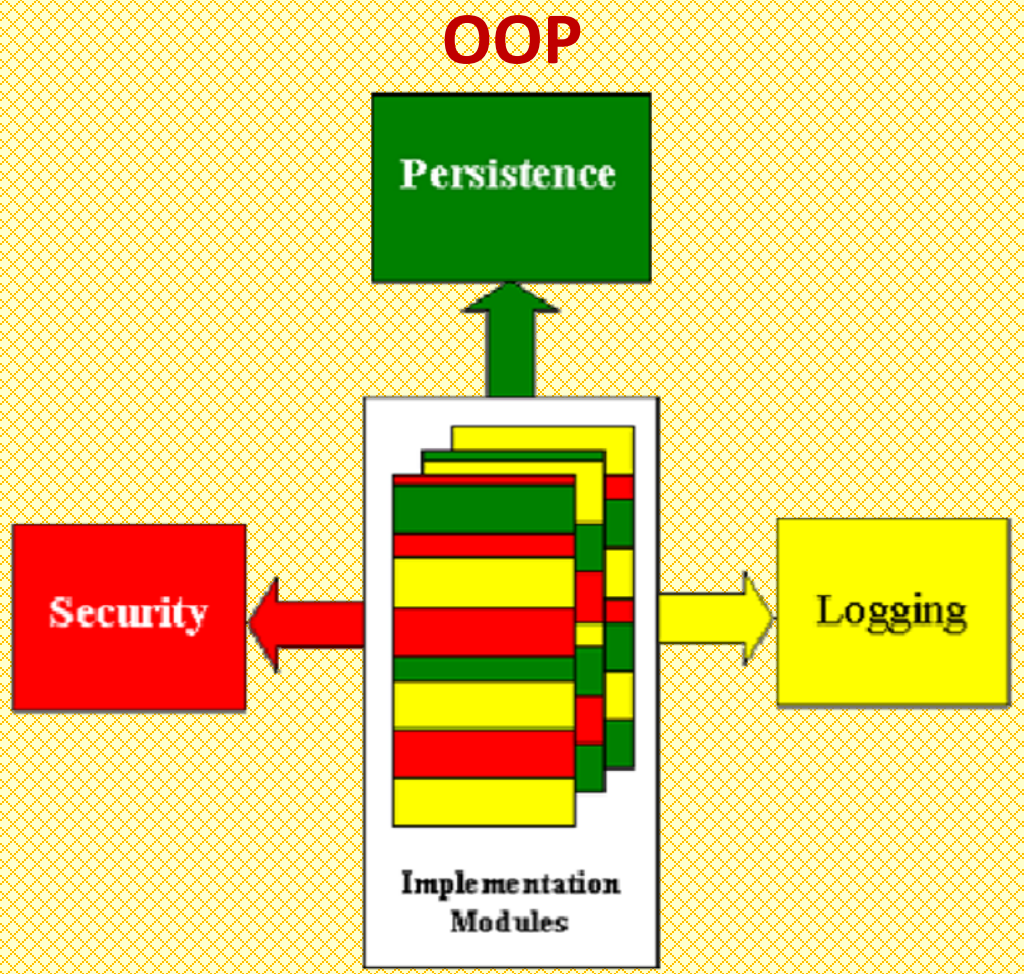
Aspect Oriented Programming (AOP) is a new **development technology** that permits **separation** of **cross-cutting concerns** that have in the past proved difficult to implement using object oriented programming (OOP).

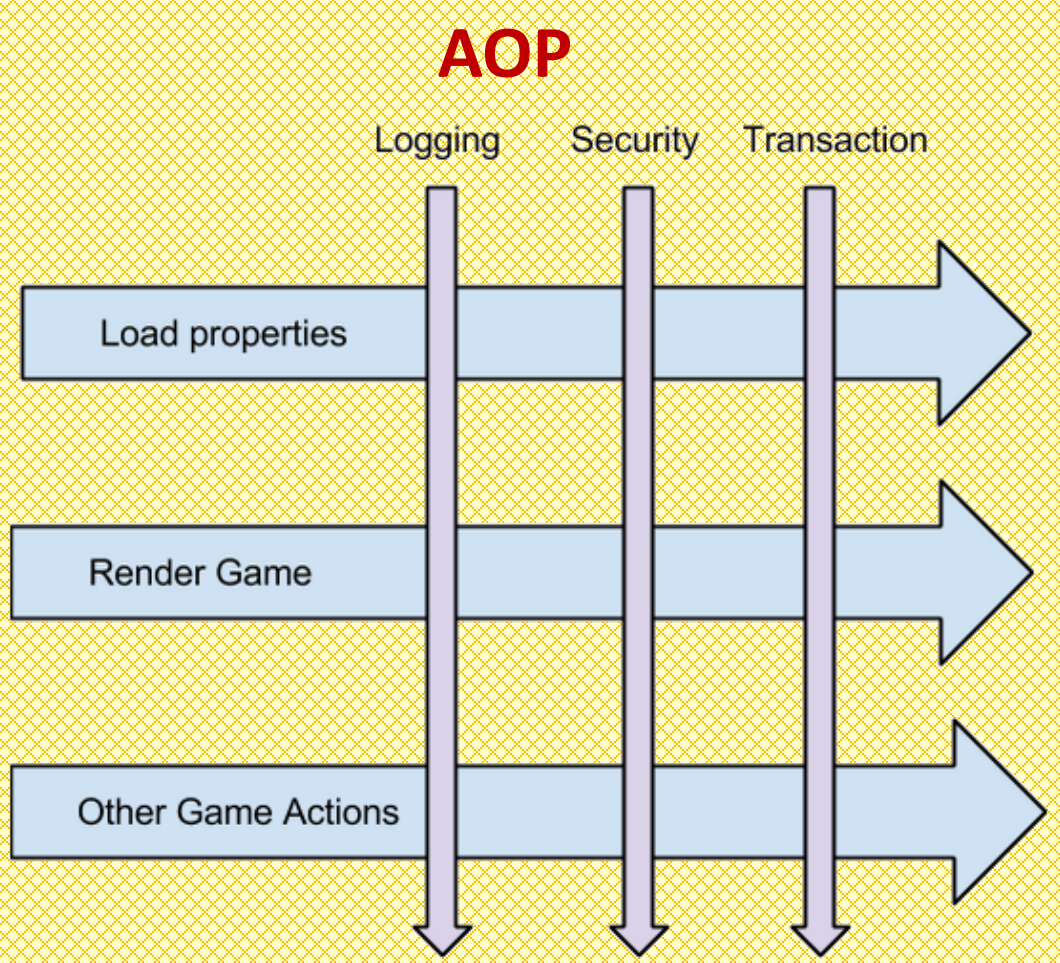
Aspect-oriented programming entails **breaking down** program logic into **distinct parts;** Of course, on the **special enamel**.

AOP Can be

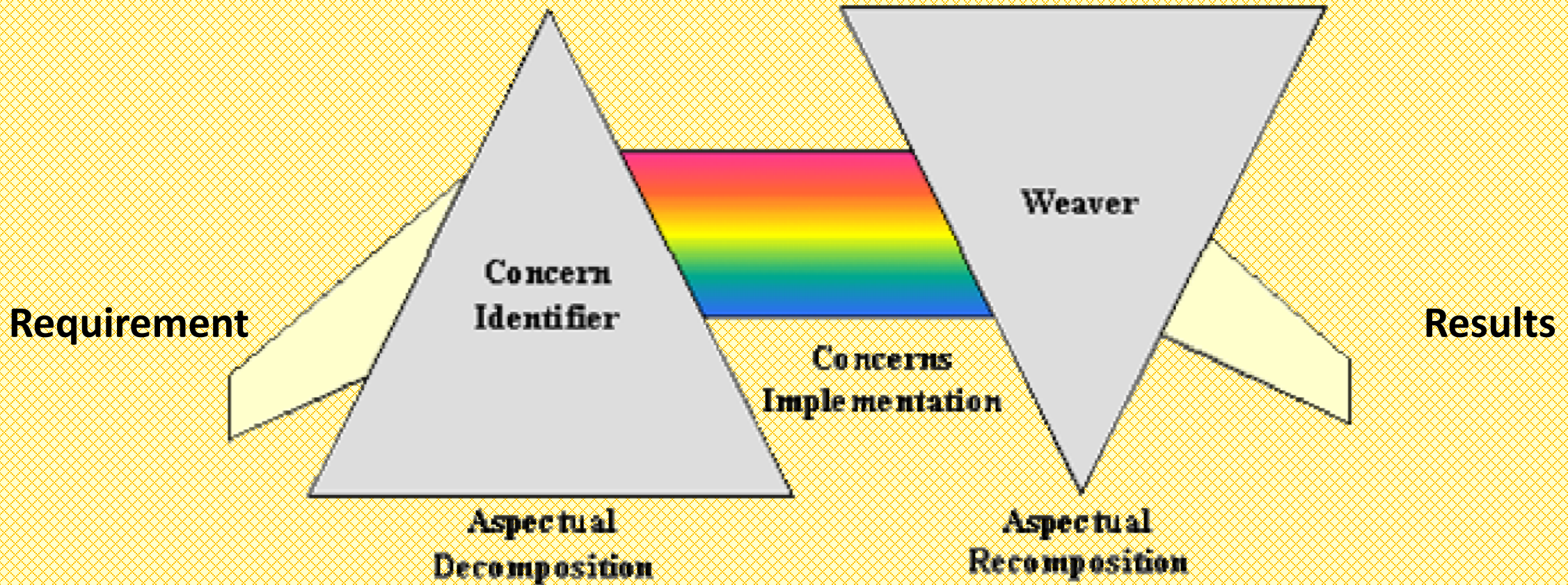
- ☑ Methodology
- ☑ Architecture
- ☑ Language
- ☑ Programming

What is AOP?


OOP
Persistence
Security
Logging
Implementation Modules


AOP
Logging
Security
Transaction
Load properties
Render Game
Other Game Actions

How is work?


Requirement
Concern Identifier
Aspectual Decomposition
Concerns Implementation
Weaver
Aspectual Recomposition
Results

Development

- Concern
  - Core level Concerns
  - System level Concerns
  - Cross-Cutting Concerns
- Concern Identifier
- Concern Implementation
- Concern Weaver
- Point Cut, Advice, Join Point

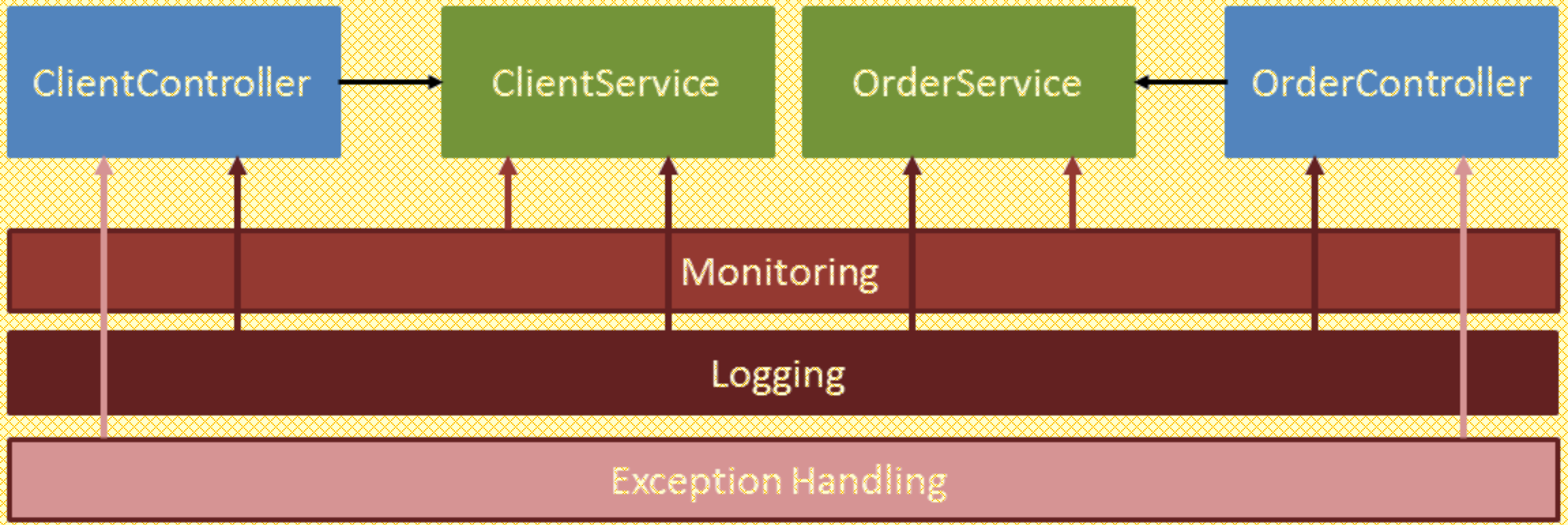
Concern

Concern is a **definite goal**, **concept** or **scope of the workflow**.

Any system is consists of several concerns

- **Core level Concerns**: Main purpose of software. i.e. The natural components of the software.
- **System level Concerns:** Other tasks Software. **i.e.** security, logging, transaction, authentication, persistence and so on.
- **Cross-Cutting Concerns:** concerns that tend to **affect several other concerns**. For instance, if a logging feature is to be implemented in an application, it is likely that all the underlying modules will have code for logging, making the underlying modules less specialized and makes it very hard to predict what effects changes in the code for logging will have.

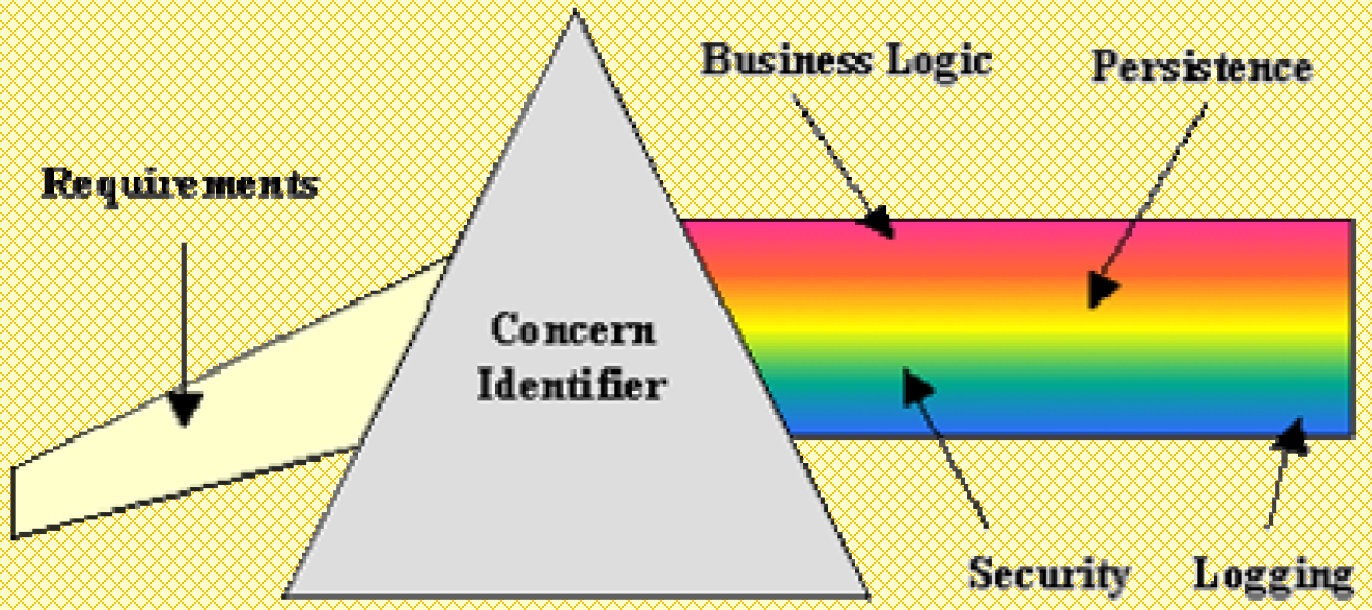
Development Concern



Concern Identifier

In the first step, the requirements are decomposed to identify cross-cutting and common concerns. Here, the core concerns are separated from the cross-cutting system level concerns.

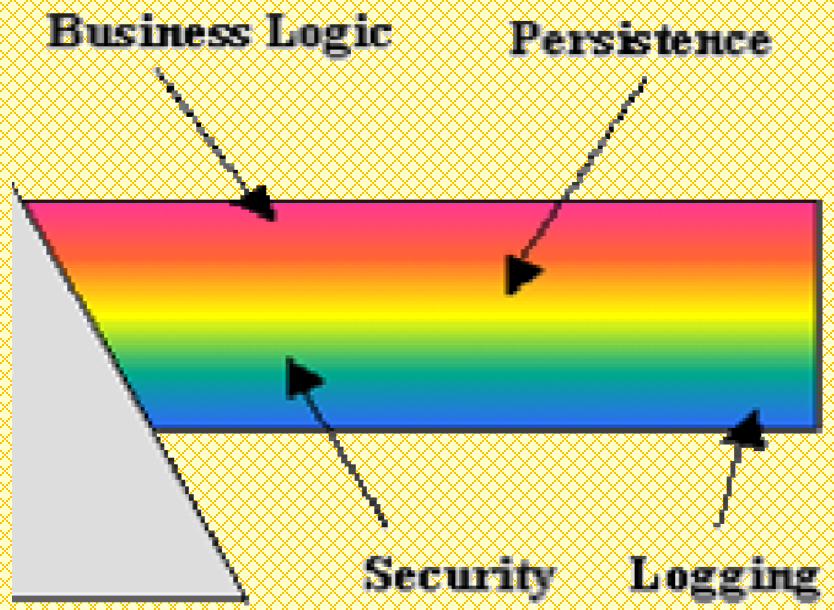
For example:
In the credit card processing module in ATM, could identify these four concerns:
Core credit card processing
Logging
Authentication
Persistence



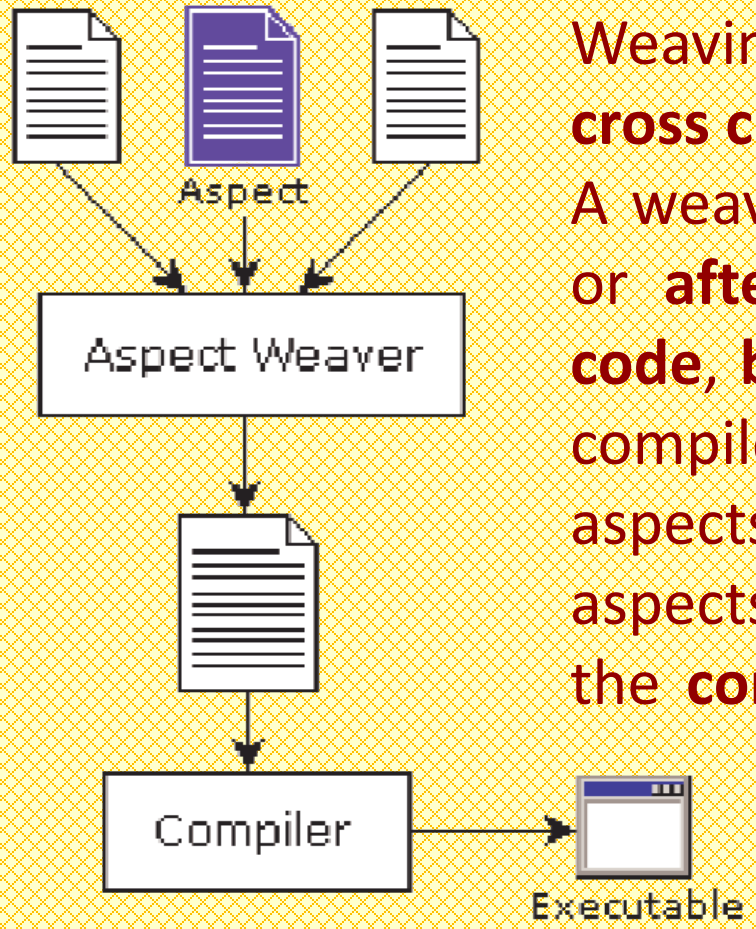
Development Concern Implementation

In the second step of the development, each concern is **implemented separately**.

In the credit card processing module, would implement the core credit card processing unit, logging unit, persistence unit and authentication unit separately.



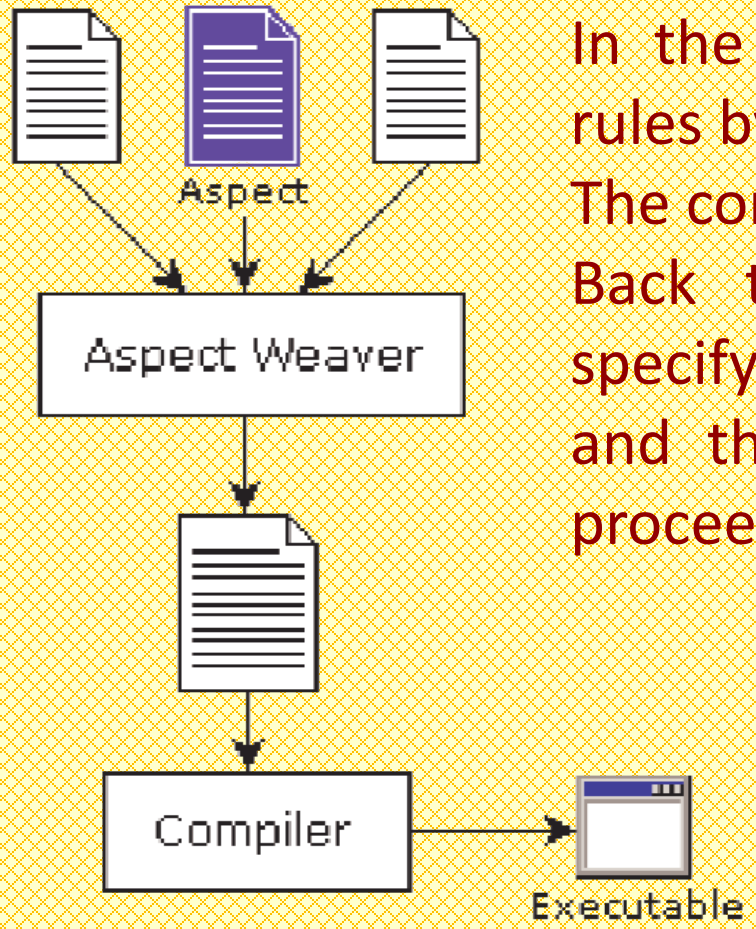
Concern Weaver



Weaving is the process of **composing a component with a cross cutting aspect**.

A weaver can **compose objects** at statically **at compile time**, or **after** aspects and components are **compiled to object code**, **byte codes** or **intermediate language** generated by the compilers. It is also **possible** to compose components with aspects **at run time** my calling a weaving library, supplying the aspects and components as parameters, or dynamically **when** the **component is loaded for use** by an application when it is **first called**.

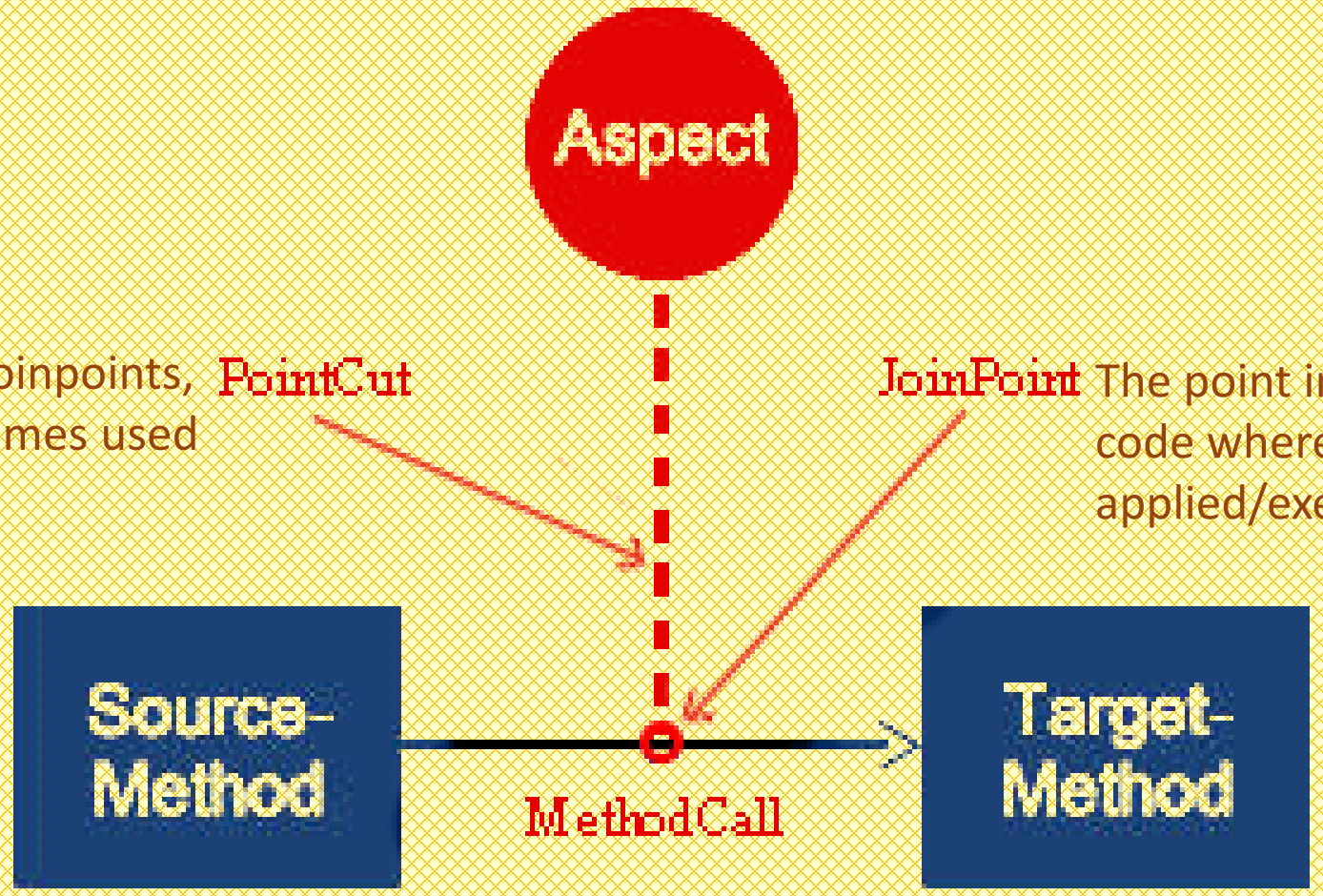
Concern Weaver



In the last step, an aspect integrator specifies composition rules by creating modularization units (aspects).
The composition process is also called weaving or integrating.
Back to the credit card processing module, we would specify each operation's start and completion to be **logged** and that each operation must be **authenticated** before it proceeds with the **business logic**.

Implementation Terms

A pointcut is a set of joinpoints, but the term is sometimes used instead of “joinpoint.”



The point in the execution of the code where the advice should be applied/executed.

Why AOP?

- ✓ Is an **elegant** and **simple** construct
- ✓ Coding is **reduced**
- ✓ Is orthogonal to the **primary purpose** of a module
- ✓ Bettering in the **encapsulation**
- ✓ Bettering **reuse** of the **arbitrarily invoked code** and **the target module**
- ✓ Can be **convert** the **cross-cutting concerns** into **an object**
- ✓ Makes more **understandable** source **code**.
- ✓ On the basis of **object-oriented programming**.
- ✓ etc...

AOP in .Net

```
public Document[] GetDocuments(string format) {
    try { using (var context = Directory.GetFiles("")) {
            var documents =
                context
                    .Documents
                    .Where(c => c.Name.EndsWith("." + format))
                    .ToArray();
            logger.LogSuccess(
                "Obtained " + documents.Length + " documents of type " + format +
                Environment.NewLine +
                "Connection String: " + connectionString);
            return documents;
        } } catch (Exception ex)  {
        logger.LogError(
            "Error obtaining documents of type " + format +
            Environment.NewLine +
            "Connection String: " + connectionString, ex);
        throw;  } }
```


AOP in .Net

Clearly, the logging code has made the original method **less readable**. It has **tangled** the real method code with logging code.
This is also a **violation** of the **Single Responsibility Prin**ciple.
In fact main purpose from that method is below method.

```
public Document[] GetDocuments(string format)
{
    using (var context = Directory.GetFiles(""))
    {
        return
            context
                .Documents
                .Where(c => c.Name.EndsWith("." + format))
                .ToArray();
    }
}
```

AOP in .Net

Also, we expect to find the same logging pattern in many methods all over the code base. Basically, we expect to find the following pattern:

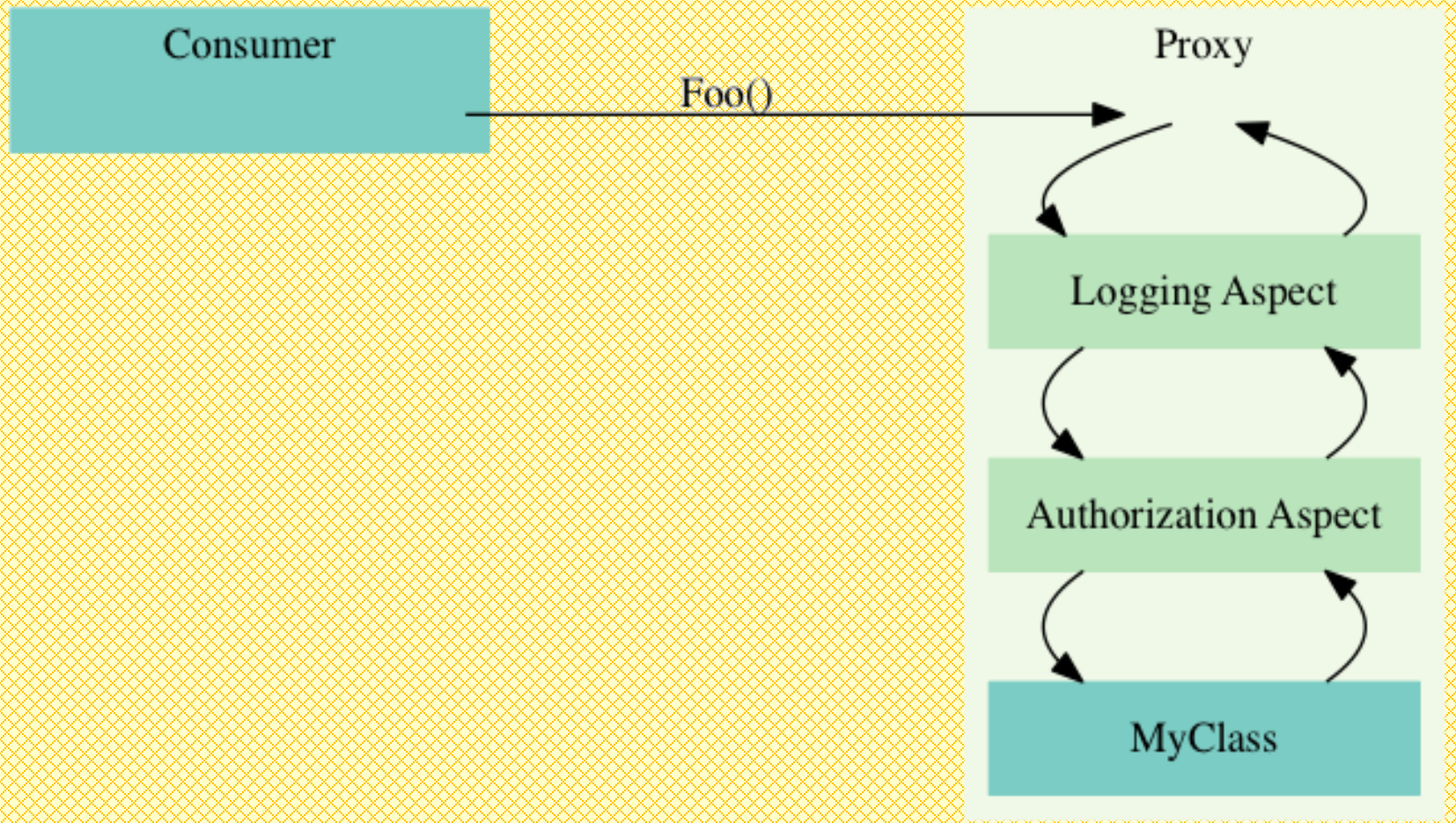
```
try
{
    //Do something here
    logger.LogSuccess(…
    //..
}
catch (Exception ex)
{
    logger.LogError(…
    throw;
}
```

AOP in .Net

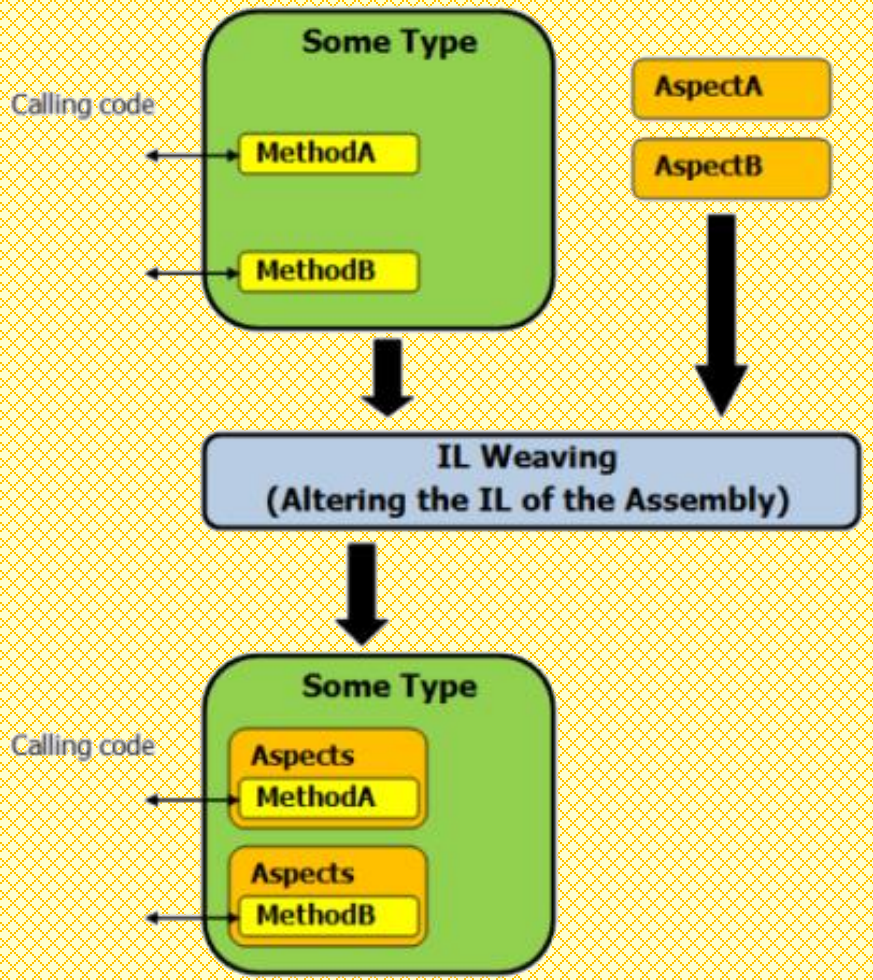
There are two types of AOP frameworks for .NET.

- Proxy-Style
- IL Rewriter

Proxy-Style


Consumer
Foo()
Proxy
Logging Aspect
Authorization Aspect
MyClass

IL Rewriter


Some Type
Calling code
MethodA
MethodB
AspectA
AspectB
IL Weaving
(Altering the IL of the Assembly)
Some Type
Calling code
Aspects
MethodA
Aspects
MethodB

End

Collected By: MiMFa