.NEXT

LUXOFT TRAINING

POSTSHARP

Рефакторинг кода с использованием PostSharp

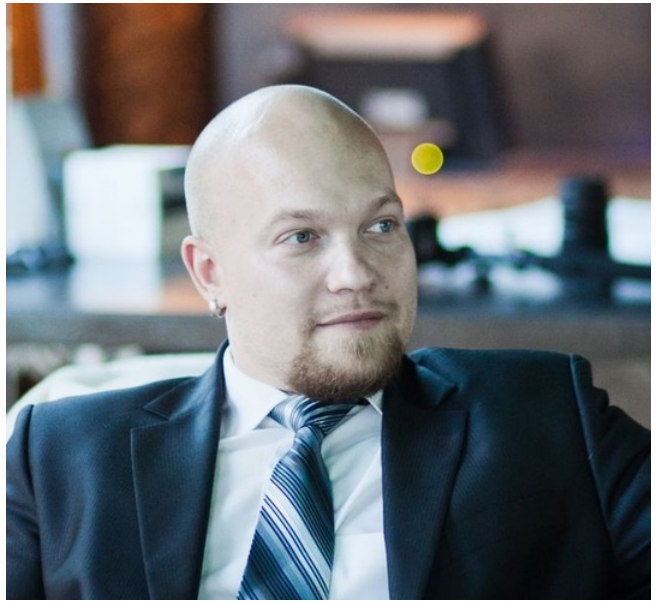
Андрей Гордиенков

my@violet-tape.net

softblog.violet-tape.ru

Обо мне

- Фанат программирования и рассказов о программировании
- Ведет свой блог 5+ года: статьи и видео
- АОП евангелист
- Очень ленив, поэтому ищет пути как писать меньше, а делать больше



Что такое АОП?

Аспе́ктно-ориенти́рованное программи́рование —методика программирования в рамках классовой парадигмы, основанная на понятии аспекта — блока кода, инкапсулирующего сквозное поведение в составе классов и повторно используемых модулей.

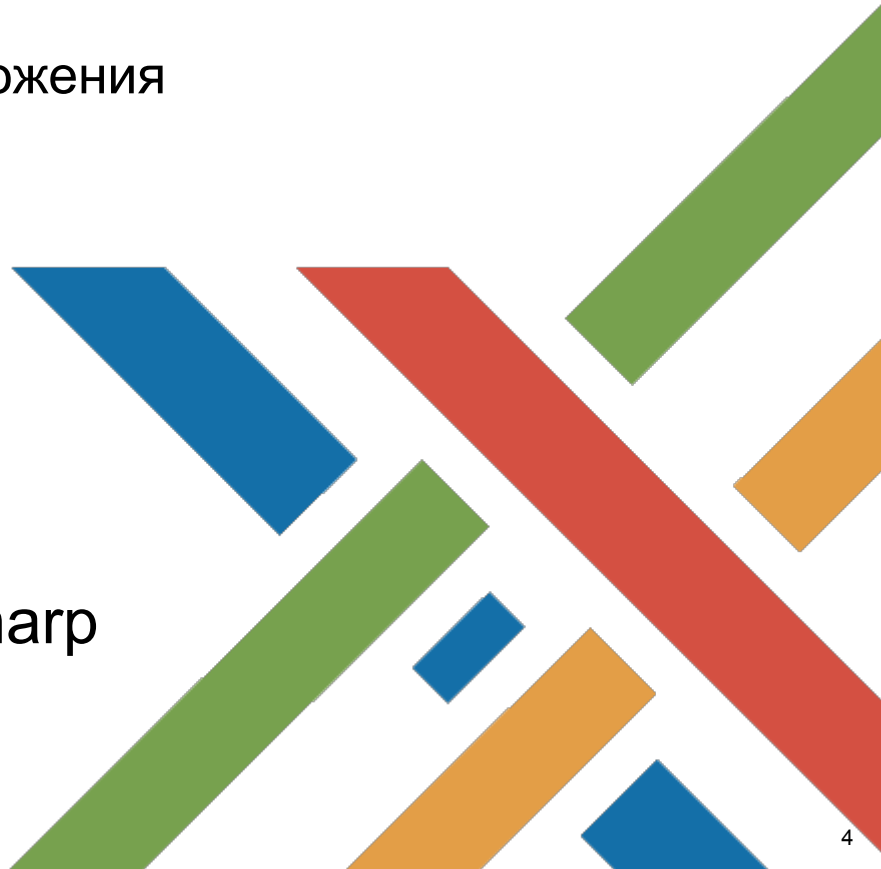
Wikibooks

- Все средства АОП предоставляют средства для выделения сквозной функциональности в отдельную сущность.

Для чего нужен АОП?

- ООП отлично себя зарекомендовала для решения доменных задач. Но есть сопутствующие задачи, которые прошивают все слои и с ними ООП справляется плохо.
- АОП позволяет:
  - Уменьшить стоимость разработки и время доставки приложения
  - Сократить количество ошибок в приложении
  - Увеличить поддерживаемость ПО

Мир Java имеет AspectJ,
в .NET сравнимым функционалом обладает PostSharp



Состав типичного бизнес кода

```
public void ProcessOrder(Period period, List<Item> items) {
#if DEBUG
    Log.Write(period);
    Log.Write(items);
#endif
    if(accessService.Check(Credentials))
        return;
    if(!items.Any())
        return;
    if(!period.IsValid())
        return;

     foreach (var item in items) {
         SomeService.Process(item, period.From);
     }
     AntherService.CreateInvoice(period, items);
#if DEBUG
     Log.Write("ProcessOrder " + executionTime);
#endif
}
```

Применимость АОП

- Логирование
- Трассировка
- Кэширование

- Но:
  - Применимость гораздо шире!
  - Например, шаблоны проектирования
  - Обеспечение уровня доступа к данным

Применимость АОП – не банальные случаи

- Автоматизация инструментирования
  - Например, расставить другие атрибуты для классов
- Реализация GoF шаблонов с помощью АОП
  - Многие шаблоны можно отделить и применять декларативно
- Безопасная работа с многопоточностью
- Обработка исключений
- Контракты на использование типов данных
- Undo\Redo

- Много примеров использования есть на postsharp.net

Пример на инструментирование кода - **до**

- Код размещается в сборке SomeLibraryExample, в неймспейсе DTO

```
namespace SomeLibraryExample.DTO {
    public class SuperHero {
        public string Name { get; set; }
        public bool CanFly { get; set; }
        public bool IsVillian { get; set; }
    }
}
```

- Точка внедрения объявляется в AssemblyInfo.cs

```
[assembly: AutoDataContract(AttributeTargetTypes = "SomeLibraryExample.DTO.*")]
```

Пример на инструментирование кода - **после**

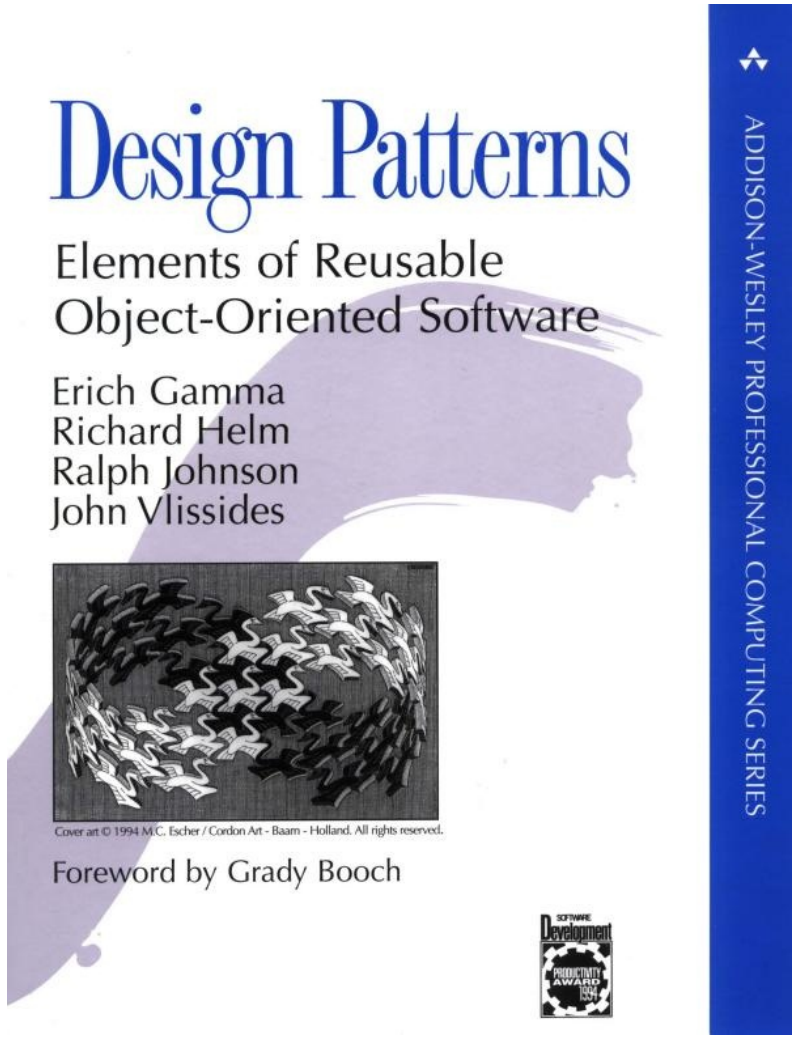
```
namespace OfftopicExample.DTO{
  [HasInheritedAttribute(new long[] {-5318251063910080884L})]
  [DataContract]
  public class SuperHero  {
    [DataMember]
    public string Name    {
      get      {        return this.\u003CName\u003Ek__BackingField;      }
      set      {        this.\u003CName\u003Ek__BackingField = value;      }
    }

    [DataMember]
    public bool CanFly    {
      get      {        return this.\u003CCanFly\u003Ek__BackingField;      }
      set      {        this.\u003CCanFly\u003Ek__BackingField = value;      }
    }

    …
```

Возможности PostSharp

- Явное указание точек внедрения
  - сборка, класс/объект, метод, свойство, поле
- Внедрение по шаблону имен
  - использование wildcards
- Инструментирование аспектами на этапе компиляции
- Проверка требований на этапе компиляции и в RunTime
- Управление порядком применения и наследуемостью аспектов
- Дебаг с точками остановки для CompileTime и RunTime
- Подсветка точек внедрения в Visual Studio

Шаблоны «Банды четырех»

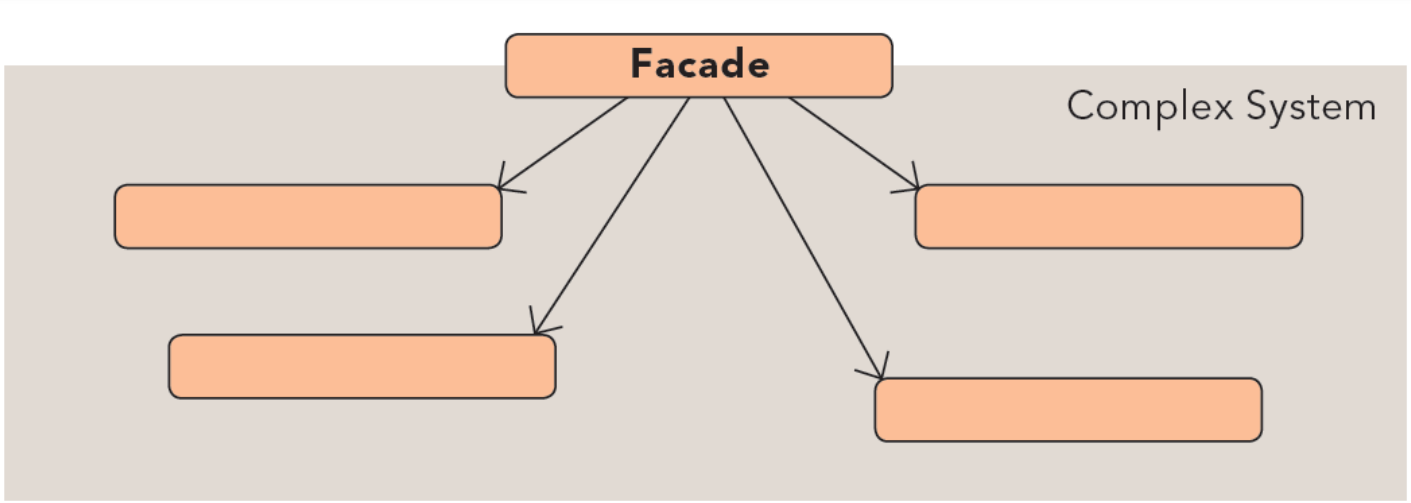
- Мировой бестселлер вышел в 1994 году
- 23 шаблона были разделены на 3 группы:
  - Создание объектов
  - Композиция объектов
  - Поведение объектов



Облегчают понимание и модификацию, но усложняют код. Инфраструктура внедряется в бизнес-логику.

Аспекты не применимы

- Façade – не выигрывает от применения аспектов.
  - Цель шаблона предоставить более удобное API для подсистемы



Аспекты не играют особенной роли

- Неочевидный выигрыш для шаблонов:
  - State
  - Interpreter
- Аспекты позволяют чуть сильнее локализировать код шаблона.

Аспекты могут помочь при определенных условиях

- Следующие шаблоны используют мощь наследования для достижения своих целей и реализации себя:
  - Abstract Factory
  - Factory Method
  - Template Method
  - Builder
  - Bridge
- Аспекты могут нести в себе код из абстрактных классов, освобождая место для явного наследования

Аспекты могут облегчить жизнь

- Большая часть поведения, может быть выделена в аспект для следующих шаблонов:
  - Composite
  - Command
  - Mediator
  - Chain of Responsibilities

```
[Node]
public class TopLevel : IComposite {
    private readonly List<IComponent> items = new List<IComponent>();

    public void Add(IComponent component) {
        items.Add(component);
    }

    public int Count() {
        return items.Count;
    }
}

[Node(Parent = typeof (TopLevel))]
public class LowLevel2 : IComponent {}
```

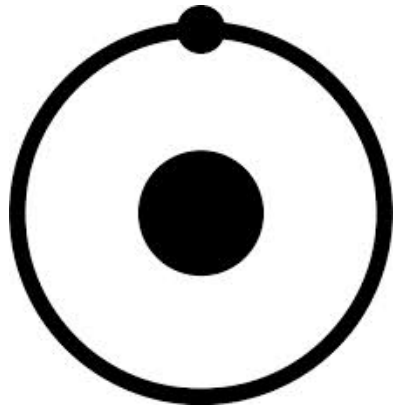
Аспекты кардинально меняют картину

- Для следующих шаблонов аспекты могут полностью изменить процесс использования шаблона:
  - Adapter
  - Decorator
  - Strategy
  - Visitor
  - Proxy
- Код шаблона выносится в аспект и применяется для всего семейства участвующих классов

Аспекты полностью реализуют шаблон

- Для следующих шаблонов реализация в участниках полностью исчезает
  - Observer
  - Memento
  - Singleton
  - Prototype
- Реализация шаблона становится универсальной

```
[SingletonClass]
public class MyClass {
    public int Counter;

    public static MyClass Instance { get; private
set; }

    private MyClass() {}
   }
}
```



Проверка структуры кода на этапе компиляции

- Проверка структуры на разных уровнях
  - Сборка, класс, методы
- Управление уровнем критичности нарушения структуры
  - Info, Error, Warning и другие

Основные выводы по рефакторингу GoF

- Для 17 из 23 шаблонов возможно применить АОП с локализацией кода.
- Для 12 из 17 можно ядро шаблона выделить в отдельные повторно используемые классы.
- Для 14 из 17 появляется более прозрачная компоновка участников шаблона.
- Знания о применении шаблона не теряются благодаря появляется

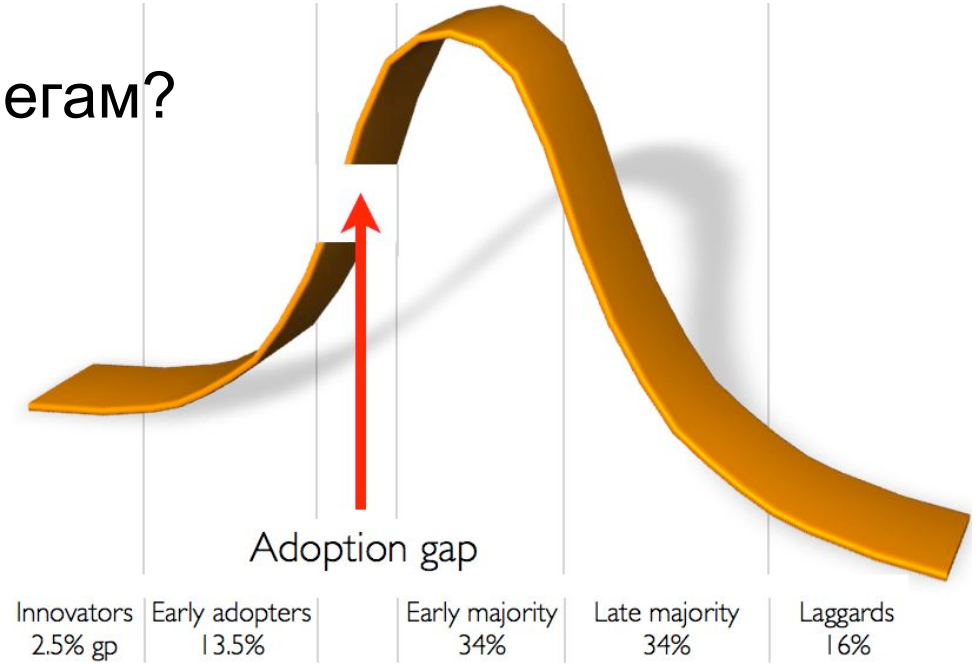
```
[SingletonClass]
4 references
public class MyClass {
    public int Counter;

    2 references
    public static MyClass Instance { get; private set; }

    0 references
    private MyClass() {}

    1 reference
    public int Foo() {
        return ++Counter;
    }
}
```

Accessor is enhanced by Single (provided by aspect SingletonClass applied on type MyClass).
Intercepted by advice OnGetValue.

Шаблон	Локальность	Повторное использование	Прозрачность компоновки	Отключаемость
Façade	-	-	-	-
Abstract Factory	Нет	Нет	Нет	Нет
Bridge	Нет	Нет	Нет	Нет
Builder	Нет	Нет	Нет	Нет
Factory Method	Нет	Нет	Нет	Нет
Interpreter	Нет	Нет	н\о	Нет
Template Method	(Да)	Нет	Нет	(Да)
Adapter	Да	Нет	Да	Да
State	(Да)	Нет	н\о	(Да)
Decorator	Да	Нет	Да	Да
Proxy	(Да)	Нет	(Да)	(Да)
Visitor	(Да)	Да	Да	(Да)
Command	(Да)	Да	Да	Да
Composite	Да	Да	Да	(Да)
Iterator	Да	Да	Да	Да
Flyweight	Да	Да	Да	Да
Memento	Да	Да	Да	Да
Strategy	Да	Да	Да	Да
Mediator	Да	Да	Да	Да
Chain of Responsibility	Да	Да	Да	Да
Prototype	Да	Да	(Да)	Да
Singleton	Да	Да	н\о	Да
Observer	Да	Да	Да	Да

Вопросы

- Приходилось ли использовать в продакшене?
  - Да, приходилось
- Для чего?
  - Как раз таки для детального трейса фин.операций в системе для этого не приспособленной изначально.
- Тяжело ли его продавить руководству и коллегам?
  - Да, очень тяжело
- Почему?
  - Сопротивление всему новому в природе человека



Ссылки для самостоятельного изучения

- Примеры АОП решений - http://www.bodden.de/tools/aop-dot-net/
- Пишем АОП сами http://habrahabr.ru/post/199378/ - для уверенных и смелых
- Пользуемся готовым – http://www.postsharp.net/ – есть бесплатная версия, которая хороша
- Теория
  - https://www.cs.ubc.ca/labs/spl/papers/2002/oopsla02-patterns.pdf
  - ftp://cs.joensuu.fi/pub/Theses/2008_MSc_Oprisan_Andrei.pdf
- Мифы и реальность АОП (ажно 15 штук)
  - https://www.ibm.com/developerworks/ru/library/j-aopwork15/
- Примеры кода https://github.com/VioletTape/GoF_PostSharp

Спасибо!

my@violet-tape.net

softblog.violet-tape.ru