

# PostSharp. Alternatives

---

*Прочитать статью на сайте*

Я тут подумал, что несправедливо было бы обойти стороной сравнение PostSharp и других продуктов предоставляющих АОП. Ниже будут рассмотрены особенности использования, плюсы и минусы различных подходов. Но не думайте что PostSharp является серебрянной пулей. =)

Можно почитать и википедию насчет АОП и фреймворков для .Net платформы. Я посмотрел несколько перечисленных там фреймворков лично и по документированности и простоте использования остановился на PostSharp.

## Containers (Инверсия зависимостей)

Большинство фреймворков направленных на инверсию зависимостей включают в себя некоторые возможности аспектно-ориентированного программирования. Для большинства достаточно простых технических требований, таких как трейсинг и обработка исключений применяемых к сервисам приложения, их возможностей хватает. Но их нельзя применить для более сложных и комплексных аспектов, которые могут применяться так же к объектам пользовательского интерфейса или к доменным объектам.

**Продукты:** [Spring.NET](#), [Castle Windsor](#), [Microsoft Unity](#)

Плюсы:

- Вы уже используете фреймворк для инверсии зависимостей
- Аспекты могут быть настроены после сборки
- Некоторые аспекты (трейсинг и обработка исключений) могут быть уже реализованы в вашем фреймворке.

Минусы:

- Очень ограниченные возможности в плане АОП
- Ваши объекты должны создаваться с помощью контейнера предоставляемого фреймворком. Такой способ нельзя применить для элементов пользовательского интерфейса, и во многих случаях для доменных объектов тоже.
- Не работает со статическими, не публичными и/или не виртуальными методами. Так же не будет работать применение на поля, свойства и события классов.
- Нет, или же плохое инструментирование для Visual Studio

## Функциональное программирование

Функциональное программирование (с помощью анонимных методов и лямбда выражений) может быть решением такой проблемы как автоматическое повторение действия при неудачно проведенной транзакции.

Плюсы:

- Стандартный C#. Не требуется никаких дополнительных зависимостей в вашей сборке.

- Работа с уже знакомыми технологиями (я о анонимных методах и лямбда выражениях).
- В некоторых случаях, результат по объему кода такой же малый как и с АОП.

Минусы:

- Невозможно автоматически передать контекст из исполняемого кода в аспект. Например, у вас не получится реализовать трейсинг с помощью функционального программирования.
- Придется дописывать код в каждый метод, к которому вы хотите добавить аспект. Настоящее АОП может применяться к множеству методов буквально одной строчкой кода и поддерживает сложные правила для выборки методов к применению.
- Композиция аспектов получается менее удобная. Если у вас к одному и тому же методу применено два аспекта, и позже вы решили изменить порядок применения, то вам придется переписать все места, где вы их использовали.

## Генерация кода

АОП и генерация кода – это два неконкурирующих подхода в метапрограммировании. Генерация кода фокусируется на производстве нового кода основанного на указанных артефактах более высокого уровня абстракции. АОП особенно хорош в объединении кода бизнес логики и технического (инфраструктурного) кода.

**Продукты:** CodeSmith

Плюсы:

- Достаточно легко сгенерировать сложный участок кода, поскольку все разработчики работают с C# или VB.
- Итоговый код на C#/VB может быть легко открыт в Visual Studio и подвергнут дебаггину, как любой другой кусок кода.

Минусы:

- В отличие от АОП, исходники ответственные за генерацию другого кода обычно пишутся в виде XML файла.
- Эта технология не может как бы то ни было подсовывать новые инструкции в существующий код или же модифицировать его поведение.

## MSIL трансформация

С ростом популярности PostSharp, авторы фреймворка предпринимали неоднократные попытки использовать трансформацию MSIL для добавления нового поведения в программы. Т.к. PostSharp сам по себе использует PostSharp SDK, то доступны самые продвинутые способы трансформации кода. Нет таких задач которые можно было бы реализовать на других платформах, а на PostSharp – нет.

**Продукты:** [Mono.Cecil](#), [Microsoft.CCI](#), [CCISharp](#).

Плюсы:

- Устраняют зависимость от PostSharp.dll для времени исполнения программы (однако то же самое можно сделать с помощью PostSharp SDK)
- Перечисленные фреймворки бесплатны и за ними стоят большие компании.



**Where can you apply aspects?**

	<b>PostSharp MSIL transformation</b>	<b>Transparent Proxy Microsoft Unity</b>	<b>JIT-Emitted Proxy Spring.NET, Castle Windsor</b>	<b>JIT-Emitted Subclass Spring.NET, Castle Windsor</b>
Methods implementing an interface	Yes	Yes	Yes	Yes
Virtual public methods	Yes	Only with MarshalByRefObject		Yes
Non-virtual public methods	Yes	Only with MarshalByRefObject		Yes
Virtual protected methods	Yes			Yes
All methods (including private, protected, static)	Yes			
External methods (defined in a different assembly)	Yes			
Constructors	Yes			
Fields	Yes			
Properties	Yes			
Events	Yes			

**Other features**

	<b>PostSharp</b>	<b>Transparent</b>	<b>JIT-Emitted</b>	<b>JIT-Emitted</b>
--	------------------	--------------------	--------------------	--------------------

	<b>MSIL transformation</b>	<b>Proxy Microsoft Unity</b>	<b>Proxy Spring.NET, Castle Windsor</b>	<b>Subclass Spring.NET, Castle Windsor</b>
Modify aspects after build		<i>Yes</i>	<i>Yes</i>	<i>Yes</i>
No impact on build time		<i>Yes</i>	<i>Yes</i>	<i>Yes</i>
Use plain old constructors instead of factory methods	<i>Yes</i>			
Support for Silverlight & Compact Framework	<i>Yes</i>			