POSTSHARP

#1 pattern-aware extension to C# and VB

Reviewer's Guide

Welcome Letter

(Please Read First – Product Positioning)

We're proud to introduce PostSharp, a development tool designed to address the shortcomings of conventional compilers and programming languages that do not support the concept of patterns. Because of these shortcomings, developers spend up to 20% of their time writing repetitive code as they implement patterns manually. We specifically developed PostSharp for pattern automation and it has become the #1 best-selling pattern-aware compiler extension to C# and VB.

Started as an open-source project in 2004 and overwhelmed by its popularity, PostSharp soon became a commercial product trusted by over 50,000 developers worldwide and over 1,000 leading corporations. PostSharp is now used by more than 10% of all Fortune 500 companies including Microsoft, Intel, Bank of America, PricewaterhouseCoopers, Phillips, BP, Volkswagen and Siemens.

PostSharp allows developers to eradicate boilerplate by offloading repeating work from humans to machines. It contains ready-made implementations of the most common patterns such as Logging, INofityPropertyChanged, Undo/Redo or Code Contracts and it also gives developers the tools to build automation for custom patterns specific to their projects. Thanks to the ready-made threading design patterns, developers can write thread-safe applications in C# and VB without rewriting them.
The result: shorter, cleaner code that's easier to write and understand, contains fewer defects and is less expensive to maintain. PostSharp typically reduces codebase by 5% to 25% with an average 19x return on investment.

This reviewer's guide was created to help you evaluate PostSharp and includes why customers should consider PostSharp, a feature list, comparison with other tools and development approaches, return on investment, useful links and instructions on how to get started with the product.

Get to know PostSharp, the #1 best-selling pattern aware compiler extension to C# and VB with over a decade experience in boilerplate reduction.

Sincerely,

Gael Fraiteur
CEO and Principal Engineer
PostSharp Technologies

With PostSharp, you're in good company

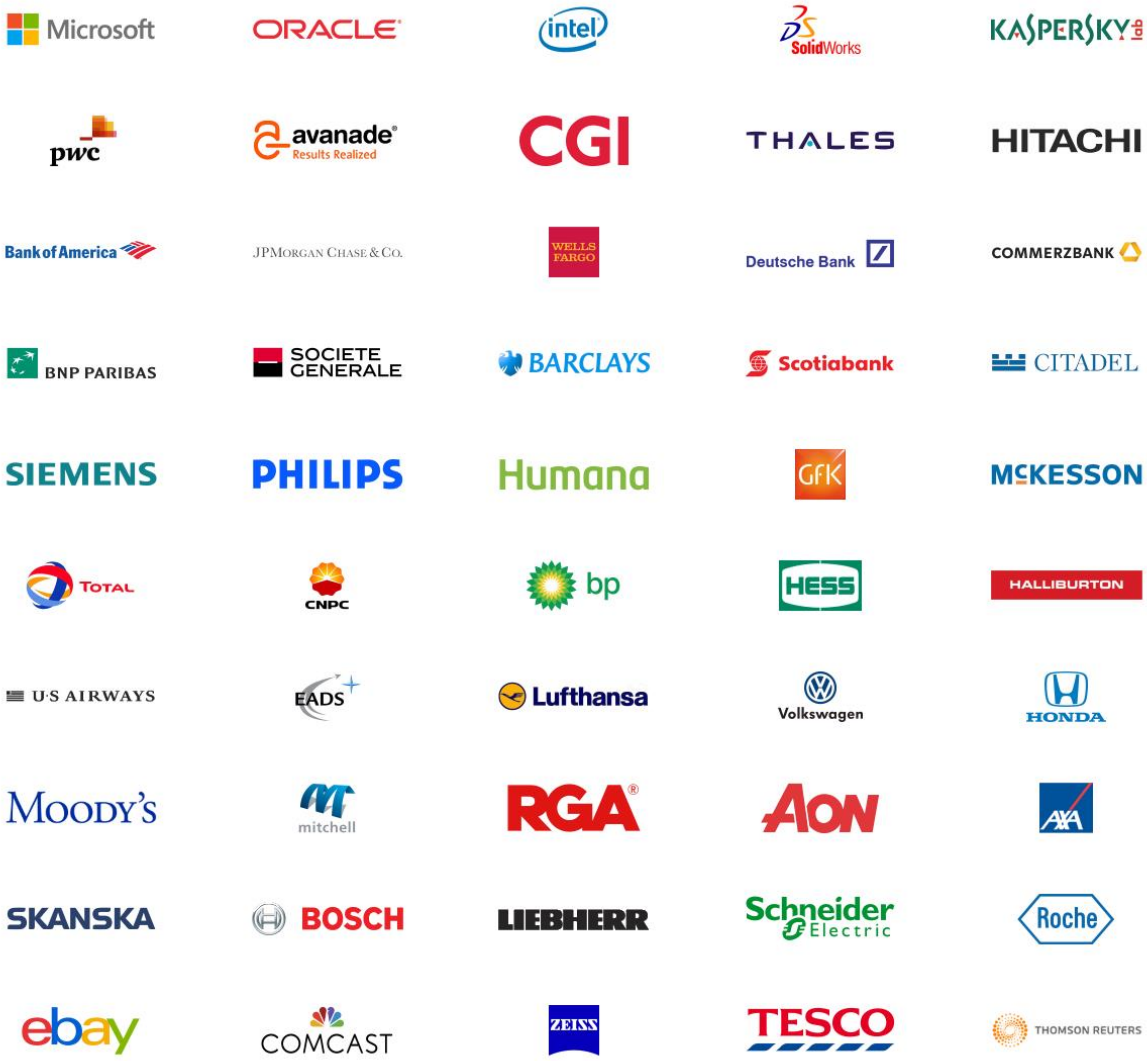

Table of Contents

"PostSharp is a pretty amazing piece of software. Abstractions like PostSharp are the whole point of what the computer is supposed to do for us, work that's not fun, like logging and transactions. So why not hide that?"

Scott Hanselman, Principal Program Manager, Microsoft

Problem: Existing Compilers Don't Support Patterns

Whether it's architecture, carpentry or mechanics, patterns are essential to all construction and engineering disciplines. Learning not only how to implement patterns, but also when and why to choose them, is an important part of the education of professionals of these disciplines.

Software engineering is no exception. Patterns have been successfully applied to software architecture and software design, two preliminary steps of software constructions. As a result, today's software engineers are trained to *think* in terms of patterns. However, when it comes to *implementation*, developers don't have the right tools.

Conventional programming languages miss a concept of *patterns*. Therefore, software developers are forced to implement patterns *by hand*. Just like artisan carpenters would manufacture dozens of *almost* (not totally) identical joints to build a cabinet, a software developer would add *almost* identical exception handling logic to hundreds of functions by hand. The difference, however, is that the artisan carpenter produces a piece of art, while the software developer builds a utilitarian application.

You would not hire artisan carpenters to build a utilitarian industrial building, would you?

Developers call this repetitive code *boilerplate*. Good developers consider boilerplate code as a boring but necessary part of their work. However, as this white paper will show, boilerplate code is not inevitable.

Boilerplate code is a major source of pain in enterprise development. It has the following consequences:

1. High development effort
2. Poor quality of produced software
3. Difficulty to add/modify functionality after first release
4. Slow ramp-up of new team members

1. High development effort

- **Large codebases**. Some application features require a large amount of repetitive code (boilerplate) when implemented with existing mainstream compiler technologies.
- **Reinventing the wheel**. Solutions to problems like INotifyPropertyChanged are always being reinvented because there is no reusable option within conventional programming languages.

2. Poor quality software

- **High number of defects**. Every line of code has a possibility of defect, but code that stems from copy-paste programming is more likely than other to be buggy because subtle differences are often overlooked.
- **Multi-threading issues**. Object-oriented programming does not deliver much value when it comes to developing multi-threaded applications since it addresses issues at a low level of abstraction with locks, events or interlocked accesses that can easily result in deadlocks or random data races.

- **Lack of robustness**. Enterprise-grade features such as exception handling or caching are often deliberately omitted because of the high amount of source code they imply, unintentionally forbidden in some parts of the applications, simply left untested and unreliable.

3. Difficulty to add/modify functionality

- **Unreadable code that's difficult to maintain**. Business code is often littered with low-level non-functional requirements and is more difficult to understand and maintain, especially when the initial developer left.
- **Strong coupling**. Poor problem decomposition results in duplicate code and strong coupling making it very difficult to change the implementation of features like logging, exception handling or INotifyPropertyChanged because it is often scattered among thousands of files.

4. Slow ramp-up of new team members

- **Too much knowledge required**. When new team members come to work on a specific feature, they often must first learn about caching, threading and other highly technical issues before being able to contribute to the business value: an example of bad division of labor.
- **Long feedback loops.** Even with small development teams, common patterns like diagnostics, logging, threading, INotifyPropertyChanged and undo/redo can be handled differently by each developer. Architects need to make sure new team members understand and follow the internal design standards and have to spend more time on manual code reviews--delaying progress while new team members wait to get feedback from code review.

Solution: Pattern-Aware Compiler Extensions

Pattern-aware programming *extends* conventional object-oriented programming with a concept of *pattern*, which becomes a first-class element of the programming language.

Most mainstream programming languages can be extended with a concept of pattern, avoiding the cost of rewriting applications in a new language.

Because patterns are supported by the compiler extension (100% compatible with your existing compiler), they do not need to be manually implemented as boilerplate code. Features such as INotifyPropertyChanged, logging, transactions are implemented in a cleaner, more concise way, making development and maintenance much easier.

There are 4 reasons to consider using a Pattern-Aware Compiler Extension:

1. Helps you stop writing boilerplate code and deliver faster
2. You can build more reliable software
3. Makes it easier to add/modify functionality after first release
4. Helps new members contribute quicker

1. Helps you stop writing boilerplate code and deliver faster

- **Fewer lines of code means fewer hours of work.** Patterns are repetitive, with little or no decision left to the developer. However, repetition is exactly what computers are good at. Let the compiler do the repetitive work and save development time and costs immediately.

2. Helps you build more reliable software

- **Cleaner code means fewer defects**. With a pattern-aware compiler eliminating the boilerplate, your code becomes easier to read, understand and modify, and contains fewer defects.
- **Reliability becomes much more affordable**. Because they no longer require so much manual coding, reliability features such as caching or exception handling are much easier and cheaper to implement, so you can spend your extra time building a more robust app.

3. Makes it easier to add/modify functionality after the first release

- **Cleaner and shorter code is easier to understand**. After the initial release, too much development time is spent reading and analyzing source code, especially if the initial developer left. With minimized boilerplate code, developers can easily focus on business logic and spend much less time trying to understanding the code.
- **Better architecture is future-proof**. Using a pattern-aware compiler, features like logging, exception handling or transactions are no longer scattered among thousands of files but they are defined in one place, making it much easier and fast to modify when necessary.

4. Helps new members contribute quicker

- **Achieve a better division of labor.** Using a pattern-aware compiler makes the introduction of new or junior team members less onerous since they can focus on simpler, more business logic-

oriented tasks rather than having to waste so much time learning complex architectural structures.

- **Implement a tighter feedback loop.** A pattern-aware compiler can validate that hand-written code respects a pattern or a model, and it can detect bugs *at build time* instead of during code reviews, testing, or in production.

5 Reasons to Consider PostSharp

PostSharp is the #1 best-selling pattern-aware extension to C#/VB. It adds a concept of pattern to the languages, resulting in a dramatic reduction of boilerplate code, lower development and maintenance costs and fewer errors.

With PostSharp you can:

1. Get more productive in minutes with ready-made pattern implementations
2. Automate more complex patterns and remove more boilerplate
3. Build thread-safe apps--without a PhD
4. Maintain your existing codebase in C# or Visual Basic
5. Benefit from much better run-time performance

1. Get more productive in minutes with ready-made pattern implementations

PostSharp includes a number of ready-made pattern implementations found in .NET. Deep integration with Visual Studio and the user interface of PostSharp ensures developers get productive in just a few minutes.

- **INotifyPropertyChanged Pattern**. Automates the implementation of INotifyPropertyChanged and automatically raises notifications for you. It also analyzes chains of dependencies between properties, methods and fields in your source code, and understands that property getters can access several fields and call different methods, or even depend on properties of other objects.

  PostSharp eliminates all the repetition and lets you go from three lines of code per property to one attribute per base class... so you will never forget to raise a property change notification again.
- **Undo/Redo Pattern.** Makes the implementation of the end-users most-wanted feature easy and affordable by recording changes at model level. Provides built-in user controls or allows you to create your own. You can deliver the familiar Undo/Redo experience to your users without getting stuck writing large amounts of code.
- **Code Contracts**. Provide validation for valid URLs, email addresses, positive numbers or not-null values and many more, right out of the box. Allows you to use contract attributes without limitations at any location in your codebase and validate methods, fields, properties and parameters. This enables you to protect your code from invalid inputs with custom attributes.
- **Logging Pattern**. Adds comprehensive logging in a few clicks – without impact on your source code – and lets you remove it just as quickly. Provides parameter and return values providing added information for maintenance and support work. Supports most popular back-ends, including log4net, NLog, Enterprise Library, System Console, System Diagnostics. You can trace everything you need in minutes without cluttering your code.

2. Automate more complex patterns and remove more boilerplate

Besides the ready-made solutions, PostSharp provides the following tools and features to build automation for advanced patterns:

- **PostSharp Aspect Framework.** PostSharp is hands down the most robust and exhaustive implementation of aspect-oriented programming for .NET and was evolved into the world's best pattern compiler. It is the most powerful toolset available to implement automation for your own patterns.
- **Largest choice of possible transformations.** Includes decoration of methods, iterators and async state machines, interception of methods, events or properties, introduction of interfaces, methods, events, properties, custom attributes or resources, and more.
- **Composition of several transformations** to easily automate complex patterns.
- **Dynamic aspect/advice providers.** Addresses situations where it is not possible to add aspects declaratively (using custom attributes) to the source code with dynamic aspect/advice providers.
- **Aspect inheritance.** Apply an aspect to a base class, specify that you want it to be inherited and all derived classes will automatically have the aspect applied to them. Relieves you from implementing the aspects manually and ensures that all derived classes using this aspect's logic are correct.
- **Architecture framework**. Validates hand-written source code against your own custom pattern guidelines. It then express the rules in C# using the familiar System.Reflection API, extended with features commonly found in decompilers, such as “find usage”, and more.

3. Build thread-safe apps – without a PhD

Starting new threads and tasks in .NET languages is simple, but ensuring that objects are thread-safe is not with mainstream programming languages. That’s why PostSharp extends C# and VB with thread-safety features.

- **7 different threading models.** Threading models are design patterns that guarantee your code executes safely even when used from multiple threads.

  Threading models raise the level of abstraction at which multi-threading is addressed. Unlike working directly with locks and other low-level threading primitives, threading models *decrease* the number of lines of code, the number of defects and reduce development and maintenance costs – without having to have expertise in multi-threading.

  PostSharp implements the following threading models: immutable, freezable, synchronized, reader-writer synchronized, actor, thread affine, and thread unsafe.

- **Model validation**. Catches most defects during build or during single-threaded test coverage.
- **Thread dispatching patterns**. Causes the execution of a method to be dispatched to the UI thread or to a background thread. Much easier than using nested anonymous methods.

- **Deadlock detection**. Causes an easy-to-diagnose exception in case of deadlock instead of allowing the application to freeze and create user's frustration.

4. Maintain your existing codebase in C# and VB

Despite the hype around functional programming languages, C#/VB and .NET remain an excellent platform for enterprise development. PostSharp respects your technology assets and will work incrementally with your existing code base – there is NO need for a full rewrite or redesign.

- **Design neutrality**. Unlike alternatives, PostSharp takes minimal assumptions on your code. It does not force you to adopt any specific architecture or threading model. You can add aspects to anything, not just interface/virtual methods. Plus, it is fully orthogonal from dependency injection. You don't have to dissect your application into components and interfaces in order to use PostSharp.
- **Plain C# and VB**. PostSharp provides advanced features present in F#, Scala, Nemerle, Python, Ruby or JavaScript, but your code is still 100% C# and VB, and it is still compiled by the proved Microsoft compilers.
- **Cross-platform**. PostSharp supports the .NET Framework, Windows Phone, WinRT, Xamarin and Portable Class Libraries.
- **Standard skillset**. No complex API. Reuse what you already know from C# and System.Reflection.

5. Benefit from much better run-time performance

Start-up latency, execution speed and memory consumption matter. Whether you're building a mobile app or a back-end server, PostSharp delivers exceptional run-time performance.

- **Build-time code generation.** Unlike proxy-based solutions, PostSharp modifies your code at build time. It also allows for much more powerful enhancements that produces dramatically faster applications.
- **No reflection**. PostSharp does not rely on reflection at run-time. The only code that is executed is what you can see with a decompiler.
- **Build-time initialization.** Many patterns make decisions based on the shape of the code which they are applied to. With PostSharp, you can analyze the target code at build-time and store the decisions into serializable fields. At runtime, the aspects will be deserialized and you won't need to analyze the code at run-time using reflection.

Key Features

INotifyPropertyChanged
Simple properties
Composite properties
Properties of child objects
Smart event triggering
False positive suppression
Support for Caliburn.Micro
Support for MVVM Light
Highly customizable

Undo/Redo
Record any change in fields and collections
Low impact on source code
Customizable granularity of operations
Customizable name of operations
Ready-made undo/redo buttons
Custom actions
Integrated with INotifyPropertyChanged

Code Contracts
Easy, using custom attributes
Apply to any field, property or parameter
Not limited to ASP.NET MVC or XAML
Inheritance – apply to even to interfaces
Not null, number ranges, URLS, email, …
Extensible
Localizable

Logging
Log *anything* in a single line.
Logging of parameter values.
Logging of return values.
Indentation.
Thread-safe.
Reentry-safe.
Support for log4net
Support for NLog
Support for Enterprise Library
Support for Diagnostics.Trace
Support for Console

Aggregatable (parent/child relationships)
Navigate children in tree structures
Automatically implements IDisposable

Thread Safety
Immutable Threading Model
Freezable Threading Model
Synchronized Threading Model
Reader-Writer Synchronized Threading Model
Actor Threading Model
Thread Affine Threading Model
Thread Unsafe Threading Model
Build-Time Validation
Run-Time Model Validation
Thread Dispatching
Deadlock Detection

Aspect-Oriented Programming
Wrap any method with try/catch/finally
Intercept any method
Intercept any field or property
Intercept any event
Introduce interfaces
Introduce methods
Introduce events and properties
Introduce managed resources
Introduce custom attributes
Build-Time Validation
Attribute Multicasting
Aspect Inheritance
Add aspects using XML
Support for Async and Iterator Methods
Call target code from aspect
Dynamic aspect providers (composite aspects)
Dynamic advice providers
Initializes aspects at build time for fast runtime performance

Architecture Validation
Validates your code against pre-defined design rules
Validates your code against custom rules
Combines code validation with code generation
Extended Reflection API
Syntax Tree Decompiler
Build server integration
Advanced Reflection

Broad Platform Support
.NET Framework 3.5, 4.0, 4.5
Silverlight 5.0
Windows Phone (WinRT) 8.0, 8.1
Windows Store (WinRT) 8, 8.1
Portable Class Libraries 4.0, 4.5, 4.6
Xamarin iOS
Xamarin Android

Visual Studio Integration
PostSharp Explorer
Pattern-aware syntax highlighting
Pattern-aware tooltips
Coding Guidance
Code Editor Enhancements
Aspect Browser
File and Line Number of Error Messages

Support
Commercially supported
Fully documented
Outstanding run-time performance
Get started under 30 minutes

"We've reduced a lot of boilerplate thanks to PostSharp. Less code means less defects and bug-fixing."
Frederik Williams, Software Developer, Queue-it

Q&A

Below are some of the most common questions about PostSharp:

I've tried such a tool in the past and it was difficult to understand my code

PostSharp comes with Visual Studio tooling that ensures you understand where and how patterns are used in your code. PostSharp applies patterns during compilation without affecting your source code in order to keep it clean. PostSharp Tools for Visual Studio push this information directly into Visual Studio so you are always aware of these patterns.

PostSharp Explorer shows which patterns are used and how they affect the codebase. Pattern-aware syntax highlighting shows which code is enhanced by patterns. Pattern-aware tooltips show which patterns are applied to the current code.

Messing with MSIL feels kind of "dirty"

MSIL is a very stable and extremely well specified ECMA standard with several open-source implementations. MSIL evolves much more slowly than the C# or VB language, which allows PostSharp to remain so stable.

Microsoft Code Contracts, Microsoft Code Analysis and several other commercial tools also work on MSIL rewriting.

Much of PostSharp benefits is possible with dependency injection (IoC containers)

Dependency injection is like a tractor on the highway: it's a great tool but often not the best one for that job.

Proxy-based AOP, which is made possible by dependency injection and IoC containers, makes a step in the right direction. However, the technology it relies on (dynamic proxies) puts severe constraints on what can be done. Basically, you can only intercept interface or virtual methods. Therefore, the number of patterns you can implement using proxy-based AOP is very limited.

Additionally, proxy-based AOP forces you to dissect your application into components and interfaces of meaningless granularity, so you will find yourself changing your architecture (with dependency injection) to get a benefit that is inherent to the technology of dependency injection, but just a side effect of it.

Finally, proxy-based AOP does not work on all platforms and is much less efficient than PostSharp at runtime.

The only right way to have thread safety is to use purely functional languages

Purely functional languages are thread-safe because they strictly follow the Immutable pattern. PostSharp also offer the Immutable pattern along with other popular threading models.

PostSharp does not force you into a specific programming model. Unlike functional programming, which is mainly popular in academic circles and in some specific industry niches, PostSharp follows a pragmatic approach where thread safety is achieved through a combination of build-time and run-time

verification. Because PostSharp does not try to reach 100% provable soundness at build time, it can focus on providing the maximum thread safety commercially realistic in a business setting.

Note that PostSharp's standard of thread safety would be insufficient for operating system kernels, aeronautic/space software, real-time financial trading or control of nuclear power plants, but these pieces of critical software typically cost an order of magnitude more than typical business applications.

Compilation will be slower

Yes, as PostSharp introduces additional steps into the compilation, there is a performance cost. This is the same for e.g. custom tools run before C# compiler is executed such as XAML compiler. How large this cost is, mostly depends on how extensively PostSharp transforms the original program which mostly depends on how much PostSharp is leveraged. For comparison, PostSharp is generally a few times faster than FxCop, which is frequently run for every build in larger companies.

Do I have to replace my existing compiler?

No. PostSharp is compatible with your existing Microsoft compiler, providing the tooling and user experience you're used to, including light-bulb integration or errors and warnings displayed in the Visual Studio Error List.

"Before PostSharp, I experimented with several alternatives, but all of them shared a common critical problem: terrible runtime performance. PostSharp solves this by doing most of its work at compile time, so everything runs smoothly at runtime."
Daniel Crabtree, Director, Rekur Ltd.

Testimonials

*“PostSharp is **very easy to use**. Releasing developers from writing boilerplate code helps my team complete the features faster.”*
Bernd Hengelein, Software Architect, Siemens Audiology

*"PostSharp allows us to **develop our applications a lot quicker**, we have a lot fewer errors as we catch them at compile time, and it also enables us to **get new developers to the projects very quickly.**"*
Adam Greene, Lead Software Architect, Cognitive X Solutions

*“The new code is much less complex and much easier to maintain. This is what really **saves time and money** in the long run.”*
Daniel Wolf, Project Manager, mobileX AG

*“Thanks to PostSharp, we were able to localize over 95 percent of the game with just one line of code and **save potentially hundreds of man hours** over the lifetime of the game.”*
Yan Cui, Senior Backend Developer, Gamesys

*“By using PostSharp, we've been able to consolidate and **vastly simplify our code**, making it more readable and maintainable. It's allowed us to more consistently leverage code throughout the code base without having to continually rewrite the same functionality over and over again.”*
Mike Lawton,Manager of Software Engineering, DaProSystems Inc.

How PostSharp Compares

Please visit the page https://www.postsharp.net/alternatives to view the following comparative matrices that show how PostSharp compares to other tools and development approaches.

PostSharp vs. tools you already know

Benefit	PostSharp	Vanilla C# / VB	Refactoring Tools	Dependency Injection
Build automation for your own patterns with comprehensive toolkit	✔			
Ready-made standard design patterns implementations	✔			
Build thread-safe applications	✔			
Detect errors and non-compliance before code reviews	✔			
Leverage the skills of your most experienced developers to the whole team	✔			
Compatible with PostSharp	✔	✔	✔	✔

PostSharp vs AOP and meta-programming

Benefit	PostSharp Pattern-aware compiler	Old-school AOP AspectJ	Meta Programming	Proxy-based AOP	Dynamic languages
Enhance programs with simple new behaviors	✔	✔	✔	✔	✔
Enhance programs with advanced new behaviors	✔	✔	✔		✔
Verify programs at build-time against simple rules	✔	✔	✔		
Verify programs at build-time against complex rules	✔		✔		
Excellent run-time performance	✔	✔	✔		
Guarantee to produce correct programs at build-time	✔	✔		✔	
Easy to use by average application developers	✔	✔		✔	✔
Detect errors and non-compliance before code reviews	✔				
Available for C# and VB	✔			✔	
Visual Studio Tooling	✔				
Ready-made pattern implementations	✔				

Detailed Comparison Matrix

Features	PostSharp Pattern-aware compiler	Refactoring Tools Resharper	MSIL Rewriters Cecil, Fody	Niche Languages F#, Erlang	Dependency Injection Unity, Castle	Static Analysis Tools FxCop, NDepend	Frameworks, Components	PreEmptive Analytics
Does not reformat your code	✔		✔	✔	✔	✔	✔	✔
Makes source code semantically simpler and more concise	✔		✔					
Empowers every team member: architects, senior developers and application developers	✔		✔					
Reduces number of defects in your code	✔			✔		✔		
Improves architecture and makes application maintenance easier and less expensive	✔				✔			
Outstanding run-time performance	✔		✔	✔				
Works with your code – with minimal rewriting	✔	✔	✔			✔		
Get started under 30 minutes	✔	✔			✔	✔	✔	
Ready-Made Patterns								
Basic automatic INotifyPropertyChanged implementation	✔		✔					
Comprehensive automatic INotifyPropertyChanged implementation with support for nested objects	✔							
Automatic Undo/Redo implementation	✔							
Thread Safety for C# and VB	✔							
Automatically add logging	✔							✔
Automatically implement parent/child relationships	✔							
Automatically implement IDisposable	✔							
Thread Safety								
Immutable model	✔			✔				
Freezable model	✔							
Synchronized model	✔							
Reader-writer synchronized model	✔							
Thread-unsafe model	✔							✔
Thread-affine model	✔							
Actor model	✔			✔				
Aspect-Oriented Programming								
Creates automation for your own patterns	✔				✔			
Adds behaviors to interface or virtual methods	✔				✔			
Adds behaviors to *any* method	✔							
Adds behaviors to *any* field or property	✔							
Adds behaviors to *any* event	✔							✔
Introduces interfaces to target class	✔				✔			
Introduces methods, events, properties, managed resources, custom attributes	✔							
Initializes aspects at build time for fast runtime performance	✔							
Visual Studio Tooling	✔							
Architecture Validation								
Validates your code against pre-defined design rules	✔					✔		
Validates your code against custom rules	✔					✔		
Combines code validation with code generation	✔							
Support								
Commercially Supported	✔	✔		✔	✔	✔		
Cross-platform	✔	✔	✔	✔		✔		
Fully documented	✔	✔		✔	✔	✔		✔

Return on Investment

Let's analyze the return on investment of a mid-sized enterprise development team working on a business software:

Consider a 20 person team working on a 3 year project. Suppose the average cost per team member and per year is $100,000, all taxes and overhead costs inclusive (an understatement in many countries). That means that the total project cost is $6,000,000. According to empirical research, a 20 person team would produce approximately 430,000 lines of code during 3 years. This gives us an **approximate cost of $14 per line of code.**

While we cannot quantify all benefits of using PostSharp, customer data shows that using PostSharp, the number of lines of code required to implement a same set of features **decreases by 5% to 25%.** Let's take a conservative 10% reduction.

Since the cost of developing and debugging software is roughly linear to its number of lines of code, it means that we can expect the **total project cost to be $600,000 lower** when using PostSharp.

Now let's compare this to software license costs. Although the whole team benefits from having a smaller code base, only developers (suppose there are 15 developers, 4 testers and 1 manager) need to purchase a license. PostSharp is in a typical price range for development tools ($600 - $2000 per developer for the first year, 30%-40% maintenance fee for the next 2 years) so we can estimate that the total licensing cost is $20,000.

Let's add the training cost. Typically, just the architecture team (say 2 people) need to acquire a deep understanding (say 3 days). The rest of the development team (13 people) just need a superficial understanding of the concepts (say ½ day). Thus the cost of training the team would be under $10,000.

We have a total cost of adoption of $30,000 for a return of $600,000. Despite all the conservative projections, this is still a **19x return on investment**! And we just took into account the quantifiable benefits. Hidden multithreading defects, late arrival to market or security leaks can cost you millions and even billions of dollars, and pattern–aware compilers help address these problems too.

Most software has nowhere near this kind of return – making PostSharp one of the most profitable approaches you can consider.

> *"Over the years, PostSharp has helped us save over tens of thousands lines of code."*
> **Yan Cui, Senior Backend Developer, Gamesys**

Screenshots

Please visit the page https://www.postsharp.net/pressroom to download the screenshots.

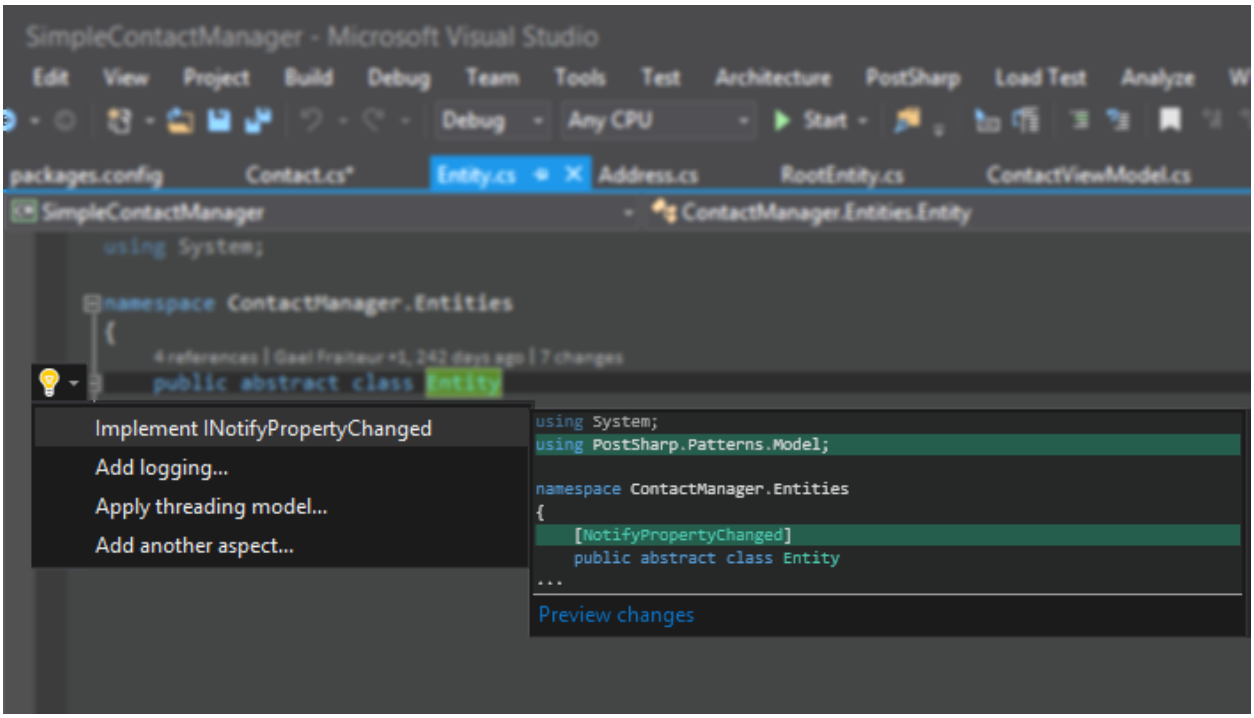

Figure 1 Implementing a pattern can be as easy as selecting an action in the Visual Studio lightbulb.

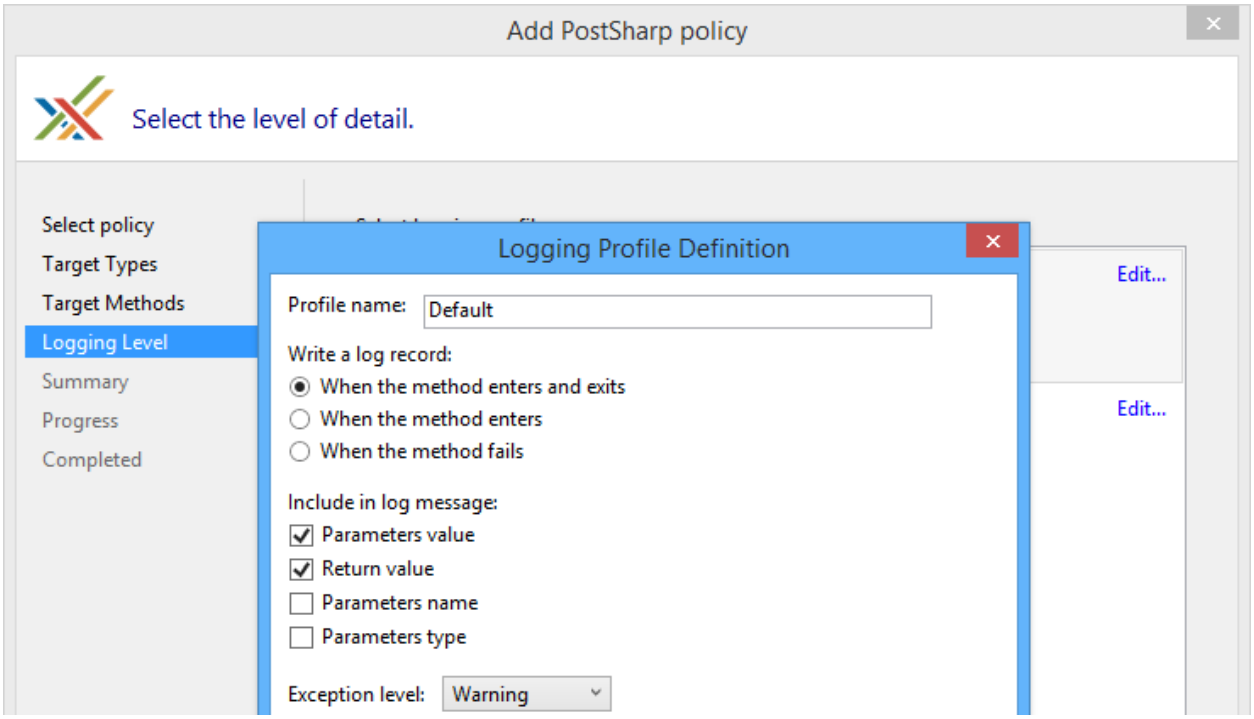

Figure 2 A look of the wizard that helps add logging to your project.

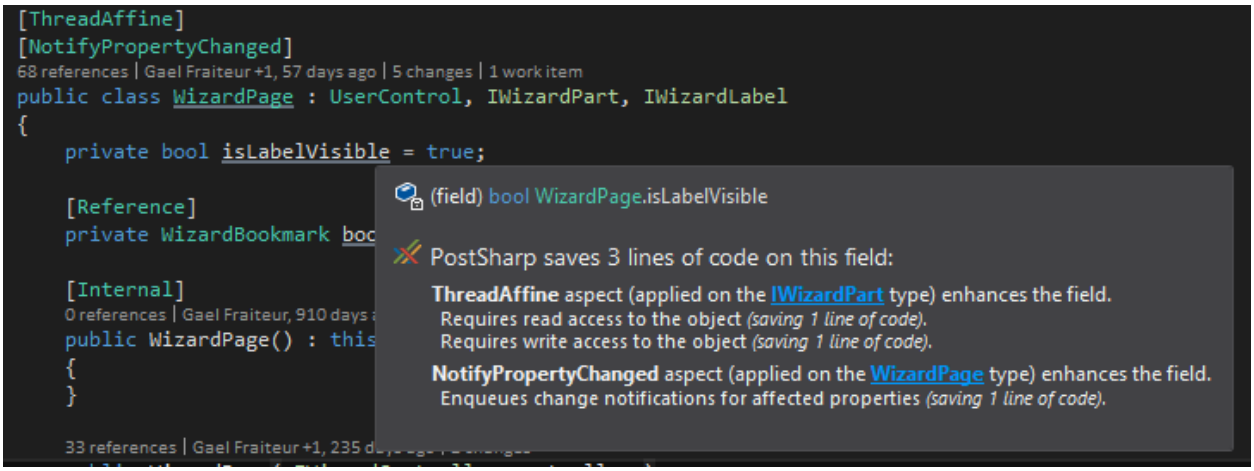

Figure 3 The tooltips show you which aspects have been applied to your code.

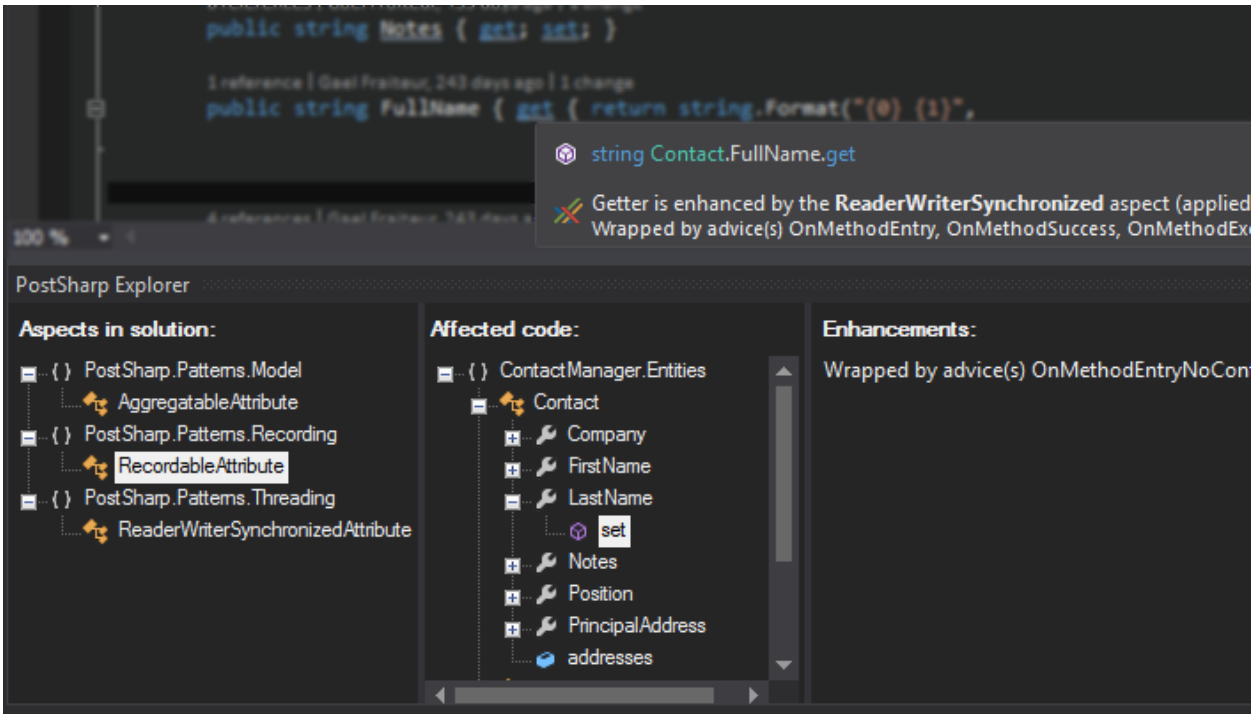

Figure 4 PostSharp Explorer shows which aspects are present in your solution and to which targets they have been applied.

PostSharp Metrics

Project	Manual KLOC #	Saved KLOC #	Saved %
PostSharp.HQ	4.4	0.9	21%
PostSharp.VisualStudio	13.0	2.0	16%
PostSharp.VisualStudio.Package.VS11.0	2.2	0.2	10%
PostSharp.VisualStudio.Package.VS14.0	4.3	0.6	13%
Total	23.9	3.7	16%

PostSharp Metrics | Error List | Output | Find Results 1 | Find Symbol Results

Figure 5 PostSharp Metrics tool window shows how many lines of code you wrote manually and how many you probably saved thanks to PostSharp.

PostSharp Editions & Pricing

Bundle Prices

	PostSharp Express	PostSharp Professional	PostSharp Ultimate
Price	Free	$369	$669
Summary	Free but limited to 10 classes per project	The #1 aspect-oriented framework for .NET.	The complete and unlimited product.
PostSharp Aspect Framework Automate implementation of custom patterns	Limited	Unlimited	Unlimited
PostSharp Diagnostics Library Logging	Limited	Unlimited	Unlimited
PostSharp Model Library INotifyPropertyChanged, Undo/Redo, Code Contracts, Aggregatable, Disposable	Limited	Limited	Unlimited
PostSharp Threading Library Threading Models, Thread Dispatching, Deadlock Detection	Limited	Limited	Unlimited
PostSharp Architecture Framework Automate validation of hand-written code against guidelines	Not Included	Unlimited	Unlimited
Premium Support 1 year of premium support and free upgrades	Not Included	Included	Included

Limited means maximum 10 enhanced classes per project and 50 per solution.

Library Prices

Pattern libraries are available for separate purchase for the following process:

Library	Price
PostSharp Diagnostics Library	$44
PostSharp Threading Library	$259
PostSharp Model Library	$169

Licensing Model

All commercial licenses are floating (non-named) and perpetual licenses.

How to Get Started

Five simple steps to get started PostSharp:

1. Download and install PostSharp. In this step, you are actually just installing the user interface: PostSharp Tools for Visual Studio.
2. During installation, choose between a 45-day trial of PostSharp Ultimate, the free PostSharp Express, or you can enter a license key for a professional edition of PostSharp.
3. After having installed the PostSharp Tools, in Visual Studio, move the caret to the class or method to enhance and choose the pattern from the light bulb or smart tag.
4. PostSharp is deployed as a NuGet package. During installation, PostSharp edits your project file and inserts itself in the build process. Therefore, there is no friction with build servers.
5. Build your project. PostSharp post-processes the output of the C# or VB compiler. It opens the intermediate assembly, adds the behaviors required by the aspects, then produces a new assembly.

Useful Resources

Product information	https://www.postsharp.net/product
23-Minute PostSharp Demo	https://www.postsharp.net/documentation/video?id=143656221
Case Studies and Customers	https://www.postsharp.net/customers
Reference Documentation	http://doc.postsharp.net/
Code Samples	http://samples.postsharp.net/
Press & Media Resources	https://www.postsharp.net/pressroom

Contact Information

Primary Technical Contact

Gael Fraiteur
CEO and Principal Engineer
gael@postsharp.net
+1 866 576 5361

Press Contact

Iveta Moldavcuk
PR Manager
iveta@postsharp.net
+1 866 576 5361

About PostSharp Technologies

PostSharp is the #1 best-selling pattern-aware extension to C# and VB. It allows developers to eradicate boilerplate by offloading repeating work from humans to machines. PostSharp contains ready-made implementations of the most common patterns and gives you the tools to build automation for your own patterns.

PostSharp is trusted by over 50,000 developers worldwide and over 10% of all Fortune 500 companies including Microsoft, Intel, Bank of America, Phillips, NetApp, BP, PricewaterhouseCoopers, Volkswagen, Hitachi, Siemens, and Oracle rely on PostSharp to reduce their development and maintenance costs.

For more information, please visit https://www.postsharp.net/.