POSTSHARP

CASE STUDY: mobileX AG

“Sure, we achieved some considerable savings in terms of LOC. But more importantly, 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

mobileX AG

A mobile workforce management solutions provider reduces code complexity and improves code maintainability in its applications with PostSharp.

Summary

When the lead developers at mobileX began a huge refactoring project – migrating the company's codebase to the .NET 3.5 framework then rewriting their flagship application from WinForms to WPF – the biggest challenge was to improve code maintainability by reducing code complexity. They chose PostSharp for its out-of-the-box solutions that eliminate code repetition without changing application architecture.

"We just wanted some out of the box solutions we could apply to some classes and not others without a huge change to our existing code base."

Daniel Wolf
Project Manager
mobileX AG

Lead developers migrate main app to .NET 3.5 framework

After struggling with the company's main product, an application that grew into a monolith over the years, the lead developers at mobileX received approval from management to upgrade to the .NET 3.5 framework and take advantage of higher-level APIs like WPF to improve client-facing customizations.

"We just arrived at a point where many things needed improvement," says Daniel Wolf, Project Manager at mobileX. "It was far too hard to change individual bits of it for our individual customers. At that point we got permission to spend about half a year completely redoing much of the main part of the application and another half a year reworking what we already had and refactoring it."

Custom application extensions required a lot of repetitive code

Daniel's team builds the front end of the company's main application. His team also develops custom extensions for clients and the logic behind them can be quite complex depending on the number of dependencies and validation rules between values controlled by different parts of the UI.

As the number of diverse dependencies between inputs rises they often become multi-layered, where the input in control A determines the number of input selections available for controls B and C. So, when the value of control A is changed, controls B and C need to be updated in exactly the right order to avoid erroneous results.

Using a traditional approach, the team had to be very careful with any implementation, attaching and later detaching, event handlers. Because attaching and detaching takes place at different times, it was easy for team members to forget to detach – leading to memory leaks.

Why mobileX chose PostSharp

During that time, Daniel and the other lead developers looked at several technologies they had not used before. They soon came to the conclusion that Aspect-Oriented Programming was the best solution for the job at hand but they weren't keen on how dynamic proxies forced developers into their architecture rather than adapting to the developer's architecture.

That's when they discovered PostSharp and the benefits of compile-time weaving.

"There was no other product on the market that could do what we wanted to do," says Daniel. "We wanted aspect orientation to be one tool among others in our toolbox. We didn't want to have to change anything to use it," he said. "We just wanted some solutions that were out of the box and that we could just apply to some classes and not apply to other classes, and they would not mean a huge change to our existing code base."

How mobileX implements PostSharp

Using PostSharp, the team at mobileX built a number of custom aspects to cover important use cases including INotifyPropertyChanged, AutoWiring and Const.

INotifyPropertyChanged

In GUI programming, one very repetitive task is implementing the INotifyPropertyChanged interface and making sure all relevant properties throw an event when changed. This simple aspect can be applied to any read-write property and will automatically add the required glue code, including the test if the new value actually differs from the old one.

"This attribute takes care of those hundreds of simple properties without too much logic in them," says Daniel. "It is a clean solution that allows us to write succinct code and it dramatically improves readability, since it allows us to use automatic properties most of the time. Without it, all those properties would have to be implemented explicitly. In the past, the required backing fields used to clutter our code without adding any value.

"Sure, we achieved some considerable savings in terms of LOC. But more importantly, 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

AutoWiring

This aspect is the "bigger brother" of the INotifyPropertyChanged aspect. You apply it to a property and tell it which values the property depends on by giving it a list of binding-style property paths. It then tracks changes to all properties along those paths, much the same way WPF does. If any of these input values change, the property value is re-evaluated, and a change event is raised if appropriate.

"Handling change notifications used to be a real challenge," says Daniel, "especially in those cases where changes to one property need to trigger a change in another property, which again needs to trigger changes elsewhere. We used to end up with lots of hard-coded dependencies, where one property would raise change events for a whole bunch of other properties that depended on it. It was hard to get right and a maintenance nightmare afterwards. Now, with AutoWiring, we simply declare what values a property depends on. And it just works – no matter how complex the dependencies."

If the team were to refactor the code, it would consider using ready-made aspects shipped with PostSharp:

"It seems PostSharp Model Patterns Library now offers a similar feature out-of-the-box, with even less manual work. Unfortunately, the feature was still in development when we needed it" says Juan, a colleague of Daniel.

Const

This attribute can be applied to any property that only has a getter. On first access, it caches the return value. On every subsequent access, it simply returns the cached value.

"This aspect may seem trivial at first, but turns out to be a real asset when it comes to writing efficient code," says Daniel. "Many properties need to perform expensive operations to calculate their value. In the past we either pre-computed and assigned these values in the constructor, which meant a delay even if the property was never actually accessed, or we calculated the value within the property getter, which helped initial load time but meant an additional delay on each property access – even for properties we knew would not change. If we really wanted to maximize performance, we had to use additional backing fields to implement lazy loading. Now, we simply put all the logic into the property's getter and decorate it with the Const attribute. And just like that, we get lazy loading and optimal performance."

Simpler code that's easier to maintain

The team calculated a total savings of over ten thousand lines of code (LOC) on the project by using PostSharp, but believes the real savings come from reduced code complexity and improved maintainability.

"Can I look at some piece of code and immediately understand what it does and how it works? Is it easy to spot errors and fix them without breaking code elsewhere? That's what really counts. And this is where PostSharp shines," says Daniel. "Sure, we achieved some considerable savings in terms of LOC. But more importantly, the new code is much less complex and much easier to maintain. This is what really saves time and money in the long run."

SharpCrafters s.r.o.
Namesti 14 rijna, 1307/2
150 00 Prague 5
Czech Republic

US: +1 866 576 5361
CZ: +420 270 007 790
www.postsharp.net
info@postsharp.net