Sometime late in 2009 or early in 2010, Microsoft will release a powerful new development tool: .NET 4.0, accompanied by C# 4.0 and Visual Studio 2010 (VS2010).

Buried in this rather large release will be a handful of power programming tools and technologies that are tailor-made for experienced .NET developers, including:

- Extensions to C# in two forms:
  
  - The ability to invoke scripting objects, Python and Ruby objects – and even COM objects -- dynamically through the Dynamic Language Runtime (DLR). This feature will even let you invoke Managed Objects dynamically without having to write Reflection Code.
  
  - Support for named and optional method parameters, eliminating the need for multiple convenience overloads in your APIs, and making it easier to invoke COM objects that use indexed and default properties.

- The F# Programming Language: a 1st Class Managed Language designed from the ground-up with Functional Programming in mind (but created to let you additionally leverage dynamic and parallel programming as well). F# will become the one accessory that every C# developer will need for implementing complex algorithms with the elegance and clarity of LINQ, but addressing programming problems that LINQ could only dream of dealing with.

- Parallel.NET: extensions to run LINQ queries, individual methods, entire tasks – even for and foreach loops -- on other threads and processors of a multi-processor system or multi-core processor, and do so without having to re-architect, re-partition or even create a thread yourself.
In this session, Richard will review and explore these features as a brief preview of his multi-day "\.NET 4.0 / C# 4.0 / F# 1.0 Power Programming BootCamp", a hands-on course for .NET developers, which will launch at the same time as VS2010 and .NET 4.0.

We'll start with a brief review of .NET 3.5, C# 3.0 and LINQ: these technologies, available since late 2007, are now the foundation on which the new .NET extensions – in particular, the Parallel extensions, C# 4.0 and F# – will be built. If you're not already familiar with Lambda Expressions, Extension Methods, LINQ queries and abstractions based on IEnumerable<T>, you'll quickly find yourself lost trying to take advantage of .NET 4.0. So in this review, you'll gain a thorough grounding on the core concepts required to move forward.

We'll follow by turning our attention to C# 4.0 and the Dynamic Language Runtime. Already in use in in other Managed dynamic language environments, the DLR is being be added to .NET to let any Managed Language utilize dynamic object binding at runtime (and – when you need to -- avoid the static binding provided by the C# compiler). While C# remains a statically typed language, C# 4.0 includes extensions to support the DLR's dynamic object binding, letting you invoke DLR language objects such as: those developed with IronPython and IronRuby, unmanaged COM code components (without requiring wrappers or Primary Interop Assemblies), and even Managed objects (but which are dynamically invoked). The import of this last point: if you've ever written Reflection code to create a Managed Plug-In architecture, you'll now be able to use DLR techniques to dynamically invoke methods and properties on unknown Managed objects without writing a line of Reflection code.

We'll wrap up this section with a study of new C# 4.0 features that support named and optional method parameters: you'll be able to avoid creating dozens of convenience overloads by defining default method parameters which can be omitted or passed by name (as well as by order). Finally, we'll look at how variance has been added to Generic Type Parameters on Delegates and Interfaces, and how this facilitates better quality code in C#. When we complete this section, you'll have learned not only how to leverage C# 4.0’s dynamic programming capabilities, but how other aspects of C#
have changed and now you can seamlessly integrate into your existing applications and components.

Next, we’ll move to F# -- but before jumping in, we’ll first review the tenants of functional programming, for which F# is specifically designed: how it works, why it works, and why you’ll want to apply it – where appropriate – in your .NET applications and components. We’ll look at how functional programming espouses stateless algorithms and immutable data (and eschews stateful algorithms and mutable data), and how eliminating these side-effects will improve the quality of your code. Then, we’ll turn to F# itself. Offering type-safety as well as type-inference, expressiveness but high readability and maintainability, F# will be the perfect accessory for every C# programmer. It will let you implement complex algorithms in ways that C# cannot, but leverage their implementations from C# itself. Unlike C# -- which was designed as a managed language that utilizes features of C++, Java and Delphi, and which has gradually been extended to include both functional and dynamic language features -- F# is a derivative of OCaml, which was designed ground-up with functional programming in mind. Adapted for .NET, F# adds dynamic language and parallel programming features -- all built-in. We’ll make a formal review of the features of F#, its notation and how it differs from C#, and we’ll compare F# and C# constructs side-by-side so you can quickly get familiar with the former’s terse, but expressive power and elegance.

The goal is NOT to entice you to move to F#: instead, you’ll learn how F# is an essential tool for C# programmers wanting more maintainable, extensible but elegant solutions in any tier. You’ll find that you can use F# to write more effective class libraries to be called from C#, letting you leverage functional and dynamic programming power beyond that available in C# 3.0 or 4.0. And by the time we’re finished, you’ll have learned how to leverage the power of F# in existing C# 3.0 (or later) applications. Finally, we’ll dig into the Parallel Programming Extensions. We’ll start with the essential problem: how do you add multi-processor – or even multi-threading –
capabilities to your existing .NET applications without having to re-architect them, isolating state that’s used by individual threads, managing state that’s shared across multiple threads, and ensuring that all threads are properly initialized, managed and – most important – terminated? And even if successful, how do you do this in a way that ensures your code will really run correctly with a 2nd core or processor on-board: after all, some multi-threaded code will fail once its run on a box with more than 1 processor, where any thread can actually run in parallel (and not serial) fashion.

The Parallel.NET extensions are designed to address exactly these problems, letting you designate – in a clear, concise, declarative fashion – everything from individual methods, LINQ queries, to foreach and for loops that may run on threads and processors other than the main thread (or default processor) that’s currently being used by the application. We’ll also look at the Managed Task Library and the thread-safe collection types provided by Parallel.NET. By the time we’re through, you’ll have a clear idea of how to leverage these techniques to run virtually any code you designate on other processors or other cores of your processor (if either is available), letting you add true multi-processing capabilities to existing managed code libraries and applications.

Come to this session to get a taste of what’s to come – and then return when VS2010 is released and take the full course for a comprehensive, hands-on immersion in these great new power programming technologies!