

The Road to .NET: The EC Wise Experience

© 2003, EC Wise, Inc., All rights reserved

This white paper will discuss our experiences refocusing our development efforts on the .NET platform, and working with a client that used us as a catalyst for refocusing their development efforts to .NET. Since our primary motivation to adopt .NET was efficiency, we will emphasize areas where using .NET has enhanced our efficiency and that of our clients. Specifically, we will discuss how .NET programming leverages programmer time, how it enables development work to become more responsive to customer requirements, and the advantages of managing .NET development projects. Before diving into that discussion, let's discuss the future of software development and present some arguments for why refocusing your development efforts on the .NET platform sooner, rather than later, is a good idea.

The future of development

Development organizations live on a technology treadmill. Software development and deployment models continue to evolve. Software development methodologies have gone through a series of "orientation" changes: we've seen structured programming, functional programming, object oriented programming, and service oriented programming, with aspect oriented programming on the horizon. Deployment has gone through the centralized mainframe, PC island, fat client server, thin client server, and web phases, and now appears to be heading into a web services phase. Much of the effort that development organizations expend is on retooling to bring their IT assets in line with whatever architectural model is contemporary. These changes are not serendipitous; each evolutionary change can be demonstrated to have cost, scalability, and functional advantages over earlier models.

Over the past few years, there have been major developments in the nature and scope of computing capabilities, based on technology development that occurred during the Internet boom. However, the economic chill that settled in since then has limited the extent to which corporations have leveraged these capabilities. In addition, there have been a series of legal and regulatory clouds that have mitigated against wholesale leveraging of new technical capabilities, particularly with regards to digital content distribution and healthcare related initiatives. Microsoft launched its .NET initiative during the boom years, largely in response to the widespread adoption of Java and Unix as a platform for Internet computing. While the end of the boom reduced the urgency of adopting new models, it did not make them unnecessary. The genie is out of its box: trends like integrated value chains with real time interactions between multiple entities and pervasive computing are here to stay. When the economy recovers, the urgency to support these trends will grow. Organizations that start to lay the technological groundwork for that eventuality now will be better positioned when it happens. The good news is that you can start implementing new models now and get benefits from them immediately.

There are several business drivers influencing the adoption of new technology infrastructures. As we already alluded to, the expectation of interoperability among the corporate systems in a

global value chain of component producers, assembly manufacturers, distribution channels and end customers is a significant driver. Both transient ad hoc working relationships and longer term merger and acquisition activity have similar impacts. All require that systems expose interoperability at the logic level, rather than simply at the data level.

A driver that's here today is pressure on IT and development organizations to reduce costs. Another driver is increasing ability to provide information and business services to a mobile workforce, through communications networks that enable deployment of relatively rich applications to mobile devices. The tension between heightened security concerns and increasing demands for interorganizational collaboration will drive changes in the way software is built and deployed. In the longer term the increasing miniaturization of intelligence is likely to cause disruptions in a variety of industries.

With these drivers in mind, it's inevitable that development organizations are going to change the way they do things. Many already have. Four years ago, we advocated and demonstrated models for building adaptive user interfaces that could support heterogeneous user communities using different types of devices. Since that time, many organizations have built applications that run on servers and generated mark up language based front ends (HTML, WML, VoiceML, etc.) for consumption by diverse client devices; some also used Java as a cross platform language.

There have been a variety of standards based and proprietary approaches to solving the interoperability puzzle, but XML Web services have been chosen as the standard and are likely to see widespread adoption over the next few years. Moving to real time operations will leverage XML web services, but will require the implementation of internal capabilities such as real-time analytics and sophisticated rules based systems. Security issues associated with all of these initiatives will have to be addressed both at the infrastructure level, by improving the performance, interoperability and scalability of certification systems, and at the transaction level, by developing effective and usable means of ensuring their integrity and confidentiality.

Why .NET?

The genius of Java was the introduction of a common software execution platform that could run on any device. This made Java a language that could be used to develop programs that could run on many different devices, not just Windows and Unix computers. In addition, Java incorporated facilities for building distributed applications and managing TCP/IP communications sessions that had previously required adopting and learning another layer of infrastructure (e.g. CORBA) and integrating libraries that added significant cost and complexity to the standard development platform. Java came to market relatively early in the Internet application cycle, and gained a significant marketshare among developers of products and services for the web. EC Wise was among the organizations that adopted Java, taking advantage of all these features, and considering it to be a highly productive, well architected object oriented language as well.

Microsoft has mimicked that approach with the .NET framework, although it relies on the availability of a Microsoft Windows operating system as its host. Microsoft will ensure that the availability of such operating systems will be widespread, and has delivered versions of Windows that drive game systems (Xbox), TV set-top boxes, handheld devices, automotive in-dash systems, and tablet PCs, in addition to computers of all sizes. Watching the growing adoption of Java, and the replacement and componentization of many legacy systems during the latter part of the last decade, Microsoft began to really understand that applications built on its platform would have to interoperate with other systems. Microsoft emerged as the leading technology advocate of XML Web services, using the technology as a leverage point to ensure that .NET could flourish in a world where centralization of software services on non-Microsoft platforms was a fact of life.

Microsoft put so much emphasis on Web services out of the chute that the message about other .NET benefits may have been muffled. Several factors drove our early adoption of .NET as a development and deployment environment. Foremost was the introduction of the new C# language, the language Microsoft introduced specifically to take the best advantage of the .NET platform. The designers of C# leaned much from Java, and built a language that was much simpler than C++, and much more powerful than Visual Basic. We found that developers coming to C# from Java, Visual Basic or C# could become productive relatively quickly, and that they produced code much more quickly than they had in C++, with performance that was better than Java. Even more importantly, we were able to quickly design and build custom user interface components and deliver much richer more flexible user interfaces than we could have with Visual Basic. Finally, we have found that we can manage .NET projects more effectively, providing clients with a higher degree of visibility into project status and trends.

Developer productivity in C#/.NET

C# Language advantages vs. Java

We moved to C# after spending a few years developing products and business applications in Java. Our development team has found a number of language advantages of C# over Java. One such advantage is C#'s support for *structs*, which are simple types defined programmatically like classes, but with the *struct* keyword. As value types, structs are more efficient than full classes, because they are accessed directly in memory. Enumerations in .NET provided the programming benefits of constants, with the added benefit that they are a type within the framework with a full set of methods. Having to explicitly indicate what methods may be overridden, and indicating the overriding methods with the *override* keyword provides a level of safety over the Java approach to overriding base class methods. Finally, the event model in .NET is much simpler than in Java, requiring only that you defining the event handling method, and attach and detach the delegated method to the appropriate events using *attach* and *detach* delegate operators.

.Net runtime advantages

.NET performs automatic garbage collection, to reclaim memory automatically when objects that are no longer being used. This removes a significant memory management burden from

application developers. .NET revolutionizes deployment of Windows applications, with a model that completely isolates each application from every other application. While you may explicitly share code modules, the default is that components are not shared, even if they have the same name and contain the same set of methods. In this case, different applications can use different versions of the same component, and the .NET framework enforces versioning policy. Code access security is a mechanism that controls how a software component may interact with protected resources and operations. To determine whether code is authorized to access a resource or perform an operation, the runtime's security system walks the call stack, comparing the granted permissions of each caller to the permission being demanded. If any caller in the call stack does not have the demanded permission, a security exception is thrown and access is refused.

Writing code and integrating multiple languages

The Visual Studio IDE understands XML, XML schema, XSLT, C#, C++, VB, ASP, CSS, among other languages, and provides code completion, syntax highlighting, and debugging support for them. We can use all of these languages in a .NET application, (and have used all except VB), switching frequently between languages. We could also expand the available languages, by using other .NET compatible languages such as Eiffel, which would allow us to take advantage of design-by-contract, etc. in a large class framework. In addition to the seamless ability to integrate .NET languages, .NET provides a command line utility that processes COM components and creates the .NET code necessary to enable our .NET applications to use them as if they were native .NET assemblies. This has enabled a significant amount of existing code to be preserved.

Integration of database, middle tier, UI tools, web, and desktop

The Visual Studio IDE automatically creates much of the code required for connecting to databases, based on inclusion of database components in the forms that you are building, and interrogates SQL Server or XML to determine the necessary structures and parameters. .NET and the IDE make it easy to create web pages where almost all of the logic is in “real” .NET classes, thus they are more easily debugged and unit tested.

ADO .Net

For data management, .NET provides the ADO .NET system of data management classes. Unlike traditional ADO and previous Microsoft data access technologies (as well as Java's JDBC), ADO.NET is much more than a set of database access APIs. Rather, they are a set of classes that allow developers to define and manipulate local in-memory tables of data, and the relationships among those tables. Developers can assert referential integrity rules and other constraints on in-memory data, and can search, sort, and insert data. These capabilities eliminate much of the need for custom classes to mirror database structures. .NET's managed providers, such as OleDbDataAdapter provide the connectivity functions necessary to move data between in-memory representations and disk based persistent storage systems.

Enhanced ability to respond to Clients

Faster prototypes

For Web applications, .NET's built-in Web Controls make it relatively easy to create data tables, links, validation, database updates, etc., and get an alpha version of an application running. Once you have a running application you can work with your clients to validate that it fulfills their requirements. From there you can use style sheets to customize the presentation and replace some of the basic controls with custom logic, increasing the visual richness and interactivity of the application, and incorporating more refined business rules.

XML Focus

.NET uses XML as a native data transport mechanism. One of the benefits of this is that it vastly simplifies report creation. For instance, in the product we're building for OnRain, the purchasing and usage reports simply call a stored procedure that returns data in an XML document, then invokes an XSL transform that applies the formatting and style rules for the report display. Reports produced this way can be created quickly, and maximize code reuse, requiring no procedural code, custom reporting data structures, explicit HTML generation, etc.

Custom Controls

The last few versions of Visual Basic have allowed custom control creation, but in .NET, the process is much more reliable. For instance, we built a financial reporting application in which our developer was able to create several custom charts in a few hours, and easily combine them to create subscreens and then entire screens. This enabled us to change the presentation rapidly as we got a better sense of what the client wanted.

Versioning

With the versioning capabilities of .NET assemblies, you can avoid DLL hell. Java had this advantage as well with its concept of packages, but .NET assemblies are even easier to deploy and maintain. You can just copy a code library to a directory, and an application resident in that directory can use the library. There is no registry to maintain. This makes it much easier to have several concurrent versions of an application available to the customer.

Creating Web services

Web services are one of the key selling points for .NET, and Visual Studio and the .NET framework make it very easy to create them. ADO can analyze XML or SQL schema and automatically generate the necessary Web services calls, classes etc. Using this facility, you can make a skeletal Web service or a consumer of an existing Web service in a couple of minutes.

More manageable projects

Training and Learning Curve for Java and C++ Developers

C# is an enhanced Java. The only thing missing is the exceptional depth of the Java libraries, which have developed over the course of several years. Many of these libraries provided specialized functionality that address programming programs that are only rarely encountered. So far our development team has not found the reduced scope of the .NET environment to be a handicap. Where .NET does not specifically offer solutions, we have been able to integrate off-the-shelf or custom developed C++ code into our .NET applications to provide them. The EC Wise development team is about evenly split between programmers doing serious Java and serious C++ coding prior to our adopting C#; all have found the transition to be pretty easy.

Scoping features

Using C# as our programming language supported our agile development process, which gave our clients a better sense of progress and increased the frequency of product deliveries and the quality of feedback. C# gave us the ability to have a developer build at least one, and sometimes more than one substantial feature within the scope of a one or two week agile development iteration. Once we understood how long features of various complexities would take, we were able to build project plans that called for sets of features to be delivered at specific times, within a context of weekly or biweekly deliveries. Any inability to build a scheduled feature within a specified iteration served as an early warning that we had issues to address. At the end of each iteration, we delivered an integrated build that the client and users could review and give us immediate feedback on the effectiveness of the implementation from a usability and requirements perspective.

Team Coordination

.NET languages lend themselves nicely to team development models. The clean separation between components facilitates dividing programming tasks among team members in a relatively granular way. Tight integration with database code makes it very feasible to move transactional code into the database, and transactional programming to database specialists. Visual Studio provides very sophisticated source code documentation facilities, which together with the high degree of organization and readability of the code itself, enable offloading of much code documentation to a technical writer, which in turn makes it easier to share code among team members.

Application Maintenance

Maintaining .NET applications is much easier than maintaining code in earlier languages, for a variety of reasons. For one thing, there should be a lot less code to write in the first place, thus less to maintain. With .NET, Microsoft introduced a number of programming conventions that most .NET programmers follow, making it pretty easy for one developer to understand code that another wrote. The language structure helps a great deal, because working classes can be extended by specialization, delegation, aggregation, and other mechanisms that avoid having to change working code. Finally, the flexibility of the Visual

Studio IDE improves maintainability with features like code outlining, automatically jumping to references, and the fact that it supports a wide variety of languages relatively seamlessly.

Conclusion

Over the past two years, we have developed a wide variety of Web and Windows software applications with C# and the .NET framework. We have found that this set of technologies empowers us to provide better solutions to our clients more quickly than we could with previous technology platforms. For Windows applications, C# and .NET provide the best set of tools that we have ever seen, offering 80 – 90% of the power of Visual C++ with none of the pain associated with C++ Windows API programming. For Web applications, ASP.NET has solved performance and debugging challenges that were endemic to earlier Web application development platforms. As clients continue to demand the ability to collaborate more effectively within and between their organizations, and as devices of various form factors continue to proliferate, we expect the advantages of working in .NET will only increase.