Introduction

Application development is usually faced with several critical factors: time to market, ever-increasing quality, and cost of development. Time to market is critical in that any project that an individual or company may have in mind is probably already in the works elsewhere. When that is the case, everything comes down to being the first to deliver…even if it falls short. A Gartner Research study showed that competing products offering similar features will take the most market share if released sooner. The study further added that the product released first, even with fewer features, typically builds market share faster. The lesson of that study is that it is important to release a 1.0 version of a new product concept as quickly as possible and then follow up with feature add-ons over time.

Product consistency and quality are also critical to the success of products making their debut. Applications with modules that look differently can undermine the application user's confidence in the product. The lower their confidence in the application, the less they use the product and come to depend upon it. The same can be said at the code level. The more modules that are implemented consistently, the easier different developers can step in to investigate and resolve problems.

Cost of development is often measured by productivity, or how much code is created to accomplish specific application tasks. The best development platform for you to use will offer the most services related to your application. This has often been measured by the size of a platform's class library and its flexibility. Because the .NET Framework is rich with data access, presentation, communication, and networking services, it can save the enterprise developer a significant amount of time that would otherwise be required to build that foundation. In addition to providing a large resource library of functionality within the .NET Framework, Visual Studio .NET provides many time-saving tools. The integrated form and component designers save a lot of time when building Windows or Web forms. The IntelliSense feature saves time searching help files to obtain quick property and method information. Furthermore, the build manager makes configuring and building projects faster. Together, the .NET Framework and Visual Studio .NET significantly accelerates the time required to build complex business and enterprise applications.

Supporting Software Tools

The application development frameworks presented within this book place a lot of emphasis on business object classes. Chapter 2, "Accessing Data in ADO.NET," introduces business objects and explains their roles within the application

framework. Although this is a scalable and flexible solution, there is one setback. It requires several C# code classes for each business object. In an effort to minimize that coding work, I have implemented an add-in for Visual Studio .NET 2003, called Business Object JumpStart.

After you download and install the Business Object JumpStart, you can seamlessly integrate it into Visual Studio. Just select Tools ➤ Business Object JumpStart from the Visual Studio menu and enter the attributes of your business object, as shown in Figure 1.

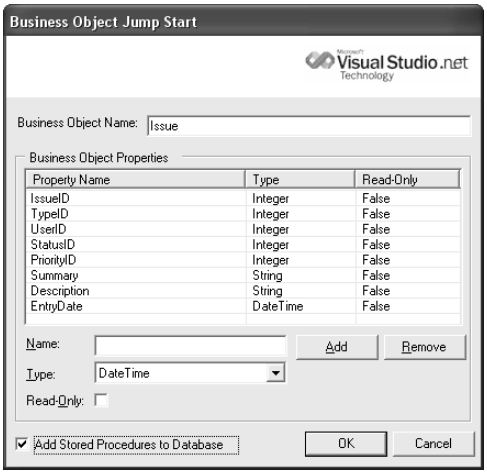

Figure 1. Adding business objects to your existing solution

After defining the new business object, click the OK button and take a break as the Business Object JumpStart adds a new class library project to your solution, creates all of the business object code in C#, and creates the database stored procedures that bind the business objects to the database. This handy tool trims a lot of mind-numbing coding from your development project and is based upon the examples within this book.

You can download the latest version of the Business Object JumpStart from the Downloads section of the Apress Web site (`http://www.apress.com`), or you can search the CNET Download.com Web site (`http://www.download.com`).

Planning Application Development

Building an enterprise application involves several steps. Chapter 1, "Introducing .NET and Enterprise Architecture," presents many of the specifics, especially with the types of documents you need to produce in preparation. Regardless of the specific documents you choose to implement, bear in mind

two fundamental concepts: clear planning and consistent implementation.

Clear application planning is required so that everyone on the development team is working toward the same goal. The entire team must understand the application functionality. This is typically reflected within a business requirements document, user interface definition, and database schema definition. In addition to these fundamental documents, a high-level application design, a component-level design, and even individual class definitions are helpful, especially when parts of the application development will be outsourced to a different team.

Consistent implementation at the application level and the code level is just as important to the finished product. From the application level, it is important to take a close look the application's basic functionality. Try to identify areas of reuse that might be packaged into components. Also, applying style templates at the forms level will help ensure that each form is implemented consistently across the application. At the code level, specify coding guidelines for developers to follow. This should include consistent naming conventions for classes, enumerations, methods, properties, class variables, database tables, and database stored procedures. At the user interface level, this should include design templates describing how form controls must placed, which fonts must be used, and how controls should be sized.

With careful planning and a solid investment in application design, team members can work individually and make intelligent decisions that result in a great product.

Adopting an Application Framework

Every project of a reasonable scale should have an application framework. An application framework can be as simple as a set of templates or as complicated as a fully integrated code generator. An application framework is essentially a clearly defined pattern for building pieces of the application. Usually, the intention is that an application framework saves development time. The framework itself provides 80 percent of the application functionality, such as data access, presentation, and so on. The remaining 20 percent is your domain-specific application code. An important factor to be aware of is to not select a framework that completely locks you into it. A framework should be powerful enough to get you started but also flexible enough for you to outgrow it.

The frameworks presented within this book meet that expectation. They provide a great starting point for your application development, but they do not necessarily lock you into them. One way to identify a flexible framework is to look for abstractions. Functional abstractions, usually enforced through interfaces, provide a mechanism for you to replace the underlying implementation with your own custom code.

The Audience for This Book

This book is targeted at the intermediate-level software developer interested in building scalable and reliable business applications. General knowledge of the .NET Framework is required, and all source code examples are implemented in the C# language. For building a .NET knowledge foundation, I recommend the following books: *C# and the .NET Platform* by Andrew Troelsen (Apress, 2001) and *A Programmer's Introduction to C#*, Second Edition, by Eric Gunnerson (Apress, 2001).

The Structure of This Book

The purpose of this book is to outline the best practices for building a scalable enterprise application. Each chapter builds a small piece of an entire application. Rather than reference the same old sample applications that are bundled with the .NET Framework, such as Duwamish, Fitch & Mather Stocks, or Northwind, this book works toward building an enterprise application from the ground up. Each chapter focuses on a specific area of application development. You can read this book from one chapter to the next or selectively reference chapters for specific examples that match problems you might be facing.

Chapter 1, "Introducing .NET and Enterprise Architecture," sets the foundation for the book by defining an enterprise application and business processes. The chapter also briefly outlines the important stages of application development from requirements gathering to application deployment.

Chapter 2, "Accessing Data in ADO.NET," dives into the most important element of an application, the database. This chapter leads off with a review of ADO.NET and follows up with the implementation of a structured data access layer. This chapter also highlights the importance of business objects and how they can effectively insulate an application from changes to a data model.

Chapter 3, "Using Networking and Directory Services," introduces directory services, explains how they work, and shows how corporate enterprises use them to manage user accounts. This chapter introduces Active Directory, Microsoft's directory service solution, to build out a directory services business service that fits cleanly into the application framework and interacts with application business objects.

Chapter 4, "Applying Reliable Messaging," adds real-time messaging services to the application and explains how they can be leveraged to build out a scalable business tier based on asynchronous communication. This includes creating messages and queues that implement a framework for building out a sample distributed analysis engine.

Chapter 5, "Integrating Mail Services," presents e-mail services within the .NET Framework, including how they work, how enterprises applications use them, and how they can facilitate online collaboration between the users and

the application. This chapter implements a messaging service that fits cleanly into the application framework and interacts with application business objects.

Chapter 6, "Automating Business Processes," takes a close look at business process automation and how it can be quickly added to an enterprise application with the help of Microsoft BizTalk Server. BizTalk provides tools that simplify process definitions that can span across multiple applications and execute for weeks. You can implement these processes using either the built-in BizTalk tools or the .NET managed code that is plugged in as an integration component.

Chapter 7, "Building Web Applications," implements the application's presentation layer with the use of Web forms, reflection, and data binding. The presentation layer finally adds the user interface to all of the services covered so far and relies upon templates and user controls to reduce errors and encourage reusability.

Chapter 8, "Developing Desktop Applications," implements an enterprise presentation layer as a desktop application. Desktop applications have different strengths and weaknesses when compared to Web applications. One thing both have in common, however, is the need for a structured development framework that minimizes errors and encourages reuse. The next chapter presents a framework for rapid application development that incorporates forms inheritance, user controls, application configuration files, and dynamic access to assemblies.

Chapter 9, "Using XML and Web Services," exposes to more details related to Extensible Markup Language (XML) and Web services. First, this chapter reviews what XML is and presents how it works. Second, the chapter shows how these concepts are broadened and used to create Web services that expose enterprise application functionality.

Chapter 10, "Integrating Reporting Services," describes the reporting services integrated with the Visual Studio .NET environment. The chapter begins with an overview of reporting and then breaks down the report-building steps. It covers everything necessary to build a typical enterprise report.

Chapter 11, "Deploying .NET Applications on Wireless Devices," takes the .NET Framework on the road and explores mobile application development. It covers developing mobile applications for two types of mobile applications: Compact Framework applications for Personal Digital Assistant (PDA) devices and Mobile Internet Toolkit applications for Web-enabled cellular phones. Both frameworks add mobile capabilities to the enterprise application.

Chapter 12, "Integrating .NET Applications," introduces application integration, .NET style. An introduction to application integration describes the value in interfacing existing applications with new .NET applications. Then, it shows how to assemble a new integration platform that puts XML, XSL Transformations (XSLT), and remoting to work.

Chapter 13, "Understanding .NET Security and Cryptography," exposes the security and cryptography services provided by the .NET Framework. You will use these services to implement user-level and code-level security within an

application. A detailed look at encryption reveals methods for scrambling data into an unreadable format that is secure for transmission over the Internet.

Chapter 14, "Installing .NET Applications," wraps up the development of an enterprise application by presenting the deployment project. Deploying Web, desktop, and mobile applications is made significantly easier with the help of the Visual Studio .NET environment. A functional and user-friendly application setup adds a polished and professional look to an application and helps ensure that user configurations are consistent.

Each chapter describes the technology as it relates to the .NET Framework and provides examples for applying it within the scope of an enterprise application. Although the examples are applied to the IssueTracker sample application, they are flexible enough to be applied to just about any business application.

A Personal Note

On a personal note, I have been charged up about Microsoft's .NET initiative from the get-go. My professional development experience comes from a mix of languages and platforms, including C++ and Java. Since Microsoft's first .NET announcement, I have followed its evolution into the robust framework it is today.

For the past several years, Java gained a lot of momentum as the preferred platform for server-side development, while C++ remained the preferred platform for desktop development. The .NET Framework has changed all of that. Sophisticated .NET applications can be quickly produced to run on or run from enterprise-wide servers and meet increasing scalability demands. I am still impressed with the myriad of capabilities .NET offers, ranging from building mobile device applications to creating code that dynamically generates new code. Overall, the .NET platform holds a lot for the future of computing.