Chapter 1

ASP.NET Development Overview

Learning Objectives

- *Understand the complexities involved with ASP.NET development*
- *State the importance of an engineer's notebook*
- *List and describe the typical development tools used to design and build ASP.NET applications*
- *Understand how multiple computer displays increase developer productivity*
- *List and describe the fundamental skills required for ASP.NET development*
- *List and describe the traits of a good programmer*
- *Understand the benefits of using virtual machines to aid development efforts*
- *List and describe the stages of the "flow"*
- *List several legitimate sources for obtaining Microsoft development tools on a tight budget*
- *Understand the rationale behind the approach taken to present the material in this book*
- *Understand what to do if you've bitten off more than you can chew*
- *List several resources you can turn to in case you get stuck*
- *List several strategies you can apply to help you learn a complex topic like ASP.NET*

Introduction

There's more to ASP.NET programming than simply ASP.NET itself — much, much more. In this chapter I attempt to provide a broad overview of various aspects of programming ASP.NET projects in the large. Some of what I talk about can be applied to any type of programming, regardless of language or methodology. Other topics are simply opinions on how to approach the task of programming in the reality of today's complex frameworks. In some cases you may scratch yourself and wonder if you're really reading a programming book, but it's all connected, whether the connection at first seems obvious or not.

Complexities Of ASP.NET Development

ASP.NET development is a complex activity primarily because so many supporting technologies must be learnt and understood before you can confidently declare yourself to be an ASP.NET developer. For example, most if not all web-enabled applications operate on some type of data. This data resides in a database of some sort whether it's a Microsoft SQL Server database or another vendor's product. To effectively integrate your web application with the database you must understand relational database concepts such as tables, rows, columns, primary keys and foreign keys. You'll need to know how to write SQL statements to create, retrieve, update, and delete data. You must know how to formulate connection strings so your web application can connect to the database. You must make a decision on whether to use the entity framework or the Enterprise Library Data Application Block. To better manage the incremental design, development, and maintenance of your database you'll want to learn how to use command-line scripts. And really I've only scratched the surface here.

A complex web application built with a modern object-oriented language like C# can have hundreds or even thousands of interacting classes. These are custom data types that you create to implement application behavior. To control the complexity of such a large number of classes and related types you'll need to know how to structure your application in such a way that groups related classes and segregates dependencies. You will have an inheritance hierarchy that allows you to place common behavior in base classes and implement specialized behavior in derived classes. All this falls into the realm of object-oriented analysis, design, and architecture.

Now, throw in the technologies that support the creation of functional, interactive web pages like JavaScript, Cascading Style Sheets (CSS) and HTML. You'll need to understand how the Hypertext Transfer Protocol (HTTP) works along with concepts related to the TCP/IP protocol. Will you be implementing your web application using ASP.NET Forms, MVC, or a combination of both technologies? What about AJAX and JQuery?

To deploy an ASP.NET web application on a Microsoft Internet Information Services (IIS) server you'll need to become somewhat of a system administrator. At a minimum you can't be timid about working with the operating system. You'll also need to know how to implement and configure application security. Will the application be deployed in an intranet environment or will it be available to the public? The answer to this question will dictate the use of either Windows Authentication or Forms Authentication and perhaps require the implementation of one or more custom role providers.

Let's not forget about logging and unit testing. Logging events in a web application significantly helps troubleshooting efforts. Unit testing ensures code works according to specification and although at first you may balk at the extra programming associated with writing unit tests, the time saved on the back end of the development effort more than makes up for the time spend writing the tests. (This falls into the category of "must see with your own eyes to believe", but is absolutely true.) For good measure you'll want to ensure the safety and integrity of the code you write by keeping it in a source code repository of some sort, like Subversion.

So you see, creating web pages with ASP.NET tags and writing code behinds in C# are only two very small parts of the ASP.NET development puzzle. But fear not. I've structured this book in a way that will help you grow your strength and competency in these and other areas. However, I can't teach you everything you need to know in one book. It'd be so impossibly large that you'd never start reading it. So, you'll need to bring to the table a few fundamental skills which I think are crucial to your success with this material. These I cover in the following section.

Fundamental Skills Required

First off, you'll need a firm grasp of object-oriented analysis, design, and programming concepts using C# as the implementation language. If you need to come up to speed in these areas please refer to my book *C# For Artists: The Art, Philosophy, and Science of Object-Oriented Programming*, Pulp Free Press, ISBN(13): 9781932504071. Specifically you'll need to be intimately familiar with the C# language, know how to implement class hierarchies with inheritance and how to implement interfaces. You'll need to know how to override method behavior in subclasses and how to define and use generic types. (Especially know how to use generic collection classes.) Lastly, and perhaps most importantly in this regard, you must be versed in the organization and contents of the .NET Framework, especially the members of the System namespace. Without an understanding of these concepts you'll make it as far as chapter 9 and then you'll hit a brick wall.

Next, I must assume you are familiar with Cascading Style Sheets (CSS), HTML and XML. There's no need to be an expert, but you shouldn't look like a deer in the headlights when I start using HTML tags in my code to create tables or implement page formatting with CSS styles.

You'll need to know how to use the command prompt. That would be the black mysterious window that opens when you double-click the Command Prompt icon. A typical command prompt window resembles figure 1-1.

Figure 1-1: Command Prompt Window

Referring to figure 1-1 — I use the command prompt heavily especially when administering Windows operating systems and scripting databases. The good news is that you only need to know a small handful of commands to be a command prompt whiz.

In addition to the fundamental skills mentioned above it helps to have certain personality traits shared by good programmers all over the world. A few of these I discuss in the following section.

Traits Of A Good Programmer

Software engineers come in all shapes, sizes, and temperaments. I've worked with many over the years. Here I'd like to discuss what I believe are a few of the most important personality traits shared by the best. I'm not trying to describe the perfect person; we all have our strengths and weaknesses. But by observing some really smart people in action, I have formulated a definite opinion regarding the traits they possess that enable them to work well by themselves while at the same time permitting them to perform well in a team environment.

Creative

The most prevalent personality trait great programmers possess is that of creativity. Solving problems in such a manner that allows them to be executed by a machine takes truckloads of creativity.

If you say to yourself, “But I’m not creative!” My advice to you is not to sell yourself short. A large part of being creative is simply having an open mind. You must be receptive to alternative solutions and not limit yourself to a “this way or the highway” way of thinking.

Tenacious

Great programmers never give up! As computers, operating systems, and programming languages grow increasingly complex, so too grows the complexity of their associated development environments and the range of issues and problems you will encounter when developing solutions for these machines. If you are the type of person who likes to bite into a problem like a pit bull and keep at it until you’ve licked it, then you’ll do well as a programmer.

Resilient

Great programmers bounce back! When a particular problem has given you a thorough trouncing you must come back strong the next day and fight the battle again. Programming is one continuous stream of problem solving. This you must be willing to repeat ad-infinitum. To paraphrase an old Timex® watch advertisement campaign slogan, you must be able to “...take a licking and keep on ticking!”

Methodical

Great programmers approach everything they do in a methodical way. This holds true regardless of if you program alone or as part of a team or if a formal methodology does or does not exist. You must be able to formulate problem attack plans and execute those plans.

Meticulous

Great programmers are meticulous. Close attention to detail is paramount in the programming profession. One identifier misspelled, one token out of place, can break entire systems.

Honest

Great programmers can be trusted to do the right thing in the code when no one is looking. They must be honest with themselves but especially towards other programmers. Honest programmers put in an honest day’s work and give realistic estimates regarding task completion.

Proactive

Great programmers recognize and capitalize upon opportunity. They get up out of their chair and go out and talk to their fellow programmers. When they see problems in the code or areas for improvement they bring it to the attention of the team.

Humble

Great programmers know when to seek guidance or help. They don’t let their ego stand in the way of the greater good. They get up off their duff and talk to their fellow programmers. They share their knowledge and wisdom so that someday they can take a vacation. Most importantly, admitting that they don’t know something early on can save hundreds of wasted work hours down the line.

Be a Generalist and a Just-in-Time Specialist

Great programmers are well versed in all aspects of computing. Rarely have I ever met anyone who referred to himself as only this type of programmer or that type of programmer. I’d rather hire generalists with solid educational

backgrounds and the proven ability to teach themselves new tricks, than to bank on a specialist who refuses to grow professionally. In other words, great programmers have a broad range of skills they can apply to the problem. Great programmers can gather requirements, design a solution, write the code, conduct testing, write supporting documentation, deploy the application if necessary, and carry on intelligent conversations with the customer to boot.

The Engineer's Notebook

If you don't already keep one, I strongly recommend you start and maintain an engineer's notebook. The primary purpose of an engineer's notebook is to record design challenges and their solutions for future reference. What you ultimately jot down in your engineer's notebook will be as personal and as varied as you are unique as an individual. In mine I write down system configuration settings when I install software like databases or application servers. I can't recall how many times I've had to refer back to my notebook to recall a particular configuration setting.

I also capture customer requirements or change requests in my notebook, which I later transfer to a formal requirements management system. I make design sketches in UML or list changes that I must make to the code during a particular day's work. In this regard it functions as a daily work journal. If your boss asks, "What did you do for me this year?" you should be able to point to your engineer's notebook and astound him with your productivity and cunning.

Mostly, however, I record particularly vexing development problems and, when I've found a suitable solution, I write that down too. This has saved me countless hours I'd normally spend solving the same problem again, which is likely to happen if enough time has passed between subsequent encounters.

Over the years I've flirted with different notebook formats from loose-leaf paper in a 3-ring binder to single subject spiral notebooks. Lately I've settled on the college-ruled, 100 sheet, cardboard-covered composition book that looks like figure 1-2:

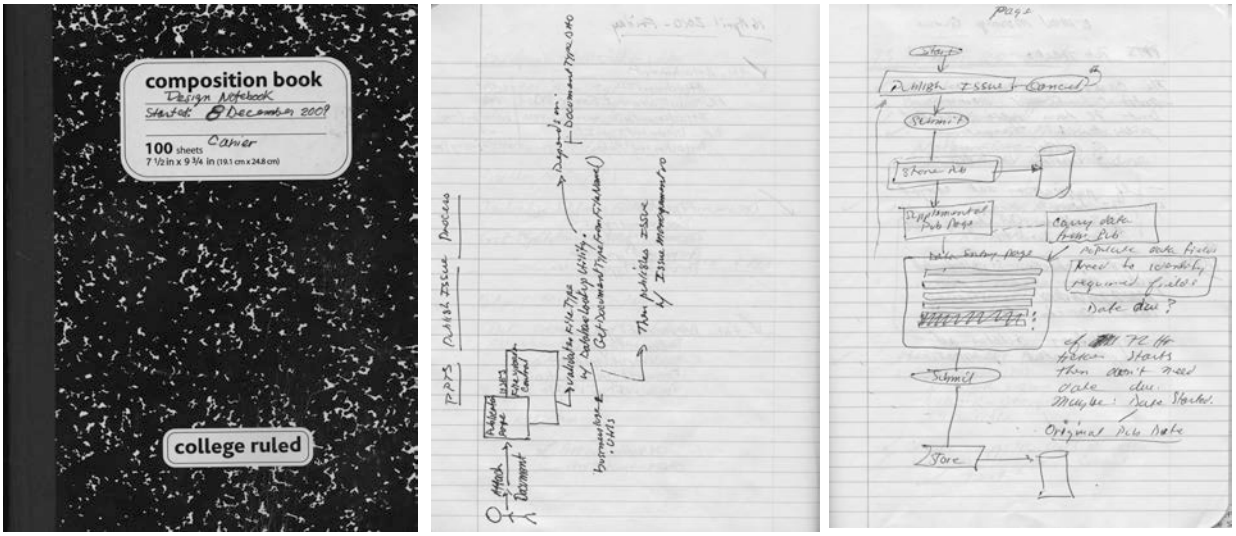

Figure 1-2: Engineer's Notebook: College-Ruled Composition Book with Sample Pages

Referring to figure 1-2 — you can find these notebooks practically everywhere stationary supplies are sold. I prefer these because they're slightly more compact than standard 8 1/2 x 11 inch paper while not being too small. The front and back covers are nice and stiff which allows easy writing on both sides of the paper from the first page to the last whereas loose leaf paper in a 3-ring binder is too bulky and clumsy to work with, especially when sitting in front of a server taking notes on deployment configuration.

When I start a new notebook I write the Start Date on the front cover and when I've filled it up I note the End Date as well so I can see at a glance the period it covers.

You'll find your engineer's notebook to be an invaluable resource as you grow as a software engineer.

Typical Development Tools

As an ASP.NET developer you'll ultimately use many different types of development tools, the flagship tool being Microsoft's Visual Studio Integrated Development Environment (IDE). In this section I want to mention a few of the tools I personally use.

In addition to Visual Studio you'll need a good stand-alone text editor. I prefer one called Notepad++ available from http://notepad-plus-plus.org/. If you're a hardcore UNIX or LINUX user you might prefer vi instead. In the right hands it's an extremely powerful text editor. You can download a version of vi called Vim that runs on Windows from vim.org http://www.vim.org/download.php#pc.

To create design documentation you'll need a good word processor and graphics tool. In my opinion, Microsoft Office can't be beat. Use PowerPoint or Visio to create straightforward architecture diagrams or process flow diagrams.

If I need to create complex UML diagrams I use a tool called Altova UModel available from http://www.altova.com/. UModel is the most capable UML modeling tool on the planet. Its reverse engineering feature is unmatched. It also produces excellent graphics for use in design documentation. I'd like to assert here that I am not a paid Altova spokesman. I am simply impressed with the quality and thought they put into developing their UML modeling tool.

If you're using MS SQL Server as your database you'll find its built-in entity-modeling tool quite capable. Even if you maintain your database with scripts, you can reverse engineer the script-generated schema, make modifications using the design tool, then generate and extract the scripts, all without any fuss.

For unit testing I use NUnit. It can be called automatically as part of the project build process or can be run as a stand-alone GUI application, which graphically shows you the status of running your test suite.

Many Microsoft and third party tools integrate with Visual Studio. One such Microsoft tool is FxCop, which can be used stand-alone or as a Visual Studio add-on tool. FxCop is a tool that inspects managed code assemblies and reports on any potential problems it finds with regards to suggested best practices as described in Microsoft's Design Guidelines. You can improve your coding skills by running FxCop and fixing the problems it finds, but more importantly, by learning from its suggestions.

For source code version control I like to use Subversion. It's a mature product that has given me years of reliable service. You can set up a Subversion repository on your machine or on another machine in your house (or office), or you can use one of several Subversion repository hosting services available out there in Internet land, one being Google Code. There are a lot of SVN hosting sites out there to choose from. Here's a site that lists many and compares their services: http://www.svnhostingcomparison.com/.

In a team development environment you'll want to set up your project's subversion repository on a reliable server that's backed up regularly or use a hosting service if you don't have the local resources.

There are other capable version control systems out there and it matters little which one you prefer to use. What matters is that for serious projects being developed by a team of engineers, a good source code version control system along with a complete understanding by team members of how it should be used as part of the development process, significantly improves team efficiency.

Another set of tools I find absolutely indispensable in developing complex ASP.NET applications is virtual machines. I discuss them in detail in the following section.

The Value Of Virtual Machines And Dual Displays

While you can certainly develop ASP.NET applications on a suitable laptop or desktop machine that directly hosts the database along with IIS and Visual Studio, you'll find things getting a little too cramped for comfort. Also, testing complex deployment configurations is difficult if not impossible on a single machine. This is where the use of virtual machines really comes in handy.

A virtual machine is an application that runs on a host computer that allows you to create instances of hosted platforms. One popular virtual machine application is VMware Workstation available from http://www.vmware.com.

Another product is Oracle's VirtualBox available free from https://www.virtualbox.org. Don't let the word "free" turn you off from trying VirtualBox. It's an awesome product.

When you use virtual machines to develop and test, it makes little difference as to what physical computer you're working on. For example, to write this book and create all the examples herein, I am using an 8-core Apple Mac Pro running the Parallels virtual machine software. Currently, as of this writing, I have three virtual machine (VM) images installed and configured: one running Windows XP SP3, one running Windows 7, and the third running Windows Server 2008. Figure 1-3 is a screen capture of my Mac running MS SQL Management Studio 2008 R2 in the Windows Server 2008 VM.

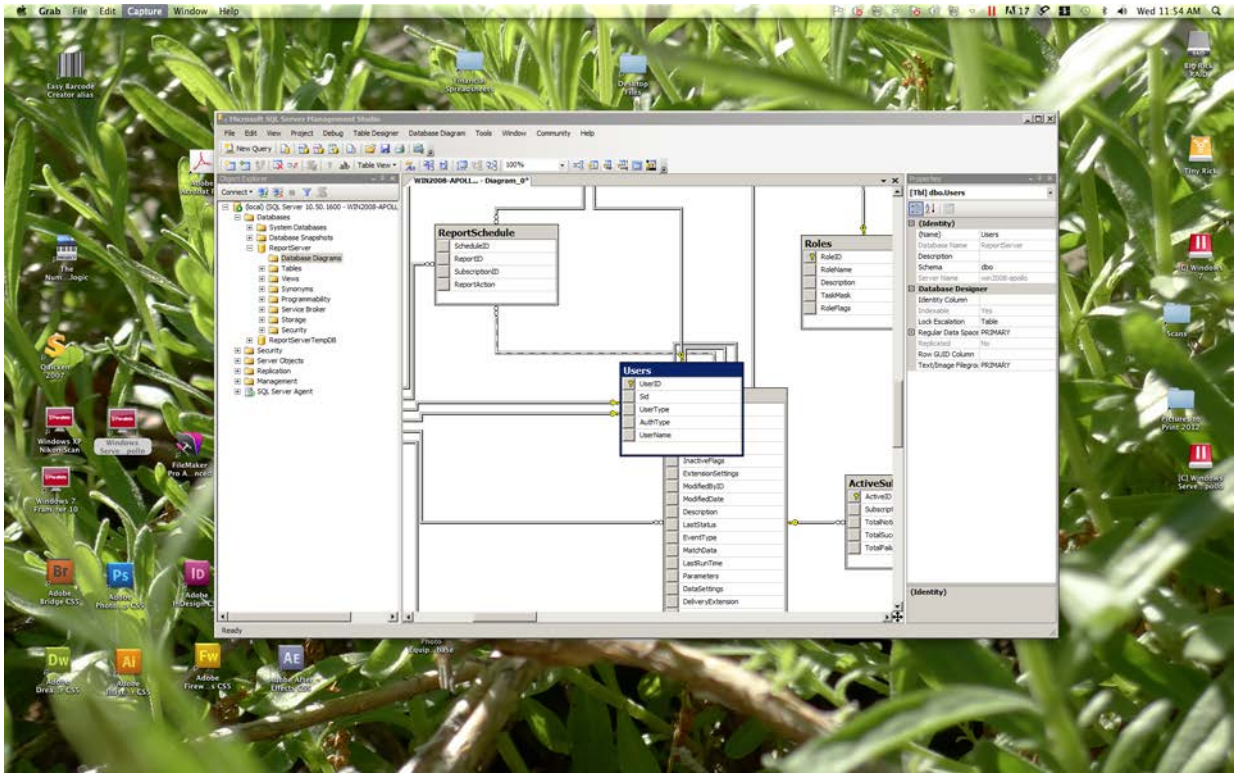

Figure 1-3: SQL Management Studio 2008 R2 Running in Parallels VM on Mac Pro

Referring to figure 1-3 — I might add that my VMs run flawlessly in this environment. Parallels integrates so tightly with the OS X environment that it's hard to tell where OS X stops and Windows begins.

The nice thing about VM images is that you can take snapshots of the image at any given time to which you can roll back to if you screw something up. This is particularly helpful when configuring deployment test environments.

VMs come in especially handy with application and system testing. No longer do you need a rack full of servers, each dedicated to some version of the operating system. Simply create a new VM, load the OS, configure as necessary, and then back it up to be used as a baseline, and make snapshots before each major change.

Another beautiful thing about VMs is that you can copy them to a shared drive, upgrade your old machine's host operating system, reload the VM application software, copy the VM image back to your machine, and voila, you're back up and running. This alone will save you days of reloading and configuring your development environment.

Dual Displays

Trust me when I tell you that your life will change for the better if you simply connect a second display to your computer to give yourself twice the screen real estate. Studies have shown that people who use two or more computer displays, programmers especially, cut down on printing and spend less time shuffling windows around on their single display.

When programming, the second display renders itself immediately useful by allowing you to display documentation in one screen while you edit code in the other. Figure 1-4 shows my dual-display configuration.

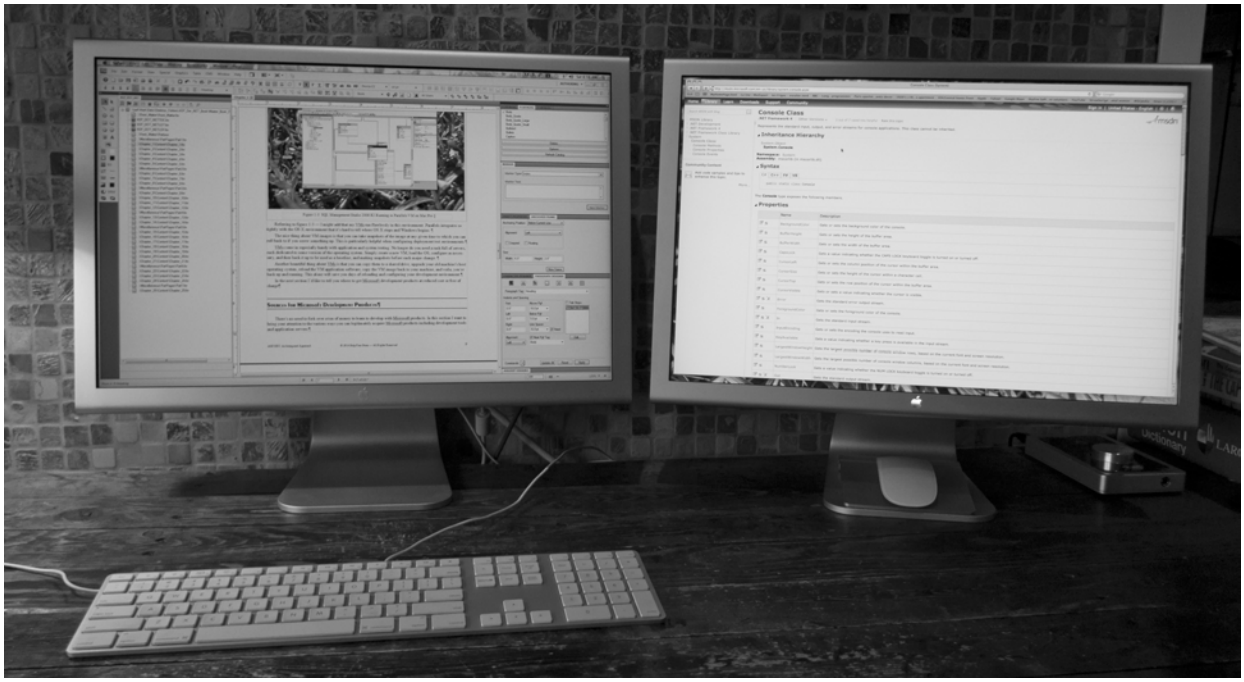

Figure 1-4: My Dual-Display Configuration

Referring to figure 1-4 — In the left-hand display I'm editing this document using FrameMaker Version 10, which runs only in Microsoft Windows, so I'm running the application in a Parallels VM configured with Microsoft Windows 7 Professional. In the right-hand display I have open a Mac OS X Safari browser window referencing MSDN .NET Framework documentation.

Several hardcore developers I know use three or more displays, and they use every inch of available screen real estate to full advantage.

In the next section I'd like to tell you where to get Microsoft development products at reduced cost or free of charge

Sources For Microsoft Development Products

There's no need to fork over a ton of money to learn to develop with Microsoft products. Likewise, there's absolutely no need to pirate their products either, as you can get practically every application required for learning software development at no charge. In this section I want to bring your attention to the various ways you can legitimately acquire Microsoft products including development tools and application servers.

MSDN Subscription

If you're a working professional programmer and you or your company has the financial resources to swing it, I strongly recommend getting an MSDN developer's subscription. Subscription levels vary from quite affordable to downright expensive. Depending on your level of need, even the most downright expensive subscription will save you a ton of cash vs. buying all the required products separately.

Developer Tool Express Editions

If you're a student or you absolutely can't spare the cash don't despair. You can download express editions of Visual Studio and SQL Server, both of which will provide a great platform for learning how to develop with Microsoft technologies. For more information on both the subscription and express downloads visit http://msdn.com.

Microsoft DreamSpark

Microsoft also allows students and faculty members to download full-fledged versions of Visual Studio and various server platforms at no cost via their Microsoft DreamSpark site https://www.dreamspark.com. You must be a registered student or faculty member at an institution of higher learning to qualify for DreamSpark downloads. A simple registration process will validate your status.

Setting The Mood For Programming

Programming is an art. Any programmer will agree — it takes a lot of creativity to solve problems with a computer. Creative people have an advantage in that they are not afraid to explore new avenues of design. Their open-mindedness and readiness to accept new ideas gives them the ability to see problems differently from people who tend towards the "cut and dry". This section offers a few suggestions on how you can stimulate your creativity and put yourself in the mood to program.

Don't Start At The Computer

Unless you have a good idea about what source code to write, sitting down at the computer without first thinking through some design issues is the worst mistake you can make. If you have ever suffered from writer's block when writing a paper for class, then you can begin to understand what you will experience if you begin your project at the computer.

I recommend you forget the computer, go someplace quiet and relaxing with pen and paper, and draft a design document. It doesn't have to be big or too detailed. Entire system designs can be sketched on the back of a napkin. The important thing is that you give some prior thought regarding your program's design and structure before you start coding.

Your choice of relaxing locations is important. It should be someplace where you feel really comfortable. If you like quiet spaces, then seek quiet spaces; if you like to watch people walk by and observe the world, then an outdoor café may be the place for you. Inside, outside, at the beach, on the ski slope, wherever you prefer.

What you seek is the ability to let your mind grind away on the solution. Let your mind do the work. Writing code at the computer is a mechanical process. Formulating the solution is where real creativity is required, and is the part of the process that requires the most brainpower. Typing code is an exercise in attention to detail.

Inspiration Strikes At The Weirdest Time

If you let your mind work on the problem it will offer its solution to you at the weirdest times. I solve most of my programming problems in my sleep. As a student, I kept computers in the bedroom and would get up at all hours of the night to work on ideas that had popped into my head in a dream.

Try to have something to write with close at hand at all times. A pad of paper and pen next to the bed or next to the toilet can come in handy! You can also use a small tape recorder, digital memo recorder, or your smart phone. Whatever means suit your style. Just be prepared. There's nothing worse than the sinking feeling of having had the solution come to you in the middle of the night, in the shower, or on the drive home from work or school, only to forget it later. You'll be surprised at how many times you'll say to yourself, "*Hey, that will work!*" only to forget it and have no clue what you were thinking when you finally get hold of a pen.

Own Your Own Computer

Do not rely on the computer lab! I repeat, do not rely on the computer lab! The computer lab is the worst possible place for inspiration and cranking out code. If at all possible, own your own computer. It should be one sufficiently powerful to use for ASP.NET software development.

You Either Have Time and No Money, or Money and No Time

The one good reason for not having your own personal computer is severe economic hardship. Full-time students sometimes fall into this category. If you are a full-time student, what you usually have instead of a job or money is gobs of time. So much time that you can afford to spend your entire day at school and complain to your friends about not having a social life. But you can stay in the computer lab all day long, even when it is relatively quiet.

On the other hand, you may work full-time and be a part-time student. If this describes you, then you don't have time to screw around driving to school to use the computer lab. You will gladly pay for any book or software package that makes your life easier and saves you time.

The Family Computer Is Not Going To Cut It!

If you are a family person working full-time and attending school part-time, then your time is a precious commodity. If you have a family computer that everyone shares, adults as well as children, then get another computer, put it off limits to everyone but yourself, and password-protect it. This will ensure that your loving family does not accidentally wipe out your project the night before it is due. Don't kid yourself, it happens. Ensure your peace of mind by having your own computer in your own little space with a sign on it that reads, "*Touch This Computer And Die!*"

Set The Mood

When you have a good idea on how to proceed with entering source code, you will want to set the proper programming mood.

Location, Location, Location

Locate your computer work area someplace that's free from distraction. If you are single, this may be easier than if you are married with children. If you live in a dorm or frat house, good luck! Perhaps the computer lab is an alternative for you after all.

Have your own room, if possible, or at least your own corner of a larger room that is recognized as a quiet zone. Noise-canceling headphones might help if you find yourself in this situation.

Set rules. Let your friends and family know that it's not cool to bother you when you are programming. I know it sounds rude, but when you get into the *flow*, which is discussed in the following section, you will become agitated when someone interrupts your train of thought to ask you about school lunch tomorrow or the location of the car keys. Establish the ground rules up front that say when it is a good time to disturb you when you are programming. The best rule is never!

Concept Of The Flow

Artists tend to become absorbed in their work, not eating and ignoring personal hygiene for days, even weeks, at a time. Those who have experienced such periods of intense concentration and work describe it as a transcendental state where they have complete clarity of the idea of the finished product. They tune out the world around them, living inside a cocoon of thought and energy.

Programmers can get into the flow. I have achieved the flow. You too can achieve the flow. When you do, you will crave the feeling of the flow again. It is a good feeling, one of complete and utter understanding of what you are doing and where you are going with your source code. You can do amazing amounts of programming while in the flow.

The Stages of Flow

As with sleep, there are stages to the flow.

Getting Situated

The first stage: You sit down at the computer and adjust your keyboard and stuff around you. Take a few deep breaths to help you relax. By now, you should have a good idea of how to proceed with your coding. If not, you shouldn't be sitting at the computer.

Restlessness

The second stage: You may find it difficult to clear your mind of the everyday thoughts that block your creativity and energy. Maybe you had a bad day at work, or even a great day. Perhaps your spouse or significant other is being a complete jerk! Perhaps he or she is being especially nice and you're wondering why.

Close your eyes and breathe deeply and regularly. Clear your mind and think of nothing. It is hard to do at first, but with practice it becomes easy. When you can clear your mind and free yourself from distracting thoughts, you will find yourself ready to begin coding.

Settling In

The third stage: Now your mind is clear. Non-productive thoughts are tucked neatly away. You begin to program. Line by line, your program takes shape. You settle in. The clarity of your purpose takes hold and propels you forward.

Calm and Complete Focus

The fourth stage: You don't notice it at first, but at some point between this stage and the previous stage, you have slipped into a deeply relaxed state. You are utterly focused on the task at hand. It is like becoming completely absorbed in a good book. Someone can call your name, but you will not notice. You will not respond until someone either shouts at you or does something to break your concentration.

You know you were in the flow, if only to a small degree, when being interrupted brings you out of this focused state, leaving you feeling agitated and eager to settle in once again. If you avoid getting up from your chair for fear of breaking your concentration or losing your thought process, then you are in the flow!

Be Extreme

Kent Beck, in his book *Extreme Programming Explained*, describes the joy of doing really good programming. The following programming cycle is synthesized from his extreme programming philosophy.

The Programming Cycle

Plan

Plan a little. Your project design should serve as a guide in your programming efforts. Your design should also be flexible and accommodate change. This means that as you program, you may make changes to the design.

Essentially, you will want to design to the point where you have enough of the design to allow you to begin coding. The act of coding will either soon reinforce your design decisions, or uncover fatal flaws that you must correct if you hope to have a polished, finished project.

Code

Code a little. Write code in small, cohesive modules. A class or method at a time usually works well.

Test

Test a lot. Test each class, module, or method both separately and in whatever grouping makes sense. You will find yourself writing little programs on the side called *test cases* to test the code you have written. This is a good practice to get into. A test case is a program you write and execute in order to test the functionality of some component or

feature before integrating that component or feature into your project. The objective of testing is to break your code and correct its flaws before it has a chance to break your project in ways that are hard to detect.

Integrate/Test

Integrate often, and perform regression testing. Once you have a tested module of code, be it either a method or complete set of related classes, integrate the tested component(s) into your project regularly. The objective of regular integration and regression testing is to see if the newly integrated component or newly developed functionality breaks any previously tested and integrated component(s) or integrated functionality. If it does, then remove it from the project and fix the problem. If a newly integrated component breaks something, you may have discovered a design flaw or a previously undocumented dependency between components. If this is the case, then the next step in the programming cycle should be performed.

Refactor

Refactor the design whenever possible. If you discover design flaws or ways to improve the design of your project, you must revise and improve the design to accommodate further development. An example of design refactoring is the migration of common elements from derived classes into a base class to take better advantage of code reuse.

Repeat

Apply the programming cycle in an iterative fashion. You will quickly reach a point in your project where it all starts to come together, and very quickly so.

The Programming Cycle Summarized

Plan a little, code a little, test a lot, integrate often, and refactor the design when possible. **Don't Wait Until You Think You Are Finished Coding The Entire Project To Compile**! Trying to write the entire program before compiling a single line of code is the most frequent mistake new programmers tend to make. The best advice I can offer is this: **don't do it!** Use the iterative programming cycle previously outlined. Nothing will depress you more than seeing a million compiler errors scroll up the screen after waiting until the bitter end to compile your project.

Learning Strategies

At first, and this applies especially if you're new to programming in general, you'll feel completely overwhelmed by the shear bulk of the material you must learn to get even a simple ASP.NET program up and running. My best advice to you is to divide and conquer. Pay particular attention to fundamental concepts. If you're fuzzy on C# and object-oriented programming then go back and come up to speed on those topics first. A firm grasp of fundamental concepts builds a strong foundation upon which to grow your knowledge. It's a lot like learning a foreign language.

I've been studying the French language for the past two years. "French?" you ask yourself. "What the heck does French have to do with anything?" Well, it's interesting being a student in class after being an instructor standing in front of a class for so long. What I observed was this: Students who applied themselves from day one of French 101 were significantly better poised to understand more complex material towards the end of the class. Conversely, it was painfully obvious to the rest of us who among us was simply not studying.

Anything worth learning is difficult to master, French especially. The trick to success is to practice good pronunciation and accumulate a working vocabulary early on in the class so that later you don't have to grope for a word and stop to think about how it's pronounced. The more studying and practicing one did early in the class, the easier it became to understand later material. This might seem obvious but it holds true in a subject where an understanding of what came first is essential to understanding what comes next. I noticed a sense of panic strike those students who screwed off during the first half of the semester. The realization that they had wasted valuable time hit them like a ton of bricks and most ended up dropping the class with only weeks left in the semester. As a fellow teacher, I would talk

about this at length with the French instructor and I said I'd noticed the same behavior in my programming classes. Nowadays especially people want instant satisfaction, but you can't get it in this business.

I'm also reminded of how I felt when, as a junior Ensign newly assigned to the aircraft carrier USS America (CV-66), I first ascended the long flight of stairs leading up to the bridge of the ship and seeing for the first time the enormous flight deck laid out before me. I thought to myself "How in the world will I ever learn to maneuver this ship?" which was my primary reason for existence as a budding Surface Warfare Officer. The trick was simple: The Navy has what they call a Personnel Qualification System (PQS) to ensure folks know what they are doing before they are allowed to do it. One must be qualified to perform any particular job on a warship even if it's to make a cup of coffee. (In the case of coffee making, the qualification may be informal, but you will certainly be trained to do it properly. And if you screw it up, stand by for remedial training!)

However, for a job that requires formal qualification there is an approved qualification sheet one must get signed off proving you've been trained in various aspects of the job. Part of qualification is formal school, but a lot is what's called on-the-job training where you team up with a qualified expert and they instruct you in some particular aspect of the position for which you are trying to qualify. If the position is serious and loaded with responsibility, like that of an aircraft carrier Officer of the Deck Underway, then once you have finished the PQS requirements, which usually takes more than a year to complete, you must pass a senior officer board headed by the commanding officer himself to prove to him you have what it takes to drive his ship.

What is a PQS card? It's simply a written list of competencies you must prove by demonstration to have obtained. You proceed step-by-step and study and work on each competency until you've mastered them. Then, one day, you realize you can drive that aircraft carrier!

How does this apply to learning ASP.NET and all its related technologies? Very simply it suggests that the best way to stem the sense of panic you may have starting out at being presented with the overwhelming amount of material you have to learn is to make a list of what you need to know, study each topic, and check each one off as you go. In the process you may need to read different books, or sit down and ask an experienced software engineer some questions. I don't know of anyone who is too busy to help someone who sincerely asks for help.

In the next section I'd like to say a few words about what to do if you find you're stuck with a particularly vexing problem.

What To Do If You Get Stuck

Lots of developers get stuck from time to time. Some developers are good at being handed well-defined tasks but lack the skills required to get a big project going from the ground up. This usually happens to students who are given non-trivial project assignments.

In my book *C# For Artists: The Art, Philosophy, and Science of Object-Oriented Programming*, I offered a *project approach strategy,* which I designed to help students overcome their sense of being overwhelmed and maintain their sense of forward learning momentum. You may download chapter 1 of that book and use the project approach strategy to get your project up and moving. Here's the link:

http://pulpfreepress.com/content/Products/Books/CSharp_For_Artists/1-932504-07-9.shtml.

If, on the other hand, you're moving along quite nicely but suddenly you come up against a particular problem, your first line of pursuit should be to look for a solution on Google. Chances are you are not the first to encounter the problem. Except in one particular instance, I don't recall ever not finding a suitable solution via Google to a programming problem I had encountered. If you phrase your Google query carefully, you'll usually find the answer within the first one or two pages of search results. One of the reasons for having a second display is to keep one handy for documentation reference and solution finding. On one recent project the customer demanded we perform the work in a classified environment with no access from our development workstations to the unclassified Internet. I calculated that this one stubborn request would triple the development effort. Developers would have to get up, go to another machine to conduct a search on the unclassified network, then return to their classified workstation and get back to programming. If you read the section above about getting into the "flow" then you'd realize that once your concentration is broken, it's hard to get right back to being 100% productive.

In some cases you may never find a suitable solution to a particular problem on the Internet. In these rare cases it would be nice to talk the problem over with an experienced developer, but this is not always possible. The next best course of action would be to search the Internet for a suitable developer's community and join up and post your ques-

tion online. Chances are there is someone in the world who's solved your problem before. I've had to do this several times and had great success with really competent, friendly developers offering responses fairly quickly.

Sometimes, a solution is right before you, in front of your eyes as they say, but you are simply too tired after a long day's struggle to realize it. In this case the best offence is a short retreat. Go home, get some sleep, and return fresh the next day and start again. Usually you'll see things crystal clear and the solution will jump out at you.

Most frequently what you really want is a good example of how to do something. In this case, especially if it concerns a particular .NET Framework component, start by looking at the examples on the MSDN documentation site. If you find no joy there then search Google. More often than not someone has posted an example of exactly what you're trying to do.

Anytime you encounter a problem that requires you to search for a solution, especially one that requires you to expend considerable effort, write the solution down in your engineer's notebook for future reference! Gradually you'll find that problem solving gets easier as you gain experience.

Summary

In this chapter I offered a glimpse into the complexities you can expect as an ASP.NET developer. Be sure to lay a solid foundation for learning by being competent in the required fundamental skills. If you're weak in some core knowledge areas like object-oriented programming and the C# language, come up to speed in these areas first and then proceed with the book.

If you can afford it, I recommend obtaining an MSDN subscription that allows you to download and use Microsoft's major development tools, servers, and operating systems. If you're tight on cash or you're a student or faculty member you can download express editions of Microsoft development products and servers via MSDN. Microsoft also provides the DreamSpark website for registered students and faculty to obtain full-fledged versions of their development products.

You can significantly increase your development productivity by using multiple computer displays and virtual machines.

If you get stuck, the Internet is really your best resource for finding a solution to your problem. If you don't have access to an experienced ASP.NET developer join an online professional community and post your question to its members. If you're looking for good code examples start with the .NET Framework documentation on the MSDN website.

Record the problems you encounter and their solutions in your engineer's notebook. You can also use an engineer's notebook to make design notes, to-do lists, informal meeting minutes, etc. What you ultimately put in your engineer's notebook will be as individual as you are.

References

Microsoft Developer Network (MSDN) website, http://msdn.com

Rick Miller, *C# For Artists: The Art, Philosophy, and Science of Object-Oriented Programming*, Pulp Free Press, ISBN(13): 9781932504071

Notes

