Part I: Design Strategies

1

Defining the Language Environment

Most professional programmers use more than one programming language. A programmer who relies on a single language for every task is like a carpenter who treats everything as a nail. Just as carpenters use a particular fastener for a specific need, programmers rely on programming languages that suit an individual development need. This chapter approaches the use of a particular programming language to meet a specific need in a development environment. Throughout the chapter, you discover how the language environment determines not only which language will work best but also how you interact with the language you select.

In some cases, you use multiple languages in a single application. Although the use of multiple languages tends to increase complexity, using the right tool for the job also tends to make the work easier. Just as a carpenter relies on both hammers and screwdrivers to build a bookcase, a programmer may use multiple languages to make creating a particular application easier. Just as the carpenter could use nails for every need on the bookcase, but will obtain a poorer result by doing so, the developer often achieves poor results by using just one programming language. This chapter views the use of multiple languages in the context of the language environment and as a prerequisite for application design.

Understanding the language environment means that you can take the next step in designing your application, which is to determine what resources you have available. Even if you're working by yourself, you must consider the language resources you have, because most developers aren't completely fluent in every aspect of all the languages they use. You may decide to learn a new development technique in order to use a particular language or live with a less than optimal language environment in order to rely on a language you use well. Defining available resources is critical in the team environment because you need to know what individual team members can do to advance the project before you design it. This chapter describes a process of inventorying the tools at your disposal and then gathering them for use in the application design and development process.

Defining the Design Strategy Elements

It's natural to say at the outset of a project that you want to accomplish a particular task. You don't know how you'll accomplish it at this point, but simply that you plan to accomplish it. Before you can define a language environment, you have to have some idea of what you want to accomplish. In many cases, you need to consider some rudimentary steps to accomplish your goal as well. For example, you may want to write an application that retrieves data from a Web service, performs some analysis on that data, and presents it to the user. Each of these tasks forms a design strategy element for your application.

Developers can become stuck in a rut when it comes to programming languages. When you talk to someone who's used Visual Basic exclusively, that person can normally tell you about all the inadequacies of C# in detail without too much thought. Likewise, developers who use C# exclusively often see the warts found in Visual Basic quite easily. A developer who takes time to learn both C# and Visual Basic will tell you that each has some distinct advantages and disadvantages, but that you can develop most applications using either language.

However, some languages do provide a specific set of features that differ from common programming languages. For example, you can use F# (I'll describe this new Microsoft language later in the chapter) to perform list processing, pattern matching, and other forms of analysis. Microsoft even makes it easy to combine C# or Visual Basic with F# so that you can obtain the best of both languages in a single application. When you consider the sample application described earlier in this section, you could use:

- ❑ C# to obtain the data from the Web service
- ❑ F# to process that data
- ❑ C# to display the data to the user

This second case points out a scenario where using multiple languages is more than a political statement; it's a necessity to make application development as easy as possible. The advanced features of a language can easily offset the added complexity of using multiple languages in many cases.

Don't assume that a developer on your team has experience in only one or two languages. The use of a particular language by the developer may simply indicate that the company hired the developer to use the language. Many developers have experience in multiple languages. For example, even though I mainly work with Visual Basic and C#, I have also worked with everything from Assembler and ROM BASIC to LISP and Prolog. I've worked relatively often with PHP, Java, and C++ as well. However, unless someone asks me about my skill set, they may simply assume my language knowledge includes mainly Visual Basic and C#.

Of course, you need someone on your development team with F# experience to write an application that uses F# for analysis purposes. Depending on the inventory you conduct, you may find that all your developers know C# or Visual Basic. In this case, the optimal language environment doesn't use F#, but you should still consider using the best language for the job at the outset of the planning process.

Your strategy has to consider the developers in your team, even if you have a team of one. You don't want to create an application where half the team twiddles its collective thumbs waiting for the other half of the team to complete a process. An application development effort may require that you write the low-level code using C# and the user display elements in Visual Basic. With proper planning, you can move the development process along rapidly by letting each team complete the part of the application most suited to its particular language.

Considering the C# Language

This book is about using the C# language to develop applications of various types. You'll work through the particulars of designing an application that relies on C# as its main language. C# is a great language for many uses. For example, it's possibly the best language that Microsoft makes available for developing applications that use a combination of native code and managed libraries. The Platform Invoke (P/Invoke) capability of C# is significantly better than other Microsoft languages. However, C# isn't a perfect language, so you need to understand its deficiencies and consider how you can use other languages to overcome these deficiencies. The following sections discuss these issues.

Understanding the Benefits

In many respects, C# and Visual Basic.NET are very much alike. Both contain all the common coding elements you might expect. It's possible to create conditional statements, loops, structures, classes, and all the other common elements found in applications using either language. The fact that the syntax differs slightly between languages isn't a major concern. However, C# does possess certain benefits not found in Visual Basic.NET that may affect your decision to use C# as your main programming language. The following sections discuss each of these benefits and define why you should consider them when defining your language environment.

C/C++-Like Syntax

The future of development for most organizations is some type of managed code. Using the .NET Framework provides a considerable number of benefits when compared to native code languages such as C++. For example, working with a managed application means that the developer no longer needs to worry about memory allocated for managed needs because garbage collection handles that requirement for the developer. The vast number of viruses and other nasty software on the market that rely on memory errors to do their dirty work attests to the need for this kind of automated memory allocation arrangement.

Unfortunately, most organizations can't simply drop all their native code in favor of a managed alternative. Consequently, you may find that you need to maintain that C++ code sitting in your library for quite some time. Because C# and C/C++ share so many syntax features, it's easy for C/C++ developers to move between C# and C/C++ as needed. The transition isn't without some learning curve, but the curve is significantly smaller than moving to some other language. It really is relatively easy to write code using C# one day and C/C++ the next.

While it's possible to create managed code using C++, most developers consider the process extremely difficult. Unlike other .NET languages, C++ requires that you follow some absurd coding practices. Most developers prefer to use C# when needed to create managed applications.

It's important to note that the brace construction used to delimit blocks of code in C# also appears in a number of other languages such as Java and JavaScript. In fact, there are enough similarities with these other languages that a C# developer can probably learn Java or JavaScript with relative ease when compared to learning a language that relies on other construction techniques such as Visual Basic.

The designers of C# and Java both wanted to improve on some primary problems with C++. Here's a list of some of the features that C# and Java share that help you create a better language environment than using C++ as your language of choice (you can find some other interesting comparisons at http://www.softsteel.co.uk/tutorials/cSharp/lesson2.html and http://blogs.msdn.com/ericgu/archive/2005/01/26/360879.aspx):

- Both languages compile into machine-independent byte code that runs in a managed environment. Although the .NET Framework runs exclusively on Windows today, you can run the C# Intermediate Language (IL) code on other platforms such as Linux using alternative environments such as Mono (http://www.mono-project.com/Main_Page).
- Memory errors occur less often because both languages rely on garbage collection to manage memory.
- You can view the internal workings of applications created with either language using reflection.
- Applications have less code because neither language requires that you use header files.
- Using classes is less cumbersome because classes are scoped to packages or assemblies. In addition, you don't have to worry about the order in which classes appear in the application. Classes also descend from objects and you allocate them from the heap using the `new` keyword.
- It's possible to lock threads with ease, providing superior thread synchronization.
- A class can contain another class to provide inner classes.
- Some potential runtime errors are avoided by eliminating global variables — every variable is part of a class. In addition, both languages provide bounds checking for arrays and strings, making some types of application bugs exploitable by virus writers a thing of the past.
- Both languages rely on the `.` operator to separate object elements. You no longer need to worry about whether to use the `::` or `->` operators.
- Both languages provide full support for the `null` and `boolean/bool` keywords.
- The managed environment automatically initializes all values before use, so you won't have any of those mystery values from C++ pop up.
- All conditional statements rely on a Boolean comparison, rather than using `integer` values directly.
- `Try` blocks provide support for a `Finally` clause that lets you perform cleanup even if a code sequence fails, which should help eliminate memory and state problems.

This list isn't meant to provide you with a complete comparison between C#, Java, and C++, but it does illustrate that there are important issues to consider as part of defining the language environment for your application. You may find situations where C++, despite its deficiencies, is still the language of choice for a particular situation. Knowing the deficiencies of each language in your arsenal is important because this knowledge helps you make the right language choice.

Unsafe Code Capability

The Common Language Runtime (CLR) constantly monitors everything your managed application does, which makes the code more reliable. CLR considers managed code safe because you can't do things like add pointers together or perform other kinds of pointer arithmetic found in older languages such as C/C++. In the past, using pointers led to all kinds of problems, including memory corruption, so modern applications generally don't perform direct pointer manipulation. In fact, direct pointer manipulation doesn't work with .NET applications because the garbage collector constantly changes the pointer addresses (which is the reason you must pin and unpin `IntPtr` managed pointers when making P/Invoke calls).

Unfortunately, you still have the vast array of native-code applications to consider when designing your applications. You may need to access Windows directly to perform some tasks or work with an older DLL where pointer manipulation is required. When you encounter this situation, you have two choices. You could wrap the code you need in a C++ DLL and make it available to your applications using P/Invoke in a manner that doesn't require the use of unsafe code. The second choice is to work with the older code directly and place the pointer code within an unsafe code block. C# is one of the few managed languages that let you use unsafe code blocks. You can find a useful tutorial on using unsafe code blocks at `http://www.csharp-tutorials.com/unsafe-code-execution/`.

In-Depth Debugging

Generally, you'll find that the C# debugger provides more information than the Visual Basic debugger, provided you're working with the same application in both debuggers. This section also bases its appraisal of C# on the use of the Microsoft debugger. If you use a third-party product, you may find that it provides more or less capability than the Microsoft product.

Some things work the same between Visual Basic and C# — a loop is still a loop. However, you'll notice a difference when working with some complex development problems. For example, a Language Integrated Query (LINQ) call in C# always shows every step of the process used to create the query results. In some cases, Visual Basic shortcuts the process and shows only a few of the steps, making it harder to diagnose problems with your LINQ statement.

Of course, the question is whether you always need the complete attention that C# provides. The answer depends on your application. As application complexity increases and you use more advanced language features, the requirement for a debugger that provides in-depth information also increases.

The debugging differences with other languages extend to other needs. C# usually makes it easy to drill down into an object to any level. This accessibility means that you spend less time trying to figure out what an object actually contains. In some cases, you have to work hard with other languages to determine what an object contains, especially when an object resides within another object.

Additional IntelliSense Functionality

When you create your own applications, you can use the triple comment `///` symbol in C# to create a documentation block where you fill out information for various application elements such as classes and functions. Visual Basic.NET 2008 adds this feature as the `'''` symbol. After you add these special code blocks, you can set the compiler to create an eXtensible Markup Language (XML) file containing detailed information about your application. However, C# still provides superior IntelliSense support for these special XML files, which means that C# provides better IntelliSense support for the code you write.

C# also provides multiline comments, a feature it shares with C++. If you want to create a multiline comment for Visual Basic, you must start each comment line individually.

Coding Additions

C# supports a number of coding additions — features that you won't necessarily find in other managed development languages. One of the more useful is the availability of strong iterator support, including the `yield` keyword. When an application iterates a collection, the `yield` keyword helps the developer control how the iteration occurs and can stop the iteration as needed.

Anonymous methods make it possible to create applications that run faster and require less code as well. An anonymous method is literally a method that has no name. The method appears in-line in place of the usual method reference. Using anonymous methods can also make it easier to debug the code because the code appears in-line where the developer can easily see it during a debugging session.

In some cases, you want to place an interface definition in multiple files. Unlike other .NET languages, C# lets you use the `partial` keyword with:

- `class`
- `struct`
- `interface`
- Method names

Sometimes you want to make class code available without requiring the developer to create an instance of the class. C# supports a static class for this purpose. For example, a math class may not have any properties or state information to maintain, so a static class works just fine. In this case, all members of the class are also static, which means you can access them directly. (A Visual Basic module is a close cousin of this feature.)

You'll also find a few coding niceties in C#. For example, C# uses the = operator for assignment and the == operator for comparison. Using different symbols for each purpose makes their use clearer in your code and helps avoid potential misinterpretations.

Exploding Developer Mind Share

At one time, most developers acknowledged that Visual Basic held a surprising lead in languages that most developers would know. However, C# seems to be gathering an increasing level of developer mind share and has possibly overtaken Visual Basic as the modern favorite. According to a *Visual Studio* magazine survey (`http://visualstudiomagazine.com/columns/article.aspx?editorialsid=2333`), 41 percent of the developers polled now use C#, contrasted to 34 percent for Visual Basic. Of course, this is only one survey, but other hints abound. For example, Microsoft often makes examples available for new technologies in C#, rather than Visual Basic as it did in the past.

For the purposes of this book, it doesn't matter whether C# or Visual Basic is the new developer mind-share king. What does matter is that using C# means you can find the developers you need for a project with greater ease when using either C# or Visual Basic than when using other languages. In short, using C# provides a big advantage in locating the specific skills your project requires. Given that many of the new C# developers on the market are converts from Visual Basic, you may very well find that the C# developer you hire to help with a project also has Visual Basic knowledge, giving you essentially two developers for the price of one.

Understanding the Deficiencies

Every programming language has a deficiency or two. The very act of addressing one need necessarily means that the language isn't addressing another requirement. C# tends to provide lower level functionality than Visual Basic, but it isn't quite as low level as C++. Consequently, C# tends to cover the middle ground of programming well, but you may find areas where you really need another language to get the desired result. The following sections describe some C# deficiencies you should consider when you begin developing your language environment.

Code Reliability Concerns

When you add the power to work with unsafe code to any language, you also make it possible for the developer to create unreliable code. The low-level functionality that makes C# attractive also makes it a dangerous tool in the wrong hands. A developer needs to know how pointers work before using them in a production environment. The detailed level of debugging that C# provides is only useful when the developer understands what the debugging displays are showing. It's possible for a well-meaning developer to create a terrifying time bomb using the wrong techniques in C#.

C# also assumes developer proficiency in other areas. For example, Visual Basic checks math operations and provides the developer with feedback about errors that result in overflows or underflows. By default, C# doesn't check numeric operations. The result is an application that can execute faster than a comparable Visual Basic application, but this feature is only useful in experienced hands. You can overcome this C# deficiency by using the `checked` and `unchecked` contexts within your application.

Another potential code reliability concern is that C# doesn't automatically correct variable name punctuation. If you use the wrong case for a variable name in Visual Basic, the Integrated Development Environment (IDE) automatically corrects the name to use the correct case. Consequently, you could write a C# application that contains `myVariable`, `Myvariable`, and `MyVariable`, all three of which are different variables.

Visual Basic has its own code reliability concern in that strong typing is optional. Without strong typing, an application can handle data incorrectly and corrupt them. Fortunately, the default in Visual Basic is to rely on strong typing. In addition, converting `True` *to a numeric value could yield* `1` *or* `-1` *depending on the conversion used.*

Too Much Information

A Visual Basic developer doesn't see anything other than essential information. Although the Visual Basic developer can optionally reveal this information, the default is to keep the programming environment simple. Consequently, while a C# developer spends time looking through unneeded entries, the Visual Basic developer can locate a needed item quickly. For example, Visual Basic hides certain project files from Solution Explorer and hides namespace information from the editing area.

Microsoft has embraced information hiding as a means of simplifying interfaces. The information hiding appears more often in user applications such as Office (where, as an example, unused menu options disappear from view), but it's interesting to see how the same approach can help developers. Of course, information hiding also means that a developer has to search for a particular item on the few occasions the item is necessary for development. Consequently, information hiding is a two-edged sword that normally works for the developer, but on rare occasions can turn against the developer as well.

Coding Complexities

Anyone will tell you that C# isn't as complex as C++ when it comes to creating programming constructs, but you'll also discover that languages such as Visual Basic make coding even easier. For example, creating delegates is considerably simpler in Visual Basic than in C# because Visual Basic lets you use `AddressOf MyObject.MyFunction` to access the delegate function.

One of the major missing elements in C# is optional parameters. Both Visual Basic and C++ support optional parameters. The lack of this feature makes it considerably harder to create Office applications using C#. In addition, C# lacks support for variables in property methods, which may make it harder to create certain classes of application.

Visual Basic provides a convenience item not found in C#. You receive automatic wire-up of events with event handlers using the handlers construct. Although this is a small feature, it does improve developer efficiency and makes it possible to see the wire-up in the same place that the event handler appears.

Developing with Multiple Languages

Most applications you create use a single language because this tends to reduce the cost of updates later, and using a single language makes the code significantly more manageable. However, your team may consist of a mix of developers, in which case, using the languages that each developer knows will make the development process faster. In addition, some languages do provide capabilities that are not robust enough. For example, even though C# does provide great P/Invoke functionality, you may still find an occasion to use C++ for native code modules. The three languages commonly used with C# are:

- ❑ Visual Basic.NET
- ❑ Visual C++
- ❑ F#

Of course, you can use any language you want with C#. Some developers use IronPython (see `http://www.codeplex.com/Wiki/View.aspx?ProjectName=IronPython` for details) with C# when they need Python flexibility matched with .NET capability. The following sections describe some of the benefits of using additional languages.

Using Visual Basic.NET

Some organizations view Visual Basic.NET with disdain due to its naming similarity with the ancient BASIC language. The two languages couldn't be more different. Visual Basic.NET provides significant capability and exceptional development performance. The fact that Visual Basic.NET often hides complexity that C# developers must deal with as part of their development environment means that you can use it to create code fast. Visual Basic.NET also tends to protect the programmer more from accidents that C# developers must keep in mind, such as numeric overflows. It isn't a bad idea to require less-proficient developers to use Visual Basic.NET until they gain experience.

Using C++

It's easy to think that C++ has seen better days and then relegate it to the dustbin alongside COBOL and FORTRAN. Interesting as it may seem, companies today are actually looking for a few COBOL and FORTRAN programmers to maintain the huge mass of code written in an earlier time for mainframes that still work today. In fact, you can read an article about the need for COBOL programmers in California at `http://weblog.infoworld.com/tech-bottom-line/archives/2008/08/calling_all_cob.html?source=NLC-DAILY&cgd=2008-08-07`. C++ is an older language, but it's also a capable language that can help you bridge the gap between the managed programming environment and Windows.

Using C++ is almost mandatory in some cases. For example, if you have a real-time application that works with specialized equipment, such as factory automation, C++ is almost a required part of your application development plans. C# does provide better capabilities than most .NET languages for working with low-level code, but it can't fulfill every low-level need easily. The coding becomes quite messy at some point with a host of unsafe code sections. You may as well use C++, a language designed to get the low-level programming task done.

Using F#

In many cases, you need to work with data in new ways in today's applications. For example, most systems today come with multiple processors, but most programming languages don't use this capability very well. F# is a new language that joins other functional languages such as ML (`http://en.wikipedia.org/wiki/ML_programming_language`), OCaml (`http://caml.inria.fr/`), and Haskell (`http://www.haskell.org/`). A functional language differs from an imperative language such as C# or Visual Basic.NET in that it works with equations. You can't assign values to F# identifiers — the value of an identifier always remains the same. However, you can interact with that identifier in a number of ways to analyze the identifier and deduce new results from existing data.

The reason you want to add F# to your repertoire is that it lets you use those multiple processors in your system with extreme efficiency. Because an F# identifier never changes value, it doesn't matter which processor works with it. You don't have to worry about state information either. The lack of tracking baggage means that F# applications are quite fast, too. In short, F# adds a new dimension to your application development environment that differs from many other languages you use. You can find a good overview of F# at `http://www.devsource.com/c/a/Languages/Exploring-the-New-F-Language/`.

Multiple Platform Scenarios

Multiple platform scenarios are becoming more common as companies merge and developers see a need to explore open source. Fortunately, you can still use C# for your project while also working with Java and back-end servers such as IBM WebSphere. Although the tools are a bit hard to find, vendors such as Mainsoft (makers of Visual MainWin, `http://www.mainsoft.com/products/index.aspx`) can make things work in a multi-platform environment. You can read a review about Visual MainWin at `http://www.devsource.com/c/a/Add-Ons/Mainsoft-ASPNET-AJAX-Linux/`.

Fortunately, you have a number of options in a multiple platform environment. Although Visual MainWin allows complete sharing of code between platforms, you could also rely on Web services or other techniques when the need for code sharing is minimal and you don't require the ultimate in application speed. Vendors also provide a number of bridging tools, such as JNBridge (`http://www.jnbridge.com/`) that offer a middle bound between the full cross-compiler solutions and Web services.

No matter what kind of multiple platform solution you use, it does affect the language environment to some extent, which means you must include this requirement in your decision. Even a full solution, such as Visual MainWin, does require some changes on the part of the developers in your organization to accommodate the tool. When working with a Web service, the developers on your team will have to create a completely different kind of application.

Many multiple platform solutions are available, so don't accept the first solution you see. For example, many developers are unaware that you can turn a COM+ application into a Web service. In this situation, you could gain a speed advantage by accessing the COM+ application locally and using the Web service for remote access. You can even turn your C# application directly into a COM+ application and then make the resulting COM+ application accessible as a Web service (see the technique at `http://www.devsource.com/c/a/Using-VS/Designing-COM-Applications-with-a-Web-Service-Appeal/`*).*

Inventorying Your Tools

You've probably collected a host of tools while working on various applications. The problem for most developers isn't a lack of tools — it's more likely too many tools. In fact, you may not even know which tools you have available. Because the tools you use affect the language environment, it's important to choose the right tools. For example, you may have two design tools, but one is geared more toward database applications and the other is geared more toward Web services. If you're creating a database application, then you want to use the first tool.

A tool inventory should include all your available tools. You should categorize them at time of purchase to keep application tasks in mind. However, when you begin a new project, you should also categorize them with your development team in mind. When you have two tools that will perform equally well in creating a particular application, then you need to choose the tool that works best with the capabilities and skills of your development team. Of course, you'll want to ask the team members about experiences they've had with the various tools in your arsenal.

It's easy to overlook some important issues when creating a tool inventory. The following list details some issues you should consider while creating your inventory:

- Do you have enough licenses for the tool?
- Are the members of your team skilled in the use of this tool?
- Does the tool provide everything needed for this application, or do you need to purchase add-ons for it?
- Are there potential alternatives for this tool that your company can acquire?
- Will the tool integrate successfully with the development environment?
- Are any of the team members reluctant to use the tool due to a bad previous experience with it?
- Will it be difficult to find replacements for a particular tool if the team member who knows how to use it leaves?
- Is the tool completely up to date with patches and service packs?

Gathering Your Resources

The vast majority of your programming resources come in human form. Even if you're using an article on the Internet as a source of information, a human wrote the article. The people on your team represent the largest potential source of change when it comes to the language environment. In fact, you should consider three potential levels of change to the language environment:

- Language resources
- Skills
- Experiences

The human resources for your project are further affected by biases, preference for particular tools, and the desire to make their mark on the project. The following sections provide an overview of the resources you need to consider as part of the language environment. The remainder of the book provides additional details on this topic as appropriate.

Inventorying Team Language Resources

Anyone with a four-year computer science degree will have experience in more than one language. The experience may not be enough to write an application, but the fact that users have experienced the language means that they have learned something about it. The experience reduces the learning curve for the language and makes it possible that the team member could potentially use that language as part of your project. The only problem with colleges and universities is that they seem to have differing language plans — one school may emphasize Java while another emphasizes Visual Basic.NET. You can't determine that someone has a certain set of skills based solely on the person's degree.

Self-taught individuals also have a wealth of experience with languages. Someone who starts with Visual Basic for Applications (VBA) as part of working with Office may move on to Visual Basic, and from there on to Transact Structured Query Language (T-SQL). After spending a lot of time on the Web, this team member may have also learned JavaScript and other languages in order to set up a personal Web site.

Unfortunately, you won't discover any of these skills until you conduct an inventory. Just because a team member is currently working with C# doesn't mean this person lacks other programming language skills. In order to create the best design for your application, you must also know that a particular team member worked extensively with C++ at one time and specialized in creating DLLs for another organization.

Considering Team Skills

It's nice that you know a particular team member knows C#, C++, and Visual Basic with a smattering of T-SQL thrown in for good measure. However, most languages are extremely complex, so it's possible for someone to become proficient in one aspect of a language, such as database design, and not in others. Consequently, knowing that the team member wrote code that accessed SQL Server databases is better than simply knowing that this member knows how to write in C#. Every application development experience that team members have improves their skill. However, skills always focus on a particular need, so you also need to know about the need that the application answered.

Skills are also transient. A team member's skill fades as time passes without exercising that skill. In addition to determining that the team member has written code to access SQL Server, you also need to know how much time has passed since the team member engaged in that activity. A team member who wrote SQL Server code eight years ago will offer far less proficiency than someone who performed the same task two weeks ago. It's important to remember, though, that people can polish their skills quickly. Even eight-year-old experience is better than no experience at all.

The final skill consideration is learning environment. A VBA programming skill developed using Office 95 is less useful for today's application needs than a skill developed using Office 2007. The same concern holds true for any skill. The skill is usually matched to a particular product version, which means you must consider what has changed since the team member developed the skill. For example, in the case of Office 2007, the Microsoft Office Fluent User Interface (RibbonX) significantly reduces the usefulness of a skill developed using Office 95 because many of the older techniques no longer work.

Understanding the Significance of Experience

Unless your team members are joined at the hip, they've had different development experiences. A development experience helps shape the way the team member views a particular application development requirement and may help you achieve a better result. The developer experience includes these aspects:

- ❑ Language
- ❑ Project type
- ❑ Project size
- ❑ Individual project task
- ❑ Tools used to write the application
- ❑ Individual or team effort
- ❑ Manager personality and management style
- ❑ Developer personality and management style
- ❑ Skills of other developers on the team
- ❑ Success or failure of the project

All these project aspects affect the members of your team in some way. The more positive experiences with projects of the same type that you plan to create, the better suited that member is for the project. However, even negative experiences are helpful. A team member who has had a negative experience and taken time to analyze what went wrong knows at least one technique that won't work for your project. Often, these team members can spot potholes in the road before they become a problem and can even warn you about them.

Some managers have a hard time accepting input from the members of their team. It's important to accept such input without necessarily implementing it. Any input can help you create a better project, but not all input is useful for your particular project. Likewise, it's hard for some team members to share experiences, especially negative experiences. A good manager can draw out these experiences in a positive manner and use them in a constructive way to improve the project. Of course, as with any skill, it takes time to develop this skill and practice is the best way to build the skill.

Defining External Resource Requirements

You may not know whether you need help from outside sources today. However, it's a good idea to begin looking for potential holes even at this early stage of the application development process in your team's language resources, skills, and experiences. The earlier you can identify the need for additional resources, the better your chances of finding precisely the right resource. In some cases, you may find that you can't hire the resource you need, which means that you'll have to set up a consulting contract with the individual as soon as possible. Otherwise, you may get to the point of needing help today only to find that the resource won't be available anytime soon.

It isn't uncommon to find that a top-notch consultant is booked for a year or more in advance. The more specialized your need, the further ahead you need to schedule time from a consultant. If you wait until the last minute, you may obtain the consultant services you need, but at an incredibly high price. In addition, because the consultant didn't have time to schedule your project, you'll find that the consultant's time is divided, so you won't get the full benefit of the consultant's experience for your project.

Of course, resources come in a number of forms. It's possible for a project to move along fine until it requires something special. You may need special graphics files to test your application. Unless you schedule the artist's time well in advance, the images may not be ready when you need them. The same holds true for any specialized need. Many projects today require developers with unique talents, such as the ability to mix sight, sound, and code into a cohesive whole. Often, you know about these needs even before you begin much planning because the basic tasks require some level of specialized interaction.

Developing Your Design Strategy

The purpose of this chapter is to acquaint you with the language environment. Many projects fail because the designer didn't start with the right language. Sure, the language the designer chose could work, but it didn't because it wasn't the optimal choice. Of course, designers have to work in the real world, so this chapter discusses real-world language environment concerns, such as the availability of staff with the correct language skills. The most important idea that you can take away from this chapter is that there isn't a silver bullet language that works in every situation. Sometimes, you even have to mix languages in order to achieve optimum results in a minimum of time.

It's important to get started with your project as quickly as possible, but it's also important to perform the right steps at the right time. The essential step described in this chapter is to create a language environment — to consider how the use of a particular language will affect the development of your application. Take time now to consider the basic tasks your application performs. You don't have to provide a sequence or anything complex, just the simple ideas of retrieving data, processing it somehow, and presenting it to the user. Inventory the languages that your team knows and then assign those languages to the various application tasks. This simple step helps you understand why you're using a particular language in a particular place. While you work on that language inventory, you may also want to create an inventory of skills and experiences.

Applications have a lifecycle. You design, write, maintain, and finally retire them. It's important to consider the entire application lifecycle before you write the first line of code. Chapter 2 presents the application lifecycle and the different strategies you can use to implement the lifecycle. Of course, you perform this thought process before you begin designing the application. Mulling the application in your mind and writing some basic ideas down on paper are the steps you take after you consider the language environment.