



The SharePoint 2010 Developer Platform

An Introduction for ASP.NET Solution Architects



David Chappell, Chappell & Associates

September 2009

Contents

SHAREPOINT 2010 AS AN APPLICATION PLATFORM.....	2
WHAT IS SHAREPOINT?.....	2
BUILDING SHAREPOINT APPLICATIONS	3
MICROSOFT SHAREPOINT FOUNDATION 2010.....	5
THE EXECUTION ENVIRONMENT	5
WORKING WITH DATA.....	8
WRITING BUSINESS LOGIC.....	10
CREATING USER INTERFACES	12
PROVIDING ACCESS CONTROL.....	14
THE SHAREPOINT 2010 EXECUTION ENVIRONMENT: A CLOSER LOOK	14
FARM AND SANDBOXED SOLUTIONS	15
USING SHAREPOINT ONLINE.....	16
SHAREPOINT 2010 DEVELOPER TOOLS.....	17
USING MICROSOFT SHAREPOINT SERVER 2010	18
THE SHAREPOINT COMMUNITY ECOSYSTEM.....	20
TARGET APPLICATION TYPES	20
CONCLUSIONS	22
ABOUT THE AUTHOR	22

SharePoint 2010 as an Application Platform

What is an application platform? It's a broad idea that can include a range of interconnected technologies. ASP.NET is a platform for creating Web applications, for example, while at the same time belonging to the larger application platform provided by the .NET Framework.

SharePoint 2010 also provides an application platform. This platform relies on ASP.NET and other .NET Framework technologies, but it adds plenty of SharePoint-specific functionality. A developer creating an application on SharePoint will use ASP.NET, but he'll also use other services that the platform provides.

Because the SharePoint 2010 application platform is based on ASP.NET, learning how to use it isn't a huge leap for ASP.NET developers. But when does it make sense to build an application on SharePoint rather than on raw ASP.NET?

The goal of this paper is to answer that question. Doing this requires describing what the SharePoint 2010 platform provides as well as looking at the types of applications SharePoint is designed to support. If your application can benefit from the SharePoint framework, building it on SharePoint will let you focus more on creating value and less on building infrastructure.

What is SharePoint?

If you're like most developers, you probably think of SharePoint as a technology that lets users create their own Web sites, then use those sites to do things like share documents. This is exactly right. Rather than needing to work with an ASP.NET developer, deal with a database administrator (DBA), and talk to their Web admin, a user can create a new site by just filling in and submitting an online SharePoint form. And since users want many different kinds of Web sites, SharePoint includes many different templates, each defining a specific style of site.

In this sense, SharePoint acts like an application. Yet the Web sites that SharePoint creates for its users are implemented with ASP.NET. Because developers can create their own ASP.NET-based applications on the foundation that SharePoint provides, it's also correct to view this technology as providing a developer platform.

Understanding that platform requires first teasing apart the main SharePoint components. The 2010 release of SharePoint can be viewed in three parts:

- Microsoft SharePoint Foundation 2010: The basis of everything else in SharePoint, SharePoint Foundation 2010 is the successor to Windows SharePoint Services 3.0. It contains the basic services that developers use to build applications on this platform, and it's available as a free download for Windows Server 2008.
- Microsoft SharePoint Server 2010: The successor to Microsoft Office SharePoint Server (MOSS) 2007, this product contains a group of technologies that address more specialized problems. Among the functions SharePoint Server 2010

provides are enterprise content management, enterprise search, and support for using InfoPath Forms. While SharePoint Server 2010 is built largely on SharePoint Foundation 2010, it's a separate product—using it requires purchasing a license.

- SharePoint Online: A Microsoft-hosted version of SharePoint technology, this offering lets both users and developers use SharePoint's functionality without installing any on-premises SharePoint software.

A note on terminology: This paper uses "SharePoint" as an umbrella term for all of these things. Whenever it's important to distinguish among them—and it often is—a specific name is used.

It's worth pointing out just how popular SharePoint has become. WSS 3.0, the predecessor to SharePoint Foundation 2010, is part of Windows Server, and it's in use at many, many organizations today. Microsoft Office SharePoint Server 2007, the predecessor to Microsoft SharePoint Server 2010, is a billion dollar business for Microsoft. Microsoft's commitment to SharePoint is evident—it's a profitable area for the company. For developers, this popularity and commitment help make SharePoint a safe platform to build on.

There's an obvious question here: If SharePoint provides such a useful platform for creating ASP.NET-based applications, why don't more ASP.NET developers use it today? A big part of the answer is that, prior to SharePoint 2010, the SharePoint platform wasn't as developer-friendly as it might have been. With SharePoint 2010, Microsoft has worked to correct this, making the platform more natural and easier to use for mainstream ASP.NET developers. Today, your ASP.NET skills are likely to fit quite well with the SharePoint world.

Building SharePoint Applications

SharePoint lets ordinary users—non-developers—do useful things all by themselves. Along with creating Web sites, users can customize many aspects of how a site looks and acts, such as its user interface, without relying on a developer. If a SharePoint-knowledgeable developer is available, though, more customization of a SharePoint site is possible. For example, adding a new menu item to a site's user interface, then writing code to carry out an action for that new option requires a developer. A significant majority of the code written for SharePoint today provides some kind of customized behavior for a SharePoint Web site created by a user.

Yet writing this kind of customization code for SharePoint isn't the focus here. Instead, imagine that you're assigned the task of creating a new ASP.NET application. You can build the application in the usual way, using the .NET Framework, Internet Information Services (IIS), and a relational database such as SQL Server. Or, if you choose to, you can build it as a SharePoint application. It's important to understand that the SharePoint development platform itself relies on the .NET Framework, IIS, and SQL Server. Because of this, creating a Web application on this platform means using Windows technologies you already know.

To get a big-picture view of where the SharePoint platform adds value for developers, look at the abstract model of a Web application shown in Figure 1.

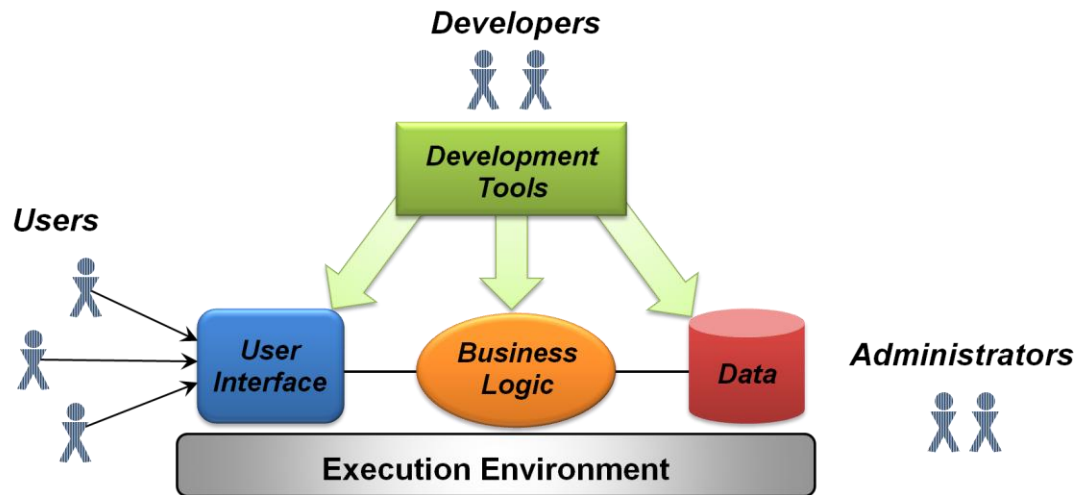


Figure 1: Web applications commonly have the same basic components.

As the figure shows, a Web application typically includes a user interface, business logic, and data. The application runs on some kind of execution environment, and it's created using one or more development tools. Interacting with all of this technology are people in three different categories: users, developers, and administrators.

With this picture in mind, it's possible to summarize the main things that the SharePoint 2010 development platform provides:

- A widely deployed execution environment for SharePoint applications, complete with trained administrators who understand this environment.
- A built-in way to work with data as lists, together with the ability to access other data, including data stored in relational databases.
- Support for building business logic using ASP.NET pages, workflows created with Windows Workflow Foundation (WF), and in other ways.
- A way for developers to create a user interface in the common SharePoint style that many users already know, then let users customize this interface themselves.
- A set of development tools, both Visual Studio-based and SharePoint-specific, to help create and maintain applications for the SharePoint environment.

An application written for the SharePoint platform can use all of these options or just some of them—there's no requirement to adopt everything.

In fact, one way to think of the SharePoint platform is as a Microsoft-provided application framework for Web-based software. Creating an ASP.NET application from scratch usually requires specifying a user interface, doing some database design, and maybe even setting up a server farm to run the application. To make this simpler, many organizations have created their own in-house application frameworks for ASP.NET, providing standard solutions for these areas. Yet for at least some applications, using the Microsoft-provided SharePoint platform is a better idea. Much of what you need can be provided by SharePoint Foundation 2010, so there's nothing

extra to buy. And as always, using a framework can let you focus more on creating the business logic that provides the application's value and less on building infrastructure.

Buying into the SharePoint world does require some changes, however. For example, ASP.NET developers are accustomed to having control of their environment, such as the ability to directly edit their application's web.config file. For SharePoint applications, this file is under the control of your SharePoint administrators, not you. While changes are possible, they must be made programmatically. Also, access to SharePoint services typically requires using the *SharePoint object model*, a set of classes specifically designed for this environment. Learning these classes isn't especially difficult, but it is different from the standard ASP.NET environment.

As with any application framework, understanding the platform you're building on is essential to writing good code. The first step in grasping the SharePoint development platform is understanding what SharePoint Foundation 2010 provides. The next section takes a look at this fundamental technology.

Microsoft SharePoint Foundation 2010

For application developers, SharePoint Foundation 2010 is the most important part of SharePoint. In fact, while SharePoint Server 2010 provides plenty of useful services for application developers, it's possible to build applications entirely on SharePoint Foundation.

The services that SharePoint Foundation 2010 provides for developers can be grouped into four categories: data, business logic, user interface, and access control. This section describes all four. First, however, we need to walk through the basics of the SharePoint execution environment.

The Execution Environment

SharePoint is built on ASP.NET. Accordingly, the SharePoint execution environment looks much like any other group of machines set up to run ASP.NET applications. This group, called a SharePoint *farm*, is illustrated in Figure 2.

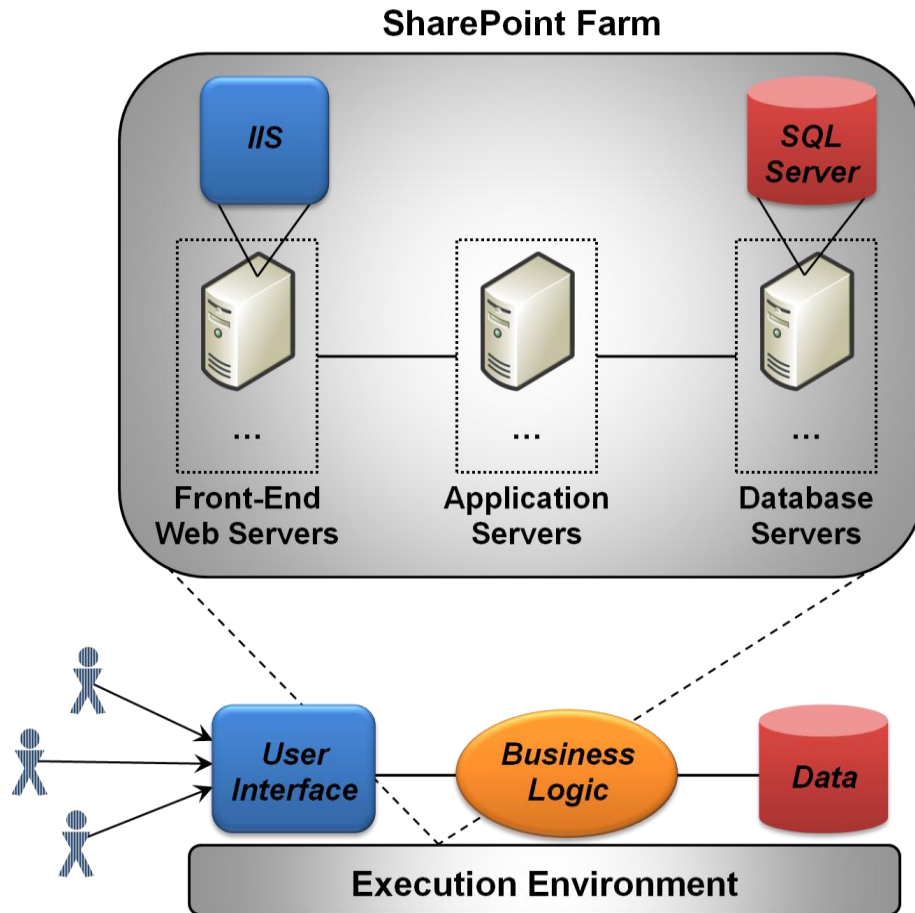


Figure 2: A SharePoint farm contains one or more machines dedicated to running SharePoint and SharePoint applications.

As the figure shows, a SharePoint farm always contains IIS and SQL Server. This example shows the most general possibility—separate machines for IIS, applications, and SQL Server—although a farm can be simpler than this. In a development environment, for instance, it's typical to run everything on a single machine. The right structure depends on variables such as how the farm is used and how many users it must support.

Whatever the physical structure of a SharePoint farm, its logical structure is defined by one or more SharePoint *sites* grouped into *site collections*. Figure 3 illustrates the relationship between sites, site collections, and a SharePoint farm.

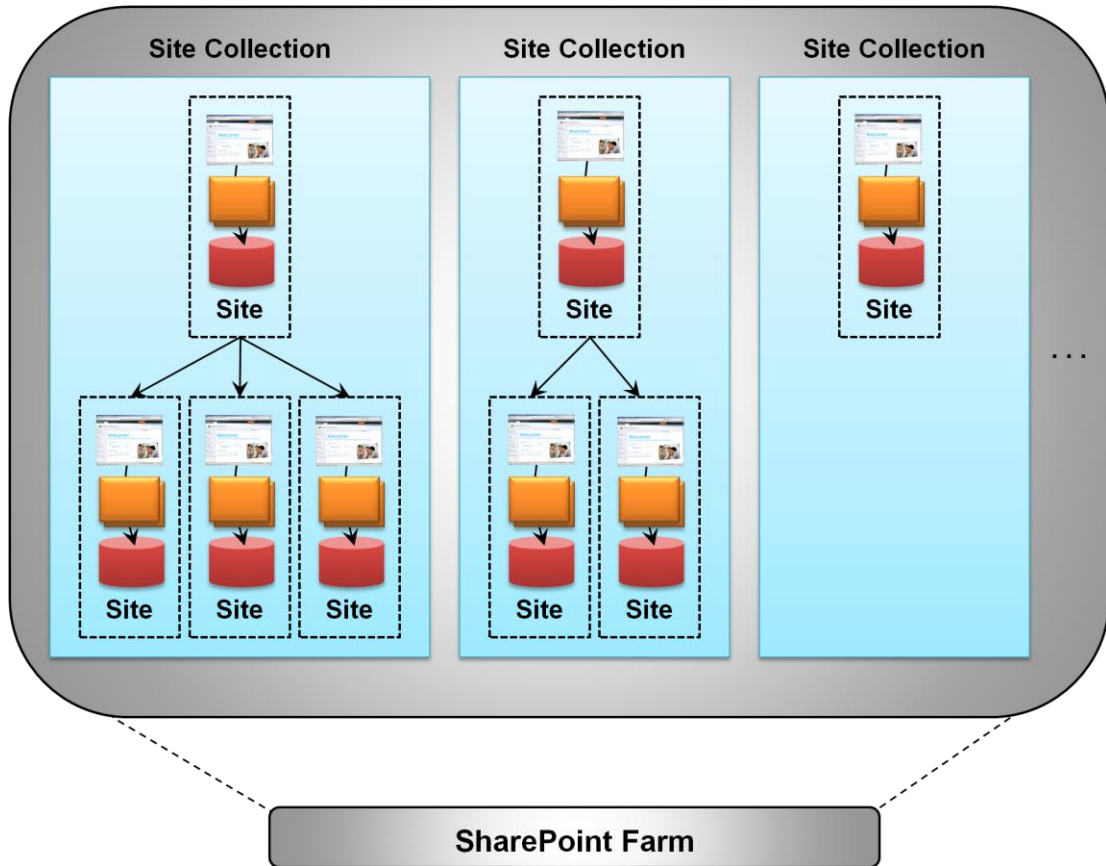


Figure 3: A SharePoint farm can support multiple SharePoint sites grouped into site collections.

A SharePoint site is just what it sounds like: It's a Web site that one or more users can access. SharePoint Foundation 2010 includes a variety of different templates for creating SharePoint sites, and Microsoft SharePoint Server 2010 provides still more. For example, SharePoint Foundation 2010 provides pre-defined sites that let users share documents, create blogs, and many others.

As Figure 3 shows, each SharePoint site belongs to some site collection. A site collection always has a root site, and this root can optionally have other sites beneath it. Although it's not shown in the figure, those sites can also have children, and so on, allowing a site collection to be a multi-level hierarchy.

To understand why SharePoint sites are organized this way, think about a primary purpose of this technology: giving organizations a quick and easy way to create and use Web sites. Making an organization's users rely on their IT department to create every site isn't an attractive option, especially in big companies. With SharePoint, a user—not an IT person—is typically given administrative permissions to a site collection. Rather than relying on IT, this site collection admin can create new sites in the collection, add and remove users, and more. She can also create site administrators, people with power over a single SharePoint site in this collection. Yet being a site collection admin or a site admin confers no administrative rights to the underlying machines in the SharePoint farm. This lower-level work, things such as adding new machines to the farm and creating site collections, is done by the

administrators of the farm itself. This multi-level administrative structure neatly separates responsibilities, giving both IT and users the rights that they need.

Working with Data

Both the people who use SharePoint and the developers who create SharePoint-based applications must work with data. And while SharePoint is built on SQL Server, it doesn't expose ordinary relational tables. Instead, SharePoint data is represented using a higher-level abstraction called a *list*. For end users, lists are simple to understand and easy to use.

Lists can also be useful for application developers. But SharePoint applications need other ways to work with information too, such as access to data stored outside the SharePoint environment. Fortunately, SharePoint Foundation 2010 makes this possible.

In fact, there are three ways for a SharePoint application to work with data:

- Lists, a SharePoint-specific way to represent data.
- External lists, which allow reading and writing various kinds of data stored outside a SharePoint farm as if that data was an ordinary SharePoint list.
- Direct ADO.NET access to relational data stored in databases outside a SharePoint farm.

Figure 4 illustrates all three options, and the rest of this section takes a brief look at each one.

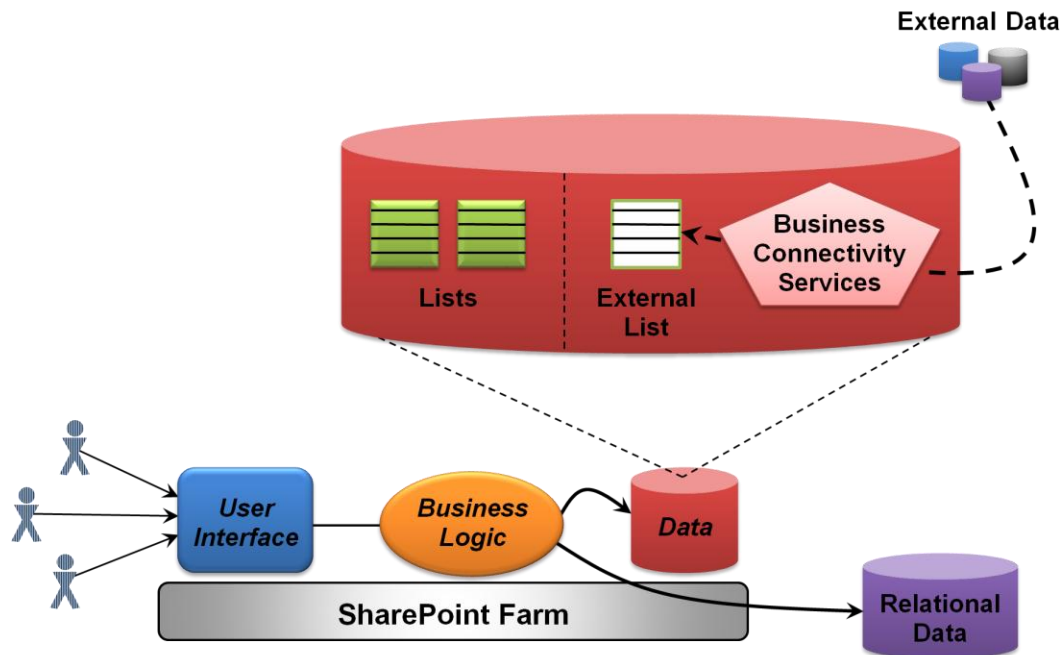


Figure 4: A SharePoint application can work with data in SharePoint lists, in external lists, and in relational databases.

A SharePoint list is just what it sounds like: a list of items, each of which has some number of fields. For example, the items in a list containing information about the students enrolled in a course might have fields such as Name and Grade, while list items describing a firm's products might have fields like Product Number, Size, and Color. SharePoint also provides a predefined list type called a *document library*, which can contain documents such as Microsoft Word files. Document libraries are a specialization of SharePoint's general list structure, adding things like a built-in facility for checking documents in and out of the library. To access lists, applications can use either the SharePoint object model, which provides SharePoint-specific ways to query and modify list data, or LINQ to SharePoint, a version of LINQ designed to be used with SharePoint lists.

Lists can be quite useful. Their attractions include:

- Lists are simpler to use than ordinary relational data. The administrator of a site or of a site collection or even an ordinary user can create new lists and add new fields. There's no need for relational table design, defining connection strings, or negotiating with a DBA.
- Users can work with list data directly. As described later, SharePoint Foundation 2010 provides built-in user interface elements that allow list access via a browser.
- It's possible to create relationships between lists. This allows creating helpful behaviors such as cascading deletes, where deleting a list item also deletes all of the list items related to it.
- SharePoint provides built-in access control mechanisms for lists, a topic discussed in more detail later.

Useful as they are, lists aren't relational databases, and LINQ to SharePoint can't always be used. Instead, things such as access to external lists require using a SharePoint-defined language called the *Collaborative Application Markup Language (CAML)*. CAML isn't especially difficult, but it is another language for developers to learn. Also, because data in lists isn't exposed as standard relational tables, making that data available to technologies such as SQL Reporting Services can take work.

Still, lists are the primary abstraction for storage in the SharePoint world. But what about SharePoint applications that need to access data outside this world? Another alternative is to use ADO.NET to access an external relational database. Just like an ordinary ASP.NET application, a SharePoint application is free to do this, as Figure 4 shows. This approach allows broad data access, at the cost of giving up the benefits and simplicity of SharePoint lists.

The third data access option, external lists, is new with SharePoint Foundation 2010. The goal of external lists is to allow access to data stored outside a SharePoint farm while still retaining many of the advantages of lists. To a SharePoint application (and a SharePoint user), an external list looks much like any other list. The big difference is that the data it exposes actually comes from a source outside this SharePoint farm.

An external list relies on a SharePoint Foundation 2010 technology called *Business Connectivity Services (BCS)*. BCS can use Web services, ADO.NET, or custom code to access external data sources, presenting it as an external list. SharePoint users and applications are free to read and write this data just as with ordinary lists.

While external lists are new with SharePoint Foundation 2010, Business Connectivity Services is the evolution of an earlier SharePoint technology called the Business Data Catalog (BDC). This technology was part of Microsoft Office SharePoint Server 2007, however, not WSS 3.0. More important, this previous version was read-only, while the version in SharePoint Foundation 2010 is read-write. These changes make accessing data stored outside an application's SharePoint farm both simpler and more general.

So far, this discussion has focused on how an application running in a SharePoint farm can access data. But what about the opposite problem? How can an application running outside of the SharePoint world access data held in a SharePoint farm? In SharePoint Foundation 2010, SharePoint lists can be accessed from the outside world using SOAP or the RESTful approach of ADO.NET Data Services. To make this easier, Microsoft also provides client libraries for JavaScript, Silverlight, and the .NET Framework. Using the client object model these libraries provide, a non-SharePoint application can work with lists stored in a SharePoint farm. For example, an AJAX application might use the JavaScript client library to access a SharePoint list, while a Silverlight or .NET Framework application might use a SharePoint document library to provide check-in/check-out capabilities for its users. The key point is that data stored in a SharePoint farm can still be accessed by other software; it's not available solely to SharePoint applications running in that farm.

Writing Business Logic

By definition, applications implement logic. If she chooses to, an ASP.NET developer building a SharePoint application can create that logic using familiar .NET Framework technologies. She can also create logic using SharePoint-specific mechanisms. Figure 5 shows the options.

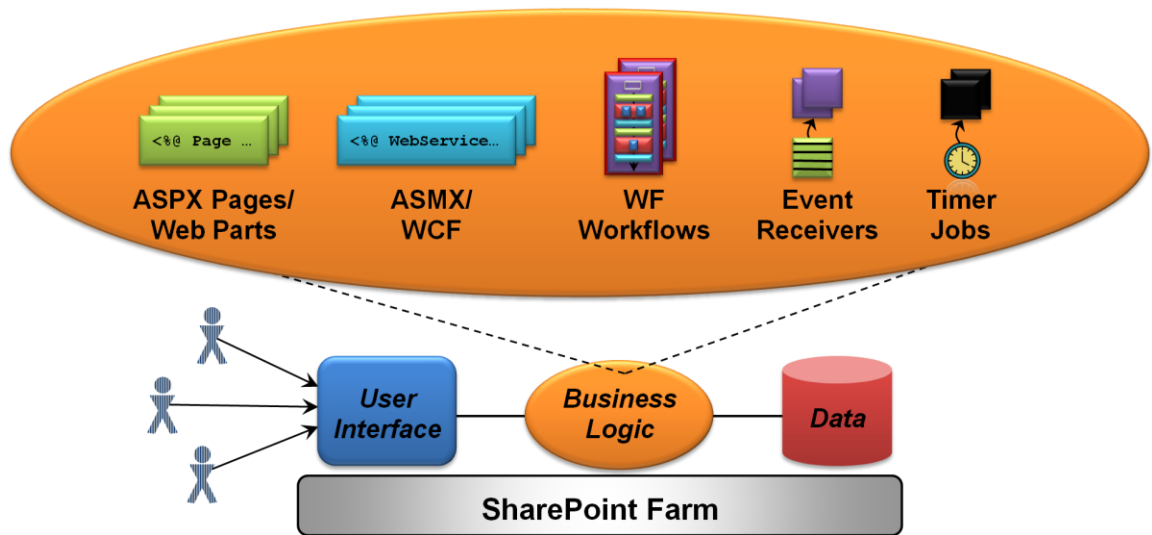


Figure 5: Business logic for a SharePoint application can be created using ASPX pages, ASMX Web services, WF workflows, and more.

To create a SharePoint application's logic, a developer can use ASPX pages, as Figure 5 shows. There are some restrictions—pages typically can't contain script, for example, so code-behind is required, and your application will commonly need to integrate with a SharePoint-provided master page—but still, a SharePoint application is an ASP.NET application. To expose Web services, a developer can use ASMX or Windows Communication Foundation (WCF).

It's common to implement a SharePoint application's logic in a *Web Part*. Most ASP.NET developers today are familiar with Web Parts—they're a standard component of that technology. (In fact, Web Parts originated in SharePoint, then were added to ASP.NET.) A Web Part is just a special kind of ASP.NET control that's deployed inside a Web Part zone. Each Web Part can contain code-behind logic in ASPX pages, and so it provides a self-contained way to implement logic and the user interface for that logic.

A developer can also implement a SharePoint application's business logic as a workflow using Windows Workflow Foundation (WF) 3.5. Workflows are a common choice for supporting long-running business processes, such as employee onboarding and claims processing. By itself, WF provides only the basics needed to create workflows. SharePoint Foundation 2010 fills in the gaps, offering a host (a SharePoint worker process), tools for creating workflows (including SharePoint Designer, described later), a way for people to interact with a running workflow (through a kind of SharePoint list called a *task list*), and more. Building a workflow application on SharePoint Foundation 2010 is significantly easier than building one using just raw WF.

Along with these familiar .NET Framework technologies, SharePoint Foundation 2010 provides two more ways to create business logic. The first, called an *event receiver*, acts much like a trigger in a relational database. For example, a developer can create an event receiver that runs when a specific event occurs for a particular list, such as a user's attempt to update data. The event receiver might, say, validate that data before it's stored in the list, making sure that the user isn't inserting invalid information. Event

receivers can also handle other list-related events, such as a user's attempt to delete a list. Note the difference between a workflow and an event receiver: While a workflow has state and typically runs for a reasonably long time, an event receiver runs only long enough to handle the event that triggered it, and it doesn't maintain any state between events.

One more option for implementing business logic in a SharePoint application is to create a *timer job*. As the name suggests, a timer job is code that runs at a specific time. For example, suppose an application needs to copy data from a SharePoint list to an external system once a day. The creator of that application might write a timer job to carry out this function, setting it to run each day at 4 am.

Business logic is an important part of most applications. Yet to make this logic useful commonly requires exposing it to people through some user interface. How SharePoint allows this is described next.

Creating User Interfaces

Just like ordinary ASP.NET applications, SharePoint applications commonly present a browser-based user interface. To make life simpler for users, SharePoint defines basic user interface elements, together with conventions for how those elements should be applied. Figure 6 shows the basics of the default SharePoint user interface.

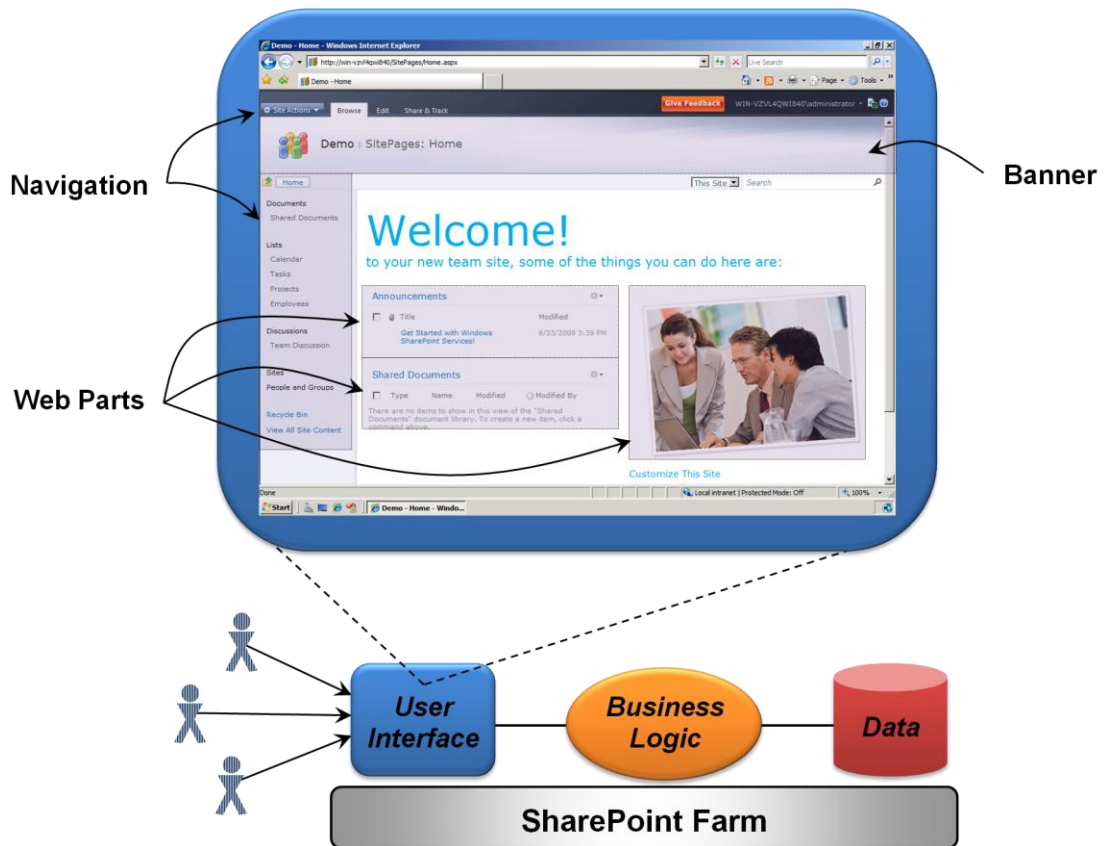


Figure 6: The default user interface for a SharePoint application has a banner, a navigation section, and one or more Web Parts.

As the figure shows, a typical SharePoint user interface provides a banner at the top of the page. This area can be used to show a company logo, display an enterprise search box, or for other things. The default interface also contains navigation areas along the left side and across the top. These typically provide a direct way to access interesting things in the site, such as document libraries and other important lists.

The third element in the standard SharePoint user interface is Web Parts. As described earlier, a Web Part encapsulates logic behind a user interface element. Each Web Part can expose actions, such as the ability to change the content this Web Part is displaying or move the Web Part to a new location on the screen. This lets a user customize the page he sees, rearranging and configuring Web Parts as desired. This flexibility makes it easier for developers to create an application that users can customize.

Developers are free to create their own Web Parts. SharePoint Foundation 2010 provides a number of built-in Web parts, however, and Microsoft SharePoint Server 2010 adds even more. For example, some of the more commonly used Web Parts included with SharePoint Foundation 2010 are:

- List View Web Part: Displays the contents of a SharePoint list
- Data View Web Part: Provides a customizable and quite general way to display a range of data types, including SharePoint lists, XML documents, data from a relational database, and others.
- Image Web Part: Allows displaying an image or graphic.

While it's fair to say that SharePoint applications usually use Web Parts and often use the interface style shown in Figure 6, it's possible for a SharePoint developer to create his own entirely custom user interface. He's not required to follow SharePoint's default conventions. For example, the creator of a public Web site built on SharePoint probably wouldn't use these defaults, choosing instead to implement a more customized look.

Many ASP.NET developers are accustomed to having complete freedom in designing an application's user interface. Why would they ever want to limit themselves by adhering to the default SharePoint interface style? Perhaps the most important reason is that, because SharePoint is so widely used today, more and more people are familiar with this kind of interface. If people feel immediately comfortable with a new application's interface, they're more likely to adopt and use that application. Adopting the default interface style also saves development time, since there's no longer any need to design (and probably argue about) the basic layout of your user interface. Furthermore, because the SharePoint interface relies on Web Parts, your application can easily present a customizable face to its users, and the built-in Web Parts that SharePoint provides can help you create your application more quickly.

SharePoint Foundation 2010 includes some user interface improvements over its predecessor. For one thing, there's now support for a broad range of Web browsers. Just as important, it's now possible to deploy a Silverlight application in a SharePoint Web Part. This lets a SharePoint application download and run Silverlight code in the user's browser. This code can then talk back to the SharePoint application through the client object model or in some other way, accessing SharePoint lists and more. The

result is the ability to create SharePoint applications with more capable user interfaces.

Providing Access Control

ASP.NET applications commonly require their users to authenticate themselves, proving who they are with a password or something else. Once an application knows who its user is, the next problem is to figure out what that user is authorized to do. Building an ASP.NET application typically requires solving both of these problems. Building that application on SharePoint Foundation 2010 can make doing this easier.

Rather than recreating the wheel, SharePoint Foundation 2010 relies on IIS for authentication. Part of setting up SharePoint is configuring an IIS Web site to support multiple SharePoint sites. All of the SharePoint sites contained within that IIS Web site will then use the same mechanism to authenticate their users.

Once a user has been authenticated, the application must make an authorization decision. If it's built using raw ASP.NET, the application will likely contain custom code to do this. If the application is built on SharePoint Foundation 2010, however, it can rely on SharePoint's built-in authorization mechanisms.

To let a SharePoint application use these mechanisms, a site collection administrator first uses standard SharePoint tools to define groups, then adds users as members of these groups. Because they're defined at the site collection level, these groups can be referenced across all sites in this collection. For example, the administrator can use them to define who's allowed access to a specific site in the collection.

More fine-grained access control is also possible. An administrator can specify which users and/or groups have access to specific lists within a site, for example, then rely on SharePoint Foundation 2010 to enforce these restrictions. Rather than requiring the application developer to write her own authorization code for site and list access, SharePoint Foundation 2010 provides this for her.

SharePoint Foundation 2010 also provides standard Web Parts for working with authorization. For example, the Site Users Web Part, which can be added to any Web Part page, displays a list of the users and groups who have permission to use a site. Especially for applications that use lists, SharePoint can make authorization significantly simpler. Once again, the goal is to let developers focus their efforts on meeting business requirements rather than on writing infrastructure code.

The SharePoint 2010 Execution Environment: A Closer Look

To an ASP.NET developer, one of the attractive things about SharePoint is that organizations that use it already have a managed environment with trained administrators. SharePoint farm admins know how to add servers, perform backups, install applications, and more. Because a new SharePoint application can take advantage of this existing human infrastructure, a developer's life can be simpler.

But while SharePoint farm administrators can help, they can also set limits on what SharePoint applications can do. To see why requires understanding how SharePoint applications are deployed, described next.

Farm and Sandboxed Solutions

When a user customizes his view of a SharePoint site, such as by rearranging the Web Parts he sees, those customizations are stored in a *content database*. All of the sites in a site collection typically share a single content database, which is stored in the SharePoint farm's SQL Server database.

When a new SharePoint *farm solution* is deployed, its code is not placed in the content database. Instead, the files for a farm solution must be installed directly in the file system of the farm's Web servers (or, if they're present, on its application servers). The good thing about this is that a farm solution is potentially available to all of that farm's users. Yet think what else this implies: First, farm solutions must be installed by farm administrators. Site and site collection administrators can't do it. Second, a solution installed directly on the farm's Web servers (or application servers) can affect anything in the entire farm. If a farm solution installed for use at just one site within some site collection behaves badly, it can hurt performance or otherwise destabilize the entire SharePoint farm.

If you were a SharePoint farm administrator responsible for keeping a farm running, wouldn't you be very conservative about the farm solutions you installed? In fact, might you even refuse to install new farm solutions at all? Why risk damaging the entire farm and all of its users for solutions that benefit only some of them?

This was a concern with the previous version of SharePoint, since it only supported farm solutions. SharePoint Foundation 2010, however, brings a new answer to this problem: sandboxing. A *sandboxed solution* no longer need be installed by a farm administrator and be made available to any site in the farm. Instead, it can be installed by a site collection administrator and be made accessible only to the sites in that collection. For this to work, a sandboxed solution can't be installed on the farm's Web or application servers. Instead, the solution's code is installed in the content database, alongside user customizations and other information. Figure 7 illustrates this situation.

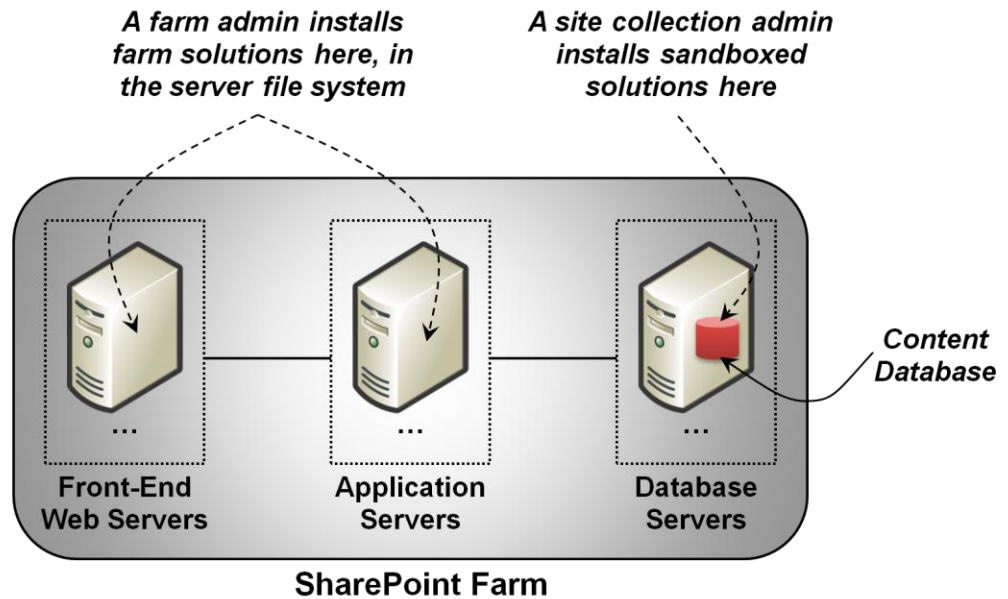


Figure 7: Farm solutions and sandboxed solutions are installed in different places in a SharePoint farm.

Unsurprisingly, sandboxed solutions are subject to some constraints. For example, there are limits to the CPU time and SQL query execution time each sandboxed solution can consume. Without this, a sandboxed solution installed for one site collection could hurt performance for the entire farm. Also, a sandboxed solution isn't allowed to access everything exposed by the SharePoint object model, nor can it access SharePoint data outside of the site collection it's installed in. There are other restrictions too, such as limitations on how workflows can be used. Still, for applications that can live within these restrictions, sandboxing makes SharePoint Foundation 2010 significantly more practical than its predecessor as a development platform.

Whether your app runs as a sandboxed or farm solution, it's packaged into what SharePoint calls a *WSP file*. This file is just a CAB file with a .WSP extension, and it contains all of the parts of your solution: assemblies, configuration information, and more. The nice thing about this is that it provides a single deployment unit for an entire SharePoint solution, wherever it will be deployed.

Using SharePoint Online

One more deployment option that hasn't yet been mentioned is hosted SharePoint. A number of companies provide this service, letting organizations use SharePoint without any on-premises infrastructure. Microsoft also provides its own hosted offering, called SharePoint Online, as part of the Business Productivity Online Suite (BPOS).

In its initial incarnation, SharePoint Online was relatively limited as an application platform. While customers could get their own site collections, running complete custom applications wasn't possible. This changes with the 2010 version of SharePoint Online. With this new release, customers can upload and run sandboxed

solutions. Organizations that want to build SharePoint applications but don't have their own SharePoint farm can now use SharePoint Online instead.

SharePoint 2010 Developer Tools

As with any other application, building SharePoint applications depends on tools. Microsoft provides two choices, each targeting a specific audience:

- *Visual Studio 2010*, which contains templates and tools targeting .NET Framework developers building SharePoint applications
- *SharePoint Designer (SPD) 2010*, targeting SharePoint users and, in some situations, developers customizing SharePoint sites and creating more limited application logic.

Figure 8 illustrates these options.

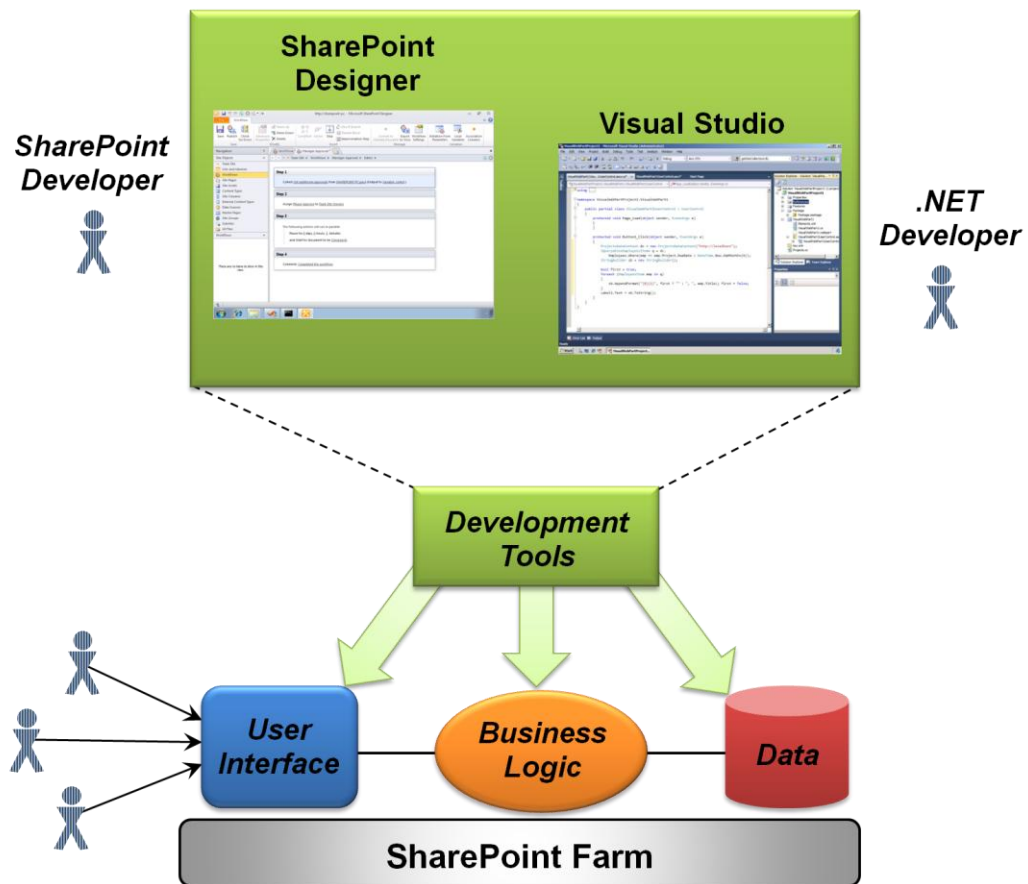


Figure 8: Both SharePoint Designer and Visual Studio can be used in creating SharePoint applications.

Visual Studio is a familiar tool to ASP.NET developers. To create a SharePoint application with Visual Studio 2010, a developer creates a SharePoint project. Once

this is done, Visual Studio 2010 provides a variety of SharePoint-specific features. For example, the Server Explorer for SharePoint lets a developer view SharePoint lists and other information from inside Visual Studio, while various wizards support things such as creating SharePoint event receivers. Visual Studio 2010 also provides tools to create a WSP file, structuring its contents as needed. There's even a checkbox that lets a SharePoint developer indicate that a project will run in a sandbox. Checking this causes Visual Studio 2010 to allow only behaviors that are legal for sandboxed solutions.

All of these are examples of an important philosophical shift in SharePoint development. Prior to SharePoint 2010, creating SharePoint applications with Visual Studio could be challenging—the tools just weren't as good as they might have been. Microsoft set out to fix this with the 2010 release, making SharePoint a first-class development environment with first-class tools. This new perspective is what underlies the significant changes in Visual Studio 2010. It motivated other changes as well, such as the ability for developers to install a SharePoint environment on 64-bit versions of Windows 7 and Windows Vista. This new release also lets the creators of SharePoint applications use Visual Studio Team System and Team Foundation Server, helping improve the development process. Making it easier and more natural for .NET developers to create SharePoint applications was a high priority for this new version of the technology.

Along with Visual Studio, SharePoint developers are also likely to use SPD. SPD can be useful in two ways: as a customization tool for SharePoint sites and as a companion to Visual Studio. In fact, some things that developers want to do are easier in SPD than in Visual Studio. For example, creating a user interface can be simpler with SPD, while customizing the Data View Web Part mentioned earlier effectively requires using this tool. Creating a workflow can also be easier, since SPD provides a workflow designer intended for less technical people.

It's also possible to use SPD and Visual Studio together. For instance, suppose a business-oriented developer uses SPD's designer to create a workflow. A .NET developer can import this workflow into Visual Studio, then add more technical detail. This approach helps professional .NET developers work more effectively with less technically oriented people, each using a tool that's appropriate for their role.

The bottom line is that effective development depends on effective tools. With SharePoint 2010, what was a sub-optimal story gets significantly better. Lack of good tools is no longer the roadblock it once was to creating SharePoint applications.

Using Microsoft SharePoint Server 2010

SharePoint Foundation 2010 is the underpinning for every SharePoint application. Yet for some of those applications, life is simpler for developers if they also use Microsoft SharePoint Server 2010. Unlike SharePoint Foundation, SharePoint Server isn't freely downloadable—it's a separately licensed product. Still, if the built-in functionality it provides saves enough effort, both in development and maintenance, writing this check can make sense.

To a great degree, SharePoint Server is focused on providing immediate value to end users. It's not just a platform for custom development. Yet the product also contains a

large number of Web Parts, .NET assemblies, and other useful components that developers creating SharePoint applications can use.

Microsoft groups the functions of SharePoint Server 2010 into six areas:

- **Content:** One of the most popular uses of SharePoint Server is for enterprise content management (ECM). In general, ECM means managing the flow of documents and other content, including things like creating the content, approving it, making it available to people, and updating it. SharePoint applications focused in this area might benefit from building on these services. SharePoint Server also supports records management, including a toolkit to help organizations build their own custom solutions.
- **Communities:** SharePoint Foundation 2010 allows creating sites with blogs and wikis. SharePoint Server adds extra functionality here, aimed at making these social computing technologies more effective for enterprises. It also includes broadly useful things such as an address book, letting the developers of a SharePoint application focus on writing business-specific code.
- **Composites:** Many applications combine assorted parts into a more useful whole, and SharePoint Server includes several services to help do this. For example, while SharePoint Foundation provides support for workflows, SharePoint Server builds on this with pre-defined workflows for document approval and other purposes. The product also includes Forms Services, a helpful addition for applications that use InfoPath forms. Without Forms Services, a SharePoint application that presents InfoPath forms to its users requires every desktop to have an installed copy of InfoPath. SharePoint Server also adds extra functionality to Business Connectivity Services, such as the ability to connect to Office clients.
- **Search:** Just as Internet search has become part of all our lives, enterprise search matters more every day. SharePoint Server provides a search engine that can be configured to search data in SharePoint as well as data in an organization's line of business applications. This can include full-text searches as well as searches over relational data. A SharePoint application can build on this facility to present search capabilities to its users.
- **Insights:** Providing business insights to users is a fundamentally important goal of enterprise software, and so several aspects of SharePoint Server fit in this category. For example, the product includes Web Parts for displaying key performance indicators. Combined with Business Connectivity Services, this allows creating a SharePoint application that acts as a business dashboard, displaying up to date information about an organization's status gleaned from its most important applications. SharePoint Server also includes Excel Services, which allows storing and processing Excel spreadsheets on a server rather than on each user's desktop. This technology stores those spreadsheets in a SharePoint document library, letting users share a single instance of a spreadsheet as well as providing auditing services for spreadsheet access.
- **Sites:** Letting users create Web sites easily and quickly is a fundamental goal of SharePoint. SharePoint Foundation allows this on its own, but SharePoint Server adds more functionality in this area. For example, it provides support for

publishing Web content, something that's especially useful for working with larger sites.

For developers, Microsoft SharePoint Server presents a buy-vs.-build decision. To make the right choice, anybody creating SharePoint applications should understand what the product offers, if only to avoid recreating what they could more easily buy.

The SharePoint Community Ecosystem

Creating SharePoint applications means becoming a SharePoint developer. As should be clear by now, this isn't a great leap for anybody who's conversant with ASP.NET. Still, even a highly skilled ASP.NET developer should take this transition seriously.

As with any other platform, books and training courses can help you use this new technology more effectively. Bookstore shelves groan with volumes on SharePoint today—it's a popular technology—and many companies offer training for SharePoint developers. There's also a SharePoint developer community, much like that for ASP.NET. Microsoft designates SharePoint MVPs, provides an official curriculum and certification exams for SharePoint developers, and runs MSDN forums devoted to SharePoint. The Patterns and Practices group in Redmond provides guidance for SharePoint development, just as they do for other Microsoft platforms.

While the SharePoint platform owes a great deal to ASP.NET and the rest of the .NET Framework, it's not the same thing. Learning how to use this platform well is essential to creating effective SharePoint applications.

Target Application Types

Every application platform has its sweet spot, the application types for which it's best suited. For SharePoint, the target application types include these:

- Business Collaboration Applications: "Collaboration" is a broad word, but it's fair to think of SharePoint whenever you need to build an application that helps people work together. Whether this application relies on a WF workflow with InfoPath forms or just logic built using Web Parts, creating a SharePoint application rather than a raw ASP.NET application can be a good option.
- Portals for LOB Application Data: By using Business Connectivity Services, a SharePoint application can have direct access to data in SAP, Oracle Applications, and other line of business software. An application that's meant to expose this kind of diverse data to business users is a natural fit for SharePoint.
- One Web Part Solutions: Small, specialized applications implemented as a single Web Part can be an attractive way to use SharePoint. Think of a custom scheduler for your organization's conference rooms, for instance, or a simple app that displays current sales volume. Because the SharePoint user interface allows snapping in diverse Web Parts, users can insert this kind of application in whatever context they find useful.

- Applications that use Microsoft SharePoint Server 2010: If an application can benefit from building on the functions provided with SharePoint Server, it's an obvious candidate for the SharePoint platform. If the app needs enterprise search capabilities, for example, or records management functions or something else that this product provides, using its pre-built services can make sense.
- Corporate Web Sites: An organization might use SharePoint to implement its public Web site. An application like this probably wouldn't use the standard SharePoint user interface style, and it would likely contain other custom code for specific functions. It might also use SharePoint Server for content management or other things.

Understanding the kinds of applications that SharePoint targets is important. It's also important to understand the kinds of applications that aren't a great fit for this platform. Building a high-volume transactional system as a SharePoint application isn't likely to lead to great results, for example, especially if the data is stored in SharePoint lists—they're not designed for this kind of load. Similarly, you wouldn't use SharePoint to support data-intensive applications that don't have end users, such as batch processes or parallel processing software. SharePoint also isn't usually the right platform for application integration. A Microsoft-oriented integration project should instead use BizTalk Server.

What about independent software vendors (ISVs)? When does it make sense for an ISV to build a commercial application on SharePoint? All of the observations above still hold, of course—some kinds of apps are better suited for SharePoint than others. And ISVs face an additional consideration: Building an application on SharePoint limits the market for that application to customers that use SharePoint. If the application uses only SharePoint Foundation, this is less of an issue. SharePoint Foundation 2010 is a free download for Windows Server, so it's easily available. Building on Microsoft SharePoint Server is somewhat more constraining, since now customers for the application must also license this product from Microsoft.

Still, the advantages of building on SharePoint, such as faster development time and a standard execution environment, might outweigh these constraints. In fact, a number of ISVs focus solely on selling to SharePoint customers—it's not a small market. Microsoft's partner program for SharePoint ISVs can help, too, both with building and selling applications. As the SharePoint environment continues to spread, we're likely to see the number of ISVs building on it increase as well.

Conclusions

For anybody who knows ASP.NET, SharePoint presents a familiar world. By letting developers reuse existing skills in a new context, it provides a development platform that can sometimes be a big step up from the raw .NET Framework. And learning how to build new SharePoint applications also makes developers capable of customizing existing SharePoint sites, a skill that's in wide demand today.

In the past, many ASP.NET developers were turned off by the SharePoint platform. It was too hard to work with and too different from the standard .NET world. With SharePoint 2010, Microsoft has made a serious effort to fix these problems. Better Visual Studio integration, support for standard .NET technologies such as LINQ, and other improvements make SharePoint a less unnatural world. The addition of sandboxing also helps, letting developers deploy SharePoint applications without risking the entire farm.

If all you needed to do was build a single ASP.NET application, climbing the SharePoint learning curve might not make much sense. If your organization plans to build a number of Web applications, however, understanding what the SharePoint platform has to offer is worthwhile. By providing a common environment with reusable elements for user interfaces, access control, and other typical functions, SharePoint just might make these applications faster to build, easier to maintain, and more attractive to their users. At the end of the day, aren't these the things that every developer is looking for?

About the Author

David Chappell is Principal of Chappell & Associates (www.davidchappell.com) in San Francisco, California. Through his speaking, writing, and consulting, he helps people around the world understand, use, and make better decisions about new technology.