

Issue

8

UNDERSTANDING SHAREPOINT JOURNAL

Bjørn Christoffer Thorsmæhlum Furuknap

Professional SharePoint Development

UNDERSTANDING SHAREPOINT JOURNAL

Professional SharePoint Development

This book is dedicated to my wife.

© Understanding SharePoint

Rubina Ranasgt. 10

N-0190 Oslo, Norway

Phone +47 91 39 85 86 • Web <http://www.understandingsharepoint.com/>

Credits

About the Author



Bjørn Christoffer Thorsmæhlum Furuknap is a senior solutions architect, published author of *Building the SharePoint User Experience*, speaker, and passionate SharePointaholic. He has been doing software development professionally for 16 years for small companies as well as multinational corporations. He has also been a teacher at a college-level school, teaching programming and development to aspiring students, a job that inspired him to begin teaching what he has learned and learns every day.

About *Understanding SharePoint Journal*

Understanding SharePoint Journal is a periodical published by UnderstandingSharePoint.com. The journal covers few topics in each issue, focusing to teach a deeper understanding of each topic while showing how to use SharePoint in real-life scenarios.

You can read more about *USP Journal*, as well as get other issues and sign up for regular updates, discounts, and previews of upcoming issues, at <http://www.understandingsharepoint.com/journal>.

Other Credits

A great big thanks to Kim Wimpsett for doing the copyedit. The quality of work in this issue is greatly attributed to her skill.









Table of Contents

Credits	i
About the Author	i
About <i>Understanding SharePoint Journal</i>	i
Other Credits.....	i
Table of Contents	1
Exercises	2
Introduction	4
The Problem with SharePoint Solution Development.....	5
Configuring vs. Development.....	5
Anarchy vs. Dictatorship	7
Solutions vs. Features.....	7
Great, So Solution It Is Then!	8
A Better Approach.....	10
Web-Based Prototyping	11
Export to STP	11
Extract Configuration from STP.....	12
Build WSP	12
Deploy and Test	12
Mission Statement.....	13
Requirements	13
Rules of Engagement.....	13
We Will Not Cover	14
Working with the Site Template	15
Creating the Prototype	15
Dissecting the Site Template	19
MetalInfo.....	20
Details.....	20
SiteFeatures and WebFeatures	21
Structure	23
Files	23
UserLists.....	24
WebParts.....	26
Core Solution Development	29

Solution Overview	29
Site Definition or No Site Definition	30
Building the Core Solution.....	31
Site Definition	32
Setup Feature.....	34
Layout Feature	37
Upgrade Feature	40
Testing the Core Solution.....	45
Rebuilding the Site.....	48
Specialization Solution	48
Lists.....	50
Web Parts	53
Web Part Provisioning Feature	54
Web Part-Adding Feature.....	56
Specialization Solution Setup Feature	59
Extending the Solution	63
Upgrading and Maintenance	66
Solution Overview	66
Modifying the Modification.....	67
Mass Upgrades	70
Some Additional Thoughts	71
Upgrading Existing Sites and Solutions	71
Hooking an Upgrade Feature to Existing Solutions	72
Common Solution Features	76
Security.....	77
Content Types	77
CSSHandler.....	78
Default Data.....	78
Final Thoughts and Additional Resources.....	81
Previous <i>USP Journal</i> Issues.....	82
<i>USP Journal</i> Affiliate Program	85

Exercises

 Creating a Prototype Site.....	16
 Dissecting an STP Template	19
 Creating a Setup Feature.....	34

	Building the Upgrade Feature	42
	Creating a Specialization Solution	49
	Adding a List Feature.....	50
	Creating a Multiscope Feature Set.....	54
	Adding a Web Part to default.aspx.....	56
	Creating a Modification Solution	67
	Feature Stapling	72
	Changing the SharePoint Logo.....	75

Introduction

Asking the right question at the right time to the right people can trigger a landslide.

Welcome to the most important publication of your SharePoint development career.

OK, that may be a brave statement, but I think you'll see, as many others have, that the approach you are about to learn is a really powerful method for SharePoint solution development.

In this issue, I am going to teach you an approach to SharePoint solution development that may revolutionize your view of developing agile SharePoint projects. I have developed this approach over many years and multiple projects, both small and massively huge.

In fact, I have used this approach to develop solutions for more than 100,000 users, with developers and participants in multiple countries and with varying skill levels.

In one case, I used the method to reduce a project that was scheduled for a two-year development cycle into just three months of development, and I deployed the first deliverables just two weeks after starting development.

You may also be surprised at the simplicity of this method. In fact, the most important component is a piece of code just 42 lines long that will transform your approach to creating maintainable and upgradeable solutions forever.

This method is equally usable in SharePoint 2007 and SharePoint 2010, and although SharePoint 2010 to some extent supports upgrading features, the flexibility offered using the method in this issue far exceeds that of the built-in feature upgrading.

So, let me take this opportunity to congratulate you on taking another step toward a professional SharePoint developer career.

Let's see what all the buzz is about!

The Problem with SharePoint Solution Development

Make sure you know what you need.

Before we begin thinking about solving any problems, we need to thoroughly understand the problems we face.

In this chapter, I'll introduce you to the issues that we have to face as SharePoint developers and that we need to overcome one way or another.





Then I'll outline the process that we'll use to ensure that we get our users on board in the development process and that we empower them to participate with both creative efforts and feedback.

I'm also going to set the ground rules for our development to ensure that we follow best practices.

Configuring vs. Development

SharePoint's greatest feature is the ease with which end users with little or no training can add or modify existing features to make SharePoint do what the user needs. The user simply adds or modifies lists, content types, and pages; activates or deactivates features; and makes configuration changes. Then—boom!—you have a fully customized solution, tailored to the user's need.

This method of development is called *configuration*. You don't actually create anything; you simply utilize what is already there and customize new lists, pages, or other artifacts to create a solution.

ICON KEY	
	Valuable information
	Test your knowledge
	Exercise
	Caution

As professional developers, we often shun this approach. First, it is difficult to maintain a coherent solution in the long run if users are given free reign over what they do. If your solution depends on the presence of a list and users may change that list, you cannot be certain that what you need really exists.

Second, designing business-critical systems is not for the faint of heart. There are so many factors to consider, such as data integrity, scalability, data models, user interface, user experience, and so on. There is a reason why software development is a trained skill and not just a hobby, at least not on the higher end of the scale.

Note

I'm not saying that business users are less intelligent; it's quite often the exact opposite, but the fact that you may be smarter than your doctor does not mean you are qualified to do your own bypass.

In addition, the options available for configuration are limited by nature. Someone has to create the lists that the user will then customize. True, SharePoint does offer a wide variety of lists, content types, and other elements out of the box, but users more often than not end up wanting more.

Something as simple as a customer list, or a list of employees, does not come out of the box. Although creating a customer list may seem easy enough, how will your solution scale, and how will you reuse that customer data once you have spent weeks or months filling the list with critical information?

Finally, some options are simply left out of the configuration method altogether. You cannot add new application pages using configuration, for example, and you cannot in any way configure them or update them if you need to add functionality.

The power we have as developers is far beyond that of configuration. After all, we have skills ranging from generic skills, such as data modeling, user experience, and programming, to SharePoint-specific skills and platform knowledge far deeper than most end users will ever have.

However, we do trade time and ease for that power, because developing SharePoint solutions takes a lot more work than simply hitting Create every once in a while.

Of course, with that added time and complexity, you may not want to do much development work at all, or at least the cost will be prohibitive for any small updates.

Or is it?

Anarchy vs. Dictatorship

So, as developers, we create features for our users and package them in a solution. For example, let's say our company needs that customer list. You create a content type and a list, allowing the user to create the customer list as needed. The user happily sees that they can now create a customer list, just like they wanted it when you created the customer list.

However, a few months ahead, the user of one department wants to add the birth date of the customer's primary contact so that the company can send a bouquet of flowers at that date. The user adds a new column to the list, and poof, you no longer have the same customer object everywhere in the organization.

Another department may find that they want to add a hobbies column to be able to track what interests the customer. Suddenly, you have two different customer objects, both different from what you created.

Ah, simple—you can just disable editing of the content type. By doing so, you are in fact just creating a new problem and not solving your current one; users still want to add the birth date or the hobbies to a customer, and you have now prevented their sole means of doing so.

Suddenly, the user may become your opponent as they try to work around the limitations you put there to protect the system.

Note

This is neither a SharePoint problem nor a developer problem. For example, security architects fear internal users far more than external ones; nothing is more dangerous than someone attacking your system from the inside.

Solutions vs. Features

When you want to deploy your custom code, whether it is lists, content types, pages, or .NET assemblies, you need to package your content as either a feature or a solution.

SharePoint *solutions* are similar to installation packages for normal Windows applications, while SharePoint *features* can be compared to xcopy-style applications that contain all the desired functionality in a single folder without the need for special installation procedures.

This simile is simplified, I know.

The best approach is to use a solution, also called a WSP. With a solution, you can package any number of features, including just a single one, and you can deploy content that is not tied to any feature at all. Examples of this is data that goes into the LAYOUTS folder, global assembly cache (GAC) assemblies, web services, and so on.

Solutions are also upgradeable, meaning that once you have installed the solution, you can upgrade it easily by running an STSADM or custom command. Any files already deployed will then get upgraded with the new version in your upgraded WSP package.

Great, So Solution It Is Then!

The problem is, you cannot expand an already deployed solution. Let's say you have created a solution to deploy a customer relations management application, containing lists, content types, and perhaps a few administration pages.

Then, after testing this solution for a few weeks, your users tell you they need a custom form for editing the customers. "Great," you think, "I can just add the new ASPX file to the upgraded WSP and upgrade the solution, and everyone will be happy."

But you can't. SharePoint will not provision the new file, since it only upgrades files or features that already exist.

"Ah," you continue, "but I can just uninstall the existing solution and reinstall a new one."

And you would be right in saying so. Well, you can if you are not using any installation features such as FeatureInstalled handlers or other custom code used for provisioning your solution. Most people don't, so that's not a big deal.

In fact, the only negative aspect of this uninstall-and-reinstall approach is the added complexity, but don't be deceived, because the complexity is a major issue in production systems.

The solution provisioning process has a lot of moving parts, so a lot can go wrong if you do not test thoroughly. As your solution grows, so does the number of moving parts, and therefore testing becomes more and more complex. For example, what happens when you uninstall a solution that has a workflow and current documents are already using that workflow?

Then there's the issue of solution availability. Uninstalling a solution means all the features of that solution become unavailable. If you have a local installation with users in a single location, you can uninstall and reinstall at nighttime, but if you have users all over the world, there really isn't a nighttime at all.

Note

Administrators absolutely love staying up at 3 a.m. because you need to add a new form to a solution.

Finally, how will you treat existing data when you need to add new features? For example, if you have a solution that utilizes a web part on the front page of a site, how will you go about changing that web part? Will you assume it is there and just modify it? What about user customizations? How will your upgrade affect possibly unknown changes that users have made?

So, again, the only issue with uninstalling and reinstalling is the complexity, but that's no small issue. SharePoint development projects can quickly become a large and overly complex, costing far more than you may initially imagine.

Another approach is to add new solution files. If you need a new form, simply add that form as a feature in an entirely new solution, and deploy that solution. This even allows you to update the form later without having to uninstall or reinstall anything.

In fact, why not make a single WSP solution for every feature? That way, you never need to worry about having to add new features to existing solutions, since each feature is a separate solution.

One problem with this approach is the massive amount of effort you need to put into building all your WSP solution files. However, this can be automated, so it may not be such a big issue after all.

Then there's the question of maintenance. Even if you can develop your own tools for managing the development side of the story, what about the administrators who must install and deploy the solutions? Each solution file requires a separate installation and deployment procedure, including an IIS reset to deploy. Again, this can be automated using scripts, at least if you don't care about the number of IIS resets.

Oh, but you still need to instruct all your site owners to activate the correct features in the correct order once you deploy your new WSP solutions. If they don't follow the procedure or if they're making the errors, you shouldn't be blamed, right? After all, complex systems should be hard to use and should be harder to use with increased complexity.

Yes, I'm joking.

A Better Approach

There must be a better way of developing SharePoint systems. After all, major companies around the world are betting the farm, so to speak, that SharePoint will be the next big thing.

In this issue, I will explain to you the method I use for developing SharePoint solutions. I'm not talking about WSP solutions but rather about full applications, even multiple applications, that work to solve an organizational need of some kind.

The idea behind this approach is to create an upgradeable and expandable development model that both harnesses the ease of use for end users and maintains the need for developers to control the environment.

The method is well suited for agile developments as well, since you can deploy a partial solution quickly and then expand on the solution as you go along. I've successfully deployed the initial solution for complex systems within a week or so of starting development, even though the final solution development might take months.

The method can further be adapted to work with multiple development models. In this issue, we will look at how we can use the web interface to prototype our solution and then extract information from our configured site for our final solution.

This approach is often one you may encounter in real life, as end users configure their sites how they want them and you are tasked with creating a deployable, maintainable, and scalable solution.

Figure 1 shows an overview of the process.

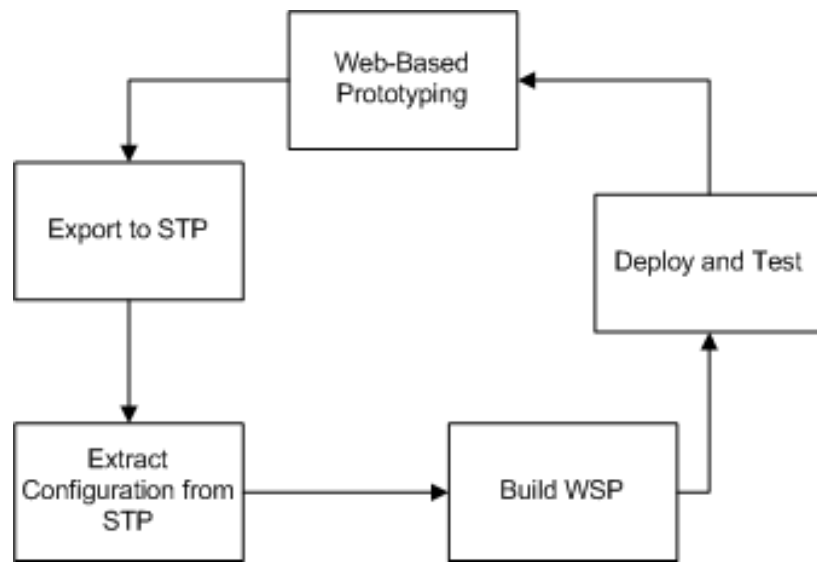


FIGURE 1. DEVELOPMENT PROCESS OVERVIEW

Let's look at the steps here, but keep in mind that this is only one of multiple approaches you may take.

Web-Based Prototyping

Web-based prototyping is where you quickly hack together a concept of how you want your solution to work. You can utilize the speed and ease of configuration to show your users how the final solution will work.

Another approach here is to let your end users design the solution themselves. This empowers end users and lets them design what they want without being limited by lack of development skill.

You may think this is completely crazy, but read on, and you'll see it makes sense.

Export to STP

Once you are satisfied with your site or list, the next step is to export it to an STP. This puts presumably everything you need to re-create the list or site into a single file that you can later download and open to see all the configuration bits.

Note

An STP file is really just a CAB file with a funny name. We'll get to that later in this issue.

At this point, you are not only thinking this is crazy, but you are certain. However, you are wrong.

Extract Configuration from STP

Note

In SharePoint 2010, sites exported from the web interface are not STP files but WSP files. The principle remains the same.

Build WSP

Now we're getting to the development part. Once we have all the moving parts from the STP, we assemble and customize the solution.

Much of the configuration will be ready to use, but there are things you need to manually build. We'll get to that later in this issue as we explore the exported STP.

This is also the stage where you as a developer will apply all your knowledge into making the solution spin. You can add new features, modify existing ones, introduce entirely new functionality, develop workflows, and create whatever your solution requires that is not easily adoptable from the exported template.

Don't go overboard, though; remember, you still have a set of requirements to meet, and anything beyond those requirements is a waste, or at least not sanctioned by the customer.

Deploy and Test

When all is done at the previous stage, you are ready to deploy and test your solution. This is not as easy as hitting Deploy and hoping for the best, because you need to work with the users. The users will now see your solution in all its glory. And they'll hate it—I can pretty much guarantee you that.

Don't be disheartened, though. As an agile developer, you know that customers will ask for changes, and your first solution's deliveries are rarely the final product. Using the method described here, working with new features and upgrading your existing ones are a breeze.

Mission Statement

In this issue, we will explore the solution development method I have mentioned. I will explain to you how you can use this method both to create entirely new solutions and to add functionality to existing solutions.

We will explore the method based on a simple scenario, which is as follows:

The customer is an organization that wants to create a project management solution using SharePoint. A project manager, Paula Monroe, has the authority to make all decisions on what to include or exclude represents the customer. The project manager will also assist you in creating the prototype site. You will be given initial requirements, but expect those requirements to change during the project.

Requirements

To fully appreciate this issue, you need to have a fairly good knowledge of SharePoint development. I will not explain lists, content types, custom pages, or anything like that.

You'll need a working lab environment, and I'm doing all my development on a machine based on the setup used in the "Beginning SharePoint Development" issue of *USPJ*. You can get the first chapter, including the setup and installation instructions, as a free preview on <http://www.beginningsharepointdevelopment.com/>. However, your regular or preferred lab setup will likely work.

The bare minimum you need is a running WSS installation and Visual Studio. I'll be using WSPBuilder for my examples. If you prefer other tools, you are daft and should upgrade, but I'm not holding that against you.

Rules of Engagement

Our rules of engagement are as follows:

- **We will use only supportable methods and not harm a single Microsoft-provided file.**
If you modify any file that ships with SharePoint, you will not be able to get support for your solution from Microsoft.
- **We will use the least amount of effort possible.**
As per the introduction, we are going to learn how to develop SharePoint solutions but not necessarily create a production solution here. I will not be covering every detail. Expect to add or improve error handling, for instance.
- **We will utilize SharePoint native techniques whenever possible.**
We might end up taking a step or two back to solve a problem. That is OK; we want to learn to utilize SharePoint.

We Will Not Cover

I want to focus on SharePoint technologies here, so I'm going to avoid certain related topics, such as unit testing, source control, and Visual Studio. I certainly encourage you to explore those topics as well. Even though the method you will learn here will speed up your development and provide powerful agile techniques, the method is only one piece of a large development project.

Let's get technical, shall we?

Working with the Site Template

The end user's solution to repeating work

So, we are going to make a project management system. I have chosen that scenario because it allows us to create a system from scratch and because it is a somewhat realistic development scenario.

In this chapter, we will take the first steps in developing our solution. We will look at how to create the prototype site and then at how to dissect that prototype into code we can use later.

This process allows end users to visualize and implement what they want in the final product. This is, in fact, the dreaded configuration stage that makes so little sense to us as developers but makes so much sense to end users.

However, as you will see, when done correctly, configuring your site saves you tons of time and perhaps even provides some goodwill to the users, who will use your solution once you are done with the development.

Creating the Prototype

Our first order of business is to create the prototype. The project manager has very clear ideas about what she wants.

Note


If you want to skip this section, I have provided the resulting STP in the source directory, named ProjectSite.stp.

The initial requirements for our solution is to have a place in which to share documents, track task progress, and maintain a list of project participants.

Yes, I know, these are simple requirements; however, we want to explore a specific method for development, not spend hours upon hours creating complex configured solutions.

In any case, let's get started.

Start by creating a new site collection in a web application. I usually start with a blank site if I want complete control. Often, users start with a Team Site or other site and modify that.

 **Creating a
Prototype Site**

The downside to starting with a Team Site is that users often end up defining what they want from what they get. In other words, if they start with a Team Site, they end up wanting a shared documents library, an announcement list, a task list, and perhaps a list of links.

From an architecture perspective, I recommend not working from a Team Site, because this tends to limit the creativity of the users. When users see shared documents, a calendar, and a task list, they tend to cite those elements as their requirements and not consider other needs.

Of course, you should consider the users' needs in each case. Sometimes, a Team Site may be the perfect starting point.

Back to our task, though—you should next create a document library titled Documents and a Tasks list called Project Tasks. These two lists fill two of the three required features from the project manager.

For the final requirement, we need to add a way to track project participants. The simplest way to do this is to add a Site Users web part to the front page, as shown in Figure 2. This also gives us an opportunity to test adding web parts to a page.

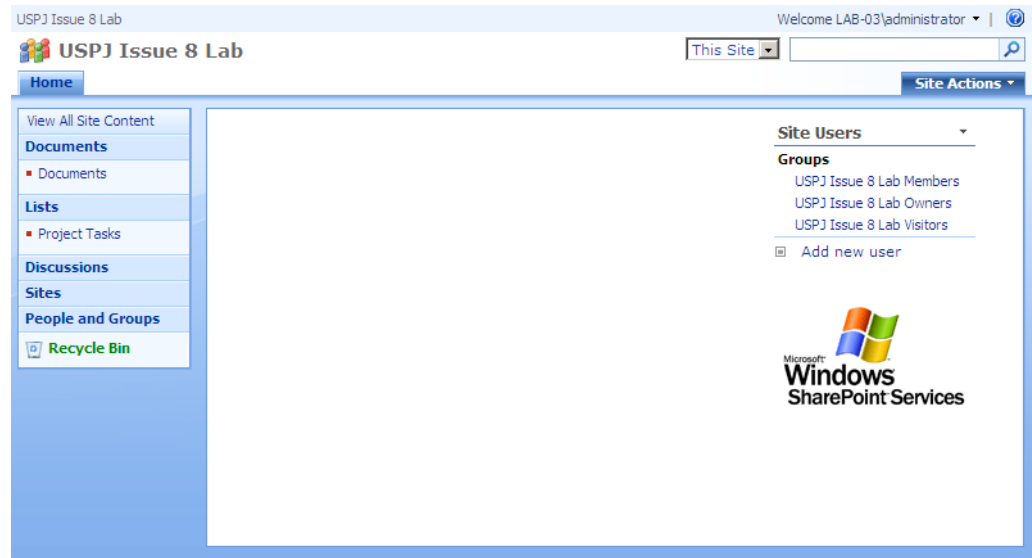


FIGURE 2. SITE USERS WEB PART ADDED

When you show this site to Paula Monroe, however, she wants some changes. First, the SharePoint logo must go, and second, she wants to have some text at the top of the page with information from the project manager of the current project. Finally, she wants a list of recent documents and pending tasks shown on the front page.

Changes at this stage are easy to implement. We still haven't written any code; we are still only playing around with the configuration method. We harness the ease and speed that the configuration method affords us and can quickly accommodate requests for changes.

To accommodate this request, delete the Image web part, add a Content Editor web part to the left web part zone, and then add the Documents and Project Tasks list web parts below the content editor.

Then, ensure that the Project Tasks web part displays only items that have a status set to something other than Completed. You can check this by going to the web part menu from the Edit Page mode and then editing the current view.

Check the Filters section, and ensure that it displays only items where Status is not equal to Completed, as shown in Figure 3.