ABSTRACT

A SOFTWARE DEVELOPMENT DOCUMENTATION INTERNSHIP
WITH BLUESPRING SOFTWARE, INC.

by Nicholas Waleszonia

This report discusses my experience as a software development documentation intern at Bluespring Software, Inc. from January 2006 to April 2006. Bluespring provides BPM Suite, a business-process management software product that allows companies to manage their business processes. Chapter 1 describes BPM suite as well as how Bluespring's history, culture, and organizational structure influenced my internship experience.

I worked on short and long-term projects at Bluespring. My short-term projects included composing online help for BPM Suite and researching Bluespring's competitors. My long-term projects included planning and writing a “Quick Start User Guide” for BPM Suite. I discuss these projects in Chapter 2.

The majority of my internship work was dedicated to creating the functional requirements for new software components of BPM Suite’s desktop software. In Chapter 3, I describe the entire functional requirements process, and in Chapter 4 I analyze each process step from the perspective of an academic theory.
A SOFTWARE DEVELOPMENT DOCUMENTATION INTERNSHIP
WITH BLUESPRING SOFTWARE, INC.

An Internship Report

Submitted to the
Faculty of Miami University
in partial fulfillment of
the requirements for the degree of
Master of Technical and Scientific Communication

Department of English

by

Nicholas Waleszonia

Miami University

Oxford, Ohio

2011

Advisor ______________________
Dr. Jean A. Lutz

Reader _______________________
Dr. Michele Simmons

Reader _______________________
Dr. James Kiper
Table of Contents

Chapter 1: Introduction ................................................................................... 1

Introduction to Bluespring Software ................................................................................................................................. 1
Bluespring’s History .................................................................................................................................................... 1
Bluespring’s Organizational Structure at the Time of My Internship ........................................................................... 3
Bluespring’s Culture ............................................................................................................................. 4
Bluespring’s Product ................................................................................................................................................ 5
  BPM Designer ...................................................................................................................................................... 6
  BPM Activities .................................................................................................................................................. 8
  Activity Configuration .................................................................................................................................... 9
Technical Communication at Bluespring .......................................................................................................................... 10
Conclusion .................................................................................................................................................. 11

Chapter 2: Overview of My Internship Projects ............................................... 12

Miscellaneous Tasks .................................................................................................................................................. 13
Internal Trial-License Renewal Process .......................................................................................................................... 13
BPM Designer’s Help Documentation .......................................................................................................................... 16
Quick Start User Guide ................................................................................................................................................ 17
Functional Requirements Documents for New BPM Designer Activities ...................................................................... 21
Conclusion .................................................................................................................................................. 22

Chapter 3: Major Internship Project – Functional Requirements for New BPM Designer Activities ........................................................... 23

Bluespring’s Functional Requirements Document Template .......................................................................................... 24
Summary of the Four Process Steps for Functional Requirements Development ........................................................................ 26
  Defining the Logic of an Activity ......................................................................................................................... 26
  Creating a Mock-Up Example of the User Interface for a New BPM Designer Activity ...................................... 27
  Writing the Functional Requirements for the New BPM Designer Activity ......................................................... 29
  Revising UI and Requirements Draft According to Mentor and Developer Feedback ......................................... 30
Discussion of the Functional Requirements Development Process for Two BPM Designer Activities .................................................................................................................. 30
  Composing Functional Requirements for the Put File Activity ........................................................................ 31
  Composing Functional Requirements for the XSLT Activity .............................................................................. 35
Conclusion .................................................................................................................................................. 40

Chapter 4: Analysis of Bluespring’s Organization and the Functional Requirements Process ................................................................................................................. 41

Bluespring’s Organization Compared to the “Organizations as Organisms” Metaphor ......................................................... 41
Adapting to the Customer Environment .......................................................................................................................... 41
Promoting Internal Organizational Health .......................................................................................................................... 42
List of Tables

Table 1: BPM Designer Activity Descriptions ................................................................. 8
Table 2: Process Requirements Outline ........................................................................ 15
Table 3: Quick Start User Guide Walkthrough Outline .............................................. 18
Table 4: Functional Requirements Outline ................................................................. 26
Table 5: Common Visual Basic UI Components .......................................................... 29
Table 6: Examples of Requirements that Provide Processing Instructions .............. 35
List of Figures

Figure 1: Bluespring Organizational Structure ................................................................. 4
Figure 2: BPM Designer Canvas ......................................................................................... 7
Figure 3: Activity Property Configuration Window for Excel Reader Activity .................. 10
Figure 4: Division of Time During Internship ................................................................. 13
Figure 5: Example of BPM Designer Online Help .......................................................... 16
Figure 6: Quick Start User Guide Call-Out Visual Design ............................................... 19
Figure 7: Quick Start User Guide Highlight Visual Design ............................................... 20
Figure 8: Division of Time for Major Project ................................................................. 24
Figure 9: Level of Detail Required in Functional Requirements .................................... 30
Figure 10: First-draft UI design for the Put File Activity ............................................... 32
Figure 11: Put File Activity UI design Before and After Feedback .................................. 33
Figure 12: Example of User Definition Requirement .................................................... 34
Figure 13: XSLT Input Step – Before and After Feedback .............................................. 36
Figure 14: XSLT Style Sheet Input – Before and After Developer Feedback .................... 37
Figure 15: Functional Requirements Describing an XSLT Transformation .................... 39
Figure 16: Example of Detailed Requirements Describing One User Action .................. 39
Figure 17: Requirements Describing Test Transformation Functionality ....................... 40
Chapter 1: Introduction

In this chapter, I discuss my experience as a documentation intern at Bluespring Software, Inc, located in Cincinnati, Ohio. The duration of my internship was 15 weeks, starting January 09, 2006 and lasting until April 21 of the same year. I served my internship during an “upswing” in Bluespring's history. The company was moving full steam ahead, enhancing its product as well as expanding its market share. What follows is a discussion of how this fast-paced, development-centric environment contributed to my internship experience. I provide an overview of Bluespring's history, organization, culture, and product, and the role of technical communication within the company.

Introduction to Bluespring Software

Bluespring Software, Inc. (Bluespring) is a small software company located in Cincinnati, Ohio that produces and sells BPM Suite®, an application that allows process owners to design, implement, and manage business processes. Business processes include simple examples like managing the approval of an expense report after worker submission and through managerial review, to complex examples like creating a new customer account at a cable company, which requires the management and collaboration of many different groups, from sales and customer service personnel who create the account, to the technicians who perform the installation. It has developed a base-product that has a clearly-defined purpose, and this clarity has allowed it to set an ambitious goal, namely that it considers any organization that has business process management needs, no matter how big or small, as a potential client. To increase its customer base, the day-to-day goals at Bluespring are to assess customer needs and to develop functionality within the BPM Suite software that meets those needs.

Bluespring’s History

When I started my internship in January 2006, the environment at Bluespring was positive and upbeat, and the company was investing heavily in the future, establishing partnerships with other companies and spending significant time and money in building and enhancing its product. This environment of company growth shaped my internship work. My primary responsibility was to help build new features in the product. However, the positive environment I experienced was in stark contrast to Bluespring's tumultuous history. Since being created in 1997, Bluespring has weathered serious financial troubles, mass layoffs, and organizational restructuring. In order to understand the environment of innovation that shaped my internship experience, one must first understand how Bluespring transformed from a company rife with financial and organizational problems into one that cannot grow or invest fast enough.

I learned about Bluespring's troubled history by reading the Master of Technical and Scientific Communication (MTSC) reports of past Bluespring interns. For instance, according to Kathryn Rudolph's work, Bluespring started as a telecommunications software company and expanded to over 100 employees in 2000, but then the company went through an intense process of layoffs in 2001 after the telecommunications industry crash. The following quote from Rudolph’s internship report illustrates the drastic measures that the company had to take to stay alive.
Bluespring’s customer base disappeared nearly overnight. The company was forced to remove employees from their positions during three rounds of layoffs over eight months. The layoffs brought the company down to approximately 25 people and naturally affected the mood and atmosphere of Bluespring. (Rudolph, Miami University . Dept. of English, and OhioLINK Electronic Theses and Dissertations Center, 2004)

After the crash, Bluespring rebounded. Over the next four years, the technology team redesigned their product offerings, creating a collection of six products called Priority CS®. This list of products had a variety of functions. One product called Customer Management System allowed sales representatives to manipulate customer accounts. Another product, Data Transport Wizard, allowed users to transfer data between databases. And yet another, Document Designer, streamlined the creation of invoices.

The variety of these products required that the small company of 25 employees had to meet the needs of a different customer type for each product, which spread Bluespring too thin and resulted in an overall reduction in quality. The task was too big, causing Bluespring’s sales to decline once again and, as Stacey Scott recounts in her 2004 internship report, demand for more profitability from investors caused Bluespring to reposition itself as solely a Business Process Management solutions provider in 2004. The goal of the repositioning was two-fold. First, it was meant to widen the company's customer base from just the telecommunications industry, which comprised its target audience before the restructuring, to any company that could benefit from managing their business processes, regardless of the industry that the company operated within. At the same time, it allowed Bluespring to focus on a specific set of customer needs. Instead of serving sales representatives, database managers, and document designers, Bluespring was now targeting its products at only business process analysts (Scott, Miami University . Dept. of English, and OhioLINK Electronic Theses and Dissertations Center, 2004). The organizational change meant that Bluespring’s product offerings had to be altered as well, so it distilled its software down from a collection of six products into one, BPM Suite.

The reorganization of the company and product was the turning point for Bluespring. For the first time, Bluespring’s single product had a clear purpose and audience, and the company was able to rally around this clarity. Soon after, Bluespring began to hit major milestones in its resurgence. In February 2005, Cincinnati Bell, a major telecommunications provider in Ohio, Kentucky, and Indiana, purchased and began using Bluespring's product to manage their enterprise customer acquisition process. Within two month's time, Cincinnati Bell saw a 66% reduction in their process cycle time and 75% in cost reduction per deal.

Bluespring didn't stop at landing new customers. Their growth also extended to adding more value to their product by partnering with other companies. For example, in July 2005, Bluespring partnered with InRule, a company that develops a Business Rule Engine (BRE), in order to integrate InRule's BRE into BPM Suite. This partnership was designed to pique the interest of Bluespring's customer base of business design analysts, who value the ability to separate a process from the business rules that drive it. Business rules allow a process to operate according to an “if / then” logic. For example, a business rule can provide instructions to a customer service representative on how and when to reward customers that have experienced problems with the goal of retaining their business, i.e. if the customer has had 2 or more problems within a six-month period, then offer a 10% discount on services, if the customer has had 2 or more problems within a three-month period, then offer a 15% discount on services. Business rules enable a company to automate decisions for common problems or situations that
may occur during a process.

Finally, right at the beginning of my internship, Bluespring achieved Microsoft Gold Partnership status. Earning gold partnership status from Microsoft meant that the company had “demonstrated technical expertise with Microsoft technologies and established a broad industry base, ranging from Financial Services & Telecommunications providers to business process optimization (BPO) and compliance consultants (Mills, September 19 Tue, 2006).” In addition to the recognition, Microsoft assigned a sales representative to Bluespring to actively sell the BPM Suite product. This was a significant achievement for the company that energized Bluespring's employees.

It was after this long string of successes that I entered Bluespring and began my internship. I wasn't aware at the time, but the varied history of Bluespring influenced my internship in two ways. First, Bluespring had survived a “trial by fire” and it had emerged as a focused and refined company. The company's survival meant that the uncertainty of the past was replaced by an excitement for the future, and my internship started right after a strong push began to invest in the BPM Suite product. As a result, I had the opportunity to directly contribute to the improvement of the product by helping design and test several new pieces of functionality. Second, the long, arduous history of the company created a close-knit, highly-skilled staff of employees to which I had open access. For example, as I will discuss in more detail later in this report, I was able to get feedback on UI designs directly from the developers as well as have my work reviewed not only by a technical writer, but also by the marketing personnel and a “customer-focus” consultant.

Bluespring's Organizational Structure at the Time of My Internship

Bluespring's string of successes can be directly attributed to the company's organizational goal of evaluating customer needs and evolving BPM Suite so that it could meet them. Therefore, it is natural that the organizational structure of the company was built to facilitate this goal.

At the time of my internship, the leader and advocate of the product and strategy was Karl Treier, the Chief Technology Officer (CTO) at Bluespring. Bluespring had a CEO, Rob Daly, who set and managed the overall direction and strategy of the company, but did not drive its daily production priorities. Bluespring was a company that lived or died by the worth of its technology, so Karl was the leader that was directly involved with developing and maintaining all the technology that went into the product. He was ideal for his role because he had many years of software engineering experience and he was in constant contact with customers, providing ample opportunity for him to assess their needs and to identify requirements for new product functionality. So, although the CEO, Rob Daly, was the official leader, the CTO, Karl Treier, was the one who defined and pushed daily, weekly, or monthly objectives. This emphasis on CTO involvement and supervision is evident in Bluespring's organizational structure shown in Figure 1. Notice that the majority of the company answers directly to Karl, as did I in my role as a documentation intern.
Figure 1: Bluespring Organizational Structure - Bluespring Software's organizational structure places heavy emphasis on leadership from the Chief Technology Officer.

My documentation internship position existed between the product management and documentation teams. My direct supervisor and mentor was a product manager, Chris Schapman. Chris is a graduate of the Bachelor of Arts in Technical and Scientific Communication (BATSC) at Miami University. I also worked closely with Jeff Stevenson, Bluespring's resident technical communicator, and Amanda Pickett, Bluespring's training analyst.

**Bluespring's Culture**

At the time of my internship, Bluespring's offices were on the sixth floor of a building on Fourth Street in downtown Cincinnati. The office consisted of modern cubicle groupings that had short translucent walls. In other words, each cubicle did not provide much privacy, and it was common to hear employees talking over or through the walls. The offices for leadership were behind but still close to all the cubicles. A 15-foot walk was all it took for most employees to talk to the CEO or CTO. The layout was professional yet stylish, and it promoted an atmosphere of high-energy and collaboration.

Generally, the environment at Bluespring was relaxed and informal. It was not uncommon to find a pint of beer waiting on my desk after a late-Friday afternoon meeting. Furthermore, employees were encouraged to take breaks throughout the day to help relieve stress, and the company had a foosball table to occupy employees during these decompression periods.

The corporate culture at Bluespring was results-orientated, and there was a heavy emphasis on personal responsibility. Managers were hands-on with the work being done, and had little time to closely watch what employees were doing. Most of the 28 employees at Bluespring
had been with the company for several years and have weathered the company’s financial storms, allowing the leadership to build a solid foundation of familiarity and trust in their workforce.

There was less role specialization at Bluespring than would normally be found at a larger company. Everyone still had a role: developers still wrote software code, documentation specialists still produced documentation. However, the small size of the company had fostered a culture of collaboration where a good idea may come from any department or any person. Leadership expected their employees to contribute without being asked. Also, Bluespring's small size ensured there were no workplace “silos” where one group is isolated from others. One example of this lack of strict specialization was the functional requirements documentation for software enhancements. Product management usually wrote these documents, but software development also occasionally created them. Furthermore, the trainer, technical communicator, and marketing staff were encouraged to read the functional requirements and offer ideas for a tweaked design or additional software enhancement.

The purpose of Bluespring’s informal, results-orientated, and flexible culture was to make it more agile in response to change, much like a successful organism adapts to a changing environment in order to survive. In Chapter 4, I discuss how Bluespring’s culture mimicked a living organism from the perspective of the “Organizations as Organisms” metaphor detailed by Gareth Morgan in his book *Images of Organization*.

The emphasis on collaboration and role sharing influenced my internship. For example, in addition to performing my standard documentation duties, I also helped quality assurance (QA) test new releases of the software so that the company could meet a release deadline. The spirit of collaboration works both ways. When I completed a draft of my “Quick Start User Guide”, I not only asked for feedback from my supervisor and the technical communicator, I was also able to get the marketing department and a “customer-focus” consultant to review it. This feedback helped me reevaluate the purpose and scope of my user guide because it better defined its audience, i.e. process designers that probably do not have the skills necessary to use all of BPM Suite’s functions to their full potential. As a result, I learned that the user guide provided too much information for a beginning user and, therefore, Bluespring could not use it as intended. However, the act of writing the guide did have a positive outcome. It clearly defined just how intimidating BPM Suite can be for a new user, a problem that inspired a new project: a business-process database that contains a list of common processes already pre-configured that customers can use to get started building processes without needing a 70 page step-by-step guide. This project started after my internship ended, so I did not work on it. I do discuss the Quick Start User Guide in more detail in Chapter 2.

**Bluespring’s Product**

Bluespring’s software product, BPM Suite, is a Business Process Management application that allows process owners to create and manage business processes from their desktop. BPM Suite consists of three components: BPM Engine, BPM Web, and BPM Designer. BPM Engine is the server-side component of BPM Suite; it performs all of the technical back-end computations and communications that occur within an automated process via a Service-Orientated Architecture (SOA). BPM Web is a web-based interface that allows a company’s employees to contribute to a process when their input is needed. BPM Designer is the desktop-based software application that allows business process analysts to design, build, execute, and
monitor automated business processes. The following lists describe the function of each component in more detail.

**BPM Engine - Foundational software of BPM Suite**
- Runs on a server
- Generates and communicates process information
- Executes tasks and invokes applications through a service-orientated architecture (SOA), which is a collection of computer functions that communicate with each other. The communication can involve passing information from one service to another or it can require a coordination of processing between services. An activity in BPM Suite is a combination of a User Interface (UI) and a service. Each activity within BPM Suite has a service component that executes the processing of the activity. A user configures an activity by providing input into the user interface, and the service performs the activity processing by following the input instructions.
- Manages information transfers between processes and activities. In BPM Suite, examples of activities are auto-generating a Word document, sending an email, or reading and writing to a database. BPM Engine operates as a SOA because, for instance, it can read a customer’s name and email from a database in one service, then auto-generate and send an email to that customer using the gathered information in the next service in the process.

**BPM Designer - Desktop program**
- Provides a canvas on which users can design and configure processes
- Allows users to view and manage active processes
- Allows users to construct the process environment

**BPM Web – Website accessed via a browser**
- Provides a Web-based portal for workers to complete assigned tasks
- Shows each user the tasks assigned to him or her
- Provides reports for analyzing process metrics

Although BPM Suite consists of three separate components, the interaction of these parts is essential for the function of the product. Using a music analogy, BPM Suite functions similarly to a symphony. Think of BPM Engine as akin to the conductor; it manages the interaction of the various parts and allocates resources. BPM Designer is the composer; it carefully designs, monitors, and tweaks processes. Finally, BPM Web is the symphony of musicians; it is the medium through which all workers provide input into a process. The majority of my work during my internship was focused on BPM Designer, which I discuss below. For more detailed descriptions of BPM Engine and BPM Web, refer to Appendix A.

**BPM Designer**

BPM Designer is a desktop application in which users plan, design, execute, and manage
processes. There are two menus that make up BPM Designer: the Configuration Management menu and the Process Management menu. The purpose of the Configuration Management menu is to establish the process environment. For instance, before a designer creates a process, he or she must first define roles for the users that will participate. If it is a sales process, then the designer will create a role for sales personnel; if it is a customer service process, then the designer will create a role for a customer support specialist. Roles indicate the type of input that each participating person will provide. Also, each person in a process can have more than one assigned role, and, therefore, can provide different types of input within the same process, but not within the same process step. Additional variables that BPM Designer defines in the Configuration Management menu include establishing connections to databases, geographies, and time zones (which help manage processes that require input from workers that are physically far apart).

After a designer establishes the foundation of the process environment, he or she builds the process using the Process Management menu, which is organized to appeal to process designers who are often comfortable working in Microsoft Visio: it offers a visual, diagram-based interface.

As seen in Figure 2, the Process Management menu consists of the Activity Toolbox in the left panel and the large white canvas that occupies the majority of the window’s right side. Designers build a process by dragging and dropping activity icons from the toolbox on the left to the canvas on the right. When on the canvas, a process designer can arrange the activities from start to finish, connect them, and configure them to make a functional, fully-automated process.

![Figure 2: BPM Designer Canvas - BPM Designer desktop tool; shows process design “canvas” and the groupings of available activities in the left pane.](image-url)
BPM Activities

The key pieces of functionality in BPM Designer are the activities in the toolbox. They are pre-made web-service programs that, when properly configured by a process designer, can complete a variety of common business tasks. BPM Designer organizes the activities into seven groups, which are detailed in Table 1.

<table>
<thead>
<tr>
<th>Activity Toolbox Group</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automated</td>
<td>Allow manipulation of data that originated from within the BPM Suite environment.</td>
<td>Retrieve information from a user's profile; change the value in a stored variable that is being passed between steps in a process.</td>
</tr>
<tr>
<td>Billing</td>
<td>Automate customer billing.</td>
<td>Auto-generate a bill and deliver it to a customer.</td>
</tr>
<tr>
<td>Database</td>
<td>Read and write information to a database.</td>
<td>Update a customer's address by writing to a database cell.</td>
</tr>
<tr>
<td>Desktop</td>
<td>Manipulate Microsoft office documents and files from other common desktop applications.</td>
<td>Write to specific cells in an Excel spreadsheet; add a customer's name to a letter template in Word; send an auto-generated email or instant message to a worker or customer.</td>
</tr>
<tr>
<td>Manual</td>
<td>Allow workers to provide input at steps that require human contributions.</td>
<td>Task a manager with reviewing an expense report; task a customer service representative with responding to a customer complaint.</td>
</tr>
<tr>
<td>Web</td>
<td>Invoke web-services on the Internet.</td>
<td>Send the code for a medical treatment to a “Medicare Reimbursement” web-service and receive the proper charge for that treatment.</td>
</tr>
<tr>
<td>XML</td>
<td>Read or write to XML documents.</td>
<td>N/A.</td>
</tr>
</tbody>
</table>

Table 1: BPM Designer Activity Descriptions - A process designer uses activities to build steps in a process. The type of process determines the type of activities that he or she must use.
**Activity Configuration**

BPM activities are the components of BPM Designer that allow a user to create a functional process via a “drag-and-drop” interface. Once a designer places an activity icon on the canvas and incorporates it into the process, he or she must configure the activity using the activity configuration window (opened by double-clicking the activity's icon). The activity configuration window for each activity is unique, and it allows the process designer to configure the activity to perform a specific task. The purpose of the configuration windows is to allow the designer to create process steps that perform complicated tasks with minimal effort. For example, refer below to **Figure 3**: Activity Property Configuration Window for Excel Reader Activity.

It is important to note that “out of the box” each activity in BPM Designer has a set function that a person must configure in order for the activity to perform a task within a process. By default, each function is generic, non-specific, and not configured for any particular process situation or task. Through configuration, a process designer places the activity at a specific point in the process, identifies input data that will pass into the activity, identifies output data that the activity will produce, and sets error-handling or other auxiliary functions. As a result, the activity’s function becomes configured to perform a task unique to the context of the process, with its own data variables and/or user participants. In summary, an activity by itself cannot perform a task, it must be configured first to function within the context of any given process.

**Figure 3** shows how a process designer can configure the Excel Reader activity to read values from an expense report Excel spreadsheet. First, the process designer identifies the input Excel spreadsheet in the “Input File Data Item” drop-down list. Data items are virtual containers that BPM Suite uses to transfer information through each step in a process. A data item can contain a complex file like a Word or Excel document, or it can contain a small piece of information like an employee's email address. Second, the process designer uses the table to identify the cells that the activity will read and the “Data Item” that will store the information contained in those cells so that the data can be transferred to and used in the next steps of the process.
As I discuss in detail in Chapter 2 and Chapter 3, my major project during my internship was to design the user interface and functionality of the activity configuration window for new BPM Activities.

**Technical Communication at Bluespring**

Following the resurgence of the company, technical communication at Bluespring received increased importance, which can be attributed to several changes in the year before I started my internship. For example, the CTO, Karl Treier, valued good help documentation and considers it an important part of the user's experience. As a result, BPM Suite's help was completely re-written in 2005 so that it is task-based. In other words, the help was put in the context of how customers would use BPM suite to complete specific tasks. Before the rewrite, the help only described the features of BPM Suite outside of any user context.

Additionally, Bluespring hired a string of four employees and interns from the Bachelor’s and Master’s of Technical and Scientific Communication programs at Miami University. The dedication to creating audience-centered documentation shared by these Miami students aligned
well with Bluespring's goal of tailoring the BPM Suite product to customer needs, thus strengthening the role and raising the profile of technical communication throughout the company.

By the time I arrived, the role of technical communication at Bluespring expanded beyond just help and training documentation. For instance, the technical communicator became responsible for maintaining all the text and updating the visual design of the company's website. Furthermore, technical communication proved useful in documenting and prescribing solutions for defects found during the quality assurance process. The marketing team even learned to appreciate the technical writing skill-set, since that team often taps the editing skills of the writers, asking them to review and provide feedback on marketing materials before they are sent out to customers. Beyond editing, marketing also asked the technical writers to research Bluespring's biggest competitors and produce white papers summarizing the competing products.

All these value-added contributions that technical communicators provided at Bluespring culminated in the biggest change to documentation at the company. In an effort to provide developers with more time to focus only on software coding, the CTO tasked those with technical writing backgrounds with helping design new features for BPM Suite by composing functional specifications. Specifically, the writers composed functional specifications that detailed the design of the User Interface (UI) and functionality of new BPM Designer activities. This task required that the writers have a general understanding of Visual Basic (a Microsoft programming language used to code BPM Suite), as well as an understanding of how best to design an activity so that a user could complete a specific task. After designing the UI of the activity, the writer had to describe the proper output for every possible user input entered into the new activity. The programmers used these specifications to create code that produced the inputs and outputs described in the document. In essence, the functional specifications acted as “blue prints” for new BPM Designer activity. My major project as an intern was designing and writing functional specifications for five new BPM Designer activities, two of which I discuss in Chapter 3.

**Conclusion**

The point in Bluespring's history during which I started my internship was one of expansion and growth for the company's culture, product, and technical communicator’s contributions to both, three positives that provided opportunities for my internship experience. I was able to learn to collaborate with software developers, members of the marketing team, and quality assurance specialists. I helped design product enhancements, ensuring that they met the needs of a specific audience, and I had the backing of leadership that viewed technical communication as an integral contributor to the product's value. All three of these factors shaped my internship into a positive experience that required me to apply the technical communication theories that I had learned in my MTSC classes to real-world work situations.
Chapter 2: Overview of My Internship Projects

The projects I worked on during my internship at Bluespring software taught me that the technical communication discipline is flexible and versatile and that it allows me to be a valuable contributor to different parts of an organization. During my 15 weeks at the company, I produced traditional documentation, such as help topics and a “quick start” user guide, but I also wrote documents that were integral to the design and creation of the BPM Suite product, such as software design requirements documents and defect reports. The opportunity to work on a variety of projects, and with many different colleagues, was facilitated by the belief of Bluespring’s leadership that the company should have no “silos” and therefore all workers should contribute to areas outside of their expertise. This environment of collaboration meant that it was not only accepted, but expected, that I get feedback on my writing from programmers and marketing personnel in addition to the normal reviews performed by my fellow technical writers. As a result, I was able to learn how to use my communication skills to add value to many different areas of the company as well as how to adapt my communication style to different groups.

Documentation still is the foundation of almost all work at Bluespring. As such, the largest chunk of my internship (i.e. 40%) was spent on writing requirements documents for new BPM Designer functionality. These documents are the first step to enhancing and expanding the capabilities of BPM Designer because all tasks required to build the product depended on them. The remainder of my work included 25% on writing a “Quick Start User Guide”, 10% on updating BPM Designer’s Help, another 10% on planning and designing an internal license renewal process in BPM designer and the remaining 15% was allocated to miscellaneous tasks (see Figure 4). In this chapter, I discuss four of these five work areas.
The majority of my time was spent on writing requirements documents and the “Quick Start User Guide;” the remainder was spent on tasks that were both beneficial and challenging.

**Miscellaneous Tasks**

Many of my assignments were long-term and, therefore, took days or weeks to complete. However, in the course of my daily routine, it was common for my mentor to assign me tasks that were high-priority but only required 2-5 hours of my time. These small projects were beneficial because they taught me new skills in areas where I had little to no experience, and they allowed me to act as a “writing mentor” of sorts because I edited the work of people who did not have communication training. Other small projects required that I communicate with development to report defects. These interactions gave me experience working with developers and also taught me how Bluespring’s software products worked. Some examples of miscellaneous projects I contributed to include quality assurance testing, defect reporting, review of customer communications, research on Bluespring’s competitors, and the development of diagrams that helped a high-revenue customer with their process designs. A detailed summary of these small projects, with examples of the documentation I produced to complete them, is included in Appendix B. For the rest of this chapter, I discuss projects that required a more significant amount of my time and effort, including my work on designing an internal trial-license renewal process, writing BPM Designer’s Online Help, writing a quick-start user guide, and my major project, writing requirements for new BPM designer activities.

**Internal Trial–License Renewal Process**

As part of an initiative to better understand the customer experience, Bluespring started to
use BPM Suite to design, automate, and manage its own internal processes. The idea was that the best way for Bluespring's employees to understand BPM Suite was to get them to use the product as intended. Therefore, the first assignment of my internship was to research, document, and implement a “Trial License Extension Process” in BPM Suite. All prospective customers are given a trial version of BPM Suite, which they then use to automate 1 or 2 of their processes, effectively testing whether or not BPM Suite provides enough value to purchase. Sometimes, the 3-month trial period is not enough time for the prospective customer to assess whether BPM Suite provides enough benefit, so they ask for an extension of their trial license. I was tasked with automating this license-extension process using BPM Suite.

This was only the second internal automated process to be implemented at Bluespring, so the method to do so was not fully developed. In fact, I helped create the requirements document for designing and implementing internal processes at the company. Below, I detail the steps I took to build the Trial License Extension Process and how I structured the requirements document to capture those steps.

The first phase to building the Trial License Extension Process was to research all the steps and variables involved. I learned this information by interviewing the people that had contributing roles. By interviewing the Human Resources (HR) manager, I learned that she is where the process began; customers called her and requested a trial-license extension. She then recorded information about the request, such as the company's name, the reason for the request, and the extension time period, and then she forwarded the request to the sales representative responsible for the customer's account. The sales representative then either approved or denied the extension request, after which the HR manager notified the customer about the decision. If the extension was approved, the HR manager sent the customer a license-extension XML file, (i.e. a system file that BPM Suite uses to authenticate the extension).

The second phase of building the Trial License Extension Process was to automate as many steps as possible via BPM Suite. Workers were instructed to only provide information when no other information gathering option was available. Therefore, I designed the process in BPM Designer to automate all communication and information transfers. My final automated process in BPM Suite was structured as follows

1. The Human Resources manager fills out and submits a web form that captures required data such as
   i. Company name
   ii. Reason for the trial-license extension request
   iii. Extension time period
2. BPM Suite reads from a customer database and identifies the responsible sales representative.
3. BPM Suite sends an auto-generated email notifying the sales representative of the request.
4. The process provides a field for the sales representative to either deny or approve the request and to provide a reason for his/her decision.
5. Based on the sales representative's response, BPM Suite automatically
   i. Generates a license-extension XML file
ii. Sends an email to the customer delivering the denial or approval and – if applicable – the XML license file

iii. Writes to an extension license database (for record-keeping purposes)

iv. Generates an Excel report for Human Resources that details the history of the customer's account

To capture all the above information required to build the process, I created six sections in the Process Requirements document that are detailed in Table 2.

<table>
<thead>
<tr>
<th>Process Requirements Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required Process Steps</td>
<td>Outlines all process steps, including branching and sub-processes. An example for a branching process step is the approval / denial of the trial license extension. An approval triggers a different set of steps than a denial. A sub-process is any series of steps that can happen while the main process continues on.</td>
</tr>
<tr>
<td>Participant List</td>
<td>Lists all workers that will participate in the process and a description of their roles or responsibilities.</td>
</tr>
<tr>
<td>Data Item List for Entire Process</td>
<td>Lists all information that will transfer between process steps. An example data item can be as small as the company name or the denial/approval of a sales representative, or as large as the license-renewal XML file.</td>
</tr>
<tr>
<td>Data Item List for Each Activity</td>
<td>Same list as above, but this time grouped according to the data items required in each step of the process.</td>
</tr>
<tr>
<td>Process Diagram</td>
<td>Visual represents the process (including all sub-processes) in the form of a screenshot copied from the canvas in BPM Designer.</td>
</tr>
<tr>
<td>Use Cases</td>
<td>Describes how the process would execute in a real-world scenario.</td>
</tr>
</tbody>
</table>

Table 2: Process Requirements Outline

I learned several important things from designing, documenting, and building the Trial License Extension process in BPM Suite. First, I learned the requirements documentation process at Bluespring. The basic requirements document structure I used came from the software requirements template. Using this document taught me how Bluespring structures and produces its internal technical documentation. Second, by using BPM Suite to design a process, I gained a hands-on understanding of Bluespring's product. While designing the process I used almost every part of BPM Designer, so I became familiar with all the different UI interfaces and their functions, including how to configure a BPM Designer activity to perform a specific task. This experience, and the resulting product knowledge, helped me to create accurate User Interface designs and meaningful functional requirements.
BPM Designer's Help Documentation

Each new version release of BPM Suite includes defect fixes or new functionality most often in the form of new BPM Designer activities. I discuss above how each new piece of BPM Suite functionality needs to be quality assurance tested and my involvement in that testing. However, new BPM Suite functionality also requires new user documentation in the form of training and BPM Designer Help topics. Part of my responsibility as a documentation intern was to write the new BPM Designer Help topics that needed to be included in each new BPM Suite release. These help topics instructed users on how to perform specific tasks within new or enhanced parts of BPM Suite.

Bluespring developed its help with a software documentation tool called Robohelp. Robohelp is a user-documentation program that allows writers to produce software user documentation with features like hypertext linking and pop-up windows. With these features, users can easily jump to other topics that relate to the current document that they are reading. Figure 5 shows a BPM Designer Help topic. The underlined hypertext is links to other relevant topics.

Figure 5: Example of BPM Designer Online Help – Pictured is a help topic for “Reading Excel Values from an Excel Document in a Network Location”

Through writing new help topics, I practiced audience-centered writing. Whenever a new release of BPM Suite was nearing completion, my mentor would assign me a list of new features that needed documentation. I then opened up the requirements documents for that new functionality, and I outlined all the features that needed new help topics. My goal was to understand the new functionality from the user's perspective and to write documentation that would help users perform specific tasks. In other words, I would not just describe all the new features in the software update. I would tailor the documentation so that it would help the user reach a specific goal or perform a specific function.

One challenge was that the help topics could not be so specific that they addressed a unique customer situation; the help had to be applicable for all customers. So, I had to strike a balance between writing instructions to perform a specific task and keeping the task generic enough so that it applied generally to all customers. For example, the screenshot in Figure 5 shows a help topic I wrote for the Excel Reader activity. The purpose of this activity is to allow a BPM Suite process to read values from an Excel spreadsheet. An update to the Excel Reader activity allowed the input Excel spreadsheet to come from multiple locations. Therefore, I wrote help topics that instructed a user on how to use the Excel Reader activity to read data from an Excel spreadsheet from a variety of input sources. The topic in Figure 5 details how to read data
from an Excel spreadsheet in a network location.

In addition to practicing audience-centered, task-based writing, developing BPM Designer Help also gave me a medium to practice and improve the conciseness and clarity of my writing. To keep my writing concise and clear, I used strategies like keeping my language simple, starting each step with a verb and using bulleted lists and pop-ups to illustrate auxiliary information. I limited the length of my help documentation to no more than 10 steps. If I encountered a topic that I could not describe in 10 steps or less, I would split it up into multiple topics. In all, I produced thirty-four (34) help files explaining task-based functions for four activities. Writing help files forced me to think about and understand, at a granular level, how activities in BPM Suite worked. This understanding prepared me to compose functional requirements for new BPM Designer activities.

**Quick Start User Guide**

The project that took up the second-largest portion of my time was the development of a BPM Suite Quick Start User Guide. The purpose of the Quick Start User Guide was to help new users become comfortable with the software more quickly. I started work on this guide in response to an analysis done by a Customer Experience Consultant with whom Bluespring began working shortly before my internship began. The consultant found that BPM Suite required a steep learning curve; it took a user 3-6 months of using BPM Designer before becoming proficient at designing processes. This long learning period was due to the complicated nature of processes in a large corporation. A typical process could contain upwards of twenty individual steps, sometimes more, and some processes are multi-layered, meaning one step in a parent process could trigger many steps in a sub-process. Process designers had to learn to build fully-functional processes that were often layered and intricate, and it took time for them to learn how to use all the tools available in BPM Suite to create these processes. On top of uncovering the large learning curve, the consultant also created a document called a “Skill Set Package,” a document that lists all the skills necessary to design, configure, test, debug, implement, maintain, analyze, improve, and optimize business processes via BPM Suite. In total, there are 36 skills required for BPM Suite proficiency. The entire list – found in Appendix C - shows that BPM Suite users must be knowledgeable in both business process design and in other technical skill sets like XML and web form development. Because the Skill Set Package provided a detailed list of what users needed to know to use BPM Suite, and because I did not have direct interaction with Bluespring customers, I designed the Quick Start User Guide so that it helped cultivate the skills found in the “Skill Set Package.”

The structure for the Quick Start User Guide is an example of how I attempted to address the required BPM Suite skill set. First, I wanted to give the new user a general understanding of how BPM Suite functions and to introduce them to the Bluespring-specific process design language used in the program. Therefore, in the first section of the guide, I described the three parts of BPM Suite: Engine, Designer, and Web, and how these three parts interact as processes execute. I also summarized the function of activities and data items within each process step. Finally, in addition to the core aspects of the program, I detailed back-end functions like reporting and metrics and the BPM Software Development Kit (SDK). Customers can use the reporting and metrics to improve a process and make it more efficient for the programming-savvy users; the SDK allows them to design and build their own BPM Suite activities.

The second and biggest section of the user guide provided a step-by-step, comprehensive
walk-through on how to configure BPM Suite to run a common process in many companies – an Expense Report Review Process. The goal of this section was to map out the steps required to build a process and to provide instructions so that the user could configure each step, down to the smallest details. To achieve the goal of guiding users through building an entire process from start to finish, I structured the second section into 5 sub-sections which are summarized below in Table 3.

<table>
<thead>
<tr>
<th>Quick Start User Guide Walk-through Section</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required Information for the Process Walk-through</td>
<td>This section instructs the user to gather information necessary to run BPM Suite, like the IP Address of the server running BPM Engine and the database server. It also ensures the user has necessary administrative information, like a user name and password.</td>
</tr>
<tr>
<td>Creating the BPM Suite Process Environment</td>
<td>This section guides the user through (1) creating users to participate in the process, (2) creating roles for the users, (3) assigning users to a role, (4) understanding how data items store information used in the process, (5) creating a list to use in the process, and (6) associating a list with a data item.</td>
</tr>
<tr>
<td>Understanding the Expense Approval Process' Configuration</td>
<td>This section is split into 4 additional sub-sections: 1. How the process gathers information 2. How the process evaluates business rules 3. How the process tasks workers 4. How the process incorporates human input</td>
</tr>
<tr>
<td>Starting the Process using Excel</td>
<td>This section outlines how to create a simple web-service application in an Excel spreadsheet that will launch the BPM Suite expense report process when a user clicks the “Submit” button.</td>
</tr>
<tr>
<td>Monitoring the Process in BPM Designer and BPM Web</td>
<td>This section details how a user can monitor the expense report process as it’s running in real time, enabling him or her to test the process and to collect metrics like how long it takes to complete each step.</td>
</tr>
</tbody>
</table>

Table 3: Quick Start User Guide Walkthrough Outline - The 5 sections of the walkthrough were meant to instruct users to build a functional process from start to finish.
Beyond the structure of the user guide, I also used graphics, arrows, and visual call-outs extensively. One good example of the Quick Start User Guide's visual design is how I indicated which part of the process each walk-through section is explaining. I started with a diagram of the entire process and then used a call-out box and magnified the steps that are the subject of each sub-section (Figure 6). These visual section “markers” served two purposes. First, they acted as dividers between sections so that a user could easily find the start of each section while flipping through pages: the graphics-heavy section dividers contrasted with the text-heavy instructions. Second, the section markers identified clearly which steps in the process each section covered, thus helping the user understand the purpose of each activity group.

Figure 6: Quick Start User Guide Call-Out Visual Design - The call-out visual design indicates the activities covered by each walk-through section.

Another visual design technique I used was to label areas of each screen shot and couple those labels with steps in the instructions. The majority of the instructions tell users how to configure the BPM Designer activities to perform a task like retrieving an email address from a user's profile, assigning an expense report approval task to a manager, or writing an approval or denial to an Excel spreadsheet cell. To configure the various BPM Designer activities to perform these tasks, the process designer must set specific values within each activity configuration window. The activity configuration windows are designed to allow users to perform as many functions as possible, and in the guide's example process the user only has to perform 1 or 2
configurations per activity. So, to cut down on user confusion, I visually coupled certain steps with specific parts of each activity configuration UI. For instance, I labeled parts of the UI screen shots with call-outs that were paired with a letter. I then inserted the letter labels into the corresponding instruction steps. Refer to Figure 7 to see an example of how I paired instruction steps with specific parts of each activity configuration UI.

![Figure 7: Quick Start User Guide Highlight Visual Design](image)

I received feedback from my mentor and from other Bluespring employees throughout the planning and composition process of the Quick Start User Guide. My mentor, Chris Schapman, focused his feedback on the accuracy and clarity of my writing. He corrected mistakes in my technical descriptions of the BPM Suite architecture, and he highlighted sections that were confusing and thus needed a rewrite. Bluespring's customer experience consultant helped me shape the scope of the guide's second section. In fact, when I completed a draft, he acted as a subject for a user test of the guide. He used my document to build a process in BPM Suite from scratch. Unfortunately, the results of the user test were not good. The customer experience consultant performed all the functions in the guide but stated afterwards that he felt like he did not retain an understanding of how to build a process. In other words, without the step-by-step instructions, he would be lost. Through the user tests, I learned that the guide provided too much information that would intimidate new users rather than speeding up the rate that a user becomes comfortable with using BPM Suite.

Ultimately, the Quick Start User Guide I developed never made it to customers because of several problems. First, because it was 63 pages, it was just too big for a “quick start” guide. It
was closer to a comprehensive user manual rather than a document that will get customers using BPM Suite quickly. Secondly, I did not request access to Bluespring customers, a big mistake on my part, which meant I wasn't able to do the one thing that Bluespring wanted a “Quick Start” guide to do, which is to instruct new users on how to build a common business process in BPM Designer, from beginning to end. Without access to customers, I had to guess what information they needed, and as a result I overwhelmed the reader with feature-based information than was unnecessary, instead of focusing on and delivering just the information necessary to perform the tasks that a new user would need and want to perform. From this experience, I learned the importance of understanding the needs of any audience, especially when composing task-based instructions. Audience understanding not only enables a writer to tailor the information to an audience's needs, it also saves the writer time and effort because he or she will be able to identify and eliminate irrelevant information that is not needed to perform and understand a task. One beneficial outcome of writing the Quick Start User Guide is that it highlighted the need for a “Common Process Database,” which would be a group of pre-built processes packaged with BPM Suite. These processes would be 90% configured and act as a template for a new company to enter their company-specific information and then operate the process within their own environment, without the need to build a process from scratch. The idea for this database with pre-built processes was in its infancy when my internship ended, so I was unable to participate in its development. An excerpt of the Quick Start User Guide is located in Appendix D.

**Functional Requirements Documents for New BPM Designer Activities**

The largest and most beneficial project I worked on during my internship was writing functional requirements documents for new BPM Designer activities. Functional requirements are important documents at Bluespring. All design work at the company, all updates and enhancements to the BPM Suite software, begin in the form of functional requirements. These documents consist of two parts: (1) a detailed description of the new functionality including a list of all user inputs and the resulting outputs, and (2) a User Interface (UI) mock-up of the new functionality produced using Visual Basic's UI design tool. The description often references the UI mock-up as it explains how the new functionality should work. For instance, some BPM Designer activities allow the user to choose between performing 2 or 3 functions. While one option is selected, the other options are suppressed or “grayed-out” (i.e. not accessible to the user). To achieve this behavior, the functional requirements must reference the UI and identify how the user selects one option (maybe via a check box) and that all other options are inaccessible once the user makes a choice. The functional requirements also identify the “default state” of the UI, meaning how the functionality should appear when a user first accesses it.

Functional requirements are used by almost all groups at Bluespring. Developers used them to determine the feasibility of the new functionality. If there are features that cannot be done within BPM Suite, they detail the updates to the platform required to make the functionality possible. Developers outline the programming code required to produce the functionality in a design requirements document. In addition to providing the foundation for development, (as discussed earlier in this report) the functional requirements were also used to develop documentation. The technical writers used them to write online help topics, and the trainer used them to develop training modules before the update was released to customers. The Quality Assurance Analyst also used the functional requirements to write test scripts for when the new
activity became part of a release candidate (i.e. a new version of BPM Designer that needs only to be quality assurance tested before Bluespring released it to customers). Finally, Bluespring's marketing group used the functional requirements to anticipate and begin advertising new features to customers.

The composition process for requirements documents is labor-intensive. I wrote five requirements documents for new BPM Designer activities. In all instances, I outlined the step-by-step functionality and described all user interactions I could identify. I also designed the UI for each activity, which was challenging because I was required to fit all the functionality into a limited UI space and organize the UI in a logical and intuitive design. After I produced a draft, my requirements documents were reviewed by both my mentor and by the lead developer. Often I tweaked my design after these review cycles, then updated the requirements and resubmitted them for review.

**Conclusion**

The variety of projects I worked on during my internship at Bluespring software required me to practice my audience-centered writing and my visual design skills. While all of these projects taught me valuable lessons, none was as challenging or as rewarding as my largest project: writing requirements documents for new BPM Designer activities. In chapter 3, I discuss in detail the composition process I used to create the functional requirements documents.
Chapter 3: Major Internship Project – Functional Requirements for New BPM Designer Activities

The success of Bluespring Software hinges on the company’s ability to understand its customer's automated business process needs and then evolve BPM Suite so that it meets them. Bluespring does this by anticipating actions that customers will want to complete in a process and building BPM Designer activities to automate those actions. Some activity examples include sending an email, creating a template Word document, writing to a SQL database, and tasking a human worker. In this chapter, I discuss my contributions to Bluespring’s product evolution process. Specifically, I detail my experience writing functional requirements and designing the User Interface (UI) for new BPM Designer activities. These activities showed me how my technical communication skills can be integral to successful software development. Not only is clear communication required to write the documentation necessary to build a piece of software, it is also a necessity in the visual design of the software's UI. The UI must communicate to the end user the purpose and function of the software using concise language and the organization of its various elements. If the UI design is not clear, the software becomes confusing at best, or unusable in the worst examples. In the written requirements, clarity is important because the developers use them as an outline for how they should code the software. If the writing is inaccurate or hard to understand, there is a risk that the developers will create errant code, and the final product will not perform as intended. All in all, writing the requirements for new BPM Designer activities gave me valuable experience in communicating to two very different audiences – software developers and end users – within the same document.

Bluespring had an established process to create activities that began after a customer need was identified and deemed important enough to require an activity. This process consists of four steps: (1) defining the activity's logic, (2) drafting the UI in Visual Basic (VB) 2005, (3) writing the requirements to explain the functionality of the UI, and (4) incorporating feedback, from both a mentor and developers, into the UI and requirements. Completing a set of functional requirements took me between two and four weeks. The length of time depended on how complicated the new activity was and my previous experience with the subject matter. If an activity needed to perform many actions, and I had little experience with its subject matter, the design would take the full four weeks. This time period was not dictated by a strict deadline. The delivery of each enhancement did need to meet a delivery date, but that date was often three to six months after an initial functionality request. So, the time allotted for me to finish my documentation was flexible, and as long as I completed it within a month, I did not risk slowing down the development process. The graph below (Figure 8) illustrates the percentage of time that each of the requirements composition steps took.
In the rest of this chapter, I detail the structure of the Bluespring requirements document template, summarize the four steps of the functional requirements process, and discuss the decisions and challenges I encountered during each step as I developed the UI and functional requirements for two BPM Designer activities: the Put File activity and the XSLT activity. I developed five functional requirements in total during my internship, but these two examples provide a comprehensive illustration of the process.

**Bluespring’s Functional Requirements Document Template**

Bluespring developed a functional requirements document template to standardize the structure and type of information provided to developers for each software design. The functional requirements template is composed of nine parts, which are detailed below in Table 4.

<table>
<thead>
<tr>
<th>Functional Requirements Section</th>
<th>Description of Section</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Header</strong></td>
<td>This section identifies the</td>
</tr>
<tr>
<td></td>
<td>• Project Name – Matches the name of the activity being developed</td>
</tr>
<tr>
<td></td>
<td>• Customer Name – Indicates internal vs. External customer, i.e. the group that requested the project. My projects were always developed for Bluespring</td>
</tr>
<tr>
<td></td>
<td>• Requirements Contact – Name of the Subject Matter Expert(s) (SME) for the project. I always identified my mentor, Chris Schapman, as the SME. For one project I also identified a developer.</td>
</tr>
<tr>
<td></td>
<td>• Document Owner – Writer of the document. I entered</td>
</tr>
<tr>
<td>Revision History</td>
<td>This section identifies the number, date, author and description of each revision.</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Project Overview and Scope</td>
<td>This section provides a high-level summary of the new software. It indicates what the end user should be able to do with the activity and how he or she should be able to do it. It also describes what the user should know in order to use the activity (i.e. a user needs to understand XML to use the XML reader activity). If necessary, this section may identify what the activity should not do.</td>
</tr>
<tr>
<td>Dependencies</td>
<td>This section details the other BPM Suite functionality required for this activity to work. For example, a dependency for the BPM Designer Sharepoint activity is that BPM Suite must be able to integrate with Microsoft Sharepoint.</td>
</tr>
<tr>
<td>Deliverables</td>
<td>This section identifies the software components and other deliverables that must be produced as a result of this project. At minimum, there are always two software deliverables: the activity User Interface used to configure the activity and the activity web-service that performs the activities function within the BPM Suite system. A possible third deliverable is documentation – both on-line help and training documentation.</td>
</tr>
<tr>
<td>Terminology</td>
<td>This section defines all the terminology for the different audiences of the requirements. For example, in the XSLT requirements, it was essential to define XSLT transformations and XML for the training, documentation, and marketing readers.</td>
</tr>
<tr>
<td>Requirements</td>
<td>This section contains a line-by-line description of the UI functionality. The goal of this section is to describe every input and output and every possible user interaction with the new activity. The requirements are presented in an outline structure, and each line indent visually connects subsections to their parent lines. Below is an excerpt from the XSLT requirements that illustrates the outline structure of the requirements.</td>
</tr>
<tr>
<td>1. <strong>Loading an XML document from a FilePointer Data Item</strong> (See Figure 1)</td>
<td></td>
</tr>
<tr>
<td>1.1. The user should leave the XML document contained in a FilePointer Data Item default radio button selected</td>
<td></td>
</tr>
<tr>
<td>1.2. The user should select a Data Item that will</td>
<td></td>
</tr>
</tbody>
</table>
contain an XML document at run-time from the drop-down list
1.2.1.1. Only FilePointer and inherited Data Items appear in the drop-down list

Parts of the requirements also reference previous parts of the outline as well as UI screen shots found in the attachments section.

<table>
<thead>
<tr>
<th>Attachments</th>
<th>This section contains a list of figures that show screen shot examples of all components of the new activity UI. Each UI screen shot is referenced by the section of the requirements that describes it.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Cases</td>
<td>This section is a narrative example of how a customer will use the final activity.</td>
</tr>
</tbody>
</table>

Table 4: **Functional Requirements Outline** - There are 9 sections to the functional requirements. The requirements and attachment sections are used most by the developers to create the new activity.

**Summary of the Four Process Steps for Functional Requirements Development**

The development of activity functional requirements starts with identifying a customer need that is currently not supported by BPM Designer. These gaps in BPM Suite’s technology are identified by leadership. For instance, the Chief Technology Officer (CTO) will meet with a customer and learn that he or she wants to be able to send and receive files using an FTP server but BPM Suite does not have an activity that allows the customer to create a process step that performs this action. At that point, if the CTO deems the activity useful to additional customers as well, he will request functional requirements for the activity by posting on Bluespring’s internal Microsoft Sharepoint site. My mentor assigned functional requirements to me from a pool of requirement requests from the CTO located on a Sharepoint message board.

**Defining the Logic of an Activity**

Once assigned a requirements document, my first step was to define the logic of the activity, meaning I listed all actions that the activity had to perform to achieve the desired function, and I organized those actions in an order that logically matched how a user would most likely perform them. For instance, the first step to configure the email activity to send an email to a worker is to identify the worker who should receive the email. The second step is to define the email's message. The third step is to identify if the email will have attachments, and so on. Defining an activity's logic is a challenging exercise that required me to look at the activity from two perspectives. First, I had to imagine the activity from the point of view of a customer. I had to define the actions that he or she would need to perform using this activity. I also had to view the activity from a developer's perspective. In other words, I defined the design elements fundamental to the performance of each activity action. The process of defining all the necessary actions was critical to an activity's development. The identified actions drove the design of the UI and the composition of the requirements as well as the development goals later on in the
process. My goal was to define steps that, when represented by visual elements, guided a user through the configuration of the activity from a logical beginning to end, meaning that identifying the logical order of steps in the activity dictates the order of the UI elements. It was important to note that the organization of an activity's elements had to be flexible enough to allow a user to complete them out of order, or at least in an order that would not match the top to bottom organization of the UI.

In addition to the steps required for an activity to perform a function, I also had to map out the logic for any optional sub-steps. These sub-steps give the user more configuration options and make the activity flexible, but do not need to be completed in order to get the activity to work. For example, most activities allow a user to type the name of output files directly in the activity. Users also have the option to name a file using the dynamic value from a data item. Using data items to create a file name, or any other value, is a common sub-step in many BPM Suite activities. Identifying logical sub-steps was challenging because they are often not obvious, and although they are not essential, they still need to be logically placed in the UI and requirement steps.

Creating a Mock-Up Example of the User Interface for a New BPM Designer Activity

After defining the logical order of the activity steps, the next phase of the requirements development was to create a User Interface mock-up. Designing example BPM Designer Activity User Interfaces posed three challenges. First, I had to design all of the UIs within a set of Bluespring activity UI design styles and parameters. Secondly, I had to work with a set of UI elements that are native to the programming code, Visual Basic. Finally, I had to use visual design principles to clearly communicate the function of the example User Interfaces with as little text as possible. I elaborate on these challenges in the following paragraphs.

By the time I arrived as an intern, BPM Designer had a set of 30+ activities. Therefore, Bluespring had already established activity UI design styles to ensure that all activities looked and functioned in a pattern that was recognizable throughout the BPM Designer program. My mentor informed me of the style rules, which, at the time of my internship, were as follows:

- All BPM Designer activity windows were 6” X 6” in space, with 5 ½” X 5 ½” of usable design space. The main UI had to match this size, although sub-windows could have been smaller. No UI window could be larger than this size.
- All activity UIs had four stock tabs: General, Error Handling, Related Activities, and Display. The fifth Activity Configuration tab was the only one where I could control the design.
- The UI should use as little text as possible. If paragraphs of text were required to explain how to use the activity, the activity was too complicated, and I had to consider splitting it into multiple activities.
- The remaining styles, like font type and size, were controlled by default Visual Basic style rules.

Given these style rules, the only design elements that I had control over were the choice of Visual Basic elements, which I used to execute the activity's purpose, and the grouping, proximity, and alignment of those elements within the UI. Even though I had a limited number of
design elements at my disposal, the visual design of the User Interfaces I developed was the centerpiece of my functional requirements.

To build the example user interfaces, I worked in the UI design portion of Visual Basic (VB). The Visual Basic development suite allows for the design of user interfaces and then the application of code underneath those user interfaces. Visual Basic has a cache of UI design elements that I used to build the activity windows. Each UI element has a specific, predetermined function. It is important to note that I wrote no code under my UIs; they were merely mock-ups used to explain the UI design. Table 5 details the function and provides an example screen shot of each VB UI element I used in my activity designs.

<table>
<thead>
<tr>
<th>Visual Basic UI Element</th>
<th>Function</th>
<th>Screen Shot Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buttons</td>
<td>To allow the user to initiate an action. By clicking a button, a user is choosing to initiate the action described by the button's label. A button can initiate any action. In the example, clicking on the button will allow the user to “Edit File Name”.</td>
<td><img src="example.png" alt="Edit File Name" /></td>
</tr>
<tr>
<td>Drop-Down Lists</td>
<td>To provide a list of choices for the user to select.</td>
<td><img src="example.png" alt="Available Data Items" /></td>
</tr>
<tr>
<td>Text Boxes</td>
<td>To allow a user to enter or view a large amount of text.</td>
<td>N/A</td>
</tr>
<tr>
<td>Radio Buttons</td>
<td>To allow a user to choose between two or more options.</td>
<td><img src="example.png" alt="radio buttons" /></td>
</tr>
<tr>
<td>Check Boxes</td>
<td>To allow a user to select a conditional option.</td>
<td><img src="example.png" alt="check boxes" /></td>
</tr>
<tr>
<td>Group Boxes</td>
<td>To visually group related User Interface elements together.</td>
<td><img src="example.png" alt="group boxes" /></td>
</tr>
<tr>
<td>Tables</td>
<td>To allow a user to enter</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Table 5: Common Visual Basic UI Components - I used the six Visual Basic UI design elements above in addition to some text descriptions to build the activity configuration UI mock-ups.

After I defined the logical order of the UI, I then used the above six UI design elements to build the interface. I didn't always use every element – the ones I did use depended on the functionality that I needed in the UI. For example, if I wanted a user to select one option from a list of many, I would use the drop-down list UI element to (a) present the list to the user and (b) allow him or her to select one option from that list. Different UI elements have different rhetorical functions, and I discuss the decision to use different elements for different reasons later on in this chapter.

Writing the Functional Requirements for the New BPM Designer Activity

After creating the initial mock-up of the UI design, the next step was to write a step-by-step requirements outline of the functionality that the UI allows. The requirements had to detail all known user input and system output of the activity and the combination of these inputs and outputs must result in the desired functionality. I wrote the requirements using an outline structure. Outline numbers (1, 1.1, 1.1.1, etc) and indents indicated parent sections and subsections. I organized the requirements to match the logical structure of the UI. In other words, the requirements described the functionality in the order that a user will most likely interact with the User Interface. I also used vocabulary that is unique to the Bluespring culture. For instance, I often referred to activities using or populating “data items.” Data items are containers that BPM Suite uses to transfer information through each step in a process. A data item can contain a complex file, such as a Word or Excel document, or it can contain a small piece of information like an employee's email address. Other unique vocabulary words described parts of the BPM Suite system with which my activity would need to interact in order to complete its function.

Because accuracy is always paramount, I wrote the requirements paying attention to the smallest details. I not only defined what was possible in each piece of functionality, I also explicitly defined what was not possible, ensuring that the programmer did not make any assumptions when developing the code. Identifying what is not possible also helped cut down on future defects because the developers built the code with a specific purpose, as opposed to making more general code that may have resulted in unintended consequences. For example, Figure 9 is an excerpt from my “Put File activity” requirements. In this excerpt, I describe functionality that allows a user to browse for a destination folder. Notice that in addition to describing the input and output of the browse folder functionality, I also indicate that the user cannot create a new folder using this window and that he or she can only select a valid folder (2.1.2.1.7), as opposed to other browse folder functionality like selecting an entire directory.
Figure 9: Level of Detail Required in Functional Requirements - A simple function like browsing for a destination folder requires eight lines of requirements in order to accurately describe what a user can and cannot do with this piece of functionality.

The requirements also heavily referenced the UI mock-up examples. Each UI window exists as an attachment in the requirements document, and I referenced them in the context of the requirements that were describing their functionality. I also constructed the figure references as HTML hypertext links, allowing the audience to quickly jump to the UI being described in each section if viewing the document electronically in Word.

Revising UI and Requirements Draft According to Mentor and Developer Feedback

After I finished a draft requirements document, I submitted my requirements for review first to my mentor and then to the CTO, lead developer, and the quality assurance analyst. My mentor provided general feedback to improve the design of the UI and the requirements text. Most of the modifications to my drafts came as a result of my mentor's review because, in addition to evaluating the design of the activity, he also critiqued my writing for clarity, conciseness, and accuracy. I discuss my mentor's feedback in more detail later on in this chapter.

The remaining reviews served the purpose of approving the document from a specific viewpoint. The lead developer reviewed the requirements and determined whether or not they could be used to write design requirements and then code the new software. The quality assurance analyst reviewed the requirements to determine whether or not they could be used to write accurate and usable test scripts for the new functionality. Finally, the CTO reviewed the requirements to confirm that the new functionality would meet the customer needs outlined in his original requirements document request.

Discussion of the Functional Requirements Development Process for Two BPM Designer Activities

I wrote a requirements document for five different BPM Designer activities. My functional requirements assignments started out simply, but as I gained experience I worked on more complicated functionality. For the remainder of this chapter, I discuss my requirements composition process for two pieces of BPM Designer functionality that are on opposite ends of a spectrum: one simple, the Put File activity, and one complicated, the XSLT activity.
Composing Functional Requirements for the Put File Activity

The Put File activity was my first functional requirements assignment. The purpose of this activity is to allow a user to create a BPM Designer process step that places or “puts” a file or files at any network-accessible location. In the following sections, I detail observations and lessons learned as I progressed through each stage of the functional requirements process for the Put File activity.

Defining the logic of the activity

I identified three actions that the activity must allow the user to perform, which are identify (1) the file to be moved, (2) to where that file should be moved, and (3) the method for how that file should be moved. A fourth action, which allows for the capture of an error message in the event that the put file fails, was suggested by my mentor. These four actions became the four sections of my Put File activity UI design shown below in Figure 10. The example of the UI is heavily marked up because it is from my first draft of the UI design. The markings are feedback from my mentor.

Designing, Drafting, and Revising the UI

To create the Put File UI, I relied heavily on the visual design principles of proximity and alignment. Each required action is wrapped in its own group box. This decision gathers related elements together and indicates their relationship using proximity. I aligned the text down the left side of the UI and the buttons and drop-down lists down the right side in order to help the user to recognize the beginning and end of each element. Finally, I ordered the elements of the UI from top to bottom in the order that the user should configure the activity, i.e. she or he had to identify the file that the activity would move before she/he identified where that file should be placed.
Although my visual design for the Put File activity clearly groups elements together, the text that I wrote to describe each element was not as clear. In Figure 10 above, feedback from my mentor highlights how my initial UI descriptions were ill-purposed or confusing. The first problem is that the initial group box at the top of the UI has a paragraph description of the entire activity, rather than just a description of the action performed by the elements in the grouping. Besides containing too much text, it also does not instruct the user on the action that she must perform. Second, the text describing the “File Destination” and “Put Method” portions is awkward in that it does not clearly identify what data the user is manipulating and how she is changing it.

The second draft for the Put File activity UI, which I developed by consulting with my mentor Chris, contains several improvements. Notably, the labels for each group more accurately describe each section. As seen in the first row of Figure 11, the label “File to put” was changed to “Date Item containing file(s) to put,” which is an improvement because the text only describes the elements near it in the group box. My initial draft had text in this group box that described the entire activity, which would be out of place and confusing to users because they are expecting instructions on how to use nearby elements. Also, “File destination” was changed to “File destination pathname and folder.” This extra detail instructed the users on where the input file was coming from and how the file destination should be identified.

The second row of Figure 11 exemplifies improvements in visual design; specifically, it shows adjustments to proximity in the “File destination pathname and folder” group to separate the destination pathname from the input functions that help the user define that pathname. I also
added more specific language to clearly describe how a user can pick a path as well as the method that the activity will use to put a file in its new location. Overall, through developing the Put File activity UI, I learned the importance of using specific language to identify the input and output variables for the user. Not using specific language such as “File destination pathname” instead of just “File destination,” would confuse the user because she would understand the action the activity will perform, but not the information she needs to provide so that the activity can perform it. The final revision in the third row of Figure 11 shows improved language that better differentiates between two options for how the Put File activity should handle a situation in which it was attempting to place files at a location that already contained files with the same name. My original language was confusing and imprecise and the new description better communicates what will happen if the user selects either option.

Figure 11: Put File Activity UI design Before and After Feedback.

Writing and Revising the Requirements

After designing the draft UI, the next step was to write requirements that describe the function of the activity. As discussed earlier in this chapter, the purpose of the requirements is to describe every intended input and output of the activity but also to define its boundaries by describing what the user cannot do while using it. In addition to these purposes, I also learned that the requirements implicitly define what type of user should use the activity. In others words, certain requirement steps describe knowledge or skills that the user should have before configuring the activity – these descriptions, therefore, define the ideal user and imply that a user without the described knowledge/skills would have difficulty using the activity.

For example, the excerpt from the Put File activity requirements in Figure 12 defines certain skills that a user needs. To give context, this chunk of text is describing the requirements...
for when a user selects the network pathname and folder to which she wants to put a file. The user can enter the destination using one of two methods: (1) she can enter a static destination by typing out the full pathname or (2) she can choose to have the process build the pathname dynamically using information contained in data items. The three requirements below describing this action make several assumptions about the user, namely that she is familiar with the Bluespring term “Date Item” and how they operate within a BPM process, that she understands network pathnames and folder structures and how to define those pathnames using slashes “\”, and that she understands both data items and pathnames well enough that she can manually create a dynamic pathname using data items as defined in requirement 2.1.1.8.

2.1.1.6. Data Items will appear with percent signs (%) around the name of the Data Item, plus a dot/period, and either DisplayValue or Value, depending on if the user clicked Insert Display Value or Insert Value, respectively.

2.1.1.7. The user has to enter the slashes (\"\" or \") when necessary

2.1.1.8. Assuming the user knows what he/she is doing, the user can manually enter the %<Data Item XML Tag>.<Value> or <DisplayValue>%

Table 6 shows three example requirements from the “Put File Activity” document that give processing instructions. I found that the processing instructions requirements are difficult to write because to develop them I had to closely analyze the logic of the activity and anticipate processing errors.

As a “lesson learned” aside, while reviewing the Put File requirements for this report, I realized that the information about the type of user required to use this activity is implicit and buried in the requirements outline, and noticing it makes me question the quality of the document. I should have described aspects of the intended user, such as the knowledge and skills that he or she would need to configure the activity, at the beginning of the requirements in a section defining the intended user, instead of just implying the necessary user attributes by including requirements steps stating technical actions that the user must perform. The information defining required user knowledge is important not just to the programmer creating the activity, but also for the trainer who will be teaching customers how to use it. As such, the information should have been pulled out in a separate section so that it could be easily identified and accessed by all interested parties.

One strategy I learned from writing the Put File Activity requirements is to provide specific processing instructions to the developers when needed. Processing instructions are more robust requirements in that they define how a specific piece of information should be processed, rather than just defining an input or output. Processing instructions ensure that the developer writes code that avoids deviation from the intended design. Table 6 shows three example requirements from the “Put File Activity” document that give processing instructions. I found that the processing instructions requirements are difficult to write because to develop them I had to closely analyze the logic of the activity and anticipate processing errors.
The activity's code should not restrict the pathname to 255 characters, but the activity should fail if the pathname exceeds this number.

The activity's code should check if the user-defined folder exists – if it doesn't, the activity should fail and log an error.

Table 6: Examples of Requirements that Provide Processing Instructions

Composing Functional Requirements for the XSLT Activity

The XSLT activity was the last requirements document that I worked on and it offered a significant challenge because I thought I first had to learn two technical subjects about which I had no knowledge: Extensible Markup Language (XML) and its transformations (XSLT). XML is a text format language that labels the structure and purpose of data so that two systems can easily exchange it. XSLT stands for XSL (eXtensible Stylesheet Language) Transformations, which are processes that change the structure of XML or convert it into an entirely different type of data using a set of defined transformation rules contained in a stylesheet. The purpose of the XSLT activity is to perform an XSL Transformation using a stylesheet and XML as input. The output is either a document or chunk of XML with an altered structure or an HTML (Hyper Text Markup Language) or a plain-text document.

Defining the logic of the activity

Before I defined the logic of the XSLT activity, I first outlined the skill set of its user. I soon understood that being an activity that manipulates XML data, the user would need the technical knowledge required to read XML and also how to transform it in order to use the activity. Talking with developers, I learned that this activity is not meant to be used by an XML novice, so I designed the UI and requirements so that the activity has very little explanatory text and instead provides tools for an experienced user to perform complicated XML functions. After I defined the audience, I listed three functions that an experienced XML user would want to perform with the XSLT activity. Those three functions are that the user would need to (1) identify the XML input to transform, (2) identify the XSLT style sheet that the activity would use to transform the input XML, and (3) define how to store the output transformed XML. In addition to these three main functions, I also identified several sub-functions. For instance, when identifying the XSLT style sheet the user needs two options. She must be able to select an existing XSLT style sheet. If one isn't available, the user must also be able to create one within the activity UI. Another sub-function is that when a user defines how the output will be stored, the activity must also allow her to define the output file name. Defining the sub-functions is important because I needed to create a UI component that performed every function and sub-function.
Designing, Drafting, and Revising the UI

My initial XSLT activity UI design included one main UI and three sub-UIs that may or may not be used based upon how a user configures the activity. Because there were so many sub-UI windows, I tried to keep the main UI as simple as possible. For my first-draft design of the XSLT activity, I built it similar to the Put File activity, i.e. I placed the elements in the order that a user would configure them. First a user chooses input XML, then an input XSLT style sheet, and finally she defines how the output should be stored. There are two sub-functions for each of these three main functions, and they concern different methods to retrieve the input XML and style sheet and how to write the output. My initial design for the XSLT activity UI underwent major revisions as a result of feedback I received from my mentor and the developers. When I submitted my requirements to development, I received extensive feedback via a message board in Sharepoint. The first piece of feedback from the developers was that my initial UI design (and the resulting requirements) forced the user to make unnecessary decisions. For example, one section of my first-draft UI required that the user choose between retrieving an XML input from a complete file or from an XML fragment. The developers pointed out that from the user's perspective, this difference is inconsequential. The user wants to transform XML and does not care how that XML is delivered. I redesigned the input step so that the activity required logic to determine the type of input, which took the responsibility to do so out of the user's hands. Figure 13 shows the before and after UI design of the Source XML step. The redesigned version is simpler.

![XSLT Input UI Before Feedback](image1)

![XSLT Input UI After Feedback](image2)

**Figure 13: XSLT Input Step – Before and After Feedback**

The second major revision to the XSLT UI design was that I changed how the user defines the XSLT style sheet. In my first design, I allowed the user to define the style sheet by either selecting a data item that contains it or by defining the style sheet in the activity's UI. If the user wanted to define the style sheet in the UI, she had to click a button that would open up a new XSLT style sheet design window. Development commented that this functionality was buried in the UI and would, therefore, confuse the user. Table 9 shows my first UI design for the input of the style sheet and my redesign resulting from developer feedback.

My initial design had several problems. First, the second radio button combines two separate actions, creating a new style sheet in the UI and importing existing style sheet code into one UI section. It does not provide enough UI components to enable the user to perform both.

My initial design had several problems. First, the second radio button combines two separate actions, creating a new style sheet in the UI and importing existing style sheet code into one UI section. It does not provide enough UI components to enable the user to perform both. The functionality for the import is instead buried under the button “Edit XSLT Style Sheet,” but this language does not imply the import functionality, thus hiding it from the user. Secondly, if a user were to enter or import XSLT style sheet code into the secondary UI window she would be
unable to see that code in the main UI because there is no window to present it. Therefore, the user would be forced to open the design window to view any imported or written style sheet code.

The main issue with my initial UI design was that essential functionality was hidden. This problem was pointed out in the developer feedback, and I redesigned the UI to raise the level of the important components so that they are readily visible by the user in the main UI window. The first fix I implemented was to create an “Import Style Sheet” button. This UI component clearly defines its purpose in the text of the button. The second improvement was to include a large text box that allowed the user to see the style sheet code in the activity’s main UI without accessing a secondary window. I still provided a secondary style sheet “editor” window that allows the user to create a style sheet using dynamic data items, but she can also enter code directly into the text box without accessing the design window. As a result of developer feedback, I improved the clarity and usability of the UI by using the logical arrangement of elements to communicate the purpose of the activity. The improvement in design can be seen in the before and after feedback screenshots contained below in Figure 14. The option to create a style sheet in the activity is communicated via the text box and sample style sheet code.

![XSLT Style Sheet UI Before Feedback](image)

![XSLT Style Sheet UI After Feedback](image)

Figure 14: XSLT Style Sheet Input – Before and After Developer Feedback
The final major revision to the XSLT UI that came from developer feedback was adding a fourth bit of functionality to the activity. The original three functions allow a user to define the input XML, the input XSLT style sheet, and the output file. The new functionality is a feature that allows the user to perform a test XSLT transformation while he or she configures the activity. The reason for the test transformation feature is that XSLT transformations are often complicated and may fail entirely or produce errors in only one or two parts of the transformation. Incorporating a test function in the activity saves time for the user because she can test the transformation when she designs it, as opposed to having to run the entire process and see if the output of the XSLT activity is correct. The need for this feature did not occur to me because of my limited experience with XML. The developers had more knowledge about my audience and, therefore, were able to provide insight into what features would improve their experience. As a result, I designed a new UI component that opened in a secondary window when the user clicked the “Test Transform...” button. I made this feature a secondary part of the UI because it was an optional function meant to improve a user's experience rather than being a required step. The test interface can be seen on Page 16 of Appendix E. It requires the user to provide two inputs, a test piece of XML and a test XSLT style sheet. The resulting output is displayed in the third text box.

Writing and Revising the Requirements

The initial draft of the XSLT requirements provided valuable experience in writing on a technical subject about which I had little experience. Also, I learned about the language differences between functional requirements, which define the general inputs and outputs of a software program, and design requirements, which define the technical details for how a program will handle inputs, process them, and produce outputs. Functional requirements are written by a technical writer who defines what an audience needs and then designs the software around those needs. Design requirements are written by a software developer and define the software code required to process an input and produce the desired output outlined in the functional requirements. Therefore, I didn't need a complete understanding of XML and XSLT transformations in order to write the requirements. I just needed to learn generally the purpose of XML, how it is structured, and that XSLT style sheets are “instructions” for how to change a given chunk of XML into different XML or a different type of document entirely.

Given this knowledge, I wrote the requirements in general terms and left it up to developers to ensure that the technical interactions of the activity worked correctly. An XSLT transformation is a complicated programmatic function from a technical perspective. However, as shown in Figure 15, my functional requirement in Step 5 describes only the input and output, not how the input should be processed from within the program. Step 5.1 gets a little more specific in that I define the file extension that should be used for each type of output. These requirements are still simple because these three lines of text describe processing that may take over 100 lines of code to produce.

I learned that describing technical functionality requirements so that they can be understood by a non-technical audience is a strength, not a fault. The purpose of functional requirements is to describe the function of a program in simple, clear prose that should be able to be understood by multiple audiences, not just highly-technical programmers. My job as a technical writer at Bluespringing was to understand the audience and define how it should interact
with a program and the outputs that should result from that interaction. Since a program can be written in any number of ways, the technical details of a program's processing should be determined by a developer with the experience to make the code efficient and such details should be left out of the functional requirements.

5. At runtime, the activity will load the XSLT XML and transform the source XML document into an XML, HTML, or Text document according to the styles in the loaded XSLT document.

5.1. The created file will have an extension that matches the type of file it is created as (XML = .xml, HTML = .html, and Text = .txt)

5.2. The activity should fail if any errors occur while transforming the XML document using the XSLT document

Figure 15: Functional Requirements Describing an XSLT Transformation

The developer feedback on the XSLT UI resulted in new logic in the input step that required that the code of the activity determine the input data type, so I had to rewrite the requirements to describe this new logic. My original requirements described actions that the user performed, which totaled four steps. The new requirements describe one step that the user should perform and the rest of the processing is completed by the activity's code. Figure 16 shows the rewritten requirements. Requirement 1.1 is the step performed by the user. In it, she picks the input that contains the XML to transform. The rest of the requirements describe the processing steps performed by the code. Basically, the code determines the type of input, whether it is an XML fragment or a complete XML file, and then it processes each accordingly. There is also an error-checking step that flags invalid input. Through working on the XSLT requirements, I learned an important lesson from the developers in that I should always design a piece of software so that it performs as much work for the user as possible. A user should provide input only when needed and the software should streamline a user's process, not encumber it with extra steps.

Figure 16: Example of Detailed Requirements Describing One User Action - Rewritten requirements that require only 1 user action; the remaining steps are performed by the software.

Developer feedback also resulted in the new UI functionality that allowed users to perform a test XSLT transformation within the activity configuration window. As seen in Figure 17, I had to write specific requirements to describe how the test transform UI would function. The user could enter the XSLT style sheet in one of two ways. Therefore, the test
transformation functionality had to be designed to handle the two different style sheet input methods. Requirement 6.4.2 in Figure 17 is an example of a requirement that describes how a chunk of data should be handled but leaves the technical details for how to best process that data up to the developers.

<table>
<thead>
<tr>
<th>6.4. The Load Style Sheet button can perform one of two actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.4.1. If the user selected the Style sheet contained in a Data Item option in the Properties for Activity window, then a Browse for File window opens and the user must browse for an XSLT style sheet file (Figure 2)</td>
</tr>
<tr>
<td>6.4.1.1. After selecting and loading an XSLT style sheet in the Browse for File window, the style sheet should load into the XSLT style sheet text field</td>
</tr>
<tr>
<td>6.4.2. If the user selected the Define style sheet now option in the Properties for Activity window, then text in the Define style sheet now text field (in the Properties for Activity window) should load into the XSLT style sheet text field</td>
</tr>
</tbody>
</table>

Figure 17: Requirements Describing Test Transformation Functionality

Conclusion

From working on functional requirements for new software in BPM Suite, I gained experience in writing documents that address a wide range of audiences. First, I designed the UI and wrote functionality requirements for the end user, keeping their interests in mind. The UI of the software had to be intuitive enough that a user could interact with it without needing paragraphs of explanatory text. Also, the logical structure of each activity UI had to guide the user through a natural progression of steps. If the software’s use was not evident in the structure of the UI, the user would give up on the software. Secondly, I had to write the requirements for development. I needed to keep the language simple and clear enough so that the functionality of the software described in the requirements could be accurately translated into usable software code. This often meant that I had to purposefully keep the technical language to a minimum by ensuring that a requirement described the input and output only and left decisions about technical data processing steps up to the developers. Finally, although my internship ended before I experienced this part of the process, I had to understand that my requirements also guided auxiliary activities like the development of testing and training documentation for the new software. The need to address many different audiences made developing the software design functional requirements the most rewarding and educational experience of my internship.
Chapter 4: Analysis of Bluespring’s Organization and the Functional Requirements Process

As I discussed previously in this report, Bluespring's CTO standardized the BPM suite development process so that the company could consistently update its software to meet customer needs. Bluespring started by identifying a customer need and worked down a series of steps in which a team containing a technical writer, developers, and testers designed, implemented, and evaluated a solution to update BPM suite to address the customer need. I was given the opportunity to be an integral part of this process during my internship; as a result, I gained experience in creating the visual design of program user interfaces as well as writing the documentation that describes their function. Through an extensive feedback and review process, I also learned how various groups within the organization, such as developers and quality assurance personnel, used my work. Receiving feedback from these groups gave me insight into how a technical document often has multiple audiences that use the information in different ways. Through these feedback interactions, I also gained the interpersonal experience of working with different groups within an organization to solve several different problems with one set of documentation.

For the rest of this chapter, I view the knowledge and skills I gained from my internship experience at Bluespring Software through the lens of academic theories and developmental models. First, I view the organization of Bluespring through the “Organizations as Organisms” metaphor detailed by Gareth Morgan in his book *Images of Organization*. Second, I discuss how my major internship activity of writing functional requirements for new BPM Suite activities aligns well with the monadist definition of technical writing proposed by David N. Dobrin in his essay “What’s Technical about Technical Writing”. Finally, I list how my internship experience reinforced specific technical communication skills and strategies.

**Bluespring’s Organization Compared to the “Organizations as Organisms” Metaphor**

The book *Images of Organization* by Gareth Morgan provides eight metaphors that compare different types of organizations to common concepts, allowing a clearer understanding of their form and function. Example metaphors compare organizations to machines, brains, cultures, or political systems. While thinking about my internship at Bluespring, I found that its organization is best understood through the “organizations as organisms” metaphor, and below I discuss how Bluespring as it existed at the time of my internship fits the metaphor by functioning as an organism and how that function influenced my internship experience.

The “organizations as organisms” metaphor posits that some organizations function like a living animal within a natural ecosystem. That is to say, much like animals, the survival of “organism-like” organizations depends on their ability to adapt quickly to changes in the environment, to maintain the health of their “body”, i.e. internal organizations and processes, and to collaborate with other “organisms” within the ecosystem. Bluespring fits well within this description because its organization focused on adapting to customer and competitive pressures, promoting organizational health despite a turbulent environment, and actively collaborating with external organizations within the same industry.

**Adapting to the Customer Environment**

Bluespring’s entire organizational structure and business plan were focused on adapting
to customer needs. This focus on adaptation is apparent in the functional requirements process in which the CTO identified a customer need and Bluespring enhanced BPM Suite with a feature that met that need in a period of 4-6 months. According to the “Organizations as Organisms” metaphor, companies that focus strongly on adapting to either competitive or customer needs rely on a contingency approach to organization. Organizations that take a contingency approach use open collaboration and communication across all levels in the organization and a high rate of information exchange in order to facilitate frequent product modifications to address changing technology or customer requirements. The previous statement is an accurate description of Bluespring at the time of my internship. There were no walls, neither literal nor figurative, between the CTO, developers, and all the personnel that helped develop functional requirements for new activities. Running a design idea by my mentor, a developer, or the CTO only required that I take a short walk and ask and, more importantly, those people had no problem stopping their work to answer a quick question or setting aside time to perform a more in depth review. For example, in order to provide feedback on my requirements design for the XSLT Transformation activity, a developer scheduled a meeting with me so he could provide detailed feedback on each section of my design, doing so not only to improve my design but also to educate me on standard software UI design strategies and practices. This flexible and organic communication style is one of the facets that made Bluespring function more like an organism, and the nimbleness characterized by the contingency approach is a common one in the technology industry. As Morgan states in *Images of Organization*:

“...the process of finding out what one should be doing proved unending, defining a mode of organization linking inquiry and action, and the process has continued. Successful electronics firms avoided organizational hierarchies and avoided narrow departmentalization, with individuals and groups defining and redefining roles in a collaborative manner in connection with the tasks facing the organization as a whole. They created innovative, team-based organizations having more in common with an amoeba than a machine (Morgan, 1997).”

Morgan also discusses that in particularly chaotic environments, organizations that function as organisms allow workers to define their own responsibilities and dictate their contributions to the organization. In other words, a “natural selection” of worker responsibilities and tasks occurs, and I experienced this natural evolution of my role as an intern at Bluespring as I was allowed to contribute to a variety of tasks that fed into each other. For instance, my initial project automating the internal trial license renewal process allowed me to learn BPM Designer and the use of activities to build steps of an automated process, and this knowledge of how Bluespring’s product worked helped me write functional requirements for new BPM Designer activities during the latter portion of my internship.

**Promoting Internal Organizational Health**

As a small company, Bluespring had the advantage of being able to establish an informal environment to combat the stress caused by the need for constant adaptation and change. For example, as I discussed in Chapter 1, employees were judged from a performance, end-product perspective, meaning that managers did not micromanage their employees’ work and instead trusted in their work ethic. Furthermore, the interactions between management and workers at Bluespring were purposefully informal, as the foosball table and pints of Friday afternoon beer attest, and this informal environment reinforced the collaborative, open culture at the company.
Morgan refers to this type of organization as an “adhocracy” and states that it is “…very informal and flexible and, although run in a highly centralized way by the chief executive, is ideal for achieving quick changes and maneuvers…[it is] a form highly suited for the performance of complex and uncertain tasks in turbulent environments (Morgan, 1997).” Given Bluespring’s uncertain and chaotic past, I believe the informal culture of the company during my internship originated as a survival mechanism in order to ensure the workers trusted and were comfortable with each other. Workers, including me, felt like they belonged to the community of the organization and that they were expected to make significant contributions to Bluespring’s success. In other words, the informal environment ensured the internal health of the company and allowed it to maintain the quality of its output.

Relying on Collaboration with External Organizations

As I discussed in Chapter 1, Bluespring worked hard to collaborate with external companies, two notable examples being the integration of InRule’s Business Rules Engine (BRE) into BPM Suite and the establishment of a Gold Partnership with Microsoft. Morgan states that a focus on collaboration, especially the establishment of strategic relationships with outside companies, is a common characteristic of organizations that act like organisms. In order to increase their chance of survival, organisms will often collaborate with each other in a mutually beneficial exchange in order to negate each other’s weaknesses.

Relating this concept to Bluespring, the Microsoft Gold partnership benefited Bluespring by giving it the endorsement of a large and established company as well as a sales associate focused on selling BPM Suite to Microsoft customers. Microsoft also benefited because BPM Suite integrated with many Microsoft products, including all products in the Microsoft Office Suite and Microsoft SQL database, meaning that each sale of BPM Suite meant that the customer needed to purchase all the integrated Microsoft products as well, thus increasing the adoption and reliance on Microsoft technology.

The collaboration with InRule was equally symbiotic. InRule’s product was a robust and comprehensive business-rule engine that contained the logic for many common business decisions. However, InRule did not have a software product that used their business rules engine. On the other side, Bluespring had a business-process management application but it did not have the time or resources necessary to develop a robust business rule engine to drive decision points in a process. By collaborating, both InRule and Bluespring strengthened each other’s product offerings by fulfilling a specific need.

Morgan states that an evolutionary perspective of organizations assumes that they exist in a constant state of competition with each other in order to gather enough resources for survival. However, organizations, like organisms, are actually one part of their organizational environment, meaning that they have a choice to either compete or collaborate with other organizations in the environment, depending on the beneficial or detrimental outcome that will result from the competition or collaboration. By making a decision between competition and collaboration, Morgan states that organizations “…can play an active role in shaping their future, especially when acting in concert with other organizations. Environments then become in some measure always negotiated environments rather than independent external forces (Morgan, 1997).” By working with other companies, Bluespring was able to benefit in ways that increased the value offered by BPM Suite while also saving time, money, and resources. Basically, like a living organism, it established strategic relationships in order to increase its chance of survival.
Bluespring’s strategy to collaborate with other external companies influenced my internship by creating an environment of rapid growth and expansion, allowing the opportunity for me to contribute to the design of new functionality in Bluespring’s software. Without the rapid growth that Bluespring experienced during my internship, my role might have been limited to more traditional tasks like writing help documentation and user guides.

**The Bluespring Functional Requirements Process and the Monadist Definition of Technical Writing**

A traditional definition of technical writing, as David N Dobrin argues in his essay “What’s Technical about Technical Writing?,” originates from a universalist view of communication. According to the universalist view, any given communication can unequivocally mean one thing and one thing only, and good technical writing transfers that one meaning to the audience without distortion. According to this view, a technical writer does not create meaning, he or she analyzes technology, finds the inherit meaning, and translates that for the audience. However, Dobrin proposes a “monadist” definition of technical writing that contrasts with the universalist view. It suggests that communication is “indeterminate and can never be precisely understood (Dobrin, 1983),” and it repositions the output of technical communication to be an artifact of negotiated meaning between the writer and his/her audience. The work that I performed during my internship at Bluespring software positions my contributions as a technical communicator more within the monadist definition of technical writing, and that definition is that “[t]echnical writing is writing that accommodates technology to the user” (Dobrin, 1983).

Specifically, the functional requirements for new BPM Designer activities that I wrote were products of negotiated meaning among me, my mentor, the developers at Bluespring, and any other reader that reviewed, approved, or used my designs. Before I wrote the functional requirements, the new BPM Designer activities existed only as a customer need that Bluespring wanted to address within their product. The developers (and other personnel at Bluespring) did not have an agreed upon understanding of the form it should take as a tangible function within BPM Designer. Only after I produced a functional requirements document and UI design was Bluespring able to view the activity not as a function, but as an idea or concept that could evolve through discussion, thereby allowing Bluespring to understand the design and decide what it should or should not be (or how it should or should not function). In essence, my writing defined the meaning of the activity and allowed it to be understood and its design to be debated over and modified to more closely align with the intentions and goals of Bluespring as a company. My writing not only accommodated the technology of BPM Suite to the Bluespring customer who would eventually use the new activity, but it also accommodated it to users internal to Bluespring. This point relates to Dobrin’s discussion of the negotiated meaning created by technical writing:

“The ambiguity of the word “accommodate,”...sets forth the mutuality and mutability of the power of relationships established by technical writing. More over, “accommodate” suggests that conversion of thing to function which is crucial to the discourse of technics. Technical writing appears to be concrete, but it is only concerned with the concrete. In fact it makes the concrete abstract; it replaces the obdurate thing with the manipulable concept. It thereby makes the strange, invasive, expensive, or inefficient into the familiar and useful (Dobrin, 1983).”
One example where my writing accommodated the technology of BPM Suite to internal users within Bluespring is my functional requirements for the XSLT activity. My initial design of the activity required that the user identify the type of input XML. During the feedback review process, developers pointed out that the user should not need to make this decision; instead, the activity itself should be able to automatically detect the type of input XML, i.e. whether it is a fragment of XML or a complete XML document. This design change not only made the processing more efficient and prevented the user from needing to make an unnecessary decision, it also allowed the developers to reinforce a design principle of BPM Suite, which is that the program should automate as much as possible within the design and execution of a business process. Before my functional requirements and UI design, the customer need for the XSLT activity was only a concrete need – that BPM Designer must provide an activity that allows users to automate an XSL Transformation. But my requirements made the concept of the XSLT activity abstract so that it could be discussed, modified, and ultimately improved so that it was a better fit within the overall design philosophy for the BPM Suite product as a whole. My requirements defined one possible meaning for the XSLT activities design, but through collaboration with other personnel within Bluespring, the meaning of the requirements were clarified and refined.

The importance of collaboration in my writing was consistently highlighted by my internship experience. My role as a technical communication intern that helped develop and mold meaning, as opposed to one that just transferred or translated meaning, was enabled by Bluespring’s organism-like organizational structure and culture, which encouraged open communication and collaboration. The flexibility and lack of departmentalization allowed me to work closely with several different groups within Bluespring and, as a result, I was able to get many different perspectives and points of feedback on my writing. In other words, I learned to adapt my writing to internal as well as external audiences. This awareness of not only the end-user audience, but also the internal audience who helps refine a communication’s message, is central to the monadist definition of technical writing, because as Dobrin writes, “the piece of technical writing or the act of writing technically are two of several foci, which must also include the practice of the group which the writer is writing to, writing for, and writing from, as well as the practices of the group in which the writer has located himself or herself (Dobrin, 1983).”

This concept of understanding the role of one’s writing within the different groups of an organization is relevant to a common goal of technical writing: using one document to communicate to multiple audiences. This practice, I discovered, is a delicate balancing act. For instance, in writing my functional requirements, I had to include the appropriate level of technical detail to accommodate the design intentions of the developers so that the final activity would fit the intended function and meet the customers’ needs; however, I also had to ensure that I didn’t add too much technical detail to avoid confusing the training, quality assurance, and the marketing personnel readers. Otherwise I ran the risk of alienating the less technical audience members, which may or may not cause inaccuracies in the communications about the new activity that they produced. Essentially, I had to learn to identify a specific “horizon” in my writing or, in other words, a target level of detail that communicated as effectively as possible to all audiences. If I went above the horizon, I risked providing too much detail, if I went below the horizon, the details I provided would be too few. I identified this horizon level of detail that accommodated the activity to the most audience members by sharing my requirements with my mentor, developers, and all other readers. Receiving feedback on my requirements and negotiating the activity design with other groups reinforced the concept that my writing was not
the final output; it was the point of origin for many other types of output, whether that output was software design requirements, quality assurance test scripts, training modules, or even the activity code and UI, all of which had their meaning negotiated, modified, and collectively finalized within a cycle of collaboration.

**Viewing Dobrin’s Monadist Definition of Technical Writing from the Perspective of My Internship Experience**

Dobrin wrote his essay “What’s Technical about Technical Writing” in 1983, which potentially makes its message dated. Since that time, there have been several critiques of Dobrin’s work. One example is the 1990 article “The Case Against Defining Technical Writing”, by Jo Allen, in which she critiques Dobrin directly, writing that his is a “behaviorist definition that is flawed with the same kind of experience-based assumptions for which Dobrin has criticized others. Like the definitions he criticizes, Dobrin’s definition stems from untested observations that present no evidence of any kind of systematic study” (Allen, 1990). Allen’s main point is that the discipline of technical writing should not be defined. It concerns too many types of audiences, contexts, formats, techniques, and purposes to be described accurately by a one or two sentence definition like Dobrin’s. Finally, she concludes that it is fruitless to create a definition of technical writing because it concerns a thing, technology, that rapidly changes and evolves and technical writing, along with its definition, must change as well to remain relevant. Allen asserts that as soon as one definition is established, it will likely be weakened by the need for the discipline to continually evolve.

In an introduction to his own piece in the anthology “Central Works in Technical Communication”, Dobrin critiques himself. He reflects that the most ubiquitous form of technical writing available at the time he wrote his article was thick and intimidating computer manuals. These manuals, many being hundreds of pages long, contradicted Dobrin’s definition because they did not accommodate technology to the user. Instead, they had the opposite effect and acted as a barrier to understanding and using the technology that they were meant to explain. Dobrin calls them “anything but accommodating” and notes that his definition does not account for how technical communication must change when he writes “[i]f I had had the clarity and common sense to see how much of my own time and effort I was wasting, perhaps I could also have seen what was wrong with these pieces, and I might then have written an article that would have killed these things off sooner” (Dobrin, 2006).

Both Allen’s and Dobrin’s critiques of the monadist definition of technical writing highlight the importance of the diversity of experiences I gained from my internship, since during it I modernized my technical writing skills by integrating them directly into the software development cycle, instead of isolating my contributions to post-development work like online help and instructional manuals that fit the more traditional definition of the discipline. The benefits of my internship are highlighted in the article “The Future of Technical Communication” in which Barbara Giammona presents the results of interviewing and surveying more than 28 professional technical communicators in 2003. When asked, “What is our future role in organizations?” many respondents indicated that the new role of the technical communicator is that of the “product developer.” The consensus among the interviewees and survey respondents was, that for technical communicators to remain valuable, they must

“...make a strong move toward inserting [themselves] in the development and innovation processes for the products we support. There needs to be a clear
connection in the minds of . . . employers between the sources of a firm's revenue and [the technical communicator's] contribution...By being more involved in product development, [technical communicators] will become more easily identifiable as contributors with a unique set of skills to offer. [They] will also be involved in innovation again, contributing actively the specialized knowledge that . . . [their] skills make [them] well suited for.” (Giammona, 2004)

Giammona’s research helps position the skills that I learned during my internship as slightly beyond the scope of Dobrin’s 1983 definition - which seems only to focus on technical writing about technology that is completed and available to the end-user - and gives them a relevant contemporary purpose. My functional requirements did not fit neatly within the monadist definition because they concerned technology that was not created yet. They did not exist between the end-user and the technology. In fact, the documents were not meant to be read by end-users at all and acted more as a conduit through which the developers and other members internal to Bluespring reviewed and understood the potential design of the technology. The resulting understanding that these groups gained about the technology then allowed them to tweak and improve its design, which resulted in a better end-user experience. In this sense, the functional requirements I created taught me to actively accommodate the technology to the user as it was being created; and, therefore, writing them helped me understand how I can use my technical writing skills to add value to the development process in the future.

**Lessons Learned**

My internship at Bluespring was beneficial and rewarding because I was able to apply my technical communication skills to many areas within the company, therefore expanding my understanding of the role a technical communicator can play in a software development environment. Also, I learned many specific lessons during my internship, and I group those lessons into two categories: lessons learned in software development and lessons learned in technical communication.

**Software Development Lessons Learned**

Through feedback from my mentor and software developers, I learned the following software UI design and functionality requirement lessons:

**UI Design**

- Build the User Interface so that the user can easily identify important functionality options; essential functionality should not be concealed within menus or embedded alongside unimportant options.
- Use succinct and direct language within a UI.
- Ensure text next to a UI element describes only that element.
- Organize the UI elements so that they communicate the software's function and the order in which a user should interact with each part.
- Identify the input variables that the user must enter to perform a task and pick the UI elements that are best suited to enable the user to enter those inputs.
- Spend time tweaking and refining a UI’s visual design. The software development
environment reinforced the importance of visual design. Many of the tasks performed by each software UI are communicated not through words, but through the visual organization of elements. For example, in the second draft of my XSLT UI, I provided a big text box for the user. The size of this text box communicates without words that the user is expected to use it to create an entire style sheet within the UI. My original design only provided one small line and thus did not communicate the functionality clearly.

Requirements Composition

- Define the boundaries of the program's function. Describe main functions of the program with explicit detail, and also define some functions that the new software should not be able to perform.
- Design the software requirements so that the final product will demand as little user input as possible to perform a given function – the software tool should perform most of the work and user input should be required only if processing cannot be completed with information currently available in the process stream.
- Compose requirements that balance between providing detailed technical information and high-level descriptions of intended functionality that can be understood by any audience. Requirements often must be understood by a non-technical audience in addition to developers.
- Identify all tasks that a user must perform with a piece of software before writing the requirements or designing the UI. The structure of a UI or requirements document can often be derived from an organized list of user tasks.
- Use the structure and visual appearance of a requirements document to communicate the relationship between information. High-level, parent requirements should be bolded and occupy a left-aligned position in the hierarchy. Sub or child requirements should be indented under each parent requirement. This visual design communicates the order of information to the user without him or her needing to read any text.
- Understand that the development of new software is never “done” - all components are continually improved to address defects, new customer needs, and new technologies. As a result, a requirements design may change over time or be modified or superseded by a new set of requirements.
- Be aware of the position that requirements have in the overall software development process. The requirements process is a smaller piece of a parent development process. Other processes are initiated based on development, i.e. marketing begins informing customers of upcoming functionality updates as developers reached the end of their cycle near a release.

Technical Communication Lessons Learned

- My internship experience reinforced the following technical communication lessons:
  - Understand that documents produced by a technical communicator are integral to a software company's success. The quality of design documentation at the beginning of a process affects the quality of all processes down the line. For instance, my requirements affected not only the software design but also the quality of the test plan, training, and
online help.

- Get feedback from non-technical communicators; it is just as valuable as getting an experienced communicator's feedback. The audience of technical communication is often not technical and their ability to understand a technical document speaks to the quality of language it uses and the information it provides.

- Ask focused questions when requesting feedback from different reviewing audiences. If having a document read by development and training, for example, ask questions about parts of the document that will benefit from their expertise. Feedback will be of higher quality if the reviewers have specific sections to review and questions to answer, as opposed to asking each audience to read the document without any direction or purpose.

- Divide technical subjects into smaller components so that the descriptions of those pieces can remain simple. The more technical a subject, the more segmented and simple each description of that subject should be.

- Label information that is meant for a specific audience in documents that have multiple audiences. Often a technical document will serve a different purpose for each audience, so it is important to make information aimed at specific audiences accessible with document structure and labeling. An audience member should not need to comb through a document to find the information pertinent to him or her. If necessary, divide the subject into different documents if the information necessary for each audience is too different.

- Explain in detail the information that the audience should gain from viewing a visual aid. In the composition of my functional requirements and Quick Start Answer Guide, I used many visual aids. In each instance, I learned to explain the purpose of the graphic. Without an explanation, the audience member might draw his or her own conclusion that could contradict the intended purpose of the graphic.

- Learn to multitask not only to increase work efficiency, but also to decrease work fatigue caused by working on the same task for an extended amount of time.

- Understand that writing about a subject about which a technical communicator has little knowledge can be an advantage. Knowing little about a subject means that the communicator can ask questions and identify issues that may not be noticed by subject matter experts who are comfortable with a particular subject.

Conclusion

My Bluespring internship experience consistently reinforced one lesson paramount in the Master of Technical and Scientific Communication program at Miami University; in all communications I create I must always consider the needs and expectations of the audience. This goal was not always as simple as it sounds, since communications often have multiple audiences with different or even incongruent goals. However, throughout my internship, I experienced again and again the benefits of analyzing one's audience. Audience analysis is the foundation for a communication because it helps a writer define a document's purpose and outline a communication's structure. It helped me define the organization of my UI designs and functional requirements for new software enhancements, as well as informed my writing in other auxiliary projects during my internship.
The lessons I learned during my internship will enrich my technical communication approach and practices throughout my career. I would not have had such a beneficial experience if Bluespring had not expected me to contribute to a variety of projects within the company. As a result of my internship, I not only practiced and refined my communication skills; I also expanded my understanding of a technical communicator's role within a company. Before my internship, I saw technical communication as a discipline with the assigned role of developing documentation that explained a piece of software. Now, I see how I can use my communication skills to help build that software from the ground up. As I continue in my career, I am excited to leverage my skills to continually highlight my value as an active technical communicator.
References


Appendix A – BPM Engine and BPM Web Description

**BPM Engine**

BPM Engine is the brain of BPM Suite. While a business process designer creates a process on his or her desktop via BPM Designer, the process does not execute locally. Instead, the designer saves the process, which sends it to BPM Engine on a server, which operates as a Service-Orientated Architecture (SOA). When it comes time to execute the process, BPM engine performs all processing and communication, which involves the execution of activity services in each process step, but also the data transfer between steps and the recording of process metrics and status. At that point, BPM designer can query process status from BPM engine, but none of the services are executed on a desktop.

When creating a new BPM designer activity the developers must create two components. The first is the activity configuration interface and the second is the new service that executes the activity’s processing within BPM Engine. The functional requirements that I wrote for new BPM Designer activities served a dual purpose; they described both the functionality required for the activity configuration window and the BPM Engine service that performed the activity’s processing.

**BPM Web**

BPM Web is a browser-based interface that allows users to enter information into a process. Specifically, it allows users to perform tasks as well as manage the tasks of others. As discussed earlier in this report, each user in BPM Designer is assigned a profile that identifies his role as well as his privileges within the system. During execution a process will assign manual activities to users based on their roles and business rules defined in the activity.

The BPM Web interface provides an interactive list of all tasks that are assigned to a user. Within the interface, a user can view and change the status of his or her tasks, reassign those tasks to different users if they are complete or need input from different people, as well as complete the task by entering information tailored to the function of each activity. For instance, if a manual task requires that a manager approve an expense report, BPM Web will provide that expense report as an attachment in the task and a field that allows the manager to either approve or deny the report and provide a reason for his or her decision. Another manual activity may task a customer-service representative with entering a new customer’s information into a database. In this case, BPM Web would provide fields for each piece of new information – the customers’ name, email, address, phone, credit card number, etc. – and then, once entered, it would store that information into data items and then write to a database in the next activity within the process.

BPM Web also provides metrics information for a given process or activity. This means that a manager can view all tasks assigned to his or her subordinates and the status of those tasks, i.e. whether they are overdue or on schedule. If a given task is behind schedule, a manager can escalate the task, which either heightens the priority of the task in a user’s list or it assigns it to a different user. The ability to reassign tasks and view metrics for a task or process is based on the privilege of each user. Lower-level users can only view their own tasks and reassign them to users at the same level. High-level users can view their tasks as well as those for all the people
below them. Figure Appendix A-1 shows the task bar presented to the user upon log into BPM Web. The user can see the list of all his or her assigned tasks on the “My Tasks” tab. When clicking upon a task, a new “Edit Task Data” window opens that allows the user to enter information needed to complete it. Figure Appendix A-2 shows an example of such a window. Notice that the user can enter information in a variety of ways. He or she can type text, attach a new file, or choose an answer from a drop-down list. In summary, BPM Web is the interface through which most users will experience BPM Suite. Anytime a process requires human input, that input is entered and managed via the BPM Web interface.

Figure Appendix A-1: Menu bar presented to users upon sign-in to BPM Web.

Figure Appendix A-2: “Edit Task Data” window that allows users to enter information to complete a task.
Appendix B – Summary of Miscellaneous Projects

**Quality Assurance Testing**

New versions of BPM Suite are released every 2-3 months. These version updates (called release candidates) include bug fixes and any recently-developed features and improvements. Before a release candidate is approved, all bugs and new features must be thoroughly tested, and since Bluespring only has one Quality Assurance Analyst, often other members of the organization are tasked with helping test new features to make sure a candidate meets a release deadline.

I helped test one release candidate. The test took three hours and it required that I follow a “test script”, which detailed steps that I must perform in BPM Designer to ensure that a new activity performs as expected. Interestingly, the Quality Assurance Analyst writes the test scripts using the functional requirements as a guide – the same functional requirements that my mentor, programmers, and in some cases, I had written. Therefore, by participating in product testing, I was able to witness how the quality of the writing at the beginning of the development process influenced the final product that was verified to work as documented before release to customers. Any deviations in functionality were documented in “Regression Reports”. After each testing session, the CTO and Chief Developer review the regression reports and determine if a release is a “go” or “no-go”, i.e. they decide if the defects still present are significant enough to stop the new software version from being released to customers or if the problems were small enough to be fixed in a subsequent release. I did not test any functionality that I designed; however, at the end of my internship, I received an email from the Quality Assurance Analyst that confirmed that one of the activities I did design – the “Put File” activity – passed testing and therefore was going to be included in the next version of BPM Suite.

**Defect Reporting**

In addition to Quality Assurance testing, I also reported defects. Bluespring had a defect management database called “OnTime”. OnTime tracks the name, reporter, severity, priority, description, and status of each defect. In addition, developers also indicate which release of BPM Suite will include the defect's fix. Since I worked closely with BPM Designer to design test processes, I used the full extent of the program's functionality and I therefore encountered any instances where the program did not perform as intended or where it produced errors. In these instances I opened a defect report in OnTime, described the problem, wrote step-by-step instructions for reproducing the problem, and then assigned the defect to development.

After assignment, I became the reference for the defect, meaning that a programmer would come to me to ask questions about the problem if the report did not provide enough information to fix the defect. As I gained more experience in writing defect reports and learning what information the programmers needed to fix the issues, the quality of my reports improved and the programmers asked me less and less follow-up questions. Therefore, writing defect reports helped me practice audience-centered writing.

**Review of Customer Communications**

Soon after my internship began Bluespring leadership decided that the company should try harder to communicate actively with customers. The first customer communication to
originate from this decision was a “Customer Support Handout” written by the manager of Human Resources. The purpose of the Customer Support Handout was to educate customers on Bluespring's support center; the document identified support services that Bluespring could provide, like “New Client Setup and Licensing Assistance” and “Technical Assistance and Problem Solving”. It also detailed Bluespring’s “Customer Response Cycle”, namely that new customer issues would receive a response within 30 minutes and 1 hour from the time they submit a question.

The Human Resources manager asked me to review the Customer Support Handout and provide feedback on both its writing and visual design. To improve the writing I suggested some changes to the labels of issues that Bluespring supports, ensuring that each label is specific, clear, and unique. The other changes I suggested were to the document's visual design. I applied the visual design principles of Proximity, Alignment, Repetition, and Contrast to make the document appear unified and cohesive. The visual design changes I made were to make sure only two fonts and colors are in the document, that each line of text aligns with another element, that a visual hierarchy is apparent through the use of bulleted lists and bold text, and that the groups of information are clearly separated from one another using white space.

Throughout the duration of my internship I also edited other customer communications like the website. Whenever a new web-page was produced by the technical writer, I would edit the new text for both typos and clarity.

**Researching Bluespring’s Competitors**

The marketing department asked me and the technical writer to research Bluespring's competitors and their products. The purpose of this research was to educate marketing and sales so that they could approach potential customers with the benefits of Bluespring's software when compared to the competition. To perform the research, I searched the competitor's websites for product information and I answered questions like

- Does the competing product manage processes with automated activities (system to system) and manual activities (people analyzing information and making decisions)?
- Do business analysts own the processes in the competing product?
- Does the competing product integrate with Microsoft office?” and
- What software code base, like Java or Visual Basic, is the competing product built upon?”

Learning about Bluespring's competitors was both intimidating and energizing. It was intimidating because I learned just how big the competition is – massive, successful companies like IBM, Oracle, Adobe, and even Microsoft have business process management products. On the flip side, it was also energizing because all these products had pros and cons and none of them are as robust or specialized as BPM Suite. For example, although Adobe's “Livecycle” allows business process owners to visually design processes and integrate worker-performed tasks into the workflow, it does not allow integration with any document or program that is not produced by Adobe itself, thus limiting the adaptability of their product. Learning the pros and cons of the competition like the example above helped me understand my role not only in Bluespring as a company, but also as a technical writer operating in a specialized segment of the programming industry, and it helped me to understand how my work was improving BPM Suite in relation to Bluespring's competitors.
One of Bluespring's newest customers, G4S – an armored-truck service for Cincinnati-area banks - needed help building one of its first processes in BPM Suite. Therefore, I was tasked with building process input diagrams. These diagrams identify a process step, all the data that needs to be captured during the step, and whether that data is user-defined, chosen from a drop-down list, or auto-populated by the BPM Suite system. G4S provided a description of each step and I converted that information into a diagram that clearly identifies the type of each input and output. The purpose of these diagrams is to be a reference for the process designer at G4S – he or she could reference the diagrams while building a process step and understand what information should enter and exit each step. The diagrams also highlight relationships between data in the processes. For example, in the “Night Deposit Processing” Diagram, arrows show that the “Region” and “Zone” fields are related to the regions and zones that a user defines at the beginning of the process.

The G4S process diagrams were visual design practice for me. I had to convey a large amount of information in a small space, and I used color codes, tables, and arrows to communicate the input and output for each process.
Appendix C – Skill Set Package Necessary for BPM Suite Proficiency

The table below was created by a “customer focus” consultant that Bluespring hired to research the BPM Suite end-user experience. The table details all the skills necessary for a user to operate every function in BPM Suite. I used this skill set to help understand the audience for my Quick Start User Guide.

The Skill Set Necessary In An Individual and / or Team To Design, Configure, Test, Debug, Implement, Maintain, Analyze, Improve, & Optimize Business Processes Utilizing BSSW BPM

<table>
<thead>
<tr>
<th>SKILL AREA AND SPECIFICS</th>
<th>Person A</th>
<th>Person B</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Easily understand Bluespring’s “language”</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>o This needs to be easy to understand in industry standard terms rather than just Bluespring terms. We should not be translating into our own language.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>o While this is probably true, we should strive to minimize this</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Business analysis &amp; basic technical skills</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>o Have knowledge and experience with a well documented BPM process / best practice</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>▪ Understand and be able to apply the principles of process flows (e.g. nested flows, decision points, etc.)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>▪ Be knowledgeable of and have experience in each stage of the business process life cycle</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>o Have the knowledge / experience to know what business &amp; technical requirements to gather</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>▪ Be able to interact with the business to gather both business &amp; technical requirements</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>▪ Be able to create data and process models, use cases, and activity diagrams</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>▪ Be able to understand data structures and data optimization</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>o Have a familiarity and working knowledge of Windows OS and Web Apps</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>o Ability to learn new technologies, terminologies, &amp; methodologies that apply to BPM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>o Be able to build / write conditional statements</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>o Be detail oriented</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>o Be able to create “rollback” process</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>o Knows how to streamline and improve business process flows</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Test &amp; debug business processes</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
- Knows how to visually monitor process as it executes to verify & validate outcome
- Gather, analyze, & change business processes
  - Be able to identify important metrics
  - Be able to configure reporting data via table writers
  - Ability to make changes to business process and compare two different production runs
  - Be able to use SQL or Crystal to present data in useable format
- Basic process integration (e-mail & web apps)
  - Understand and be able to use SOAP
  - Understand and be able to use xpath / xml
  - Understand and be able to use web form development (Perl, PHP, ASP, ASP.net)
  - Have knowledge of the specific systems that they need to link to (e.g. MS business solutions)
- Advanced process integration & extending our application
  - Be able to write SQL
  - Be able to write Vb
- Monitor & troubleshoot business processes in production
  - Be able to identify error handling
  - Be able to troubleshoot web services & SQL (network knowledge)
- System administration
  - Be able to plan IT infrastructure
  - Be able to administer SQL
  - Have knowledge of and be able to administer windows servers (event logs, services, iis, etc.)
Appendix D – Quick Start User Guide

This appendix contains a 7 page excerpt (out of 52 total) of the Quick Start User Guide that comprised my second-most important project during my Bluespring internship. I designed the layout, determined the styles, captured and inserted all screenshots, and composed 90% of the text. My internship mentor, Chris Schapman, composed 10% of the text via feedback suggestions during the review process. Most of his contributions were clarifications or corrections of text that I had written.

As discussed in the report, this user guide did not make it to customers because we decided it contained too much information and therefore was intimidating for the primary audience, which is a brand-new BPM Suite user. Instead is it more of a comprehensive user guide for an experienced user. The amount of information contained in this guide required to introduce a new user to BPM Suite highlighted the need for a “process database” that contains a host of commonly-used, pre-configured processes. With such a database, most of the activity configurations would already be made within each process, allowing a user guide to be more “quick start” in its explanations and less detailed or overwhelming.
How the Process Gathers Information

Reading information from the expense report – Excel Reader Activity 62

Reading Information from a User's Profile 63
How the Process Gathers Information
This section covers:

- **Reading Information from an external document (Excel Reader activity)** – This activity reads information from the expense report and stores the information in Data Items

- **Reading Information from a User’s Profile (Get User Profile activity)** – BPM Suite reads the profile of the worker who submitted the report; records the worker’s email address and the user ID of the worker’s manager, both of which are used by activities later in the process
Reading information from the expense report – Excel Reader Activity

The Excel Reader activity reads specific values from the expense report and saves the values to Data Items.

To access the Excel Reader activity
• Double-click its icon in the process diagram and click the Excel Reader tab in the window that opens.

Configuration of the Excel Reader tab
To finish the configuration of the Excel Reader activity, follow the steps below:
1. Notice how the activity is configured to receive the expense report from the ExpenseReport Data Item (A)
2. Notice how the activity is configured to read values in the Expense Report and store the values in Data Items (B)
3. Configure the activity to read additional values by following the steps below:
   o Click the last cell of the Data Item column (C).
     A drop-down list appears.
   o Using the drop-down list, select EmployeeDepartment
   o In the last field under the Sheet column, enter Expense Statement
   o In the last field under the Cell column, enter E14

What does the Excel Reader Activity Accomplish?
At the beginning of the process, the Excel Reader activity gets the expense report from a BPM file repository and reads values from cells in the document that are specified in the activity. In the example activity shown above, the activity reads the employee’s name, the employee’s user ID, and the total value of the expense report. Also, you configured the activity to read the employee’s department.

Now you are ready to move on to the next step in the process, which is a Get User Profile activity that uses the employee’s user ID that the Excel Reader activity read from the expense report to reference the user’s profile.

Click OK to close the Excel Reader activity window.
The **Get User Profile** activity reads information from a user’s profile that is used later in the process.

**To access the Get User Profile activity**
- Double-click its icon in the process diagram, and click the **Get User Profile** tab in the window that opens.

**Configuration of the Get User Profile tab**
In order to configure the **Get User Profile** activity to retrieve needed information from the profile of the user that submitted the expense report, follow the steps below:

1. Notice how the activity selects a user profile to read by referencing a Data Item containing the User ID of the user who submitted the report (A). The User ID Data Item is populated by the **Excel Reader** activity, which reads the User ID value from the expense report.

2. Notice how the activity selects the Manager User ID profile attribute (B) and assigns it to the Manager ID Data Item (C).

3. Click the second row under the **Profile Attribute** column. A drop-down list appears.

4. Using the drop-down list, select **Email Address**. Text appears in the Data Type column (D).

5. Click the second row under the **Data Item** column. A drop-down list appears.

6. Using the drop-down list, select **Employee Email** (E).

**What Does the Get User Profile Activity Accomplish?**
Instead of making a user enter his/her contact information every time it is needed, the **Get User Profile** activity allows a process to retrieve various types of user contact information without involving the user. The **Get User Profile** activity in the approval process retrieves two pieces of information:

- The user ID of the manager of the worker that submitted the report – which may be used later in the process to assign an approval task
- The email address of the user that submitted the report – which is used at the end of the process to send an email to the user, notifying them of the approval decision

Now you are ready to move on to the next section of the process, in which the process branches down one of three paths according to the evaluation of business rules.

Click **OK** to close the **Get User Profile** activity window.
How the Process Evaluates Business Rules
How the Process Evaluates Business Rules

This section covers:

**Business Rule Controlled Branching Paths** – Once the process places all necessary information into Data Items, the process evaluates the total value of the expense report against two business rules. Those rules are as follows:

- **Requires Manager Approval** - If the total expense report value is over $100, the manager of the worker that submitted the report needs to approve it.

- **Auto Approval** - If the total expense report value is under $100, then the expense report is automatically approved.
Business Rule Controlled Branching Paths

In the approval process, the logical branching paths make the process follow one of three paths based on the evaluation of a specific business rule that checks the total amount in the expense report. In addition to choosing one path among several, you can configure a logical branching path to follow two paths simultaneously. Basically, branching paths provide a means for business logic to control what parts of a process execute.

Now you are ready to move on to the next section of the process, in which human workers are tasked to either approve or deny the expense report via Manual activities. Click OK to close the Link Properties window.

Custom and Pre-Built Business Rules

Before this guide details how human workers can contribute to a process, it is important to note that business rules can be applied in other places besides logical branching paths. BPM Designer has a Logical Expressions window in which you can create custom business rules. Then, you can use the Logical Expression Rules activity to apply the business rules you create to a process.

Finally, BPM Suite comes packaged with InRule, an expansive business logic repository that you can access to take advantage of pre-built business rules.
Appendix E – Functional Requirements

This appendix contains two documents, the functional requirements that I created for the Put File activity and the XSLT activity. These are the final versions of these documents. The feedback I received from my mentor Chris Schapman was hand-written on paper copies of the drafts. I’ve included scanned screenshots of the pertinent feedback in the body of the report. The other feedback I received from developers via a Microsoft Sharepoint discussion board. I did not capture this feedback during my internship and I lost my access to the Sharepoint site as soon as my internship ended. I discuss the nature of the feedback I received and its effect on my draft requirements in the body of this report.
Requirements Document

Project Name: Put File activity
Customer Name: Bluespring
Requirements Contact: Chris Schapman
Document Owner: Nick Waleszonia

Revision History

<table>
<thead>
<tr>
<th>Rev #</th>
<th>Rev Date</th>
<th>Author</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>02/10/06</td>
<td>NWW</td>
<td>Submitted Draft</td>
</tr>
<tr>
<td>1.1</td>
<td>02/15/06</td>
<td>CWS</td>
<td>Reviewed; ready for requirements feedback review</td>
</tr>
</tbody>
</table>

Project Overview & Scope

The goal of this project is to create a Put File activity that will allow users to place one or multiple files at a specified location. A user will enter the destination pathname of the files while configuring the activity and the activity will put them in the last folder of the pathname when it executes. A user will also be able to use BPM to generate all or part of the destination pathname. In this instance, a user will browse for the destination in a folder browser window. Upon selecting a destination, BPM will generate and display the pathname used to get to that location.

If a folder in the pathname does not exist, the user can configure the activity to create folders with names specified in the pathname. A user will also be able to configure the activity to either overwrite files with the same name in the destination folder, or create a similar file with an incremented filename.

Dependencies

None

Deliverables

<table>
<thead>
<tr>
<th>Deliverable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Put File Activity Web Service</td>
</tr>
<tr>
<td>Put File Activity UI</td>
</tr>
<tr>
<td>Total</td>
</tr>
</tbody>
</table>

Terminology

none

Requirements

The following requirements detail the functionality of the Put File activity:

1. Data Item containing file(s) to put

   1.1. The user will select a FilePointer Data Item that contains the file or files that he/she wants to place at a specific location

      1.1.1. The user can only select one FilePointer Data Item. The user selects this Data Item from the FilePointer Data Item containing file(s) drop-down list

      1.1.1.1. The FilePointer Data Item containing file(s) drop-down list only contains FilePointer Data Items and Inherited Data Items.

2. File destination pathname and folder

   2.1. The user will enter the pathname for the destination folder to hold the file or files in the Data Item

      2.1.1. When entering the pathname, the user can use any combination of Data Item Display Values and Data Item Values

      2.1.1.1. The only Data Items that can be used are non-system, non-FilePointer, non-UserID Data Items

      2.1.1.2. The available Data Items are listed in the Data Items drop-down list
Requirements Document

Product Management

2.1.1.3. After selecting a Data Item, the user can click the Insert Display Value button to insert the Display Value of the selected Data Item or click the Insert Value button to insert the selected Data Item’s value.

2.1.1.4. The same Data Item can be used more than once

2.1.1.5. The Data Item is added to the Destination pathname and folder text box at the last known position of the cursor

2.1.1.5.1. If the last known position is unknown (meaning the user has not entered anything in the Destination pathname and folder text box manually, then the Data Item will be added after the last character in the text box

2.1.1.6. Data Items will appear with percent signs (%) around the name of the Data Item, plus a dot/period, and either DisplayValue or Value, depending on if the user clicked Insert Display Value or Insert Value, respectively.

2.1.1.7. The user has to enter the slashes ("\" or ") when necessary

2.1.1.8. Assuming the user knows what he/she is doing, the user can manually enter the %<Data Item XML Tag>%<Value> or <DisplayValue>%

2.1.1.8.1. Manually entered Data Item values are also acceptable, because the Destination pathname and folder will be parsed for Data Items at run-time

2.1.2. The user can browse for the destination folder

2.1.2.1. To browse for the destination folder, the user will click the ellipsis button next to the Destination pathname and folder text box

2.1.2.1.1. The ellipsis button opens the Browse for Folder window (see Figure 2)

2.1.2.1.2. After browsing to and selecting a folder, the user clicks OK to close the window. The folder the user selects appears in the Destination pathname and folder text box

2.1.2.1.3. The user can manually change the pathname in the text box

2.1.2.1.4. Each time the Browse for Folder window opens, it starts at the default location; not the location of the folder previously selected

2.1.2.1.5. The Browse for Folder window does not allow the user to create a new folder at design-time

2.1.2.1.6. The Browse for Folder window can only access folders that the machine on which the Designer is installed can access

2.1.2.1.7. The Browse for Folder window only allows a user to select a folder and will only let the user select a valid folder. For example, the user cannot select My Computer.

2.1.2.2. The destination pathname can start with a drive letter (ex: C:\) or the name of a machine (ex: \servername).

2.1.2.2.1. If a drive letter is used, that drive letter is relative to the machine on which the BPM Engine is running

2.1.2.2.2. The drive letter has no relation to the BPM repository; considering the point of this activity is to get a file out of the repository

2.1.2.2.3. If the Designer is installed on a computer that is different than the server running BPM Engine, and the user selects C:\, then the C:\ will be the server’s C drive, not the C drive on the computer with the Designer

2.1.2.3. The pathname cannot be a URL

2.1.2.3.1. Enabling the functionality to push a file to a URL will be considered as an enhancement for a future version of this activity
2.1.4. Pathname, including the name of the file, is limited to 255 characters at run-time, not at design-time.

2.1.4.1. After replacing Data Item values at run-time, if the pathname is longer than 255 characters, the activity should fail and the log should state that the activity failed because the pathname was longer than 255 characters; the exact character count should also be listed, so the user knows how many characters need to be removed from the name before it will be OK.

2.1.5. The user has the option to create folders at run-time if they do not exist

2.1.5.1. By default, the Create folders if they do not exist check box is checked, which indicates that the activity should create a folder if it does not exist

2.1.5.1.1. This folder can be anything after the drive letter or machine name (ex: C:\ or \servername, respectively)

2.1.5.2. If the user unchecks the Create folders if they do not exist check box, then if the activity does not find a folder at run-time, then the activity will fail. The log should record that the activity failed because a folder could not be found. The name of that folder should be listed.

2.1.6. Whether or not the user is using Data Items, the user can click the Preview Pathname button to open the Preview Pathname window (see Figure 3)

2.1.6.1. When the Preview Pathname window opens, it parses the Destination pathname and folder for any Data Items

2.1.6.2. Data Items are listed in the order they are found in the grid

2.1.6.2.1. If the same Data Item is used more than once in the Destination pathname and folder text box, then the Data Item will only appear once in the list and be listed relative to its first occurrence

2.1.6.2.2. If there are no Data Items in the Destination pathname and folder, then the grid will be empty

2.1.6.3. The pathname does not appear in the Pathname text box until the user clicks the Preview Pathname button on the Preview Pathname window

2.1.6.3.1. The Pathname text box is not enabled.

3. Put method

3.1. When the file or files are placed in the destination folder, BPM checks to see if any files by the same name already exist in the folder

3.2. The Overwrite existing files radio button is selected by default, which indicates that if a file with the same name as a file being placed in the destination folder exists, then that existing file will be overwritten by the new file

3.3. If the other option, Increment file name if the file exists, is selected, then the file name of the new file will be incremented by 1.

3.3.1. Example: If a file in the destination folder is named Customer Statement.doc and a new file is being added called Customer Statement.doc, then that new file will be renamed to Customer Statement-1.doc.

3.3.1.1. If the length of the pathname and file name is close to the 255 character mark, then the addition of the counter, may push the pathname total over 255, which may break the process after it has run 10, 100, 1000, 10000+ times.

4. Save error message to Data Item

4.1. Optionally, the user can save any error messages returned by the machine to which the file(s) are being placed to a Data Item

4.1.1. The error message will also appear in the logs, but saving the message to a Data Item allows for additional branching options and gives the user a way to notify someone of a specific issue with the process
4.1.2. Possible error messages include Directory Not Found (only returned if the Create folders if they do not exist check box is not checked)

4.1.3. An error message is only saved to the Data Item if the activity fails

4.2. If the user checks the Save error message to Data Item check box/group box label, then the Data Item drop-down list is enabled

4.2.1. By default, the Save error message to Data Item check box/group box label is not checked

4.3. If the user chooses to Save error message to Data Item option, then the user mush select a Data Item from the Data Item drop-down list.

4.3.1. If the user does not select a Data Item from the list, then the Activity will not be fully configured and the process cannot be placed in either of the Active modes.

4.4. The drop-down list of Data Items only contains non-system, non-FilePointer, non-UserID Data Items.

5. Error logging

5.1. To help error logging, the following information should be logged in the log file:

5.1.1. Destination pathname before Data Items have been replaced

5.1.2. Destination pathname after Data Items have been replaced

5.1.3. Number of files uploaded

5.1.4. Final name of each file uploaded; including incremented files

5.1.5. What files were overwritten, if any were overwritten

5.1.6. Name of any folders created during the upload

Attachments

- Compiled OCX
- VB6 project file available.
Properties for Activity: New Put File

Data Item containing file(s) to put
Select the FilePointer Data Item containing the file to be placed in the destination folder. If the Data Item contains multiple instances, each file will be placed in the destination folder.

Data Item:

File destination pathname and folder
Enter the pathname and folder to which the file or files in the selected FilePointer Data Item will be saved.

Data Items:

Destination pathname and folder

Create folders if they do not exist?

Put method
Select whether to overwrite existing files or increment the name of the new files.

- Overwrite existing file
- Increment new file name if file exists

Save error message to Data Item
Optionally, select a Data Item to store any error message returned when the activity runs.

Data Item:
Figure 2 (Browse for Folder window)

Please select the destination folder for the file(s) stored in the selected FilePointer Data Item.

- Desktop
  - My Documents
  - My Computer
    - Local Disk (C:)
    - DVD Drive (D:)
    - Christopher.Schapman on 'yorkie.bluessw.ni'
  - My Network (BLUSSW)
  - temp

Figure 3 (Preview Destination window)

Test values for preview:

Pathname: [input field]

Preview Pathname

OK
Use Cases

1. User wants to place a file or files into a specific folder location. User identifies the files he/she wants to place and the location at which he/she wants to place them. User selects Filepointer Data Item that will contain the file(s) to be placed. Then the user:
   a. Enters destination pathname manually
   b. Builds all or part of destination pathname using Data Item placeholders
   c. Builds all or part of destination pathname using Browse for Folder window (which inserts pathname of selected location into Destination field)
   d. Enters destination pathname using a combination of manually entering values, inserting Data Item placeholders, and/or browsing for destination in Browse for Folder window

2. A user can copy multiple files to one folder location by ensuring that the Filepointer Data Item he/she she uses is a multi-instance Data Item.

3. If a user wants to copy one or more files to multiple locations, then he/she should use multiple Put File activities (i.e. one for each destination).

4. The user can preview the destination pathname to see if the Data Items will build it correctly.

5. User wants to place a file or files into a specific folder location, but he/she does not want to overwrite previous versions of files that may already be in the destination folder. User should configure the activity using one of the methods in Use Case 1, and then select the Rename put files radio button under the Put method group. As a result, any files being placed with the same name and file extension of files in the folder will be renamed with an incremented file name.

6. User wants to ensure that files are only placed in pre-existing folders. User should configure the activity using one of the methods in Use Case 1, and then uncheck the Create folders that do not exist? checkbox. As a result, the activity will only place files at a destination that matches the exact pathname entered. By using this option, a user can ensure that any misspelled folder names are not created. Also, this option causes errors in pathnames to be caught, since the activity fails if BPM cannot find the pathname.
**Requirements Document**

**Project Name:** XSLT Requirements  
**Customer Name:** Bluespring  
**Requirements Contact:** Chris Schapman, Chris Counselman  
**Document Owner:** Nick Waleszonia

---

**Revision History**

<table>
<thead>
<tr>
<th>Rev #</th>
<th>Rev Date</th>
<th>Author</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>3/10/06</td>
<td>NWW</td>
<td>Started Draft</td>
</tr>
<tr>
<td>2.0</td>
<td>3/20/06</td>
<td>NWW</td>
<td>Incorporated comments from Chris S.</td>
</tr>
<tr>
<td>3.0</td>
<td>3/29/06</td>
<td>NWW</td>
<td>Incorporated feedback from Sharepoint discussion</td>
</tr>
</tbody>
</table>

---

**Project Overview & Scope**

The goal of this project is to create a XSLT activity that applies an XSL style sheet to an XML document, transforming it into a document that follows the styles in the XSL style sheet. The output file can be a XML (.xml), HTML (.html), or text(.txt) document (but these three can reference a large variety of documents like PDFs or SVGs)

The user can configure the activity to receive an XSL style sheet at run-time, or he/she can create or load an existing XSL style sheet at design-time.

To properly configure this activity, a user must be familiar with XML, XLST, and how XSLT style sheets transform XML documents.

---

**Deliverables**

<table>
<thead>
<tr>
<th>Deliverable</th>
<th>Effort (in Days)</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSLT activity UI</td>
<td></td>
</tr>
<tr>
<td>XSLT web-service</td>
<td></td>
</tr>
<tr>
<td>Documentation</td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
</tr>
</tbody>
</table>

---

**Terminology**

XML – A general-purpose markup language designed to facilitate the transfer of data across different systems  
XSL – A family of languages that describe how XML should be formatted or transformed. XSLT is a subset of XSL.

**Requirements**

This activity should transform an XML document by applying styles from an XSLT document. The XML document is determined at run-time while the XSLT document can be determined at run-time or selected/created by the user at design-time.

The following requirements detail the UI functionality for this feature:

**Source XML**

The user should select a Data Item that will contain the XML to be transformed.

1. **Loading an XML document from a Data Item** *(See Figure 1)*

   1.1. From the Data Item containing source XML drop-down list, the user should select a Data Item that will contain XML or an XML file at run-time

      1.1.1. All non-system Data Items should appear in the drop-down list in ascending alphabetical order (0 - 9, A – Z)

      1.1.2. The activity should determine the Data Type of the Data Item at run-time and load the XML accordingly

         1.1.2.1. If the Data Item is a FilePointer Data Type, then the activity should load the appropriate XML file from the file repository

         1.1.2.2. If the Data Item is a Data Type other than FilePointer, then the activity should load the XML contained inside the Data Item
Transform XML using XSLT style sheet
The user can retrieve the XSLT style sheet in one of three ways: loading the style sheet from a Data Item at run-time, importing an existing style sheet at design-time, or creating a style sheet at design-time

2. Loading the style sheet from a Data Item at run-time (See Figure 1)

2.1. The user should leave the Select a style sheet contained in a Data Item default radio button selected; this radio button disables the Define style sheet now text field, the Import Style Sheet button, and the Launch Editor… button.

2.2. Using the Style sheet contained in a Data Item drop-down list, the user should select a Data Item that will contain an XSLT style sheet

2.3. All non-system Data Items appear in the drop-down list

2.4. The activity should determine the Data Type of the Data Item at run-time and load the style-sheet accordingly

2.4.1. If the Data Item is a FilePointer Data Type, then the activity should load the appropriate XSLT style sheet file from the file repository

2.4.2. If the Data Item is a Data Type other than FilePointer, then the activity should load the style sheet contained inside the Data Item

2.5. The activity should read the default value of the Data Item; if the Data Item is multi-instance then the activity should fail, because multi-instance Data Items do not have a default value

3. Creating a new style sheet (See Figure 1)

3.1. The user should select the Define style sheet now radio button

3.2. Enables the large text field below the Define style sheet now radio button, the Import Style Sheet button, and the Launch Editor button; disables the Style sheet contained in a Data Item drop-down list

3.3. The Define Style sheet now text field is writable; the user should be able to enter values into it

3.4. By default the XML large text field is blank

3.5. The user should create a style sheet by typing additional XML into the large text field

4. Importing an existing style sheet (See Figure 1)

4.1. The Import Style Sheet button opens the Browse for File window (See Figure 2)

4.1.1. Using the Browse for File window, the user should browse the accessible file system for an XSLT style sheet

4.1.2. The Browse for File works like the standard Windows file browser window

4.1.3. Once the user selects and XSLT style sheet and loads it, the XML in the style sheet should appear in the Define style sheet now text field; the default XML in the text field should be cleared before the new XML is loaded

4.1.3.1. The scroll bars should adjust to the size of the style sheet in the text field, so that the user can view the entire style sheet by using the scroll bars

4.1.3.2. The user should be able to edit the loaded style sheet by typing directly into the large text field

5. XSLT Style Sheet Editor (See Figure 3)

5.1. The Launch Editor… button opens the XSLT Style Sheet – Editor window

5.2. The XSLT Style Sheet – Editor window should be:
• Sizable – the user should be able to resize, or maximize the window. The user should not be able to minimize the window.
• Modal – the user should not be able to access any other windows while the XSLT Style Sheet – Editor is open.

5.3. When the XSLT Style Sheet – Editor opens, the XML that is in the large text field in the main activity window should appear in the XSLT style sheet field in the XSLT Style Sheet – Editor window.

5.4. The user should be able to copy, cut, and paste text into, or out of, the Define style sheet now text field.

5.4.1. Right-clicking the Define style sheet now text field should bring up an editor menu (Figure 4).

5.5. The user can insert Data Item tokens, both Display Values or Values, into the XSLT style sheet field.

5.5.1. Using the available Data Items drop-down list, the user should select a Data Item to insert into the style sheet.

5.5.1.1. The drop-down list should display all Data Item Types except FilePointer.

5.5.2. After selecting a Data Item, the user should click either the Insert Display Value or Insert Value button.

5.5.2.1. The Insert Display Value button should insert the Display Value token of the selected Data Item into the XSLT style sheet field; the Insert Value button should insert the Value token of the selected Data Item into the XSLT style sheet field.

5.5.2.2. The Data Item token should be inserted at the cursor’s position in the XSLT style sheet field.

5.5.2.3. If the cursor’s position is unknown, then the Data Item token should be inserted at the top-left position in the XSLT style sheet field.

5.6. Clicking OK closes the XSLT Style Sheet – Editor window and transfers the XML the user entered into the large text field into the main activity window.

6. Testing the XML transformation (See Figure 5)

6.1. The Test Transform… button opens the Test XML Transformation window, a modal, fixed-sized window.

6.2. The Test XML Transformation window should be a modal, fixed window.

6.3. The Load Sample XML… button opens up the Browse for File window (See Figure 2).

6.3.1. The user should browse for a sample XML document to transform.

6.3.2. After selecting a document and loading it, the XML in the document should load into the Sample XML text field.

6.3.2.1. The Sample XML text field is writable; the user should be able to enter values directly into it.

6.4. The Load Style Sheet button can perform one of two actions.

6.4.1. If the user selected the Style sheet contained in a Data Item option in the Properties for Activity window, then a Browse for File window opens and the user must browse for an XSLT style sheet file (Figure 2).

6.4.1.1. After selecting and loading an XSLT style sheet in the Browse for File window, the style sheet should load into the XSLT style sheet text field.

6.4.2. If the user selected the Define style sheet now option in the Properties for Activity window, then text in the Define style sheet now text field (in the Properties for Activity window) should load into the XSLT style sheet text field.

6.5. The Transform XML button transforms the XML in the Sample XML text field using the style sheet in the XSLT style sheet text field and displays it in the Transformed XML text field.

6.5.1. If the style sheet contains Data Item tokens, the Expected Data Item Values window opens (Figure 6).

6.5.2. Within the Expected Data Item Values window, the Data Items in style sheet table has two columns:
6.5.2.1. Data Item – Displays the name of each Data Item in the style sheet; displays Data Items in the order that they appear in the style sheet

6.5.2.2. Value – Provides a field for the user to enter an expected value for each Data Item

6.5.3. The Run Transformation button closes the Expected Data Item Values button, replaces the Data Item tokens in the style sheet with the user entered values, and runs the XML transformation

6.5.4. The Cancel button closes the Expected Data Item Values window and does not run the XML transformation

6.6. The Close button closes the Test XML Transformation window

7. At run-time, the activity will load the XSLT XML and transform the source XML document according to the styles in the loaded XSLT document

Output destination (See Figure 1)
The user must select a Data Item to contain the result of the XML transformation

7.1. Using the Data Item drop-down list, the user should select a Data Item to hold the result of the XML transformation

7.1.1. All non-system Data Items appear in the drop-down list

7.1.2. If the user picks a non-FilePointer Data Item, then the output XML will be stored in the Data Item as a display value

7.2. If the selected output Data Item is a FilePointer, then the user must set the name of the output file

7.2.1. The Output file name text field and Edit File Name... button become enabled; both buttons are disabled by default

7.2.2. The user should enter a file name into the Output File Name field

7.2.3. The Edit File Name... button opens the Configure Output File Name window, which should be a fixed, modal window (See Figure 7)

7.2.3.1. The Configure Output File Name window allows a user to add Data Items to the output file name

7.2.3.2. The user does not have to use the Configure Output File Name window, he/she can enter a name into the Output File Name field.

7.2.3.3. The user does not have to determine a file extension, since the XSLT style sheet determines that in a output method declaration

7.2.3.4. The output file can be an XML, HTML, or text format

7.2.4. For a description of the functionality of the Configure Output File Name window, please refer to the Excel Writer requirements for BPM Designer 4.3, page 4

7.2.5. If the process is started without a file name being set in this activity, then the activity should fail with the reason "Output File Name not set" written to the error log

8. Error checking

8.1. The activity should record any errors that occur while:

8.1.1. Loading or parsing the source XML

8.1.2. Loading or parsing the style sheet

8.1.3. Transforming the source XML using the style sheet
**Properties for Activity: XSLT**

**General**
- Source XML: 
- Data Item containing source XML: 

**XSLT**
- Transform XML using XSLT style sheet:
  - Style sheet contained in a Data Item: 
  - Define style sheet now:

```xml
<xsl:stylesheet>
  <xsl:template match="*"/>
</xsl:template>
</xsl:stylesheet>
```

**Error Handling**
- 

**Related Activities**
- 

**Display**
- 

---

**Output for transformed XML**
- Select a Data Item that will store the result of the style sheet transformation. If saving the output as a file in a FilePointer Data Item, enter a name for the file.

- Data Item: 
- Output file name: 

---
Figure 2 (Browse for a XSLT Style Sheet window – window should not be sizable)

Figure 3 (Create a New or Use an Existing XSLT Style Sheet window)
Figure 4 (Right-Click Style Sheet Options Menu)

- Insert Value
- Insert Display Value
- Undo
- Cut
- Copy
- Paste
- Delete
- Select All

Figure 5 (Test XML Transformation window)
Figure 6 (Expected Data Item Values window)
Use Cases
1. A user wants to transform an XML document according to style in an XSLT style sheet.
2. The user selects whether an entire XML document in a FilePointer Data Item or XML text in a Data Item will be transformed.
3. The user selects an XSLT style sheet using one of the following four methods:
   3.1. The user selects a Filepointer Data Item that will contain the XSLT style sheet at run-time
   3.2. The user loads an existing XSLT style sheet into the activity at design-time
   3.3. The user creates an XSLT style sheet at design-time
   3.4. The user loads an existing XSLT style sheet and modifies it at design-time
4. The user selects a Data Item that will contain the transformed XML