Chapter 1

Business, Functional, and Technical Requirements

The goal of this chapter is to help you address and answer questions from the people around you in the form of a common language. *Requirements* are essential for implementing Exchange 2013 successfully.

Exchange can be a daunting product to contemplate, with over 20 million lines of code. There are many, many features from which to choose, though some have not changed significantly between Exchange versions (address book generation, for example). Furthermore, there's a discussion of how these features are to be implemented and the entire best practices conversation, which comes with the territory. How do you choose which features to implement and which to leave behind? The answer is requirements. And how do you decide which best practice to apply? Again, the answer is requirements.

Building the Foundation for Requirements

Requirements elicitation can sometimes be seen as boring, tedious, and overly complex. This perception can often derail this most critical part of a new project. Requirements elicitation is traditionally associated with software engineering, which implies a long list of requirements to satisfy the discipline of creating or modifying software. With the exception of writing scripts, most administrators who wish to implement Exchange don't need to know or care about the difference between a functional and a business requirement, since they're not creating software from scratch. However, we still need to capture the essence of "why" we are taking certain actions, as well as the "what" and the "how" we are doing them.

This chapter is particularly important for the Exchange administrator or consultant who may have been tasked with installing, upgrading, or migrating to Exchange for the first time in a formal manner and who doesn't know where to start. Even if you have successfully implemented Exchange, this chapter will still be of tremendous value to you if this is the first time that you are the one documenting a design.

Requirements are the core of an Exchange project. Based on the requirements, a host of other documentation items can be affected. These may include the following:

- Vision and scope document
- The Exchange 2013 design
- Testing plan
- Migration plan
- The bill of materials required to implement Exchange

- Test case documentation
- Adjustments to the disaster recovery plan (DRP)

A good place to start is to learn how to identify and document requirements correctly and with enough detail to satisfy people from different parts of IT and within the business as a whole. A bad place to begin is by installing Exchange on the basis of a diagram only. Since we are in IT, we often start with a diagram of something and then wind up making design changes on the fly.

Documenting requirements is thus a critical part of the design process, as we will explore later in this chapter. In summary, this chapter will equip you with the tools to ensure that "why," "what," and "how" are addressed and documented adequately, without requiring a degree in software engineering in order to do so.

Establishing Project Roles

Establishing roles at the outset of a project of significant magnitude is critically important. Most of you reading this book fall into one of two camps:

Camp 1: You are a highly respected individual with demonstrated competence in your field. You tend to be external to the companies you consult with, even if you are part of a multi-year project or outsource contract.

Camp 2: You are responsible for messaging within your company, or you may be a project manager who needs background reading for your first Exchange project. You have just been tasked with delivering an implementation of the newest version of Exchange.

Whichever camp you fall into, either external or internal to the organization, your attitude about such a project determines its success or failure. Your mindset needs to be that of a consultant toward a client. If you work inside the company, this may be harder to adopt; however, the methods within this chapter work equally well with either camp.

Getting Started with the Exchange Design

Often we are asked to review a design, which may be technically brilliant, architecturally sound, and mindful of the newest features introduced in the latest service pack. However, designs are very often written without capturing the essence—or even the reason for the existence—of the design. The essence or the reason for existence for an Exchange design is documented by capturing the requirements.

Exchange designs and implementations are often driven by a version's new features list, as opposed to having captured and wrestled with requirements and extrapolated the features necessary to satisfy the needs of the business. Having a solid set of requirements to work against, among other reasons, makes your technology choices defensible, since designing or building Exchange without a solid set of requirements is like going food shopping without a list and putting anything that suits you into the shopping cart.

Documenting requirements also makes a document much easier to review. A structured document may have one section summarizing the security requirements and then a separate section encompassing the technical detail on how those security requirements are manifested as configuration detail or product features. This would allow the security or compliance officer to sign off on that portion of the document without having to wade through the storage design, unless of course the storage design captured a secure requirement.

When reviewing designs, we often see that the author has discussed the technology and then made a statement that a feature or technology should be implemented in a certain way. For example, the author may wish to implement Database Availability Groups (DAGs), which were introduced with Exchange 2010, and allow databases to be highly available. The author may say something like, "DAGs are great, so we're going to implement a DAG." However, the "why" of implementing a DAG is not captured. The question of why you are recommending the implementation of DAGs and which requirement you are fulfilling must be answered. Your reasons for choosing technology should always be clearly documented.

Most designs that we reviewed have little or no longevity. That is, if you were to review your own design in five years' time, would you understand the "why" of the design? Why did you choose to implement a DAG in the manner that you did, and so forth? Keep this in mind when eliciting requirements and as a continual thread throughout your document. When your design document is reviewed, arguing that the reasons behind the design were not enumerated because you didn't think it through or were ignorant of other options available at the time is not adequate for explaining why the "why" part of documentation is missing.

Requirements as Part of a Larger Framework

If you are a member of a consulting organization, then you will be quite familiar with the following. If you are doing this for the first time, then this section should be considered a primer as to where requirements fit into a larger framework.

There are several available methodologies from which to choose, including the Microsoft Solutions Framework. Irrespective of which methodology you choose, the steps are often quite similar:

Envisioning Phase This is the "thinking phase" of the project where you and others work to blue sky the project. In its simplest form, this involves the critical people considering the version of Exchange to be implemented and addressing such issues as *how* things are being done today and *why* they need to be done differently.

Requirements Definition Phase This phase captures the requirements, which is the focus of this chapter.

Design Phase This phase molds the requirements into something deployable and practical.

Testing Phase This is a standard phase of larger projects, and it is based entirely on the defined requirements. If the requirements are clear, the tests can be written well before the design is complete.

Deployment Phase This phase implements the design.

The first three phases will naturally generate a document set, which at minimum is similar to the following:

Vision and Scope Document This document captures the reason for the existence of the project, and it defines the business's vision of the technology to be implemented. It also defines what is in and out of scope.

Functional or Technical Specification Document This can be separate from the design document. It defines the business requirements and lists the derived functional or technical requirements. Often, the consultant will use the scoping meeting to document the basic scope of the project and then try to derive the business and functional or technical

requirements for further clarification. We will cover this process in much greater depth shortly.

Design Document This document captures the resulting design.

Depending on the methodology selected, many other documents, such as testing plans, will also be expected.

If you are doing this for the first time, the level of detail required for a large project may overwhelm you. Nevertheless, as a consultant, it is likely that you are indeed working with this level of detail. While there is a wealth of material available about the available methodologies, the basic problem that requirements are often captured poorly, or not at all, remains.

Understanding the Types of Requirements

Classical requirements elicitation is a very deep and mysterious discipline, unless you're a business analyst, systems analyst, or the like. The aim of this chapter is not to address classical requirements elicitation from a systems analysis point of view, since that would only help you become a better systems analyst. The goal of this chapter is to determine what kinds of requirements are important for your Exchange design and to give you the ammunition you need to defend your technology choices.

You may be tempted to wrestle with the nuances among some of the more esoteric requirements. At minimum, however, you need to examine the business and technical requirements. We will take a moment to define each of these later.

The purists among you may question why we don't break technical requirements out into functional and nonfunctional requirements. *Functional requirements* describe what a system is required to do, while *nonfunctional requirements* describe how the system behaves. As mentioned at the beginning, this chapter is focused on eliciting the requirements necessary to implement existing software, not on writing software from the ground up.

If your project is small, the lines between functional, nonfunctional, and business and technical requirements may blur and add unnecessary complexity. Unless you find a compelling reason to include functional and nonfunctional requirements, we suggest that you focus exclusively on the business and technical requirements. Once your project reaches a certain level of complexity, however, you will need to define technical requirements in much more detail. Thus, you will then break out the functional and nonfunctional requirements aspects of the project.

Business requirements may also be captured separately in a vision document. Consulting organizations will be familiar with this procedure, and they will require completion of a specific document set in order to capture this level of detail.

You may argue that you have been given a requirement, and that sounds something like, "We need to upgrade." This statement is, in itself, only half a requirement, and it is an insufficient rationale for a business today to invest in your project. Now let's examine our two chosen requirements in more detail.

Business Requirements

In this section, we are going to discuss business requirements in the context of our upcoming Exchange implementation. This is the "why" part of your requirements. The project sponsor or management team member provides the business drivers for the project. You want to be sure that you don't get stuck in "analysis paralysis," since business requirements tend to be broad statements lacking the detail expected in technical requirements.

For our purposes, businesses tend to have a few simple drivers. These tend to fall into the category of decreasing costs, increasing/retaining revenue, or decreasing risk. A good set of business requirements should address all of these drivers if possible. You may not often be in the position of being able to drive sales up by, say, 40 percent, but you are certainly able to reduce risk by implementing a well-thought-out high-availability strategy for email, if email is a critical business function.

Let's look at a few examples of business requirements:

- Revenue requirement: A company may choose to implement a mobility app to increase productivity of its sales staff.
- Risk requirement: A company needs to protect itself from a failed audit because of a lack of support for its existing version of Exchange.
- Risk requirement: A company has made a strategic decision to migrate their technology base from Lotus Notes and Domino to Microsoft Exchange and Microsoft SharePoint due to in-house development moving to .NET based languages.

We will examine business requirements in the context of a sample customer, *XYZ Bank*.

Business Requirements for a Sample Exchange Upgrade

XYZ Bank has retained you to design and implement a replacement for its aging messaging system. XYZ's current messaging system is implemented in Exchange 2003, which was state of the art at the time. XYZ has an extensive branch network, with many individual Exchange 2003 servers across the country.

When it was implemented, email was not considered to be a critical application. XYZ is growing fast, adding a branch every two to three months. XYZ has also made it known that it intends to list itself on the stock exchange, which will subject XYZ to regular security and process audits.

The bank has recently decided to use email as one of the primary tools for communicating with its customers. XYZ currently limits mailboxes to a few hundred MBs. Because of this limit, employees are forced to move email data to PST files on desktops and file shares, exposing XYZ to risk from theft and corruption. Even with stringent restrictions, a number of branches are complaining that email performance is decreasing, even though the number of users on the respective servers remains the same.

Because of its critical nature, XYZ would like email to be centralized in a datacenter alongside its existing critical banking applications, with a similar level of redundancy and availability. Of course, XYZ is concerned about cost, and it wishes to explore a number of storage options before committing to purchasing a new Storage Area Network (SAN) solely for Exchange's use. Finally, XYZ has stated that it would like similar messaging functionality as provided by Exchange 2003 on day one of implementation while reserving the right to add features in the future.

This story is quite typical. It includes a mixture of requirements, including a clue that the bank is researching later versions of Exchange and that it is aware that several storage options are available. This is an indication that the bank has a number of well-defined feature-based requests.

We can discern a number of business-specific requests from this scenario:

- Replace the unsupported Microsoft Exchange 2003 platform with a currently supported Exchange 2013 environment.
- Increase the availability of the email environment to match the XYZ bank standard.
- Design for future growth.
- Allow for auditing administrative activity with the ability to demonstrate such processes.

Notice that the requirements are broad and contain little technical detail. The business requirements captured as part of a design along with an executive summary, for example, allow management and key staff to assimilate quickly the reasons why the Exchange upgrade is being deployed without getting bogged down in technical detail. In our example, XYZ concentrates on mitigating risk (support, availability, and auditing), while also supporting the stated goals of the business (expansion and increased communication).

Technical Requirements

Staying with the theme of combining technical requirements with functional and nonfunctional requirements, they are quite different from business requirements. Technical requirements are the "what" and "how" parts of requirements. Furthermore, business requirements are written as broad statements, while technical requirements are designed to be precise statements. Technical requirements should be simple lists that are both individual and granular. One of the biggest causes of confusion for implementers is ambiguity in technical specifications. Adding too much explanation within the requirements can cloud the specification.

When dealing with a product like Microsoft Exchange, we take a lot of functionality for granted, and so we should. However, there is a fine line to walk in terms of writing specifications. Let's take, for example, the last line from our XYZ Bank scenario:

> *Finally, XYZ has stated that it would like similar messaging functionality as provided by Exchange 2003 on day one of implementation while reserving the right to add features in the future.*

We could take for granted how to interpret this statement. However, it is actually quite subjective and needs clarification. For example, taken alone, we do not know who or what XYZ is, what "similar messaging functionality as provided by Exchange 2003" means, and what types of features could be added in the future. There are two possible paths of interpretations of this statement: One is broad interpretation based on our knowledge of Microsoft Exchange, and the other is "analysis paralysis."

An example of analysis paralysis would be to specify Exchange functionality as follows:

- A user must be able to generate an email.
- A user must be able to display the contents of an email.
- A user must be able to share their calendar with another user.
- Calendar sharing must support granular rights.

This would be a massive book on its own. What we want to do is to clarify what "similar messaging functionality" means and to specify it. This may include the following:

- Sending and receiving of email internal to the organization
- Sending and receiving of Internet mail
- Rich client access using Outlook 2013
- Thin client access using Outlook Web App (OWA)
- Mobile client access using Exchange ActiveSync (EAS)-based compatible devices

New systems require signoff or acceptance testing criteria to be fulfilled. Each technical specification should be capable of being tested and proved or disproved. Your technical and/or your functional specification should translate easily into testing criteria, which will allow for a relatively easy signoff. In other words, you should be able to write a test plan based on your technical or functional specifications.

If you need to break out technical requirements into a separate document listing functional and nonfunctional requirements, consider the following example.

Based on the following business requirement, we are able to infer these functional and nonfunctional requirements:

> *Design for future growth in mind*
>
> **Functional Requirement**
>
> Hub transport server queues must be located in a separate storage area from the system volume so that growing mail flow volume will not overwhelm the OS drive. (We could list many other functional requirements that pertain to the scalability of the system.)
>
> **Nonfunctional Requirement**
>
> Infrastructure supporting email services should be designed to meet the XYZ Bank's forecasted growth of 20 percent a year.

Constraints

A *constraint* in a design is a non-negotiable item, which has been specified in advance or required by the project. For example, you have a requirement to do X but are constrained by factor Y. Constraints have a direct bearing on the project and may have a significant impact on the final result. Constraints should be listed as a separate heading in the requirements section of your document.

Some constraints are economic, for example, when the customer has already purchased and installed the new hardware without knowing if it will fit the project's ultimate requirements. Another constraint can occur when the customer has specified that Exchange must utilize an existing investment in virtualization or storage. Other constraints may be time-based; that is, the project must be completed within the financial year, or before the change freeze period around a given holiday, and so forth.

Whatever the constraints, make sure that they are documented in your requirements so that you may reference them later in your design. Depending on the nature of the business, security, risk, and compliance may pose significant constraints for a new project.

Assumptions

We often see assumptions listed in documentation. However, assumptions have no place in your documentation. When you review the documentation, endeavor to clarify such assumptions as facts and then list them as either project requirements or constraints.

Requirements Elicitation

Now let's discuss how to get requirements and who creates them. Notice that we use the term *requirements elicitation* and not *requirements gathering*. Gathering implies that requirements are easy to find and include in your documentation. More often than not this is not the case. Requirements elicitation is a much clearer description of what you're trying to achieve. Elicit means "to draw out," which is much closer to how requirements are brought forth, that is, through interactions with teams and individuals.

During this phase, you need to manage constantly the fine balance between assumption and fact. This applies as much to you, the consultant, as to the rest of the project group. You may or may not have been briefed before you joined the project. However, more often than not, as the consultant you may have several assumptions that, if left unspoken, will filter into the design. Your assumptions, and the assumptions of the assembled group, must be verified as facts in order to be considered valid requirements.

High availability is a classic example of the difference between an assumption and a fact. Someone may state, "We want 99.9 percent availability." Your assumption might be 99.9 percent availability during work hours, not including scheduled downtimes. Their assumption might be 99.9 percent availability on a 24/7/365 scale. Your job is to take the "We want 99.9 percent availability" statement and eliminate any ambiguity immediately by eliciting how that availability is measured and then update the statement. For example, "We want 99.9 percent availability on a 24/7/365 basis, not to include any scheduled downtime." If this request sounds unreasonable or implausible, then part of your role is to educate the group as to why this is not feasible and drive consensus on what is stated in the final requirement.

Requirements Elicitation and the Long Tail of Obsolete Best Practices

Exchange has many features that can be implemented in many ways. The subset of the best ways to implement Exchange is known as "best practice." Best practice may be specific to a particular version of Exchange. For example, Exchange 2003 and earlier mandated that data should be stored on tier one storage, that is, fast disks and a redundant storage array of some sort.

To this day, we must often begin a storage discussion by dispelling this notion and educating our audience about how dramatically Exchange has changed, often using the Mailbox Server Role Requirements Calculator spreadsheet. Storage best practices as well as many others have evolved, but it is very tempting to reference obsolete best practices as your base model when thinking about newer versions of Exchange. Best practices, obsolete or not, may fall into the category of assumptions, which must be transformed into facts before they can form part of a design.

Summary

As you read through the subsequent chapters in this book, you will be reminded that Exchange is a feature-rich and exciting product. However, without clearly defining the requirements—that is, the "what," "where," and "why"—your Exchange implementation will likely not deliver the results you hoped for. Learn to document the reasons for your choices at all times, make your designs defensible and justifiable, and know why you wrote what you did when you review your design document again in the future.