Implementing ITIL Change and Release Management

Larry Klosterboer

The ITIL Series

IBM Press Program Managers: Tara Woodman, Ellice Uffer

Cover design: IBM Corporation

Associate Publisher: Greg Wiegand
Marketing Manager: Kourtnaye Sturgeon
Acquisitions Editor: Katherine Bull
Publicist: Heather Fox
Development Editor: Julie Bess
Managing Editor: Kristy Hart
Designer: Alan Clements
Project Editor: Jovana San Nicolas-Shirley
Copy Editor: Gayle Johnson
Indexer: Lisa Stumpf
Compositor: TnT Design
Proofreader: Water Crest Publishing
Manufacturing Buyer: Dan Uhrig

Published by Pearson plc

Publishing as IBM Press

IBM Press offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
1-800-382-3419
corpsales@pearsontechgroup.com

For sales outside the U.S., please contact:

International Sales
international@pearsoned.com

Library of Congress Cataloging-in-Publication Data

Klosterboer, Larry.

Implementing ITIL change and release management / Larry Klosterboer.

p. cm.

ISBN 978-0-13-815041-9

1. Configuration management. 2. Information technology—Management. I. Title. II. Title: Implementing Information Technology Infrastructure Library change and release management.

QA76.76.C69K65 2008
004.068’8—dc22

2008040421

ISBN-13: 978-0-13-815041-9
ISBN-10: 0-13-815041-9

Text printed in the United States on recycled paper at R.R. Donnelley in Crawfordsville, Indiana.

First printing December 2008

CHAPTER 3

Defining Change and Release Management Processes

The heart of ITIL is in processes and the disciplines surrounding them. Thus, it should be no surprise that early in the planning for change and release management, you need to begin defining processes. Although version 3 of the IT Infrastructure Library defines the processes in more detail than any previous version, it still doesn't describe exactly which process every organization should use. This is because one of the best practices is that each organization should define its process somewhat differently.

This chapter starts with a generic description of how to define any process at four levels. After you understand the basics, the ITIL suggested high-level processes are introduced, followed by specific suggestions about how to customize your own change and release management processes.

How to Define a Process

Like most specialties, process engineering uses its own vocabulary. The words aren't complex or difficult, but they are used in a specific context and must be understood that way. To work effectively with process engineers, you must know this vocabulary. This section introduces the important terms and concepts used to define a process. Figure 3.1 shows the steps required to build out a process.

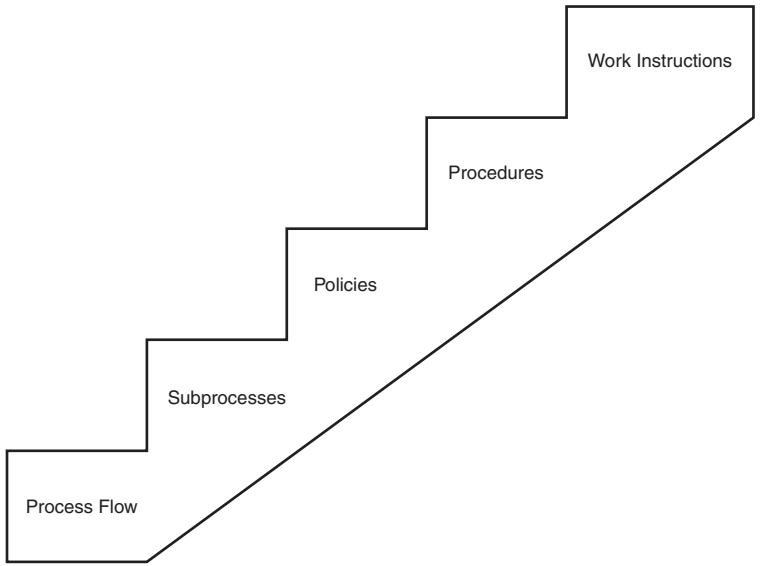

Figure 3.1 Five basic steps are needed to create a process.

Start with Process Flow

At the onset of defining your process, you need to grapple with what is meant by a *process*. Although more technical definitions are available, the working definition of a process is a sequence of defined actions that produces a measurable and desirable outcome. Because a process is a sequence, the place to start with definition normally is a flow diagram of some sort. Whether you favor a "line of visibility" diagram that delineates the different roles in the process or a simple flowchart that captures only the steps, your organization should adopt a standard way of defining a process flow. Look around for other process documents, and copy whatever style is in use for your organization.

Creating a workable process flow requires a solid starting point. Fortunately, the ITIL service transition book provides a solid start. In Chapter 4 of that book, you'll find some excellent sample process flows that can serve as a starting point. If your organization already has change management and/or release management processes documented, those can also serve as good reference points.

After establishing a starting point flow, the next step is to look back at the requirements you documented. Look very closely at the process requirements, and see if any of them dictate that you change your starting flow in specific ways. It would be unlikely at this high level, but occasionally a requirement will cause you to add a step to an overall flow.

Be sure not to get too detailed in your high-level flow. A good rule of thumb is that the top level should fit on a single page. Most of the process requirements will be around the details rather than the high-level process. Those details will be worked out eventually, but you have the

greatest chance of success if you start with a single sheet that all your stakeholders can agree is the top level of the process.

After you have used existing starting points and your own judgment to define a high-level process flow, it is time to validate it. Take the single sheet to your sponsor, your stakeholders, and the project team. Get their ideas and incorporate them. Be sure to keep everyone aligned with the scope of your project. This is not a second round of requirements gathering, but an attempt to meet the requirements and scope you have already defined. It is important, however, that the high-level flow meets everyone's understanding of the project scope, because you are about to base many hours of work on this single page.

Frequently one or more activities on the high-level flow will be worthy of a separate flow by themselves. In the language of the process engineer, these are subprocesses. Identify these subprocesses as you validate the high-level flow, and work with the stakeholders and project team members to build these flows as well. As before, constrain each to a single sheet of paper, and focus on getting the flow documented in a consistent format. The subprocesses do not need to be especially detailed at this point, but they should provide enough information to allow later definition of the details.

As soon as all the flows are complete, you have the basic outline for all the process work. From these simple flows, you will determine which policies need to be defined, how many procedures will be written, and ultimately how many work instructions need to be documented.

Identify Needed Policies

The next term to understand is *policy*. A policy is an axiom or rule that is always true because your organization says it is true. Change and release management abound with policies, and throughout this chapter, you'll find many examples of policies that you may want to adopt for your process work. For example, many organizations insist on a policy that no IT component or service can change without an authorized change record. Policies are declarative statements about how things should work, and many times these policies result directly from the requirements.

When the high-level process flow and subprocess flows are done, the next step is to read the requirements and determine where policies will be needed. One clue is that policies often are associated with making a decision. Look at the decisions on your process flows, and determine how that decision will be made. Is there a policy statement to be defined? Unless the decision is very simple, you will probably want to provide some guidance in how it is to be made, and that's the perfect reason to create a policy.

Not all policies are large and encompassing. Instead of a single grand policy on change approvals, let the process flow guide you in creating smaller policies on change advisory board (CAB) membership, voting rules, handling of emergencies, and so forth. Try to keep the policies focused on one area at a time, and later you can combine them into a single policy document if that's more convenient to manage.

Create enough policies that no important decision or action is left to the imagination of one person. That is when you know enough policies have been defined.

Create Procedures

Procedures are the meat of the process definition. They are the narrative description of the step-by-step actions that someone will take to follow the process. Although the high-level flow provides a good overview, the procedures are the details of how to actually execute the process.

Procedures should be detailed enough that two people executing the same process step will always get the same result. They do not need to be so detailed that two people following them will use *exactly* the same method to get the desired result. Finding the correct level of detail requires an understanding of your organization and the skill level of the people who will be assuming the change and release management roles.

Procedures normally are documented as a numbered list of the specific steps that need to be taken. Not every activity in the process and subprocess flows will require a procedure to be written, but many of those steps will. If the outcome of an activity on the high-level flow is critical, or if that activity will involve a lengthy set of steps, document that activity with a procedure.

Often procedures are written into the same document as the high-level flows. This is a good format, because it keeps all process documentation in one place and is easier to maintain. On the other hand, if you have many procedures and policies, keeping everything in a single document can result in a very long document that is difficult to review and read. Some organizations choose to keep their process and procedure documentation online in a web page or wiki. That format lends itself much more readily to having each procedure and policy written as a separate short document that can be indexed from a process home page.

Document Work Instructions Where Needed

Sometimes procedures can get too lengthy and cumbersome, or the same set of steps needs to be repeated in multiple procedures. This is where a work instruction can be useful. A work instruction is a very specific bit of guidance on how to achieve a specific task within the process domain. For example, in your change management process, you may have several procedures that start with someone looking for a specific change record in an online tool. If the tool has been in your organization for a while and everyone knows how to look up records, you can probably just include a single step in every procedure that says "Look up the change record." On the other hand, if the tool is new, you might want to document exactly how to look up a record. You could do that in one place in a work instruction and refer to it in your procedures.

Normally, work instructions are tool-specific and procedures are not. This is a good practice, because it allows you to change tools in the future without having to rewrite your procedure documents by simply creating additional work instructions. If this is important to your organization, you will have general procedures with many more work instructions.

Keep the work instructions at a higher level than a tool user guide, however. They should be used only to describe specific uses of a tool where many options might be available. For example, your release management tool might support a variety of approaches to documenting a release policy. You could write a work instruction that directs people to always follow a very specific method. This ensures that people within your release management team always use the tool in the same way regardless of the latitude offered by the software publisher.

Standard ITIL Process Activities

Now that you understand how to define and document a process, it is time to think specifically about change and release management. As stated in Chapter 1, "Change and Release Management: Better Together," these processes are intimately linked. We will consider them separately here because in all likelihood, you will be defining them separately, but throughout this section as well as the rest of the book, examples will point out the linkages between these process areas.

Change Management

Almost every organization has a change management process in place already, whether or not it is aligned with ITIL best practices. Surprisingly, most change management processes are similar at the highest level. This happens because there is really a basic flow that works for almost everyone, and that is exactly the high level suggested by the service transition volume of ITIL. This basic flow is shown in Figure 3.2.

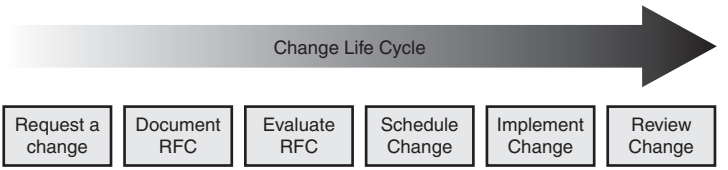

Figure 3.2 Change management follows a standard high-level flow.

The change flow begins with someone proposing a change. This person is called the change requester, and his or her proposal is called a request for change (RFC). The RFC is documented using a set of standard fields, most likely in a change management tool, but sometimes just on paper.

After the RFC has been documented, the evaluation stage begins. Evaluation can be very simple or somewhat elaborate. The most basic form of evaluation is someone looking at the documented RFC and deciding whether it makes sense to proceed with making the proposed change. Some organizations split this decision into a technical evaluation, aimed at making sure the change is technically feasible, and a business evaluation, designed to assess the business risks versus the potential rewards of the change. Throughout this book, you'll learn much more about ways to evaluate RFCs to determine whether to enact the changes they propose.

Assuming the evaluation is positive, the change gets woven into the operational schedule. This can be a complex task, depending on the size of the environment and the number of changes happening near the same time. You must consider the urgency of making the change, any available maintenance windows for the environment being changed, other business activities needing the resources that are being changed, and several other factors. The change is eventually placed into the operational schedule, known in ITIL terms as the forward schedule of change (FSC). Normally a change is scheduled with a specific start date and time, as well as a specific end date and time, so that others will know exactly when the change will be finished.

After scheduling, the next major step in the process flow is implementation. Some preparation activities may take place before the start date and time, but normally implementation starts when the schedule indicates that it should. A change may be implemented successfully, or the implementation may fail. Failed changes may be retried, or perhaps the change may be backed out to restore the environment to its state before the change was attempted. Besides the actual change itself, the status of the change as either successful or failed is the most important aspect of implementation.

After implementation is complete, the process concludes with a review of the change. If the change is successful, this review may be very brief. For failed changes, the review normally includes much more detail, including a recovery plan to indicate how the change will be retried later with a greater likelihood of success.

Based on their needs, different organizations will emphasize different aspects of this basic cycle. I've worked with organizations that separated evaluation into separate steps of evaluation and then approval. I've seen organizations that did hardly any review, even of failed changes. Frequently the request and documentation are combined into a single step. It isn't necessary that you have the same number of high-level steps as ITIL, but it is very important that you think through each part of this high-level flow and determine how they will be handled by your change management process.

Release and Deployment Management

The official name for release management in the service transition book gives some indication of the emphasis of the process—it is called "Release and Deployment Management." The emphasis is very much on the rollout of new or significantly modified services into the environment.

You should be aware of two levels of activity when thinking of the release management high-level flow. One set of activities occurs once per service and makes high-level plans for the entire life cycle of the service. This macro-level planning involves determining the overall business goals of the service, managing risks associated with the service, and evolving the architecture and design for the services. Some of the key issues settled at the higher level include how many releases or projects will be used to initially deploy the service, how often the service will be enhanced with new features, what cost model will be used to fund the service, and when to retire the service. This higher level involves strategic thinking and long-range planning.

At the same time, each service will be broken into a series of individual projects called releases. Each release adds incremental function to the overall service and represents a separately deployed part of the service. At the lower level, release management is about orchestrating these releases through a cycle that includes planning, building, testing, and deploying the necessary components. Normally each release is a project, involving a project team, a scope, a design, and a project plan of its own.

ITIL would say that the top level is called "service design" and is described in the book by that name, and that the lower level is properly called "release and deployment management." For the purposes of your implementation, however, it is almost impossible to achieve the lower level without a solid understanding of the higher level, so you should plan to include both in your release management process document, as shown in Figure 3.3.

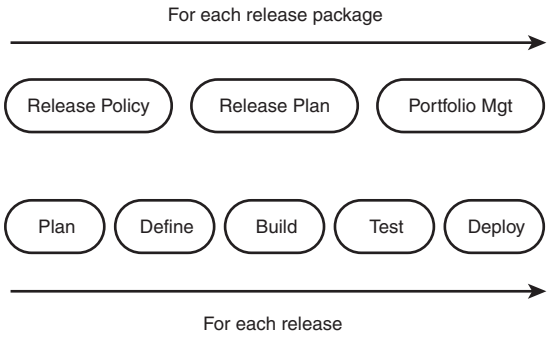

Figure 3.3 Release management spans both service design and service transition.

At the lower level, release management is much like traditional IT project planning. Actually, many organizations leverage the same process for both project management and deployment management because they are so similar. If your project management process covers a full life cycle, including planning, designing, building, testing, deployment, early support, and transition to full support, you can also use it for the lower-level process of release management.

Change Management and Operations

Now that you understand the basics of creating a process and what the standard ITIL documents offer, it is time to look more deeply at the change and release management processes. This section covers change management, and the next section discusses release management. The goal is to provide an overview of some of the more interesting process issues that you will undoubtedly face. The issues are not necessarily resolved for you, but at least they are outlined so that you can begin to find the best resolutions for your organization.

The First Policy

When beginning to define change management in a more formal way, every organization struggles with the question of exactly which activities need to be controlled with a change record. After establishing the highest-level flow, you should try to get agreement about the policy of when a change record is needed. This is the first, and perhaps most important, policy to define.

In most organizations, data center changes are already under change control. Moving a server to a new rack, deploying a major business application, modifying firewall rules, and decommissioning a network appliance are all activities that normally are subject to change control. But why does everyone agree that data center activities need to be controlled? It is really because of the potential impact of something going wrong. Perhaps a good policy to use is that if the change could cause a serious negative consequence, it must be controlled through the change-management process.

That is a great working definition, but it is too ambiguous to be of direct use. What seems like a serious negative consequence to one person might seem like a minor inconvenience to another. For example, swapping the 21-inch display attached to my desktop PC with a 14-inch display probably wouldn't seem like it could have serious consequences to the CIO or most of the IT organization. To me, however, the results of even a successful swap would drastically reduce my screen size, and thus my ability to multitask, create illustrations, and monitor events all on the same screen. Therefore, I would interpret this swapping of monitors as having serious negative consequences. So would a change record be necessary?

Consider a weekly batch job that updates data in your customer support database. The same job runs each month, picking up data from several sources and then integrating that data into the database that all your customer support representatives use to respond to customer calls. Certainly if this job fails in such a way that the database is scrambled, you will have serious negative consequences. But the job has run for years now and has never failed. Does your policy call for a change record to control this job? Sometimes data changes require significant controls, and other times they might be considered standard operations and require no additional controls.

Spend some time at the beginning of the change control process considering as many different scenarios as possible and creating a clear, concise, and helpful statement of when a change record is needed and when it is not. This first policy will probably need to be amended from time to time, but making it comprehensive early on will save effort later in the implementation project.

It is worth taking some time to document the scenarios you use in creating this policy. Those scenarios will be valuable when it comes time to validate your policy and to test the change management process flow.

Documenting the Request for Change (RFC)

The content of a request for change (RFC) is the second issue you will face. Many people assume that whatever tool they choose will determine the contents of the RFC, but this is a mistake. Certainly all tools will come with a comprehensive list of fields that can become part of every RFC, but all good tools also allow you to customize these fields, and you should certainly take advantage of this flexibility.

Most organizations will adopt common basic fields for their RFC content. A number; a title; information about the requester, sponsor, and implementer; scheduled and actual dates and times; and some indication of status are all essential. You will probably also want to have information about the approvals necessary for the change and some implementation information such as an implementation plan, a plan for backing out if necessary, and perhaps a plan to verify that the change is successful.

The best practices contained in ITIL indicate that each change should reference one or more *configuration items*. These are entries from your configuration management database (CMDB), and each change should record exactly how the configuration of your environment will be modified by the change being proposed. If you already have a CMDB with well-structured identifiers for each item, you're in great shape, and you will want to be able to attach one or more

of these identifiers to an RFC. On the other hand, if you haven't yet implemented a CMDB, you still need some way to allow the requester to specify which parts of the environment he or she is changing. This will be very important in assessing the technical impact of the change.

Some organizations like to include fields in every RFC to help understand the compliance implications of the change. Often these are simple check boxes or flags that indicate whether the proposed change will affect audit posture. You may also need something more complex, such as pointers to a separate compliance management tool. The requirements you documented will guide how much you need to customize the fields that make up the RFC.

Reviews and Impact Assessment

The number and types of reviews needed is another significant process issue to explore. Some organizations choose a single review that focuses on the question of whether a change should be made. Other organizations like to use separate technical and business reviews to focus on the technical and business risks and implications of the change. Your process definition should consider the number and order of the reviews and should assign appropriate roles for each review. The names of the people reviewing each RFC might change, but the roles should be consistent from review to review.

Part of the review process should involve assessing the impact of each proposed change. Impact assessment consists of two parts—technical analysis followed by risk management. The technical analysis phase determines what components of the overall IT environment might be affected by the proposed change. For example, if the change calls for rebooting a specific server, it would be natural to understand which business applications depend on that server. Those applications could potentially be impacted by the change. The technical analysis would also determine whether those applications could be switched to other servers or whether an outage of the applications would be certain with a reboot of the server. Having a complete and accurate CMDB makes technical assessment of a proposed change much simpler.

The second phase of impact assessment deals with risk analysis. This involves using your imagination and technical understanding to guess what could go wrong with the proposed change. Consider the possible ways the change could fail, and build a two-dimensional matrix. The first axis in the matrix is the likelihood of any potential failure happening, and the second axis is the damage that would result if that failure actually happened. In the server rebooting example, for instance, it is possible that a hard disk failure might keep the server from restarting. Given the reliability of modern disk drives, there is a low likelihood of this happening, but the impact of the server's not restarting might be quite significant. This kind of analysis could be repeated with all the potential failures for the change, and the aggregation of risk data will help assess whether the change should be attempted.

Approval, Authorization, or Both

One of the key questions to be determined in your change management process is to what extent you will require changes to be approved before they are implemented. Many different models for approval exist, and the one you choose should allow sufficient control without undue bureaucracy.

Experience suggests that you might want to use two different kinds of permission—approval and authorization. Approval grants you permission to invest time (and therefore money) to plan a change and is essentially a business approval. For something like a major release of a business application, this might involve a team of programmers or the purchase of a vendor software package. For hardware implementation, approval may be required to purchase a new router or to invest in architect time to define a new SAN layout. Any change that requires investment to get ready for implementation might require an approval to make that investment.

The second kind of permission is authorization. Whereas approval grants permission to expend resources, authorization grants permission to alter the production environment and thus is a technical or IT approval. Consider again the implementation of a major business application. Many months might pass while the developers are working on building and testing the application. Approval was granted to spend money during those months, but there is no guarantee that the results of the effort are safe for deployment. Authorization is an acknowledgment that the testing of the application has been sufficient and that plans for implementation have considered and mitigated the risks involved.

You should spend a significant part of your change management process work on this question of approvals and authorizations. You will probably determine that some changes require only authorization and that others require both authorization and approval. Be sure to define a policy that will help everyone understand which kinds of permission are required for which changes.

Post-Implementation Review

ITIL recommends that you review each change after implementation, and it gives you some general ideas of what to look for in that review. That guidance is sound as far as it goes, but you will certainly need to fill in many details concerning how post-implementation reviews will be conducted for your organization.

The central purpose of reviewing a change is to learn how to improve your future implementations. We learn more from our failures than our successes, and this is also true in change management. The changes that fail have the most to teach us about future success and thus should be thoroughly reviewed. Understanding the reason for a failure can make future changes more successful. This is why each failed change should be reviewed.

Your process definition should include the specifics of how post-implementation reviews will work. Identify the roles to be involved, the potential actions to be taken, and the ways in which discovered information will be fed back into future changes. If these reviews are new to your organization, you need to specify even more closely how they will be conducted to ensure that they provide the maximum value.

There are many more topics to understand when documenting change management. The topics common to nearly every implementation are covered throughout this book, but some topics may be more specific, so you need to deal with them on your own. In dealing with any issue in process definition, the best resolution always comes from forming agreements between your stakeholders and sponsors. Introduce the issue, generate lots of communication around it, and

then proceed with the resolution that makes the most sense to everyone involved. Remember that policies, procedures, and processes can always be modified later to be even more useful. In the continuous service improvement book, ITIL suggests that each process document be reviewed at least annually to find potential improvements, so don't be too determined to get everything perfect on the first pass.

Release Management and the Project Life Cycle

Almost every organization has a change management process, but very few have a specifically named and defined release management process. Unless your organization does a lot of software development, you may not have given much thought to release management. Whereas many of the issues described earlier for change management are familiar to you, those described here for release management may cover new ground. This doesn't, of course, make the issues any less important. You will soon discover that release management is every bit as important as change management, and that together change and release management form the core of how new services get introduced to your environment. This section covers the highest-level process issues in release management.

Release Unit Identification

Just as change management begins with defining which activities will require a change record, release management begins with documenting which components will be released simultaneously. ITIL defines a release unit as the set of components that get upgraded, installed, or changed all at the same time. As a simple example, often the next version of a business application requires new versions of middleware software products. This means that the application and the middleware form a release unit, because they are deployed at the same time.

There are many reasons to form a release unit. Vendor prerequisites might determine release units, as in the business application example. Sometimes purchasing considerations define a release unit, such as when new PCs come with a new operating system already installed. The PC and operating system become a single release unit. Project management often determines release units based on an analysis of the risks versus rewards of implementing multiple parts of a complete project at the same time. In some cases, there are valid architectural reasons to create a release unit out of multiple components. Whatever the reason, when your organization determines that multiple components should be joined for the sake of introducing a new service, you have defined a release unit.

You should try to create release units along consistent lines. Some people find that releases based on business application environments work well. They change out the operating system, middleware, and business application all at the same time as part of the release management process. Others like to create release units based on technology types, creating a desktop PC release consisting of bundled hardware, operating system, and standard software.

It takes a great deal of communication to create a release unit policy. It would be extremely difficult to identify in advance every situation that might cause a release unit to be formed, so you

should focus instead on creating some guidelines that help people decide how to best create them. Work with the various deployment teams in your organization to understand and document these guidelines. Ultimately, deploying two or more things at once is always more risky than deploying only one component at a time, but most organizations find those risks worth taking in certain circumstances. Understand what those circumstances are, and document them as part of your release unit policy.

Release Policies

As soon as you understand release units, you can begin defining some general policies concerning release management. Most organizations find it useful to define a policy about how many releases should be produced per year. This policy helps in IT planning activities, because the organization can lay out the annual plans based on how many release units are active and how many releases each of those units will go through per the policy.

Of course, the number of releases per year will most likely depend on the number and type of components that make up the release. If a significant component of your release package is a software product, you won't be able to create releases more often than the software publisher produces releases. If you are bundling hardware refresh into your releases, the release cycle will depend on how often you choose to refresh your hardware. This will lead to a release policy that determines the frequency of releases based on the kinds of components that will make up the release unit.

An alternative to defining numbers of releases is to constrain the size of releases. You can constrain the size by either project budget or hours expended. For example, your policy might say that each release will require less than 2,000 hours of planning, testing, and deployment. This kind of policy ensures that your organization doesn't attempt huge projects that have correspondingly large risks. Limits of this kind will force projects to break their desired results into multiple releases and allow your organization to stop those releases if the cost or risk of achieving all the benefits appears too high.

Regardless of how you choose to define release policies, they are worthwhile to define. Release policies help create consistency in your organization and tend to create deployment projects that are roughly the same scope or size. This consistency helps you better evaluate successful and failed projects, and you can tune your release policies to optimize the size and scope of the projects for your organization. By creating fewer, larger releases, you will get larger projects that run longer, consume more resources, and return more value. By optimizing toward smaller releases, you wind up with small projects that generally return value more quickly.

Releases or Bundled Changes

Somewhere in your definition of the release management process, confusion is likely to arise about the difference between a release and a set of bundled changes. Although these may seem similar on the surface, they are really quite different.

Normally changes are bundled as a scheduling tool. There might be three different activities that all require the mainframe to be restarted, so rather than restarting the mainframe three separate times, these changes are bundled. All three things are done, the mainframe is restarted, and the changes are marked as complete. This is a convenient grouping of changes that happens one time because the schedule works out that way.

A release, on the other hand, is determined by a set of permanent policies that define release units and release frequency. The components of the release are related to one another by technology or business purpose, and the relationship is permanent rather than transitory.

A release might be deployed as a single change, or as a group of changes that are related to one another. For example, if the release includes an operating system and a middleware product, these might be deployed through two changes that take place on consecutive weekends. If the first change fails, the second change cannot happen, because the release control process ties the two together into a single release, and the release cannot be only partially deployed. In other words, a release can result in a group of bundled changes, but there are perfectly legitimate reasons to bundle changes that have nothing to do with release management.

Support and the End-of-Life Cycle

One of the key benefits of release management is that it causes an organization to think about the entire life cycle of a release unit. Many organizations have no policies or, at best, ineffective policies, around the end of support. I've been involved with companies that had six or even seven separate versions of an application all being supported because they just didn't know how to sunset that application. A key part of the release management process definition should be a policy surrounding the end of life for your releases.

Normally a release reaches end of life because a newer release replaces it. It might take some time to fully deploy the new release, and during this time both releases will be part of the supported environment. Your policies should take this situation into account and define how long the older versions will be supported. Your policy might insist that each release deployment project include the costs of supporting and then removing the old release.

In addition to the end of any specific release, your policy should consider how to define the end of a release unit. For example, imagine that your release unit consists of a payroll application, web server, database middleware, and common server operating system. You can define new releases as the middleware changes or new versions of the application become available, and each release retires the previous release. But you should also consider when you will move to a new payroll application that requires different infrastructure and thus creates a new release unit. If you make it an organizational policy to include this kind of long-range planning in release management, you will be able to forecast the large number of resources required to actually launch such a large project. This kind of complete life-cycle thinking is one of the hallmarks of a mature release management process.

Looking Ahead

Process definition is at the heart of ITIL. It is impossible to achieve best practices without documenting what those practices will look like in your organization and training your people to use them. In this chapter, you've learned about the different parts of a solid process document and how to assemble them. Using this knowledge, you can now read the ITIL books and begin to build your own implementation of change and release management. In the next chapter, this idea is extended to included logical work flows, which are repeatable procedures that cover a variety of common situations.

Index

D

E

F

G

Q

R

S

T

U

V

W