



realtimepublishers.com[®]

The Definitive Guide[™] To

Enterprise Change Management



Dan Sullivan

Chapter 2: Examining the Nature of Enterprise Change Management.....	25
Modeling Enterprise Change	25
Constructing a Model of Enterprise Change.....	27
Assets	28
Dependencies	29
Policies.....	30
Roles	31
Workflows.....	31
Using the Reference Model to Define ECM Applications	32
Functional Characteristics of ECM Systems	33
Communicating About Change.....	33
Tracking Changes with AMRs.....	33
Implementation Characteristics of ECM Systems	36
Synchronizing Process and Data.....	36
Using a Shared Metadata Repository.....	37
Sharing the User Interface	37
The Enterprise Change Lifecycle.....	38
Specializing Change-Management Operations.....	39
Identifying Software Configuration Management Requirements.....	39
Identifying System Configuration Issues.....	41
Identifying Content and Document Management Issues	41
Supporting Enterprise Operations with Change-Management Services.....	44
Summary	45

Copyright Statement

© 2003 Realtimerepublishers.com, Inc. All rights reserved. This site contains materials that have been created, developed, or commissioned by, and published with the permission of, Realtimerepublishers.com, Inc. (the "Materials") and this site and any such Materials are protected by international copyright and trademark laws.

THE MATERIALS ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. The Materials are subject to change without notice and do not represent a commitment on the part of Realtimerepublishers.com, Inc or its web site sponsors. In no event shall Realtimerepublishers.com, Inc. or its web site sponsors be held liable for technical or editorial errors or omissions contained in the Materials, including without limitation, for any direct, indirect, incidental, special, exemplary or consequential damages whatsoever resulting from the use of any information contained in the Materials.

The Materials (including but not limited to the text, images, audio, and/or video) may not be copied, reproduced, republished, uploaded, posted, transmitted, or distributed in any way, in whole or in part, except that one copy may be downloaded for your personal, non-commercial use on a single computer. In connection with such use, you may not modify or obscure any copyright or other proprietary notice.

The Materials may contain trademarks, services marks and logos that are the property of third parties. You are not permitted to use these trademarks, services marks or logos without prior written consent of such third parties.

Realtimerepublishers.com and the Realtimerepublishers logo are registered in the US Patent & Trademark Office. All other product or service names are the property of their respective owners.

If you have any questions about these terms, or if you would like information about licensing materials from Realtimerepublishers.com, please contact us via e-mail at info@realtimerepublishers.com.

Chapter 2: Examining the Nature of Enterprise Change Management

The need for change within organizations comes from many sources. Innovative technologies, market pressures, and regulatory changes are just a few. These drivers affect organizations in a variety of ways; many set in motion complex sequences of change that propagate effects far beyond the original point. As we explored in Chapter 1, a simple upgrade in a word processor program can change the way staff collaborate and share documents as well as introduce security vulnerabilities that eventually lead to a widespread disruption. This chapter will examine the nature of ECM by breaking down complex processes into constituent parts and analyzing their interactions.

I'll begin by describing a generic model of enterprise change and showing how that model provides a guide to understanding ECM tools. The sections that follow discuss the process of managing enterprise change and examine its lifecycle. Finally, the chapter concludes with a discussion of the need for domain-specific additions to the generic model for software development, systems configuration, and document management. Each of these three domains will be examined in more detail in Chapters 3 through 5.

Modeling Enterprise Change

Although we can talk about changes at the enterprise level in broad terms, it is more useful to start with a more precise discussion. To understand the nature of enterprise change, we need to identify specific *assets* that change and understand how the changes in one asset modify another. For example, specific assets include software packages, Web services, networking hardware, security policies, manufacturing processes, and marketing initiatives. Each of these assets has both static attributes and dynamic lifecycles.

Attributes are general characteristics of a resource that you use to describe the resource. An instance of a CRM system, for example, runs on a particular platform, has a specific version, and uses a relational database for storage. Similarly, marketing initiatives are associated with product lines, have budgets, and have executive sponsors. It is likely that during the life of the CRM application, the application will be upgraded to a new version or migrated to a different platform. While executing a marketing initiative, the budget might change in response to an unexpected downturn in the market. These changes in the core attributes of an application or marketing initiative characterize the *lifecycle* of the resource.

Lifecycles, like attributes, vary among resources. An application's lifecycle entails development, testing, release, maintenance, and retirement. Documents are created, reviewed, approved, published, revised, translated, and archived. Hardware systems are evaluated, installed, integrated, monitored, and replaced. Strategic initiatives are formulated, revised, approved, implemented, and evaluated. Fortunately for those who manage change, the lifecycles are generally predictable, moving from one state to a small set of other states as Figure 2.1 illustrates.

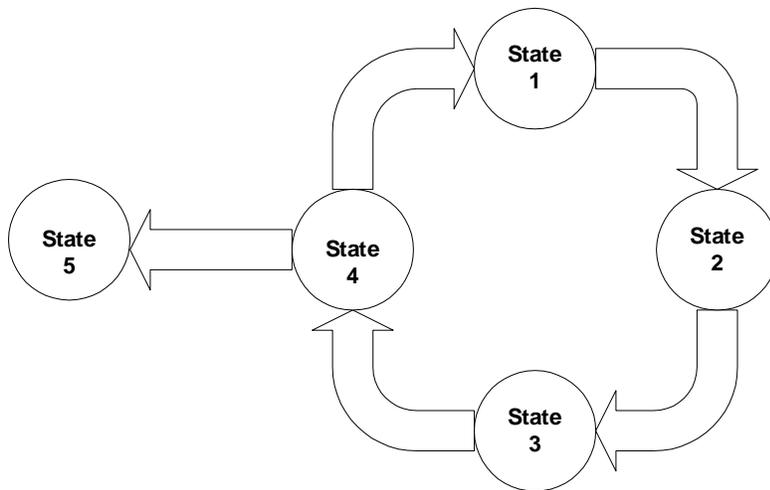


Figure 2.1: The lifecycle of resources follows a predictable pattern although multiple outcomes are possible from any given state.

Individual assets, like individual organisms in an ecosystem, have lifecycles that interact with others. To understand the dynamics of a complex system, whether it is a biological ecosystem or an enterprise IT infrastructure, you have to understand how lifecycles influence each other. For example, an increase in a fox population leads to a decrease in a rabbit population. However, as the rabbit population declines, the food source for the foxes declines leading to a reduction in the fox population. The declining fox population decreases the threat to the rabbit population, which, in turn, will grow over time. The changes in one population directly affect the other. Similar effects are seen in organizations. As enterprise assets change with events in their lifecycles, those events trigger actions in other resources' lifecycles (see Figure 2.2).

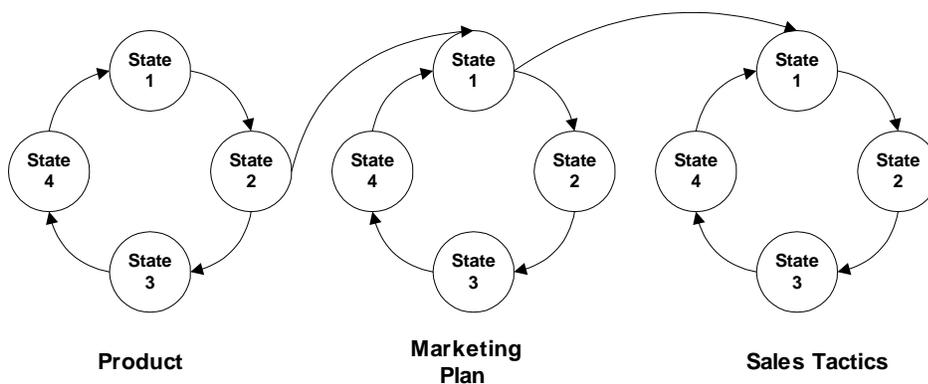


Figure 2.2: Events in the lifecycle of one event can initiate changes in the lifecycle of other resources.

For example, suppose an LOB manager decides to reposition a product line to take advantage of a new trend in the market. The marketing department updates the marketing plan, which is then passed on to regional sales directors who revise their sales tactics. They, in turn, inform their sales teams who change how they present the product to prospective customers.

Nature has numerous mechanisms propagating change throughout an ecosystem—populations migrate, species adapt, and animals change behavior according to their situations. Enterprise ecosystems, especially IT ecosystems, do not adapt as gracefully as biological ecosystems. Changes in one resource can disrupt services, degrade performance of other systems, and force unwanted changes on operations. When changes are not communicated, customers and partners receive inconsistent information and service. Rather than depend upon a nature-like cycle of change and adaptation, dynamic organizations need more centralized, proactive controls over enterprise change. The first step to developing those controls is modeling their requirements.

Constructing a Model of Enterprise Change

Models make complex systems understandable. They also allow us to see patterns that are not necessarily obvious when we see systems in detail. The characteristics shared by the processes of introducing a new version of an application, a policy document, and a network device are often not obvious—especially when we think of common domains such as software, documents, and networks. For example, when we think of software applications, many of us immediately think of programming languages, software design patterns, version control systems, release management policies, and a host of other software engineering techniques and tools. Similarly, when a new policy document is introduced, we think of document revisions, metadata, approval workflows, publication processes, and other document management tasks. Network device configurations bring to mind protocols, security issues, network outages, and hardware compatibility. Identifying the unique and obvious characteristics associated with each domain is quite easy compared with identifying the more nuanced but equally important shared characteristics of each process. Without an understanding of these inter-relationships, organizations tend to rely on silo-oriented change-management procedures. For each domain, IT staff has particular methods for dealing with change rather than using a single approach for change management across the enterprise.

Complex organizations need to extend these change-management techniques to better work across functional domains. To begin, IT staff needs to understand the similarities between functional domains (such as software development, system configuration management, and document management) and describe them with models.

 Recognizing *shared* characteristics of different change-management domains is the foundation of modeling ECM.

Managing any operation or process requires that we understand its essential elements. Models help us do so by distilling processes to their most basic components and operations. Many of us are familiar with some change-management methods for particular domains (such as software configuration management) and not with others (such as document management). These domains share common characteristics, and a generic model helps to highlight those properties. In addition, a generic model provides a reference for evaluating and comparing ECM tools. ECM models are made up of five components:

- Assets
- Dependencies
- Policies
- Roles
- Workflows

Assets

Assets are objects that change. Software, policies, network devices, strategic plans, and marketing initiatives are all examples of assets. All assets in ECM models have several characteristics:

- Versions
- Issues
- Attributes

Assets change over time, and versions track logical groups of changes. In the case of software, documents, and other digital assets, organizations frequently keep older versions as well as the latest release; however, only a single version of a hardware asset usually exists. For example, an asset called Boston Router 2 might not be the same physical device today that it was 4 years ago, and the earlier version may have been redeployed, sold, or disposed of.



How versions are realized (such as with a new file or a new piece of hardware) is unimportant for the model. ECM models represent the shared characteristics of assets.

Issues, as you might have guessed, are topics about an asset that must be addressed. Defects, system incompatibilities, missing features, proposed content changes, and legal questions about policies are examples of issues. Again, the differences in the types of issues are unimportant. Software has bugs; documents do not. Documents have copyrights; servers do not. Nonetheless, the general property of tracking issues applies to resources that change over time.

Attributes are characteristics that help describe an asset. (Versions and issues are attributes in the strictest sense. For our purposes, however, consider attributes to be the set of characteristics that change with asset type. All assets should be modeled with versions and issue attributes.) Documents kept in a content-management system will have a creation date, an archive date, key terms, categories, summaries, language, description of rights, and access controls. Software modules have major and minor release numbers, and a description of change history that includes the name of the programmer, the type of change, the date of the quality control check, and the release date. To model change, you don't need to know the exact list of attributes for each asset type you want to manage. This freedom is one of the benefits of a model of ECM—it allows you to focus on the unchanging, core properties of enterprise change, and leave the varying details until later.

Dependencies

Dependencies identify assets that are required by other assets. When modeling dependencies, you need to consider the granularity of assets and the types of dependencies.

Assets can be described at a number of levels. At a high level, you can model an ERP system. A more detailed model breaks the ERP system into modules such as inventory, scheduling, and database. Each of these components can be further decomposed. For example, the database module includes the relational database engine, backup software, performance monitoring tools, and network client software.

So what is the appropriate level of granularity for describing assets? In general, assets should be modeled at the level that dependencies apply to all sub-modules of an asset. Consider Asset A, which consists of Modules 1, 2, and 3. Asset B depends upon Modules 2 and 3 of Asset A. Figure 2.3 shows the appropriate level of granularity.

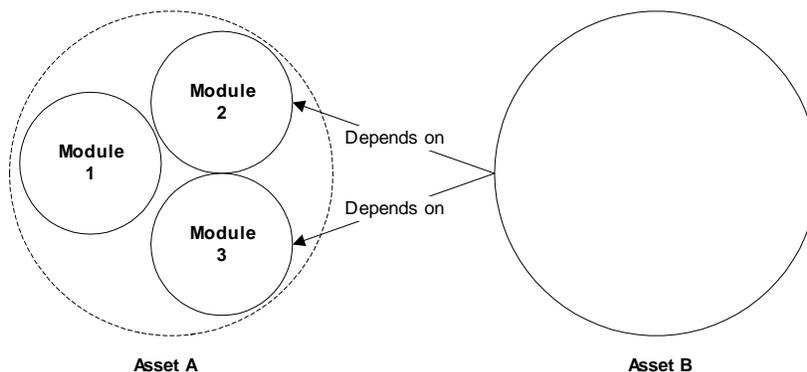


Figure 2.3: Asset granularities should be chosen to precisely depict dependencies between assets.

Models that do not precisely describe dependencies can misrepresent relationships by depicting them as more restrictive than they are. For example, Figure 2.4 shows an imprecise, or course-grained, dependency model that inaccurately depicts Asset B as dependent upon all three modules, when, in reality, it doesn't depend on Module 1.

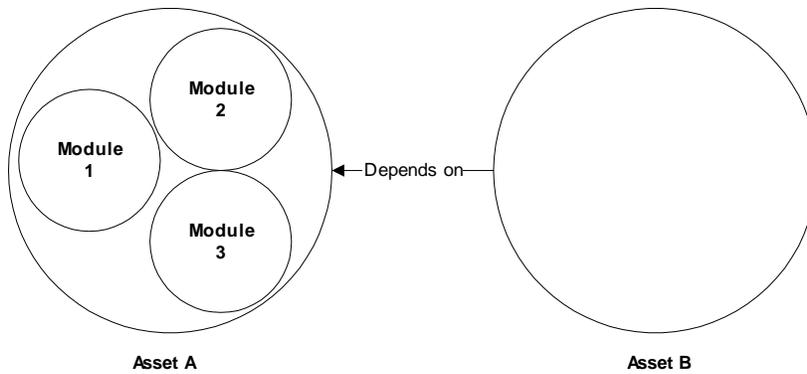


Figure 2.4: Course-grained models can erroneously depict dependencies.

Similarly, models that are too fine-grained will depict dependencies in a more complex manner than necessary. If Asset B depends upon Modules 1, 2, and 3, then Figure 2.4 depicts the appropriate level of granularity; Figure 2.5 depicts a version that is too fine-grained, which creates more work than necessary to accurately model dependencies (most models will be much more complex than this simple example).

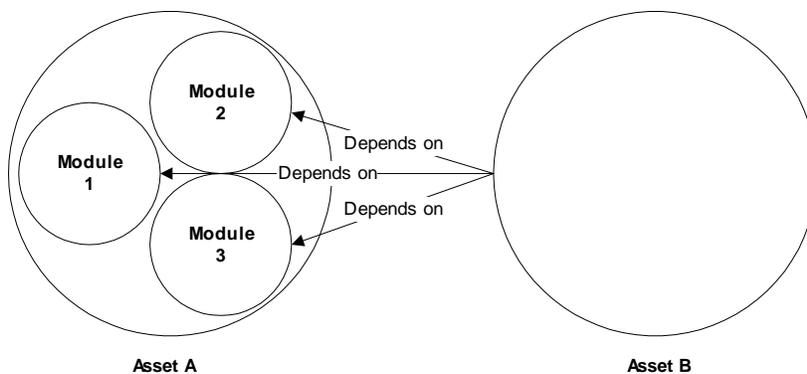


Figure 2.5: Because Asset B depends on all modules of Asset A, this model is too fine-grained.

Policies

Policies describe the rules that govern the use of assets and the mechanisms of change. Some policies are asset-specific, such as procedures for introducing a new piece of network equipment, creating a new branch in a software configuration management system, or approving changes to an HR policy. Other policies are independent of assets. For example, rules about notifying others about changes and workflow approval policies are not restricted to particular types of assets.



Policies themselves have change-management issues. The lifecycle of a policy should be governed by change-management procedures.

Policies have similar lifecycles to other enterprise assets. For example, a change in federal regulations may cause an organization to change its policies about sharing customer data. This external change forces the company to revise its policy, which, in turn, creates changes to policy content. The change in content initiates a workflow of reviews, approvals, and publishing that ultimately requires communicating information about the new policy (see Figure 2.6).

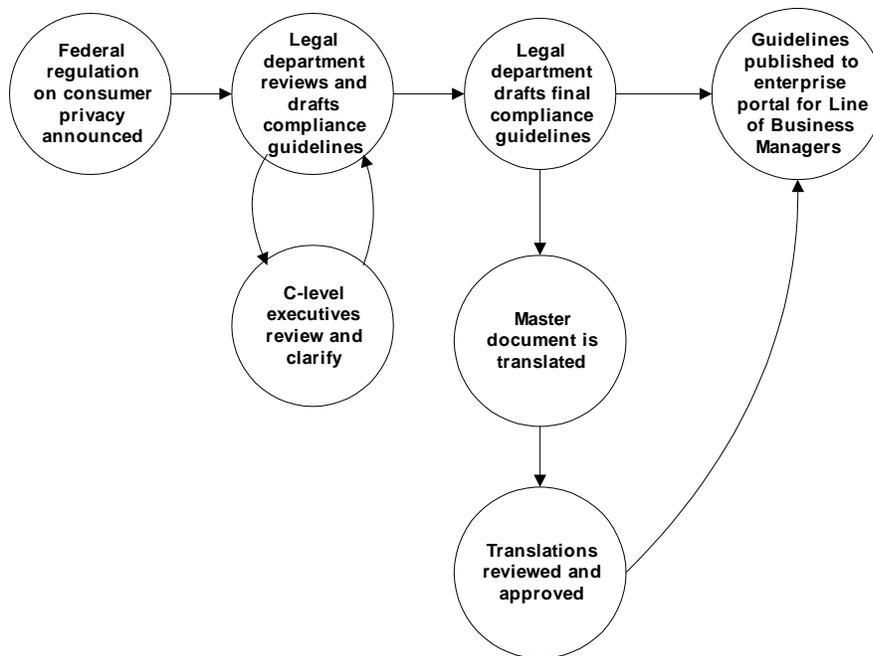


Figure 2.6: Changes to policies are also managed through controlled workflows.

Roles

Roles are assigned to individuals or groups who make decisions and execute workflows. These people follow the rules described in policies to create and maintain assets and to control change across the enterprise by:

- Approving change
- Deciding on the specifics of a change
- Reviewing a proposed change
- Preparing for change
- Implementing change

Roles are frequently used in security systems to control access to resources and serve a similar purpose in change-management systems.

Workflows

Workflows are processes that change the state of assets and related objects, such as change request forms. Workflows manage the change process, coordinate people and assets, and enforce policies. The purpose of workflows is to enable a consistent, rule-based process for executing change. At its most basic level, workflows entail:

- An initial start state
- Rule-based transitions to intermediate states
- One or more end states

The initial state is the activity that starts the workflow process. This activity can range from creating a change request to installing a desktop application to compiling a CRM upgrade proposal for an enterprise change review board. Workflow processes move from initial states to intermediate states according to transition rules.

Transition rules are conditional requirements that, when met, change the state of a workflow. For example, *If the change request to install a desktop application is approved by the support desk supervisor, the request is added to the job queue* is an example of a simple transition rule. Once in the new state, a new set of transition rules applies. For example, *If the job is to install a new desktop application, the priority is 4 and Assign jobs to support desk personnel according to increasing priority* are two rules for controlling transitions to other states.

A workflow is finished when the end state is reached, such as when an application is installed, documentation is updated, and acceptance tests are passed. For example, *Notify Requestor of Status* is the end state of the workflow illustrated in Figure 2.7.

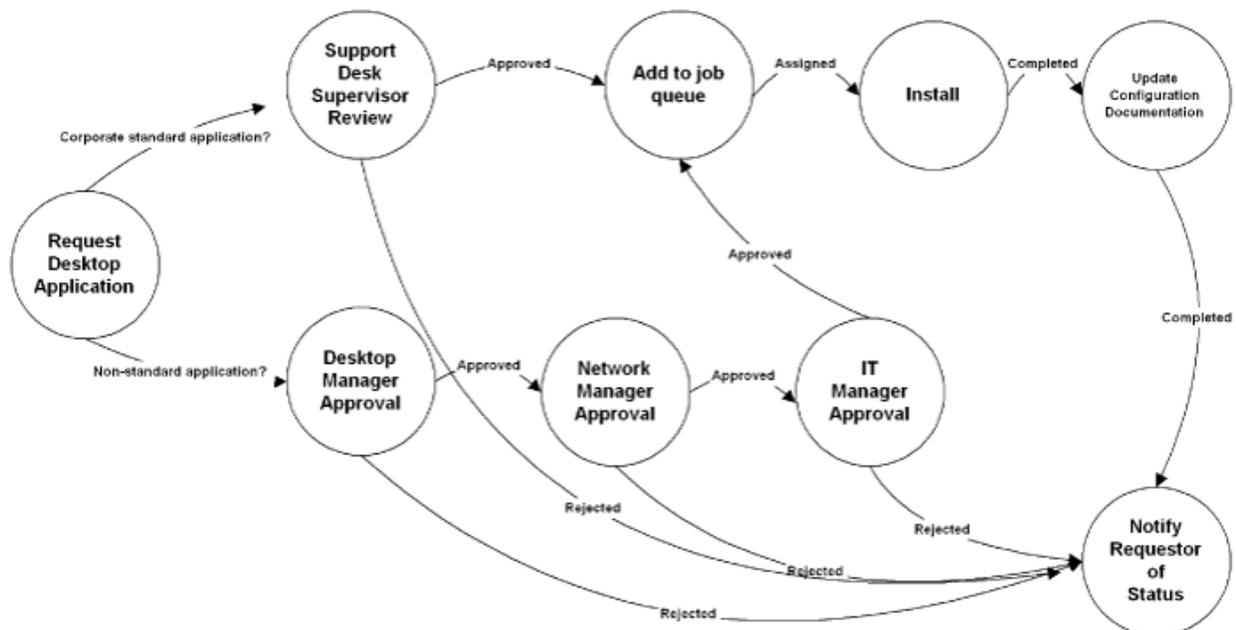


Figure 2.7: Workflows consist of states and transitions between states.

The generic model of enterprise change is made up of assets, dependencies, roles, policies, and workflows. By themselves, they do not completely describe the ideal ECM application; however, they give us a tool for evaluating and comparing such tools.

Using the Reference Model to Define ECM Applications

We need to remember that the need for change management encompasses virtually all parts of an organization. Changes in one department or application can quickly ripple through to other areas and systems. To compound the complexity, change crosses functional divisions, so a change in one software module can impact other software modules, network configurations, and documentation. Developing a model of enterprise change is the first step to creating an ECM management system. The second step is to figure out how to implement that model as an ECM support system.

Functional Characteristics of ECM Systems

Managing change in an enterprise requires three elements:

- Tight communication
- Asset metadata repositories
- Workflow management

We've already discussed workflow management, so in this section, I'll focus on the other two elements.

Communicating About Change

It is clear that changes emerge from multiple sources. A user may request a feature change to a program. A merger introduces new hardware to a WAN. New government regulations mandate additional procedures for product development and production. When a change is communicated in advance of its implementation—for example, when a user requests change to a program—the change is readily managed. But when the impact of a change is discovered after implementation, such as when the upgrade of an enterprise application creates unanticipated demand on network bandwidth, the consequences are more difficult to manage.

To control change, managers and administrators need to understand the impact of external changes to their internal operations. The first step in this process is being aware of the specific details about forthcoming changes. This, in turn, requires that those who are introducing changes understand the scope of their modifications and the potential impact. For example, a Web server administrator may not realize that upgrading to a new release of the Apache Web server will cause a business intelligence reporting tool to crash. The Web server administrator needs a tool that identifies such a dependency. This type of situation is where asset metadata repository (AMR) comes in.

 Dependencies between resources are too complex to manage manually. ECM requires tools for tracking dependencies between resources.

Tracking Changes with AMRs

AMRs track information about individual assets and dependencies between them. As noted earlier, assets are described by attributes. Assets often require other assets to function properly, as Figure 2.8 shows. ERPs depend upon databases. Databases require OSs and servers. OSs depend on networking hardware to communicate with other servers, such as domain controllers.

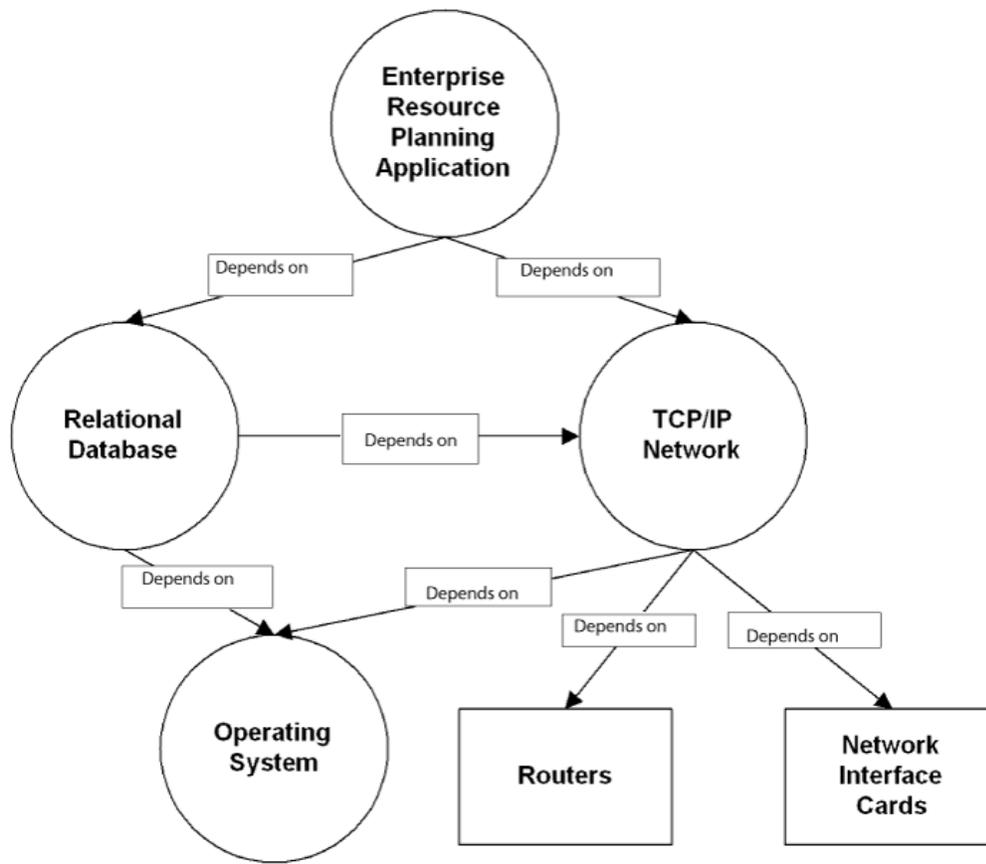


Figure 2.8: Resources can depend upon layers of other resources.

Dependencies are logical relationships between one asset and another, or more specifically, between one asset and a property of another asset. For example, the latest version of an ad hoc reporting tool, Reporter X, requires portal software, PortalSoft Y, version 2.3 or later. Dependencies imply constraints not only on the type of an asset, but can restrict particular attribute values as well.

Dependencies vary by type. One way to categorize dependencies uses the following three types:

- Requires—An asset needs another asset with particular properties to execute correctly
- Consumes—An asset uses the output of another asset
- Supplies—An asset produces data used by another asset

By keeping track of assets and their dependencies on other resources, AMRs enable administrators and managers to perform impact analysis of proposed changes. For example, consider an administrator who wants to upgrade a server to use the latest version of the Linux kernel. The new kernel provides performance enhancements that will benefit all applications, but the kernel also contains changes to code libraries that could adversely affect some applications. The administrator wants to know which applications on the server are affected. Using an AMR, the administrator can list the applications running on that server and the dependencies of each application.

 AMRs are essential to answering the question, what is affected by a particular change?

Let's assume that a machine is a dedicated ERP server that is compatible with the latest Linux kernel, so there should be no problem with the upgrade with the first level of dependencies. We need to keep evaluating the effects of change on deeper levels, though. The ERP server depends on an Oracle database that has not been certified with the latest Linux kernel. However, the database runs on a different server, so it is not affected by the change. Or is it? Although the ERP and database run on different servers, communication software runs on both to enable the distributed system. The database communication software on the ERP server has dependencies on the OS. Although the overall database system is not affected by the OS change, a small, client module is affected. To manage fine-grained dependencies such as this, the AMR should model details of individual modules within larger applications (see Figure 2.9).

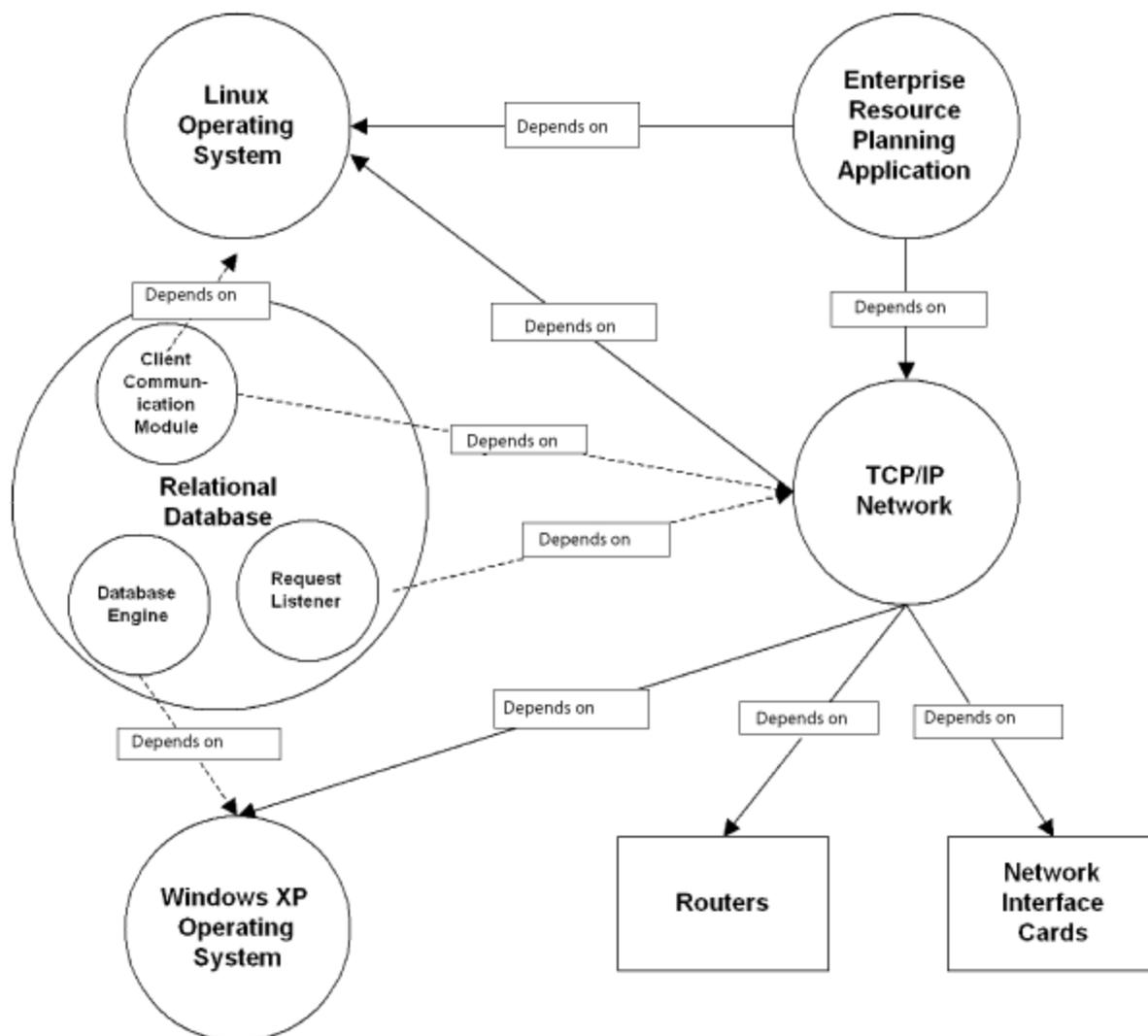


Figure 2.9: AMRs should model fine-grained dependencies—especially in distributed systems.

Tight communication is essential to understanding upcoming changes that affect other parts of the organization. AMRs manage information about enterprise resources and the dependencies between them. Workflow management systems coordinate tasks between people and automated processes. Together, these three elements act as the foundation for managing change, as Figure 2.10 shows.

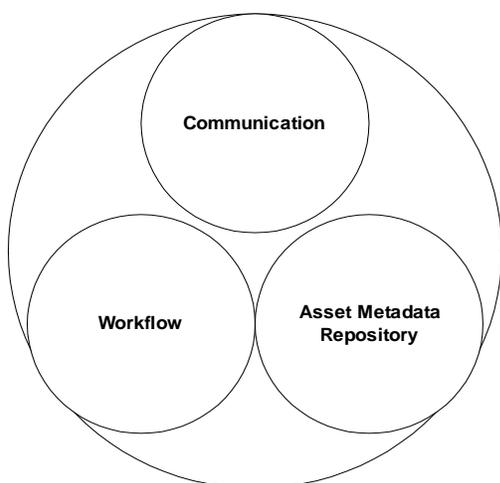


Figure 2.10: The ideal ECM environment consists of AMRs and workflow processes that support tight communication.

With an understanding of the mechanisms required to control change, let's turn our attention to the design characteristics of the ideal ECM system.

Implementation Characteristics of ECM Systems

ECM is a dynamic system that inextricably links process and data. To support the breadth and complexity of change found in large organizations, ECM systems must deeply integrate processes and data. This requires:

- Bi-directional synchronization at the process and data levels
- A shared common repository
- A shared common user experience

Synchronizing Process and Data

A common operation in the ECM is defining workflows. These processes require that you specify assets and rules for manipulating those assets. Once defined, a change to either the workflow process or the assets can affect the other. For example, consider the following business rule: *If a program is on the list of corporate approved applications, the program can be installed on client machines with the approval of the support desk supervisor.*

Let's assume that users submit change requests through an online system that automatically routes requests through a workflow process. When a user requests an installation of a word processing program, the workflow system will need to check the AMR to determine whether the program is on the list of corporate-approved applications.

Similarly, if the workflow process changes the property of an asset, the AMR should be updated. For example, if a project manager approves the release of a new revision to a software module, the repository is updated to include a record of the new version and its properties. The workflow mechanism and the AMR are closely linked in their basic operations (see Figure 2.11).

 ECM depends upon deep integration of workflow and metadata about assets.

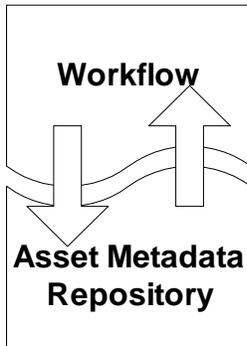


Figure 2.11: Workflow processes depend upon the AMR, and the AMR is updated by workflow operations.

Using a Shared Metadata Repository

The AMR is a common asset repository. Information about software, hardware, network configuration, and content is kept in the AMR. This shared repository design enables deeper integration of ECM than individual repositories. For example, consider the design that Figure 2.12 shows. Silo-based repositories—such as this design—lead to well-known problems, such as inconsistencies among the data and the need for ad hoc data exchange mechanisms.



Figure 2.12: Silo-based configuration management is not an effective option for ECM.

Ideally, releasing a new software module will initiate or correspond with a change in documentation about the module. When the software configuration management workflow and the document management workflow are independent, this change is difficult to automate. The result is a potential inconsistency between the repositories and a break in the logical chain of ripple effects caused by the initial change.

Sharing the User Interface

Another benefit of the deep integration is that it allows for a shared user experience. Users learn a single interface and work in a single environment. Of course, some operations are specific to particular types of change. Documents have keywords and authors; network hardware does not. Nonetheless, by using the model of ECM described in this chapter, you can model different assets and operations within a single application and provide the users with a consistent and unified experience.

Users benefit from both cross-functional change management and the ability to work within a single environment. The single model design also reduces the number of applications that must be integrated with the existing IT infrastructure. The more the ECM system, especially the workflow elements, can be directly integrated with other applications, the more you can automate the change-management process. With an understanding of the static elements of ECM systems, let us turn our attention to the more dynamic characteristics of those applications.

The Enterprise Change Lifecycle

The three elements—tight communication, AMRs, and workflow management—describe the anatomy of ECM. The lifecycle of ECM consists of four stages:

- Initiating change
- Reviewing change
- Implementing change
- Assessing change

The first stage of the lifecycle is initiating change. This stage could entail creating a new asset, such as a policy document; changing the configuration of an asset, such as a router; or creating a new version of an asset, such as a revised program. Proposed changes are then reviewed. This process can range from rapid, single-person review (such as approving the installation of a desktop application) to more formal and detailed plan reviews by a change-control committee. Once approved, the change is implemented following procedures specific to the type of change and type of asset. Finally, the effects of the change are assessed and are either left in place or, if unanticipated consequences emerge, rolled back. Figure 2.13 illustrates this concept.

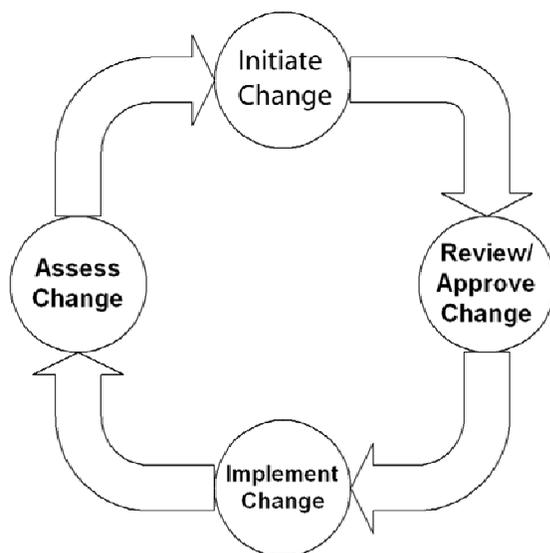


Figure 2.13: *The four-stage enterprise change lifecycle.*

With an understanding of the general elements and lifecycle of the ECM model, let's shift our focus to specialized types of change management.

Specializing Change-Management Operations

To this point in the chapter, I have described a generic model for enterprise change and discussed how several basic elements and processes can support a wide array of changes. This design works well as a reference model for understanding the nature of ECM and its basic elements (assets, roles, policies, and so on). To be truly useful in day-to-day operations, however, we need to support the particular needs of specific processes and assets.

In this section, I will further expand the ECM model to include characteristics of software, system configuration, and document change management. I chose these three types because they are commonly required and highly interdependent. We can imagine higher-level asset types, such as strategic initiatives and marketing campaigns, that build upon these and require change-control mechanisms as well; however, for clarity, I'll keep the focus limited.

 The ECM model described here is generic enough to apply to other organizational processes that entail assets, processes, and dependencies.

Identifying Software Configuration Management Requirements

The requirements for software configuration management are driven by the unique lifecycle of software. Large software systems are modularized to control complexity. The number of modules and the levels of dependencies can grow quickly. Software engineering practices promote code reuse, so a module designed for one application may be used in several others. Modularization and reuse force, at least implicitly, agreements between module developers and module users.

The simplest module is a function that returns some data based upon other data provided as input. Users often make assumptions especially when dealing with simple functions. For example:

- The function does not have side effects (such as changing a global variable or updating a database record)
- The function accepts a reasonable range of inputs (such as 0 to 110 for a customer's age)
- The code will degrade gracefully if there is an error (such as returning an error status instead of crashing the process)

These are reasonably safe assumptions because they fit with common programming practices—especially when dealing with simple functions. As functionality becomes more complex, behavior of the module must be made explicit through detailed documentation.

Explicit descriptions of behavior allow software developers to effectively combine modules to build complex systems in a layered manner. As Figure 2.14 illustrates, a programmer responsible for developing the user interface (UI) might need to work with procedures in the data access layer, but the developer should not have to work directly with database or network components. However, the interface code still depends upon those lower-level modules. Changes to those modules can directly impact the interface.

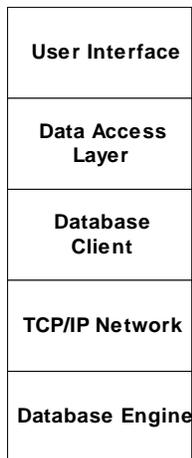


Figure 2.14: Modularized and reused code creates indirect dependencies between modules in complex systems.

Software engineers have developed configuration management practices to manage change in complex software systems. Typical development teams use version control, system builds, and system releases to control change.

 Software engineers that change code need to understand not only what the program does but also where and how it is used. The AMR tracks the required information.

Version control, as the name implies, is a mechanism for tracking changes to program code and documentation. Programmers check out code from a repository to work on the code. While the code is checked out, others are prevented from changing the code in the repository. The programmer checks in the code when he or she has finished making changes and creates a new version of the code. Although this technique keeps developers from overwriting others' changes, it is not enough to ensure that a change does not adversely impact a dependent module.

To ensure that programs continue to work as expected, software developers frequently combine the latest versions of all modules and test the results in the build process. This step consists of compiling source code, linking libraries, executing test suites, and comparing the test results with those of previous builds.

Once the system is built and tested, it is released for use. Release management practices govern how executable code is distributed to users, how dependent modules—such as OS components, link libraries, and Web services—are put in place, and how users are trained to use the new application.

Like other types of change, software configuration management is highly dependent upon rule-based workflows. Programmers check in and check out code, project managers approve code for inclusion in a build, quality control analysts evaluate test results, and release engineers roll out the application. System configuration management, like software configuration, has its own particular issues.

 We'll explore software configuration management in more detail in Chapter 3.

Identifying System Configuration Issues

One of the trickier aspects of system configuration management is continuing to provide services while making changes to the IT infrastructure. If a network engineer needs to change a router, another router must be in place to keep network traffic flowing. Large organizations often have dual ISPs. Critical servers are configured to failover to backup servers that run in stand-by mode. Redundancy is common in hardware systems, and change-management procedures need to account for them. Some characteristics particular to system configuration include:

- Identifying backup systems
- Describing how failover occurs (for example, automatically in the case of routers, or manually in the case of offline servers brought online only after a failure)
- Describing clusters and load-balancing configurations

Security issues might be tracked independently of particular assets. Of course, one can track security issues of a particular device or software (such as a bug in a firewall or router). In other cases, security issues arise as a function of two or more assets operating together.

 In Chapter 4, I'll discuss system configuration management in greater depth.

Identifying Content and Document Management Issues

The most commonly encountered change-management issues in content and document management are:

- Archiving
- Indexing
- Publishing
- Translating
- Access controls
- Reuse and structuring

As the number of documents grows in a document management system, the task of administering the repository becomes more difficult. To ease some of the burden, many administrators use archive dates to indicate when documents should be removed from the repository. Archive dates can be specified manually along with other metadata or automatically generated using simple rules (such as completion date plus 2 years) or with more complex business rules based upon the type of document, the author, the category of content, and other metadata attributes. For example, 21 CFR Part 11 regulations governing pharmaceutical manufacturing processes dictate strict requirements on records retention.

When the content or metadata attributes of a document change, the document repository index must be updated. This task includes updating automatically generated categorizations. If there are significant changes to a repository, such as adding documents about new topics, the taxonomy or other classification scheme will require updating as well.

Changes in a content management system may also trigger the publication of documents in multiple formats. Interested users might automatically receive copies of or notifications about the new documents. In multi-language environments, translations must be revised to reflect changes in the source content. For example, consider a global document management system for product documentation. Master documents are written and revised in the company's primary language, then translated into other languages. Each translation leads to a new version of the document that must, in turn, undergo review and approval procedures defined in a workflow.

As with any IT system, the particular requirements for access controls will change over time in a content management system. Updates are driven by both changes in the user base and changes in requirements to protect different document classes. For example, the state of California has enacted a law requiring companies to notify California customers when a security breach leads to a disclosure of personal information. This requirement may prompt companies to change data management procedures for California customers, move California customers' data to separate repositories, or even lead some to stop conducting business in that state.

 Supporting multiple languages and maintaining effective security are two of the most challenging aspects of document management.

Reuse and restructuring of documents is much easier now than in the past. Extensible Markup Language (XML) is radically changing content management. Prior to the widespread adoption of markup languages, such as XML and HTML, documents were treated as single, logical units. A short memo and a 300-page budget were both stored as a single document. (Large documents, such as books, could be broken down into a series of files stored in directories, but this approach still retained many disadvantages.) These documents are generally referred to as unstructured text because the documents are not structured like databases or spreadsheets with easily identifiable pieces of data. From a content management perspective, unstructured documents have two drawbacks: they make it difficult to reuse content and they only provide document-level metadata.

Unstructured documents make it difficult to reuse content. For example, a product brochure might contain a brief description of a product that could be used in an online catalog. Traditionally, that description was cut from the brochure document and pasted into a description field for the product in a content management system. For companies with a global market, the problem is compounded with the need to reuse multiple versions of translated content.

Another drawback is that unstructured documents only provide document-level metadata. For example, the metadata about a product brochure document might describe the author, creation date, version number, keywords, and document description. Unstructured documents do not specify details such as where the title begins and ends, where the short product description begins and ends, where pricing information begins and ends, and so on. Without complex programs to analyze the text, it is impossible to reliably extract pieces of information from unstructured documents.

 For more information about the role of XML and related technologies to business, see the Organization for the Advancement of Structured Information Standards Web site (<http://www.oasis-open.org/home/index.php>) and for more technical details, see XML.org (<http://www.xml.org>).

For these and other reasons, content managers are turning to XML and related technologies to better manage content. The most important are XML, XPath, and XQuery.

XML is a markup language that lets you structure parts of a document. The documents still contain free form text—they are not as structured as databases or spreadsheets—so they are called *semi-structured documents*. For example, a simple product description in an unstructured document follows:

```

WP 1234 Wireless Phone

"The WP1234 2.4GHz cordless phone offers caller-id, call back,
100 number memory, speaker phone ..."

"Recognized for superior sound quality and reliability the WP
1234 ..."

```

A corresponding XML document adds structural information and metadata such as the example in Listing 2.1.

```

<product-description>
  <product-name> WP1234 </product-name>
  <features>
    <feature>2.GHz</feature>
    <feature>caller-id</feature>
    <feature>100 number memory</feature>
    <feature>speaker phone</feature>
  </features>
  <short-description>The WP 1234 2.4 GHz cordless ... </short-
description>
  <long-description>Recognized for superior sound quality and
reliability,
the WP 1234 ...
  </long-description>
</product-description>

```

Listing 2.1: An example XML document.

The semi-structured XML version includes the same information as the unstructured file as well as indicators, or *tags*, that identify specific information such as the product name and a short description. These additional tags make the files more difficult to read for humans but greatly enhance our ability to automatically process these files. XML technologies, such as XPath and XQuery, provide the means to identify a particular piece of information in a document (for example, the short description), then query for it much like we query databases.

As XML, XPath, XQuery, and related standards are adopted, organizations will have more opportunities to reuse document components for multiple purposes. For example, a single product description maintained in an XML database may be used in brochures, catalogues, and email solicitations. (Identifying and extracting appropriate content requires managing the XML schemas, which, in turn, introduces change-management issues. There is no end to the need for change management).

 We'll explore content and document change management in detail in Chapter 5.

Clearly, there are similarities and differences across domains in modern enterprises. Effective ECM tools will support both.

Supporting Enterprise Operations with Change-Management Services

Change management is evolving. Understanding and managing change is becoming more complex as businesses integrate operations and cross-traditional functional boundaries. One of the most important aspects of ECM is the interdependence of changes. A change in one type of asset often triggers a change in another type of asset. A change in one department or line of business frequently initiates a change in the operations of another department or functional area. Analyzing and planning for the scope of change that ripples through an organization when silo-based change-management systems are used is challenging at best. These systems do not provide the broad, cross-functional impact analysis that is needed to effectively manage enterprise change.

Impact analysis allows managers to model the effects of a change. Consider the following examples:

- A change in a client application requires changes to the hardware configuration of client workstations.
- An upgrade to an application server will not function properly without reconfiguring a firewall and the virtual private network (VPN).
- Changing a marketing plan will change the product presentation for the sale staff.

To understand the detailed effects of these changes, managers must have access to configuration and dependency information. That data, along with impact analysis algorithms that trace paths of dependencies through networks of assets, are the key to impact analysis reporting. Impact analysis allows administrators to conduct what-if planning but can also help determine:

- Who to notify about forthcoming changes (for example, other systems administrators and application managers)
- When to implement a change to minimize down time
- When to reconfigure services to prevent a service outage
- What changes to hardware are required
- If there are any incompatibilities with layers of software
- Any potential increase in calls to the Help desk

The benefits of ECM extend beyond impact analysis. These techniques allow organizations to maintain compliance with auditable standards, such as HIPAA in the healthcare industry, 21 CFR Part 11 in pharmaceuticals, the Gramm Leach Bliley Act in financial services, and the Sarbanes Oxley Act in publicly traded companies.

 See Chapter 1 for more details about how external requirements such as audit standards create the need for ECM.

As studies of the use of software capability maturity models have demonstrated, controlled methodologies improve the quality of application development. Although it is too early to measure the impact of ECM techniques, it's reasonable to expect similar levels of improvement in non-software development areas.

 For more information about software engineering and capability maturity models, see the Capability Maturity Model Integration Web site (<http://www.sei.cmu.edu/cmml/>).

ECM procedures provide C-level executives with enforceable policies and procedures. They provide impact analysis tools for planning and managing change. In short, they enable proactive planning and serve to minimize the need for post-change damage control.

Summary

Organizations are constantly changing to adapt to new situations. The process of continual change is a fundamental process for large enterprises and cannot be avoided. Even if an ideal internal configuration for an organization could be found, external factors, originating from customers, competitors, and governments, will change the environment in which the organization operates.

Change-management practices have long improved the quality of software development, system configuration, and document management. To move beyond domain-specific techniques, we need a generic model of enterprise change. A basic model, like the one described in this chapter, helps to elucidate similarities between change-management domains, which, in turn, provides a foundation for understanding how to manage changes that cross domains.

As organizations move from silo-based change-management techniques to ECM, they will need systems that describe assets and their dependencies, policies and roles for controlling assets, and workflows to use assets. Organizations will also need to understand the lifecycle of change. Enterprises are like ecosystems—changes in one part of the environment have ripple effects on distant parts. Unlike natural ecosystems, today's organizations cannot sustain uncontrolled change.

Organizations control change with tools that support tight communication and workflows using AMRs. The tools we need are evolving along with the need for ECM. If you are assessing ECM tools, use the generic model described here as a reference point. The generic model might not describe all features required to control change, but it does include the minimum functionality needed to support ECM.

In the next three chapters, we will use the generic model to discuss how to effectively manage change in software development, system configuration, and document management, respectively. At the same time, we will examine how changes in these domains cause change in other domains, and how you can control that cross-domain change.