CHAPTER TWO

OVERVIEW OF CONTENT MANAGEMENT

It's been a hard day's night,

And I've been working like a dog.* *—John Lennon and Paul McCartney*

Beyond the mountains

there are again mountains. *—Haitian proverb*

Executive Summary

The best way to understand content management is to watch a web team in action. This chapter tells a story about a team that manages a web property; it shows how the assets evolve over several years from an exploratory prototyping effort into an enterprise-critical business operation. As the story unfolds, we highlight the core issues of content management via commentary to illustrate how each issue originates, and to point out the approach the team takes to overcome the challenge.

*A HARD DAY'S NIGHT by John Lennon and Paul McCartney. Copyright © 1964 (Renewed) Sony/ATV Tunes LLC. All rights administered by Sony/ATV Music Publishing, 8 Music Square West, Nashville, TN 37203. All rights reserved. Used by permission.

From Prototype to Enterprise

It has been a wild four years. Rita's original idea was to build a web site in her spare time to improve the wooden suggestion box outside her company's lunchroom. Over the years, the penciled suggestions on scraps of paper yielded insightful ways to improve product quality, manufacturing efficiency, and employee morale, to name a few. The suggestions bypassed the traditional chain of command, and although the suggestions were almost always anonymous, Rita's group tirelessly made special efforts to respond seriously to each one. Perhaps the reason the aging wooden receptacle led to the changes that it did was the immediacy of the follow-up. Paradoxically, her group had never been formally charged with being the keepers of the suggestion box. It just seemed to be the right thing to do. Rita herself could not have predicted the events that would follow.

2 A.M. Software

It is 1996, and the Internet Age has dawned. No one in the company recalls where the idea came from, but perhaps it doesn't matter. Why not augment the wooden suggestion box with a web site? That way, employees outside that immediate location can participate as well. Rita takes on the challenge during evenings and weekends. She refers to it as "2 A.M. software."

As a skunkworks project, word of the web site's existence spreads by word of mouth and internal e-mail. The suggestion box web site enjoys a steadily increasing flow of visitors. At first, visitors from other company locations visit to see first-hand how the democracy of ideas within that humble support facility can lead to tangible improvements. Soon thereafter, suggestions from other sites begin trickling in. Many of the suggestions pertain to company-wide operations.

Rita is the sole part-time web developer on the project. Just prior to leaving for an out-of-town conference, she demonstrates a prototype to a group of product managers to explain the value of the suggestion system. The product managers offer many suggestions to improve the usability of the interface. After returning from the out-of-town conference, Rita dives into implementing their recommendations.

Without the usual distracting chatter of phone conversations filling the air, Rita adds the pull-down menus to the entry page surprisingly easily. She likes the way that her new menus remove the visual clutter from the page. Eager to show off the new interface, she asks a colleague to try it out. When he tries it, Rita realizes that she hadn't tested her changes with the Netscape browser. They notice that her menus don't erase properly. Because she hasn't tested regularly with different browsers, some new code she recently added is the likely culprit. But which change is it?

The one-week hiatus renders Rita's recollection about the web site internals hazy. She needs help recalling what worked and what didn't work when she last did the demo. Luckily, Rita versioned the entire source tree of the demo web site before leaving on her trip. She's able to figure out what changes are candidates for the browser sensitivity that seems to have been induced by her recent changes.

Web site versioning plays a crucial role in the web development process. It is essential to periodically capture known snapshots of the web site, as we see in Rita's early efforts with the web site. Snapshots serve several purposes. First, with a known snapshot of the web site a developer can roll back to a known-good copy of the web site. Second, if the web site or a section of it becomes hopelessly broken, a developer needs to be able to selectively pick assets to revert to. An *asset* is an electronic artifact that embodies the intellectual property of an organization. Having a known working set of web assets lets a developer proceed to make changes with the assurance that there's always the ability to compare to a working copy for ongoing development. A working copy can be used as a reference copy, to discern what changed and what didn't. Typically, only a small fraction of a set of files changes from one day or week to the next, which makes having a reference working copy of a web site invaluable for locating the changes that led to a problem. The core issue of site versioning typically arises distinctly early in a web site's lifecycle.

The Pioneers

The team expands to four members. Rita is now the web architect, an informal leader of her band of renegades. They soon find themselves doing independent tasks. Sandy is the quality assurance leader whose primary goal is to build a test harness for the business logic embedded in the web application. She has also volunteered to prototype the online help system. Max is the CGI developer, and he is converting the text-file-based suggestion repository to use the corporate-standard commercial database. James is the interface designer, and he explores ways to simplify the interface.

continued

It is Tuesday. James breezes past the unattended reception desk. He hears a few clattering keyboards from the early risers in the office. As he settles into his cubicle, he listens to the reassuring sound of the coffee maker gurgling the morning's first pot of coffee. James reloads the page that he changed yesterday before he went home.

"What happened to my changes?" James cries out, as he throws his hands into the air. He rechecks the URL to verify that he's indeed pointing at the right location. The reality sinks in. Someone has overwritten his changes with their own changes last night. James gets to his feet. He silently paces the aisle.

Later that day, after much wrangling, the team decides to adopt a practice that gives each developer separate areas to do their work. James does his best to recreate his changes from memory.

Two months pass. The team works 18 hours per day 7 days per week to prepare for their presentation to a corporate Internet task force. To build the team's pitch, Rita gathers recent examples of business improvements that gathered momentum from their web site. She hopes to solicit support from the high-level executives charged with selecting and funding a promising set of web initiatives identified by the Internet task force.

James has numerous changes to the homepage file, `index.html`, and to the first-level pages, such as `suggest.html`, to deliver the promise of his slick new user interface. Max has changes, too. Although he has been working independently and in relative isolation on his database subsystem, it is now time to integrate that code into the homepage and first-level pages as well.

They hammer out an integration plan on Friday after their weekly status meeting, but when 2 A.M. Sunday morning comes around, Max finds that James hasn't completed the changes that they agreed to. James has either forgotten, or his progress was delayed. Unfortunately, when Max snoops around in the directories on James' development machine, he sees various versions of half-completed sets of pages. Max is stuck. He has to either make guesses about what James intended to do, or he needs to call James at home. He relishes neither option. He had intended to finish his integration Saturday night because he promised to help with his niece's sleepover birthday party on Sunday.

We see the issue of *managing concurrent changes* coming to the fore when the group reaches four members. Because each person is off doing different projects, and especially because of the nature of web technologies, they hit a "web-wall." This is the point in the lifecycle of a web site when the combination of the number of developers, the number of assets, and the pace of development exceeds the ability of informal coordination mechanisms to adequately do the job.

The Tornado

Named a finalist by the Internet task force, the team gains additional funding, and grows to 20. The pace of change outstrips the ability of any single person to keep track of changes. As of August 1998 the team is tackling the following projects:

1. Converting to template UI for rebranding
2. Writing scripts to regenerate the UI
3. Implementing better help system
4. Reimplementing form entry based on templates
5. Building client portal

Each project involves a cross-functional team of two to four developers. The project to regenerate the template-based user interface has a scripting component, an HTML component, and a user interface design aspect. Keeping track of any one of these projects on its own would be difficult enough, but keeping track of any two of these projects simultaneously is beyond the grasp of a single person. All five projects taken together require major infrastructure assistance.

In August, James leads a three-person project to convert to a template-based user interface for branding purposes. The release date is near, and the team is deeply involved in testing. Meanwhile, Joe leads a two-person team getting started on implementing the help system. The help team feels strongly about checking in partial releases of their subsystem. Because they have the suggestion system itself to receive client feedback on their help system, they feel that the potential gains outweigh the risks. The help team has content that has been approved and is ready to send to production. On the day before that, the template team plans to have their subsystem ready.

Project completion skew occurs as the team grows to a point that individuals are doing different things and multiple groups each have a different project focus. In other words, each project is coping with contributors on a project who are working on diverse activities, and each project alone has a need to develop, integrate, test, and review their work before their project can be integrated into the live web site. Inevitably, projects run concurrently, and they don't all finish at the same time. One project might be getting underway while another project prepares to wrap up their work. There needs to be a way to keep the work separate.

This problem became especially acute for Rita's group when James and Joe led separate projects. The changes from one project interfered with the other. Because they hadn't introduced separate edit areas, some of the unfinished changes for the branding project mixed in with the finished changes for the new help system. The mixture of finished and unfinished work proved problematic.

> Joe works with the marketing department to build a weekly news section of the web site. For the first few weeks, Joe moves the weekly updates to the production site himself every Saturday at midnight. When that becomes tiresome, he and James agree to share duties on alternate weeks. It gets worse when the marketing department decided to push changes more frequently, now three times per week: a Monday edition, a mid-week edition, and a weekend edition of the newsletter. Both of them agree that they need a more automated system. One day over lunch, they wish for a system that will deploy precisely the assets that they specify, at a predetermined time, and to notify them by pager if the automated system encounters errors.

Deployment comprises the processes and practices by which web assets that have been reviewed and approved are copied from a development environment to a production environment. The goal of a deployment infrastructure is to copy assets to the production server into the right location at the appropriate time. Assets no longer on the development side are deleted from the production side.

An important organizational underpinning of a deployment infrastructure is the "release agreement," which binds the development and production groups into a social contract.[1] Content and application developers agree to approve and formally submit any asset to be deployed, and production server administrators agree to use only released assets on a production server. In a well-designed deployment infrastructure, only someone that is authorized to initiate a deployment job does so. A well-designed deployment infrastructure copies assets into production with minimal or no effort, with full control, notification, and the ability to roll back to a known-good version.

Rita's gang needs an efficient and reliable deployment mechanism almost from the very beginning. Although moving a handful of files to production is simple when taken in isolation, the small overworked staff soon finds itself buried in small trivial tasks that nonetheless are prone to error. It becomes especially difficult when the person copying the files isn't the one who made the changes and therefore isn't familiar with the files.

[1] Chapter 7 discusses the release agreement in more detail.

With multiple editions of the newsletter per week, and the increased visitor volume, a misstep in the handling of the numerous content sources is bound to happen. Sure enough, it happens at a particularly inopportune time. The company brass had quietly begun investigating the feasibility of selling the web operation to outside buyers, or alternatively, conducting a public offering. Therefore, it is especially embarrassing that the lead article on a Monday edition of the newsletter misspells the company name of a new partner company. Worse still, it incorrectly identifies the job titles of three of that company's executives.

Workflow is the process by which people collaborate to develop assets within a content management system.[2] This issue becomes important when several people collaborate on a job, where wait time is a significant proportion of the total job time, and where patterns of interaction are repeated frequently. Workflow improves productivity by minimizing the wait time between successive steps, and it automates the business logic of an organization.

One important benefit of workflow is its ability to automate routing, review, and approval of jobs. A second benefit is the ability to enforce a formal business process, as the web operation becomes larger. This advantage becomes especially important when the operation spans different departments.

Rita's group introduces a formal workflow system after the misspelling fiasco in the online newsletter feature article. The rapid growth in the staff justifies investing in a workflow solution because it ensures that each set of changes has had proper review and approval.

Go Dot-com

Rita's department spins out as an independent corporation in April, 1999. The marketers adopt a dot-com name, ezSuggestionBox.com. Previously, they were a division of a small manufacturing company building educational aids for K–12 and the higher education markets. Now they are an independent 100-plus person company. Their customer base is growing, and they have seven major initiatives moving at web-speed simultaneously. The industry analysts refer to them as an "application service provider" in the untapped higher-education market segment. Inside the company, they realize that their growth and ultimate survival as a standalone company rest on their ability to continually enhance and extend their service offering, while maintaining their 24 × 7 uptime promise.

continued

[2] Chapter 6 discusses workflow in more detail.

Some of the company's initiatives immediately change the current web site. For example, without advance notice, the marketing group decides to insert a series of banners on the homepage calling attention to an improved notification service. Other initiatives move at a more deliberate pace, such as the ongoing effort to completely replace the homegrown script-based notification system with a commercial product from a third-party vendor.

Each banner project goes from conception, to assignment, to implementation, to review, and to approval within four hours, through the rollout of the new service. But the integration of the commercial notification engine takes six weeks. Neither change can afford to wait for the other.

The issue of *long-term versus short-term projects* becomes important when there's a long-term web development effort going on concurrently with short-term changes to a web site. The essential point is that there are changes for the long term that overlap with changes for the short term, and the changes cannot go to production together. For this reason, the development efforts are split apart and done separately.

Rita's gang encounters the challenge of managing what essentially amounts to two distinctly different web properties. On one hand, there is an ongoing sequence of short-term changes to insert a new banner on the homepage, for which each takes only a few hours to go from implementation to approval. On the other hand, the deeper and more pervasive changes to integrate the third-party notification system take weeks to complete. The solution to this problem involves setting up separate "branches" of development, where each branch corresponds to a logically independent web site.

As the scope of the web operation expands, corporate marketing plays a larger role in directly creating content or sponsoring the creation of content by outside contributors, known as stringers. Nina, the corporate marketing manager, is an example of an inside contributor. When she assembles a press release, she focuses on the market positioning spin, the strategic sprinkling of third-party endorsements contained within the press release, and the go-live date of the release. At the same time, she defers to the art director's guidance on the layout of the press-release detail pages on the corporate site. The same goes for the choice of font for the title on the press-release summary page. Most of all, Nina appreciates that previously written press releases will be automatically reformatted to the current design rules.

Separating content from presentation is also known as *templating*.[3] The assets that comprise a web site must be factored in a way that allows many members of the web team to make changes concurrently. A content contributor is someone whose domain of responsibility focuses on the information content within a web site. Another department or group typically holds responsibility for determining the form of the presentation of the content. Because a large organization finds it essential to separate content from presentation, it centralizes art direction decisions, while it decentralizes content creation. This separation becomes more pronounced as an organization grows.

This is exactly what we see in this story. Nina, the corporate marketing manager, focuses on creating content. Divorcing the content from presentation lets her reuse content in many different situations, some of which was not anticipated when she created the content. For example, another manager uses a one-sentence summary of her piece as a caption on a promotional graphic. For Rita, the key is to design a content-capture framework that maximizes the opportunities for the content to be reused over its lifetime.

By the middle of 2000, ezSuggestionBox.com expands its operations worldwide. Two regional development centers operate out of Boston and Chicago. To satisfy the customer traffic, production web farms operate in San Jose, Boston, Chicago, and London. The total asset base has grown to 500,000 web assets. Multiple content servers, geographically dispersed, host the assets.

The issue of *handling of very large-scale web sites* arises when web operations spread over development locations around the world, or when the volume of web assets exceeds the capacity of a single server to handle. Scalability of the software and hardware infrastructure is an essential consideration, regardless of the particular choice of technology.

In the case of ezSuggestionBox.com, the developers find the need to distribute assets among several content management servers when their operations expand into regional development centers. They reap the benefits from the fact that each development center uses a common infrastructure for content management and that the basic framework extends to handle future growth in assets.

[3] Chapter 5 discusses templating in greater detail.

Terminology

A *web property* is an aggregated presence on the World Wide Web with a single entry point, such as www.ezSuggestionBox.com or www.yahoo.com, or it is a disaggregated presence that is reached from multiple entry points, exemplified by banner ads hosted by doubleclick.net. One or more servers render a web property's presence on the Internet.

Web development encompasses creation, modification, review, and approval of assets. Examples of assets that we consider are web sites, source code repositories, document repositories, or any combination of these. Our focus is the development of assets. An individual or a group creates an asset, and it is costly, sometimes impossible to replace. A graphic file containing the company's logo is an asset. Similarly, a Java source code file that is used in an online commerce application server is also an asset.[4] The text of a press release or a product support bulletin is an asset. All of these embody intellectual property of an organization and are difficult or costly to replace.

Universality of Assets

When we speak about assets, it is sometimes tricky to distinguish a web asset from a nonweb asset. For example, a company's web banner logo graphic is easy to categorize as a web asset. On the other hand, it is more difficult to classify program code that contains an SQL query that extracts a customer's pending order that is invoked by an application server running in conjunction with a web server. Is that a web asset? We'll be pragmatic. If the logic or other intellectual property embodied in the program code affects the success of the web property, we'll consider it an asset within our area of concern. To emphasize the point, we'll refer to these as *extended web assets*. An extended web asset is directly or indirectly used to implement a web property. An extended web asset may not be a web asset in the strictest sense, but its ability to be developed in a timely and correct manner does affect the overall success of the web property. We'll use the shorter term, "asset," to refer to these extended web assets. For example, an automated voice-response telephony system uses prerecorded audio messages. Working in conjunction with a web property or working standalone, the audio message files are assets by our definition.

Web assets, we must emphasize, are passive. A web property comes alive through hardware and software that *renders* web assets. (See Figure 2.1.) For example, a "classical" web property renders HTML and images by using a combination of web server software running on stock hardware. The HTML and images constitute the web assets. Because web assets are passive, we store them, version them, index them, and

[4] Chapter 3 introduces the distinction between a source asset and a derived asset. For example, a Java source file is a source asset, while a compiled Java class file is a derived asset.

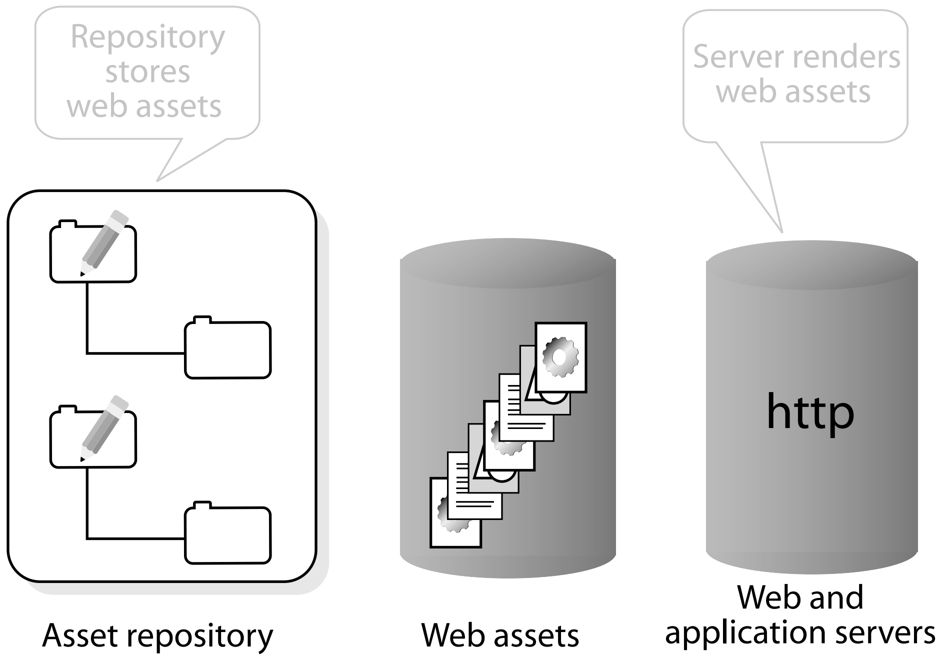
Repository stores web assets
Server renders web assets
http
Asset repository
Web assets
Web and application servers

FIGURE 2.1
A repository stores assets while a server renders them to bring them to life.

retrieve them. In spite of their passivity, web assets define the essence of a web property. As such we must carefully protect and care for web assets by storing them in a content repository. (Paradoxically, the hardware and software to render assets consist of standard replaceable components: off-the-shelf computers and shrink-wrapped software components.)

The distinction between web assets and the rendering of web assets is critical; it underscores the proposition that the enterprise of web development focuses on creating and managing web assets. The fundamental challenge of content management is to protect and care for the web assets that give life to its web property.

Organizations choose varying approaches to rendering their web assets. One company chooses a traditional approach of HTML files, image files, augmented with CGI scripts. Another chooses a database-driven content server. Another devises a hybrid solution that combines a database-driven content server with a Java servlet-driven application server. Common application servers include products from IBM, Art Technology Group, BEA Systems, BroadVision, and Vignette. All of these mechanisms represent different approaches to the problem of rendering web assets. This idea extends to assets that implement extensions to the web server (such as Microsoft Active Server Pages) as well as assets hosted on the web server but downloaded into the client browser (such as Java applets). Content management transcends the specific choice of rendering technology. The requirement to manage the assets, preferably in some kind of repository, is universal across all the approaches.

Managing Web Assets

Web development groups change rapidly, especially the successful ones. They hire more staff. Teams tackle business objectives that expand and transform over time. The pace is relentless. Above all, the size of the operation tends to grow quickly. (See Figure 2.2.) Sometimes the business grows. The operation grows as fast as people can be hired, or the operation expands as it validates its business model.

As the size of the web operation increases, different techniques for managing the web property come into play. As we'll see, each technique overcomes important problems faced during development. Each technique has advantages and limitations.

In the following sections, we'll introduce four approaches to managing assets for a web property. The first one is suitable for a small web site consisting of fewer than a hundred assets. The others make sense for successively larger web operations, up through enterprise-class web sites with millions of assets.

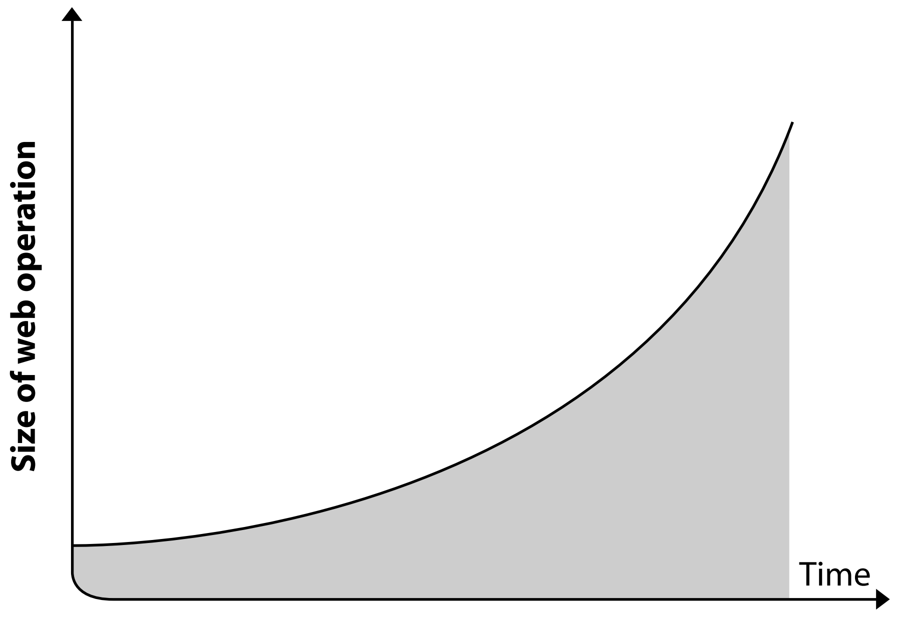

FIGURE 2.2
Web operations tend to grow rapidly.

Live Editing

For a very small web site produced by one or two developers, where the site consists of fewer than 100 files, it is common for the developers to edit the live web site directly. (See Figure 2.3.) The approach is simple. There is one copy of the web site, the live production copy. To make a change, edit the asset directly.

Since small web sites are typically exploratory efforts, uptime isn't essential. The number of hits on the web is small, and the fact that files may be temporarily missing or incorrect on the production web site doesn't affect many visitors. In Rita's story, live editing is adequate in the early days when her primary goal is to establish a proof of concept, during her "2 A.M. software" era.

This arrangement has the advantage of being simple to administer. Fix what needs to be fixed. Visit the live site to see if the site works. This simple scheme works because the entire site is under one person's management, and the live site is the latest working copy of the site.

The major disadvantage of this arrangement is that there is little control of the site. Typically, the only version control consists of making an entire copy of the web site occasionally. If a problem is discovered, and there's a need to revert some of the site, or the entire site to a previous working version, the copies are examined to determine which copy to roll back to.

Directly making changes on the production site has other unfortunate consequences. First, the solution doesn't scale on a number of dimensions. Only a handful of people can work on a live production site if a mistake becomes visible to the user base. This makes it impossible to accomplish objectives more rapidly by boosting staffing. Since the site must be fully functioning at all times, this severely limits the kinds of projects that can be undertaken. Any kind of change that requires multiple files to change in a coordinated fashion, or that require any testing at all, cannot be undertaken on a live site.

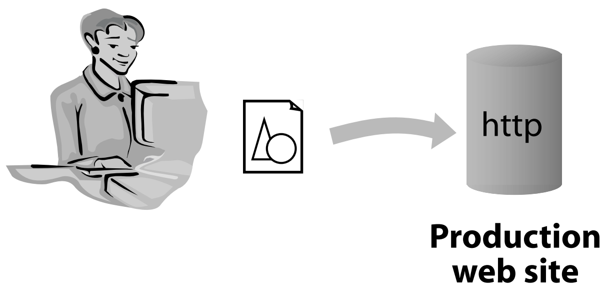

FIGURE 2.3
Developers on small sites often edit the live web site directly.

Finally and perhaps most insidious, live editing promotes a self-limiting work culture that impedes future expansion. By its nature, live editing encourages a "web cowboy" mentality that hinders future development. One example is the ability of different production web servers to host a web property without rewriting the internal references. For example, if the web property of a hypothetical firm, the General Bot Corporation, uses fully qualified references, such as "href=http://www.generalbot.com/privacypolicy.html," then this limits the ability to test the web assets before moving them into production. Suppose for instance, that the company evaluates competing web-hosting vendors to more economically and reliably serve the corporate web site. This requires copying a snapshot of the web property to a test server to measure how different server configurations handle simulated loads. In our example, we want the test web server to handle the reference to the privacy policy, instead of unconditionally redirecting the viewer to the main corporate site. Internal references relative to the "docroot" should be made to "href=/privacypolicy.html" instead. This more closely expresses the true intent of the reference. The practice of live editing tends to mask the distinction between internal and external references, which impedes future expansion of the development team and the testing effort.

Staging the Web Site

As the number of assets grows, and the number of developers increases, it is no longer practical to edit the production web server directly. Enter a separate web server, or staging server. It runs a copy of the production web site, with the difference that we copy changes that the developers intend to put into production on the staging web server first. (See Figure 2.4.) This solution works on sites up to 1,000 assets, when the number of developers is less than five.

The staging web site solution is adequate when Rita's gang consists of a handful of developers. In the story, we see that as the efforts of the developers begin to take divergent paths, managing the changes and the testing within a single staging web site has significant limitations.

The staging server introduces the important ability to test changes before they go live. Developers are able to detect errors before they reach the production site. This solution is similar to the live edit procedure, except that developers point their browser at the staging server, instead of at the production server. As long as the number of developers is less than five or so, then the improvement in the ability to test outweighs the additional burden of the two-step procedure. First, move changes to the staging server. Second, copy the changes to the production server.

This solution begins to break down as the number of developers increases beyond a handful. With more developers, it becomes harder to keep track of individual changes. That trap befell Rita's organization in the opening story. Although Rita's development organization deserves credit for having progressed to a level of maturity where they recognize the importance of testing web content before going live, they

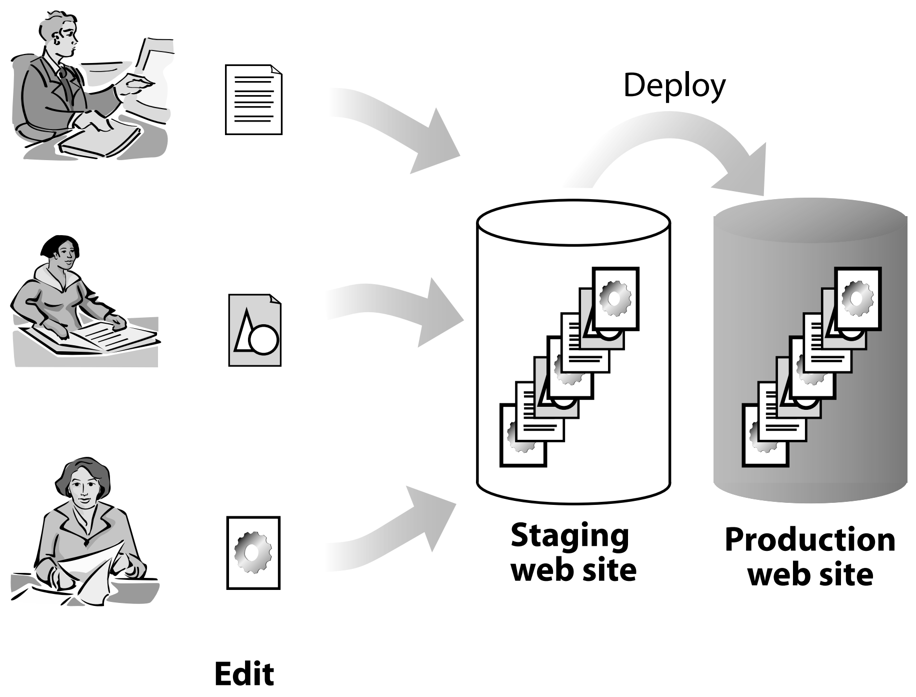

FIGURE 2.4
Copying a modified asset to a staging web site allows a development team to test changes before deploying them to the production web site.

are encountering the principal limitation with the staging server model. Their development team has too many members for each person to understand which changes belong to whom, and which changes are or are not ready to go into production.

Independent Edit Areas

At the next level of sophistication, development groups retain the staging server, but proceed to give each developer an independent area in which to make and test their changes. (See Figure 2.5.) This partially solves the problem of developers stepping on one another because each developer has a web server and a separate area in which to make their changes. This has the additional benefit that each developer is able to test changes independently. This approach has some amount of success with development teams up to eight, with number of assets fewer than 2,000 to 5,000.

Rita's band of renegades find the independent edit areas approach useful when they branch out into small project teams working on different assets. Keeping an accurate

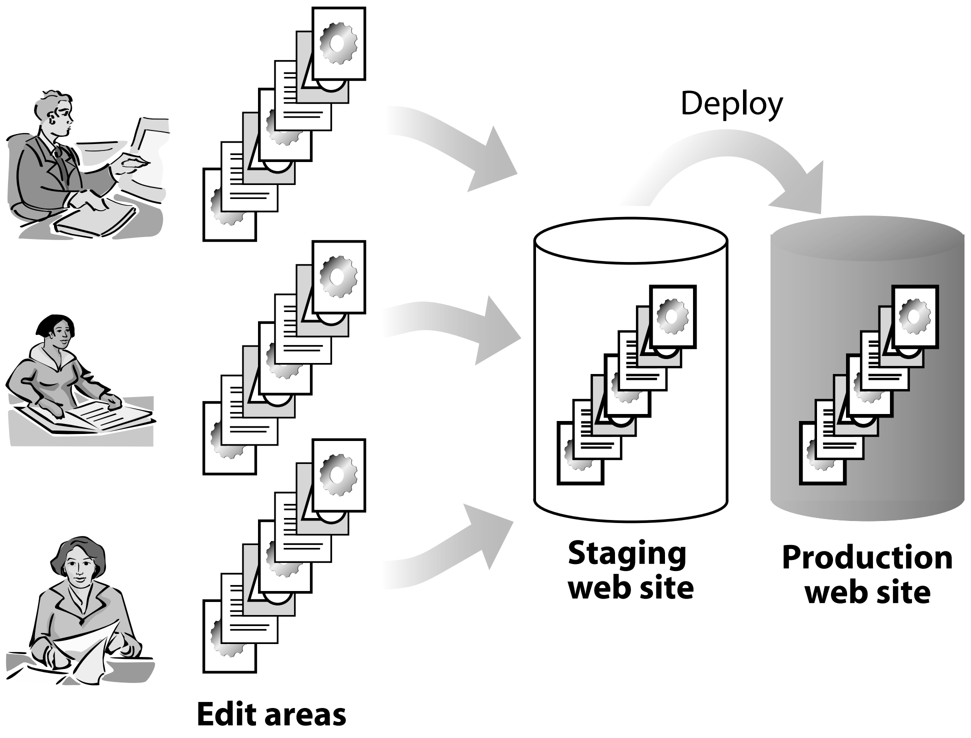

FIGURE 2.5
Developers on larger sites use separate edit areas to make and test changes.

version history is important, however, and it is wise to use this technique in conjunction with source code versioning tools.[5]

There are two drawbacks to this approach. First, as the number of independent areas increases to accommodate the developers, the total space consumed in the file system for each copy of the assets increases. Second, conflicts between areas become harder to keep track of. This happens because, with many developers, the likelihood of two people changing the same file increases. Let's suppose that two developers need to change `index.html`. One of them completes their change, and copies their change into the staging server. Assistance from a version control or content management tool is required to make a second developer aware that the changes to `index.html` are now

[5] A limitation of source-code versioning tools is that a person who isn't a software developer typically finds such tools difficult to use. The result is either frustration on the part of the developer, or a frequent "forgetfulness" in using the tool. Both are detrimental to the organization.

in conflict with the latest version of `index.html`. Without this kind of assistance, it is very likely that the second developer will overwrite the `index.html` in the staging server with their modified version of `index.html`. This effectively overwrites the changes from the first developer.

Content Management

When the asset count exceeds 2,000 to 5,000 files, or the web team exceeds 10–12 content developers and code developers, then it usually becomes necessary to adopt a content management tool. Formal support from a content management system overcomes the drawbacks of informal solutions, such as the ones described earlier. *Content management* is a discipline that manages the timely, accurate, collaborative, iterative, and reproducible development of a web property. (See Figure 2.6.) It combines a mechanism to store a collection of web assets with processes that seamlessly mesh the activities of people and machines within an organization. Content management responds to the unique combination of problems posed by web development.

Rita's gang should reasonably expect to support their activities with a content management approach by the time their group size reaches 12. If they allow time to evaluate, purchase, and implement such a system, and if they consider their rapid growth, they could reasonably start the process when their team is 8–10 people, depending on their hiring rate.

Since web efforts tend to expand quickly, both in terms of number of assets and size of staff, it often makes sense to introduce formal content management well in advance of crossing the asset and team size threshold suggested earlier. As rule of thumb, you should initiate the introduction of content management before your effort crosses the file count and team size thresholds, say, six months in advance. This gives time to evaluate tools, solicit budgetary approval, complete the purchase, implement the tool set, and train your staff, before the need becomes so critical that the absence of a content management solution impacts your business. In addition, the overall training cost

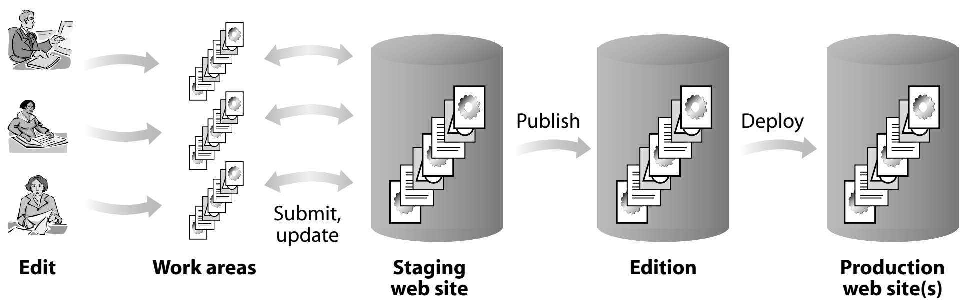

FIGURE 2.6
Content management orchestrates the development, testing, review, and deployment of web assets.

is lower if you introduce content management techniques when your staff is smaller, and fewer people become entrenched in bad habits.

Content Management Architecture

A content management infrastructure consists of subsystems that fulfill the following four functions (See Figure 2.7.):

- Content creation and editing
- Content repository and versioning
- Workflow and routing
- Deployment and operations management

Content Creation/Editing Subsystem

The content creation/editing subsystem consists of content editing tools, such as HTML editors, word processors, image editors, and XML editors. A wide variety of tools reflects the fact that each kind of content specialist, such as an artist creating an illustration or a telephone support associate creating a trouble ticket, becomes most efficient when that individual uses a tool suited to his or her expertise and problem domain. The job of the creation/editing subsystem is to accept input, effect appropriate processing on the input such as error checking, bounds checking, and whenever possible present direct feedback to tool users about the results of their actions. For

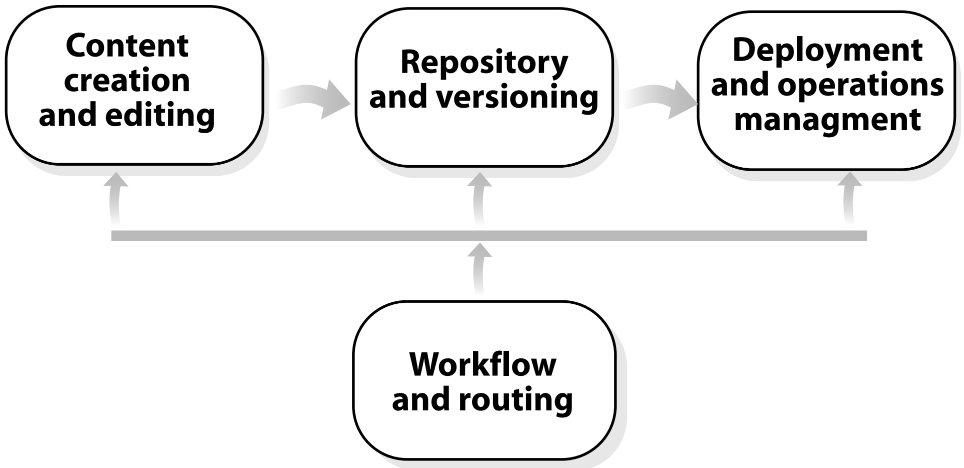
Content creation and editing
Repository and versioning
Deployment and operations managment
Workflow and routing

FIGURE 2.7
Four major subsystems of content management infrastructure.

example, developers creating servlets by writing Java code should see the effects of their code changes on an application server, preferably in real-time. We measure effectiveness of the creation/editing subsystem by the productivity of the content specialists that the subsystem serves. In addition, we can expect that the ability to support metadata tagging, using standard mechanisms such as XML, will become increasingly important.

Repository Subsystem

The repository subsystem provides storage, access, retrieval, indexing, versioning, and configuration management of content. Content includes files, database assets, and structured assets (e.g., XML). Access to content is available through standard means, including file system access, standard database access API's (ODBC, JDBC), and browser interface. We measure effectiveness of the repository subsystem according to its ability to store assets reliably, with scalability and with excellent performance.

Workflow Subsystem

The workflow subsystem manages assignments, routes jobs, handles notification to tie together the activities of the many specialists required to create, edit, test, review, and approve the plethora of simple and composite asset types in the repository. This subsystem needs to be aware of the content repository to allow job actions to be driven from repository changes, and conversely to have job actions to cause repository changes. Job specifications may be predefined statically, or ad hoc jobs may be created dynamically according to a "wizard" interface. The effectiveness of the workflow subsystem is measured by it ability to sustain work throughput, while offering simple means to define, modify, and specify new kinds of job routings.

Deployment and Operations Management

The deployment and operations management subsystem copies a variety of asset types from the development environment to production. It does so efficiently and reliably. Operation is 24 × 7. There needs to be simple mechanisms to provide positive assurance when systems work properly, and immediate notification and escalation when it detects an error state. Because operations often span the globe, crossing organization boundaries and multiple time zones, administration needs to be simple to set up, easy to learn, and provide detailed monitoring and audit trails. The effectiveness of the deployment and operations subsystem is measured by the reliability and throughput of the service, while minimizing the administrative effort.

Summary

The story of Rita and her gang illustrates predictable issues that confront a web operator. A web site swells from a skunkworks effort tended by one person to an enterprise-wide web site supported and maintained by hundreds of contributors. Some tend to the web site full-time, such as a Java developer implementing core functionality. Others contribute on a more casual basis, such as a contract writer adding a promotional piece against a predefined template.

Beneath the surface clutter of events, the swirl of growth in staff, and the expanded role of the web site in the business, the web site has a lifecycle of its own. It has a predictable set of universal lifecycle issues related to content management. Savvy managers consider these issues. We can see these issues in the fictitious story of Rita and her gang.

We have seen how the development of very small web sites, ranging in size up through enterprise-class web sites, can be managed with different approaches. (See Figure 2.8.) For a very small site, especially one that has a single developer, it is common for editing to occur on the live site. At the next level of sophistication, introduce

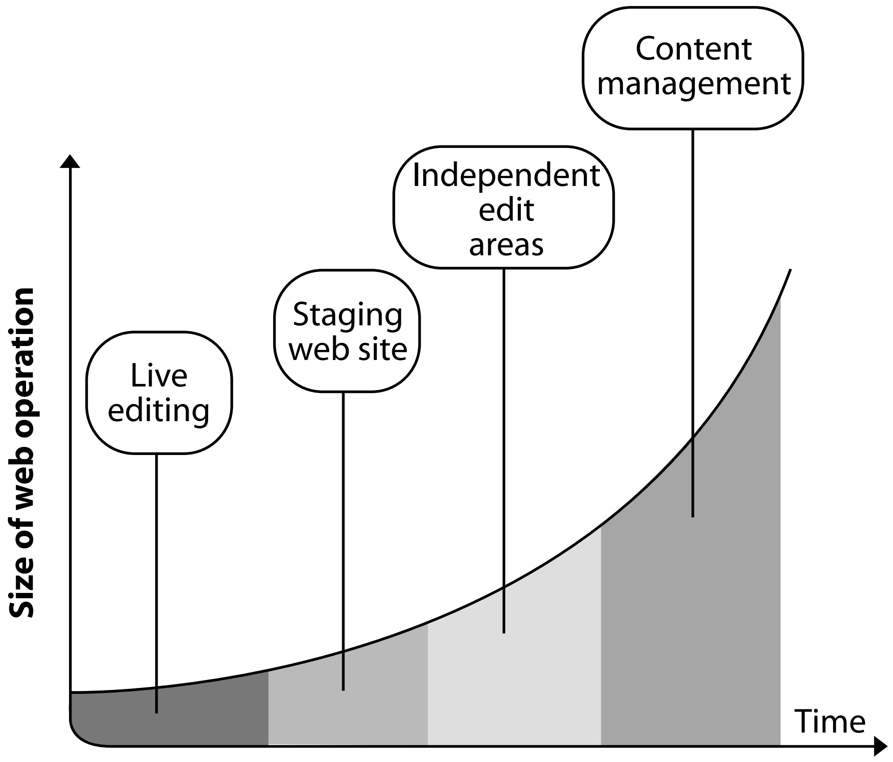

FIGURE 2.8
Choosing an approach to managing web assets depends on the size of your web operation.

a staging web site where one or more developers copy their changes for testing, review, and approval. For a larger team, or to support concurrent projects, introduce independent edit areas. Beyond 5,000 assets or a team of ten to twelve, it is highly recommended that a content management system be used.