7

Planning Your ClearCase Deployment

How often do you visit fortune-tellers? Not many of us make a habit of visiting the local palm reader before making major software decisions. But how much different is a visit to the friendly neighborhood soothsayer compared to many of the "shoot from the hip" decisions we make? If you're about to implement a very large ClearCase solution, you might find yourself cutting corners in the name of time. How different is corner-cutting from fortune-telling? Let's hear the rationalizations: You manage a team of smart, competent people who should know exactly what they're doing, right? The software comes with full documentation, and the default install process is fairly simple, right? The new software should plug right into your team's existing infrastructure and development processes, right?

Since we're leaving so much of your future to fate, let's look at your options from a fortune-teller's perspective: Down one path we see pain, misfortune, and possibly failure. This might equate to poorly

defined requirements, improperly defined handoffs and procedures, and a nonscalable architecture. Down another path, we see hard work but smart management of your intellectual property assets and, ultimately, success.

Designing and building a software configuration management solution is much like following the counsel of a palm or tarot card reader if you don't plan for it: without proper planning, you're letting fate decide your success. Remember the old adage, "If you fail to plan, you plan to fail."

Gosh, that always sounded so hokey when our parents or teachers threw it into a lecture or reprimand—and yet we've seen firsthand how poor planning has affected far too many teams and projects, usually with the justification of shaving a few man-hours from the schedule.

Here's another way to think about it: You're getting ready to take a long trip overseas. Do you just grab a backpack, throw in some odds and ends, and head for the airport? For most of us, of course not. You think about what you'll need, maybe you jot down a list of items to bring along. You do some planning, think about how many days you'll be gone, which clients you'll see and in what social settings, the weather at your destination, and the extracurricular activities you might partake in. Yes, you could throw into your bag an extra pair of shoes, some socks, and a few pairs of underwear and probably make it for a few days. But is skipping some planning steps up front really worth all of the trouble? Four days down the road, you don't want to be stuck in a meeting with clients in Singapore wearing the same outfit you wore on the plane, without fresh socks and underwear . . . metaphorically speaking, of course.

Planning to Plan

It's quite simple: you need to plan. Here's a start: think about all of the actors. Understand their relationships. Build your use cases. Apply them to your business rules for validation. And then build

your plan. It all begins with a few targeted questions. But do you even know which questions to ask?

The questions posed here are nothing but a starting point to building a system and tool set that meet your organization's long-term development needs. We all suffer from "writer's block" from time to time, so think of this list as a tool to help grease the wheels in your brain. Here's where we recommend you start.

How Big Do You Think the Project Will Be?

This is an interesting question and oftentimes difficult to answer. How do you determine the size of the project? Do you use lines of code (LOC), number of classes, function points, or what? In most cases, the LOC is a good start for determining the size of the project. You can use several different methods to project the LOC for a project. Check out *A Discipline for Software Engineering,* by Watts Humphrey (Addison-Wesley, 1995), for some ideas.

How Many People Will Be Involved in the Product Development?

The size of the project is determined not only by the amount of code you will produce, but also by the number of people on your development team. Many times you can "guesstimate" the number of people that you will need on the team. When doing this, don't forget about the supporting roles of software development: QA, project managers, technical writers, and even managers. These people need to access the code and can therefore benefit from using ClearCase just as much as—if not more than—the software engineers.

Some might say, "We don't need to worry about the size of our VOBs because the new ClearCase architecture allows us to have really large VOBs." Just because you can have *huge* versioned object bases (VOBs) does not mean you should have them. Let's not forget that segmentation and the use of multiple VOBs have their benefits.

At How Many Locations Will the Product Be Developed, Tested, and Deployed?

If your first inclination is to answer, "Well, we're only here at the one location," or that your team can just telnet into the central office to do their work, then your project is either really small or you are not thinking big enough. With the expansion of broadband communications throughout the world, more companies are looking for cheaper ways to get teams to work together. It is expensive to move your whole team to the Silicon Valley when you can just as easily put a VOB server in each employee group's geographical location. Or maybe the product is being tested or developed overseas. The number of locations and the work that each location is performing will play an important role in your VOB architecture, triggers, and integration with third-party tools.

Is the Product External or Internal?

If the product is external, what delivery mechanism are you going to use? Are you burning a CD, are you providing an Internet download, or are you supplying a service? Can you use a VOB to control your releases? How often do you need to release the product to the customer? Is the product mission-critical for your customers?

When a product is internal, you typically have to deal with internal integration problems in addition to the normal external customer problems. Are other teams going to use your project to develop their product? Are they using ClearCase? (In a perfect world, everyone is using ClearCase.) How often can they get releases from your team? And what kind of coordination do you need to provide?

What Third-Party Tools Will You Be Using?

How many third-party tools do you plan to use? Make a list of all of the third-party tools that your development team (that is, your expanded team, as mentioned previously) is using. Don't forget about project management, testing, and technical writers. It makes sense to have some kind of VOB strategy for these tools. This includes

things such as OS patches, compilers, Quantify, Purify, FrameMaker, MSWord, and so forth. There is nothing worse than having to spend weeks patching a mission-critical product that is over five years old, and you don't have access to all the tools you need to build, test, and document the thing.

What Is the Development Cycle of the Product?

Are you using a traditional waterfall method to develop your product? Is your cycle time Internet-fast (three months) or telecom-slow (seven years)? This can play an important part in the way you lay out your VOBs. You want to make sure your VOBs and ClearCase are not getting in your way, and actually help you get your job done faster. If you have a plan, you can use triggers to help enforce policy and automate things that are typically done by hand.

If you cannot answer this question about development cycle, let us point you to the Rational Unified Process. It is a good place to start for those in need of some direction.

How Do Your Current Development Methodologies Fit into Your Tool Plans?

We have seen far too many groups change their process to fit their tool set. This can be a dangerous exercise in bureaucracy. You might find processes that nobody can remember instituting, or that have no defined purpose. We have even seen processes created in an effort to circumvent the original process that no one could remember creating or instituting. Most of the time, we view these as shortcomings of the tools your team uses. It's important to find the right tools to fit your methodologies. Most today are very flexible and customizable and can handle whatever processes you might come up with.

Are Any Key Roles Missing from Your Team?

Make sure you have included everyone in the development team you need. Think through several scenarios. Were you asked recently for

information about the project from someone who you don't have on your list as a key player in the development of the product? What was his or her role, and should your team be expanded to include this individual? If you are working with more than one location, who is the person responsible for coordinating meetings with all locations? Who is your multi-site administrator? Does your customer need access to ClearCase release VOBs? Make sure you have everyone written down.

What Are the Security Issues?

Don't fall into the trap of believing that just because you are behind a firewall your project is automatically secure. There are several issues that could pop up which you might not have considered. For example, you may have customers that want you to test your product against their proprietary systems, or some other protected intellectual property. You may need to restrict access to parts of your data while keeping other parts open between the customer and your development team. Another consideration: how can you protect your code from theft by a former employee? These are questions you need to ask.

What Types of Artifacts Are You Creating?

If your answer is, "I produce code—nothing else," then you are actually in the very small minority. Most products include documentation, binaries, project plans, images, and various marketing collateral. Map out all of the types of artifacts you need to version. A good rule of thumb is this: anything that is shipped or consumed by the end customer should be put in the source control system.

Do You Have the Hardware You Need?

It is really hard to deploy a development environment and ClearCase without hardware. Although it can be done, there is nothing worse than putting your VOB server on someone's desktop machine. You never know when you will have a janitor turning machines off in the middle of the night to save energy or, more commonly, have the air

conditioning in your office turned off on the weekends to save money. That one is always fun. Make sure you have the hardware that you'll need up front. It is much easier to do it right the first time than to try and fix things later.

Do You Have the Infrastructure Ready to Support Your Plans?

Another item that people often forget is the infrastructure to support their development plans. Air conditioning, power, floor space, and networking are important to ensure you have a successful development environment. For instance, if you have to boot your machines in a specific order so you don't throw circuit breakers, you definitely need to get more power. If the $12.99 "blue light special" fan from Kmart keeps your computer from overheating, you need more A/C. Another thing to consider: if you are developing in multiple locations, you will need to make sure that you have connectivity to all locations (it's the little things—like connectivity—that always come back to haunt you). The hardest part, of course, is convincing someone who has the money to make the investment. If you do the work up front and have your project properly planned, it will be much easier to convince the senior executives to support these kinds of expenditures.

Preparation is the key to successful deployment of ClearCase, pure and simple. Ask the right questions, gather the right requirements, connect to all of the necessary tools and processes—and you've got it made.

Now You Know . . . and Knowing Is Half the Battle

Planning how to set up your VOBs and ClearCase environment can seem like you're trying to peer into the future, because you need to predict the future of the product. But unlike the telephone fortune-tellers who make baseless recommendations on how to live your life,

you need to start asking questions—and making some key decisions based on the answers you get. Many efforts start out as if they're relying on fortunetellers, but as you slowly start answering these questions, your project will become a self-fulfilling prophecy. By properly planning things, you rely less on "predictions" and greatly reduce the risk.

Smart deployment is all about risk management—planning reduces risk. And properly planning for a strong source-code management system will roll over into other practices. With a strong set of tools and processes driving your development machine, you'll find your team planning their development efforts with more rigor and logic. The resulting products will be . . . well, better.

Think of it this way: when you were growing up, did you ever notice the difference between doing homework in a messy room versus one that was clean, from top to bottom? Clinical research shows that students in a clean and orderly environment excel over those in a messy environment. So how different is this from your work environment?

How clean is your room?

8

Modeling Your Configuration Management System

Bridging the CM Gaps

Building a bridge is one of the most fundamental ascents of mankind over nature. Few endeavors over the last 100 years have posed such risk while also providing such great utility, linking regions and communities otherwise locked away behind distance, and usually water. Without the vast systems of roads and bridges, humans would be forced to travel less efficiently, and society would be disjointed.

In the United States, for example, the development and success of two major geographic regions came about largely due to the building of bridges.

The island of Manhattan is surrounded by major waterways, including the East River, the Hudson River, New York Harbor, and the Harlem River. Before the development of our modern modes of

transportation, travelers depended on boat and ferry service—but this could be unreliable and even dangerous, particularly in winter, when weather often prevented boats of any size from crossing between Manhattan Island and the mainland. In these early days, the construction of a bridge was a major event, allowing remote communities to expand and extend themselves across an entire region. For Manhattan, the Brooklyn Bridge was the first major connector to the island, and it was quickly followed by a system of bridges.[1]

On the opposite side of the country, the communities of San Francisco and those in the North Bay shared the century-long dream of spanning the Golden Gate Strait, which is the entrance to the San Francisco Bay from the Pacific Ocean. The strait, a treacherous body of water with currents ranging from 4.5 to 7.5 knots, is approximately three miles long by one mile wide. Construction of the Golden Gate Bridge commenced on January 5, 1933, with the bridge opening on May 28, 1937.[2]

Bridge over Troubled Waters

There are bridges in software development, as well. Building a configuration management system can be a daunting task for any organization. Yet many teams, in the rush to solve immediate problems and focus their time on "billable" development activities, will push some kind of CM solution out the door without properly analyzing the organization's needs, short term and long term.

As we're sure you've all seen, far too many teams are tackling problems without understanding the full scope of the project in front of them. And yet, time after time, that's how projects get under way—with arbitrary deadlines—and usually behind schedule from the start. Customer demands and project deliverables require a speedy implementation. But a speedy deployment and a well-planned and well-developed system do not have to be mutually exclusive.

1. http://users.commkey.net/Daniel/bridge.htm
2. http://www.goldengatebridge.org

One common management dream is to start on a project at the beginning of the lifecycle, rolling out process and procedure at the inception phase of development. However, this is rarely the case. Luckily, configuration management systems are available to aid in your development efforts. They can be a powerful asset for increasing communication, productivity, and quality through process automation and integration of the tools that most engineering groups use today. You could call CM the "bridge" that links your company's development teams. Implementing some kind of CM solution will organize your development efforts around solid and repeatable processes—and by helping your team prioritize and manage the development lifecycle, you are more likely to meet your customers' needs.

As we've stated previously, configuration management and the teams that manage these tools are in a unique position because they are the glue between engineering and the rest of the product development structure in your organization. The product cannot move forward in a timely manner unless the CM team coordinates with the build-and-release teams and, generally, manufacturing. And unless your project is proceeding quickly, it's a safe bet that your customers are not happy.

Now . . . back to the steps that lead you to the solution. The process of determining the path and structure of the system as it is applied to your organization is called *business modeling*. The purpose of business modeling is to determine who and what the customer is—but not necessarily the requirements of the project. The point is to seek to understand the client's perspective, and not make any judgments about possible solutions or what the customer thinks he or she needs.

The purpose of this chapter is not to walk through the dos and don'ts of business modeling—there are plenty of chapters available on the Rational Developer Network on the subject.[3] It's probably safe to say

3. For an excellent review of the whys and hows of business modeling, we recommend "Business Modeling with UML," by Bryon Baker. It's available online at http://www-106.ibm.com/developerworks/rational/library/content/RationalEdge/dec01/BusinessModelingwithUMLDec01.pdf

that business modeling is the most undervalued part of the software development process. With that said, we feel it is important to remind everyone that we are not trying to identify the requirements at this stage. The business model is the "problem domain" while the requirements are the "solution domain," which we address later in the development process. Instead, it is critical to know what you are building and why.

> Models help a software development project team visualize the system they need to build that will satisfy the various requirements imposed by the project's stakeholders. The aphorism "A picture is worth a thousand words" holds especially true when it comes to work involved in developing software: much of what's interesting about a given system lends itself to visual modeling better than plan test does. The UML is specifically designed to facilitate communication among participants in a project.
>
> ~~*From*~~ UML Explained, *by Kendall Scott (Addison-Wesley, 2001)*

Identifying Actors

When analyzing a system, we first like to draw a box around it, to define the boundaries of that system. This is best done by defining the actors—the people, software, hardware, and others— that interact with the system. In the configuration management world, we can usually write down a list of actors that participate in a CM system fairly quickly. Remember: think of the CM system as a black box. Just make sure you define all of the outside parts of the black box.

Figure 8.1 is a diagram of the actors within a basic CM system, along with a brief description of each.

Software Engineer

The software engineer is responsible for the design and development of the software that makes up your product. The software engineer will use the configuration management system to store designs, code, and sometimes tests, so that they can be integrated into a releasable product.

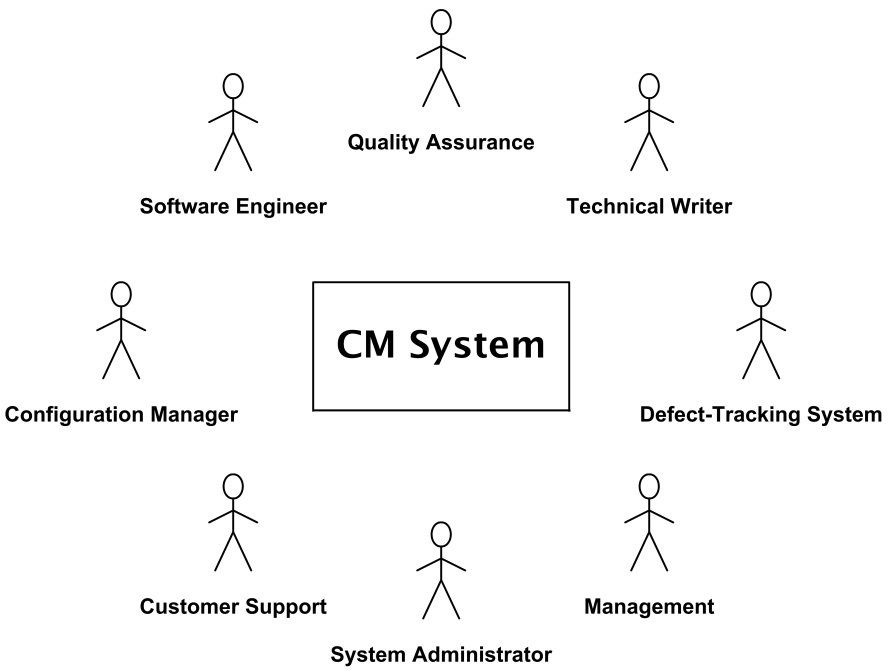

Figure 8.1 *Actors*

Configuration Manager

Those of us who come from the CM world might consider the configuration manager as the most important actor in any CM system. It is typically the CM manager's job to control the "crown jewels" of the company: the source code. We typically see the CM people not only administrating ClearCase and the process of software development, but also performing the builds and releases of the product as well. This is especially true in smaller organizations, where resources are tight.

Quality Assurance (Product Validation)

We prefer the term *product validation* to *quality assurance,* because these individuals do much more than check the product to find bugs the software engineers created. These teams play a very important

role as the final validation of the product use cases before going to the customer. They will use the CM system to store their tests and test suites, and to get controlled release of the product from the configuration manager.

Management

Whether we like it or not, management is a necessary evil. We deal with management on a daily basis (some days better than others), and as the providers of the company's CM solution, they recognize that we supply them with the information they need to make decisions about product direction and strategy. The CM system contains the information that management needs, but it is our job to present the data in a manner that is useful to them.

Customer Support, or a CRM System

Another necessary actor is the customer support organization. Customers just might have problems with your software—no matter how well it was designed—and you need to be able to give information to your customer support organization about product releases, customer requests, and bugs that have been fixed in product releases. This information can either flow directly to the support team or be tracked within a CRM solution.

Defect-Tracking System

This can be part of the system or not. If you do not define this as an actor, you should at least be ready to include it in your system later down the road. If you are not integrating your CM solution with a defect-tracking system, you should seriously consider this option. Joining the defect data with the source code can help you make improvements in project management and the overall quality of your product.

Technical Writer

Make sure you include everyone who will develop artifacts for your product. There is nothing worse than getting documentation that is two versions behind the shipped software. If your tech writers are not using ClearCase, then teach them how to use it. If it is too hard, then write the scripts that they can use to make their jobs easier. Everything will run much more smoothly if they at least know the basics. Just keep it simple.

System Administrator

Your configuration manager is not a system administrator. Of course, in a small organization they could be the same person, but these roles are very different. The system administrator is there as a support for your hardware and, potentially, software problems. He or she should be making sure that the systems are configured according to product specifications and fine-tuned to the requirements of the CM team.

Use Cases

Now you need to look at how the actors use the system. As shown in Figure 8.2, for each actor, write down how the actor will use the system—or how the system will use the actor. Draw an arrow in the direction of information flow. Don't forget the administration use cases, such as "Backup the Source Code," "Restore from Backup," and so on. Don't include use cases that do not involve the CM system. For example, "Reboot Machine" is probably not a good use case, because it does not directly involve the CM system.

Pick the most used use cases and start working from there. Look at how the actor will use the system and how the system will guide the user. The CM world has some well-defined scenarios, so try to focus on what is different about what your team does compared to other teams.

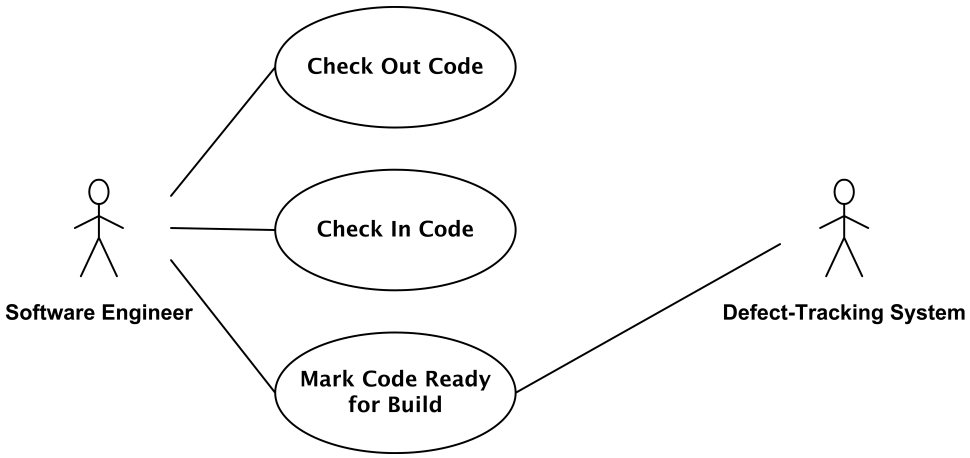

Figure 8.2 *Sample Use Case*

Activity Diagrams

The next step is to look at activity diagrams and process analysis. The first thing you will find out is that your organization most likely already has some process in place (whether you like it or not). Whether or not there are formal guidelines for development, people typically work the same way over and over again. They get into a mode of working so they don't have to think about what comes next. They would rather spend their time working on new things instead of worrying about what is the next step in the process. What you need to do is figure out what your team is currently doing and model it. Then you can find ways to optimize it.

Don't fall into the trap that process engineers run into time and time again: "I am going to develop a new process that will make everyone more productive." In reality, it usually has nothing to do with the team culture or habits, and it hinders more than helps the organization. First, find out what your team is doing and then try to change with little steps of improvement.

Figure 8.3 is an example of an activity flow of a typical CM system.

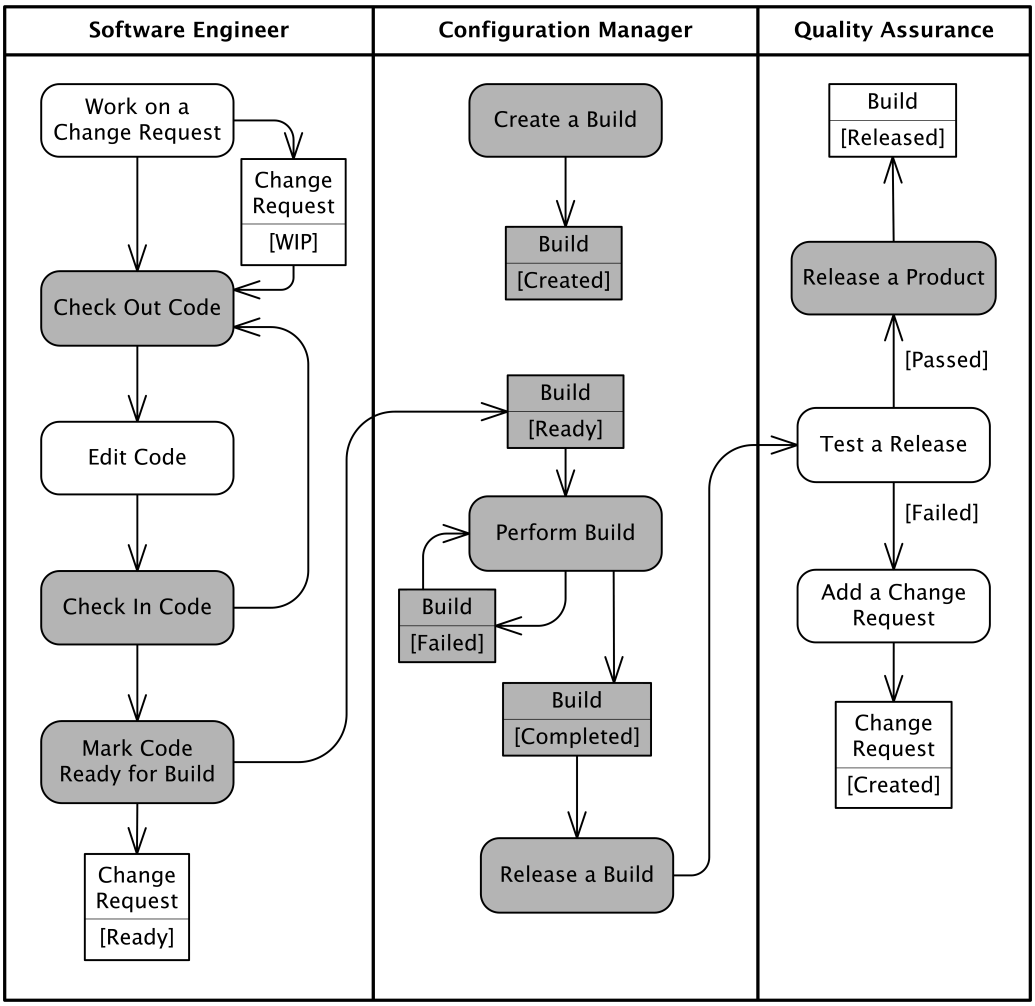

Figure 8.3 *Sample Activity Diagram*

Component Diagrams

One often-overlooked aspect of designing a configuration management system is the definition of components. Although you won't know what all of the components of your system are at the beginning, with your knowledge of the system you should still be able to group

things pretty well. You should also consider the actors of your CM system when defining components. Don't forget about the product's supporting files, such as licenses, documentation, releases, and so forth.

Consider the following small project named "Kish," as illustrated in Figure 8.4:

- kish_adm—Administration VOB for labels, branches, triggers, and supporting CM scripts.
- kish_process—This is a process VOB. UCM likes to use this to store information about the project in the VOB.
- kish_doc—Documentation VOB. Your technical writers need a place to put their information.
- kish_src—Most products have some kind of source that gets compiled.
- kish_release—We prefer to store the releases in a separate VOB from the source. It gives us the opportunity to multi-site the VOBs separately and scrub them differently.
- kish_test—Most test harnesses become larger than the code base itself. This is great again for multi-site purposes.

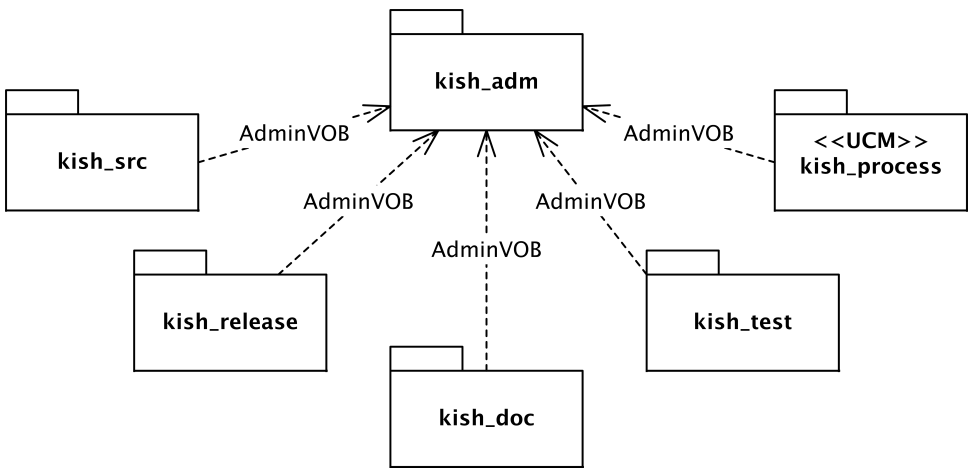

Figure 8.4 *VOB Layout*

These VOBs can be broken down into more VOBs, according to the project size. We will talk about that in the next chapter.

Deployment Diagrams

For the context of this CM effort, deployment diagrams, as shown in Figure 8.5, can be used for three things:

1. To show the different machines you need to support the size of your team: VOB servers, view servers, build machines, test machines, and workstations.
2. To show the different platforms that you need to support. This is overlooked far too many times, and it can have an impact on the way you set up your VOB structure and your multi-site strategy.

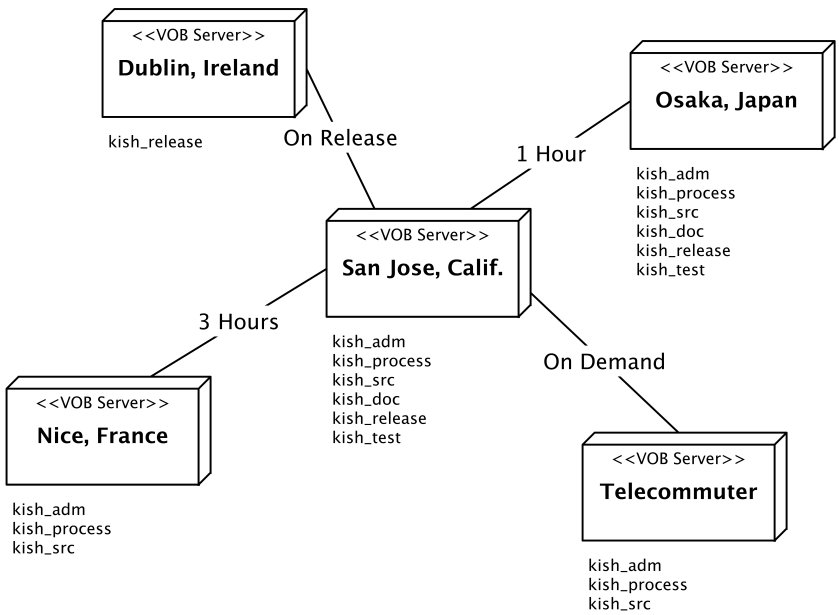

Figure 8.5 *Sample Deployment Diagram*

3. To show the different locations that will be working on the product and the VOBs that they need. This helps you with your multi-site strategy dramatically.

Bridge Building

Well, we've attempted to illustrate clearly the steps to designing your new CM solution, and hopefully you've figured out how to get started on this planning phase. As outlined in their book *Use Case Driven Object Modeling with UML* (Addison-Wesley, 1999), Doug Rosenberg and Kendall Scott summarized the goals of use case modeling:

- You've built use cases that together account for all of the desired functionality of the system.
- You've produced clear and concise written descriptions of the basic course of action, along with appropriate alternate courses of action, for each use case.
- You've factored out scenarios common to more than one use case.

Once you have accomplished these things, you should be ready to design the next stage of your CM system.

Remember: bridges don't just happen. They take years of planning, followed by systematic implementation. The result is a resource that can improve the lives of those who use it. The same can be said for a solid configuration management system.

Our next few chapters will continue to explore the different aspects of configuration management deployment, including VOB architecture, security and data integrity, integration of other back-office applications, and customer-facing deployment strategies, such as training and support.