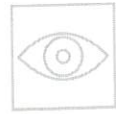


chapter **3**

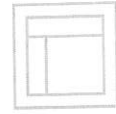
The Strategy Plane

Site Objectives and User Needs

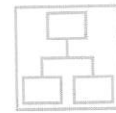
The foundation of a successful user experience is a clearly articulated strategy. Knowing both what we want the site to accomplish for our organization and what we want the site to accomplish for our users helps inform all the decisions we have to make about every aspect of the user experience. But answering these simple questions can be trickier than it looks.



Surface



Skeleton



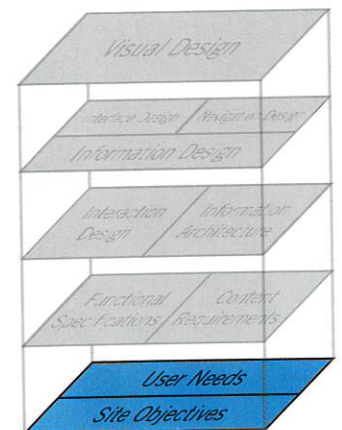
Structure



Scope



Strategy

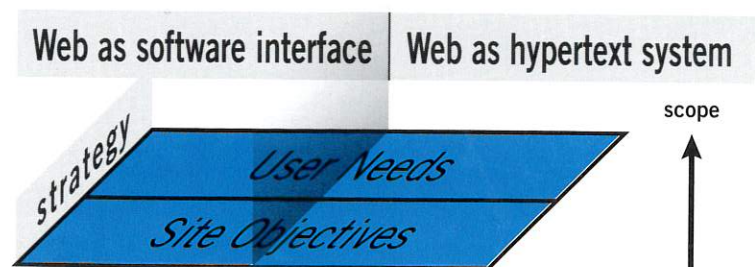


Defining the Strategy

The most common reason for the failure of a Web site is not technology. It's not user experience either. Web sites most often fail because, before the first line of code was written, the first pixel was pushed, or the first server was installed, nobody bothered to answer two very basic questions:

- ▶ What do we want to get out of this site?
- ▶ What do our users want to get out of it?

By answering the first question, we describe the **site objectives** coming from inside the organization. The second question addresses **user needs**, objectives imposed on the site from outside. Together, site objectives and user needs form the strategy plane, the foundation for every decision in our process as we design the user experience. Yet, amazingly, many user experience projects do not begin with a clear, explicit understanding of the underlying strategy.



The key word here is *explicit*. The more clearly we can articulate exactly what we want, and exactly what others want from us, the more precisely we can adjust the choices we make to meet these goals.

Site Objectives

The first part of making our understanding of the strategy explicit is examining our own objectives for the site. Too often, site objectives exist only as an unspoken understanding among those building the site. When that understanding remains unspoken, different people often have different ideas about what the project is supposed to accomplish.

Business Goals

People commonly use terms like “business goals” or “business drivers” to describe internal strategic objectives. I’m going to use the term “site objectives” because I think these other terms are both too narrow and too broad: Too narrow because not every internal goal is a business goal (after all, not every organization has the same kinds of goals that businesses do), and too broad because our concern here really is to identify in the most specific terms possible what we expect the site itself to accomplish, the rest of our activities notwithstanding.

Most people start out describing site objectives in very general terms. At the most fundamental level, business Web sites generally exist to serve one of two purposes: to make the company money or to save the company money. Sometimes it’s both. But exactly how these sites are supposed to do that is not always clear.



On the other hand, objectives that are too specific don't adequately describe the strategic concerns at issue. For example, stating that one of your objectives is "to provide users with a Java-based real-time communications tool" doesn't explain how such a tool helps advance the objectives of your organization, or how it helps meet the needs of your users.

In trying to strike a balance between being too specific and being too general, we want to avoid jumping ahead to identify solutions when we don't yet fully understand the problems. To create a successful user experience, we have to make sure that nothing is determined by accident—that every decision we make is rooted in a firm understanding of its consequences.

Brand Identity

One essential consideration in formulating the objectives for any site is brand identity. When most of us see the word "branding," we think of things like logos, color palettes, and typography. While these visual aspects of brand are important (we'll revisit them in more detail when we get to the surface plane in Chapter 7), the concept of brand extends far beyond the visual. Brand identity—a set of conceptual associations or emotional reactions—is important because it's inescapable. In the minds of your users, an impression about your organization is inevitably created by their interaction with your site.

You must choose whether that impression happens by accident or as a result of conscious choices you have made in designing your site. Most organizations choose to exert some control over the perception of their brand, which is why communicating brand identity is a very common site objective. Branding isn't just for commercial entities either—every organization with a Web site, from non-profit foundations to government agencies, creates an impression through user experience. By codifying the specific qualities of that impression as an explicit objective, you increase your chances that it will be a positive impression.

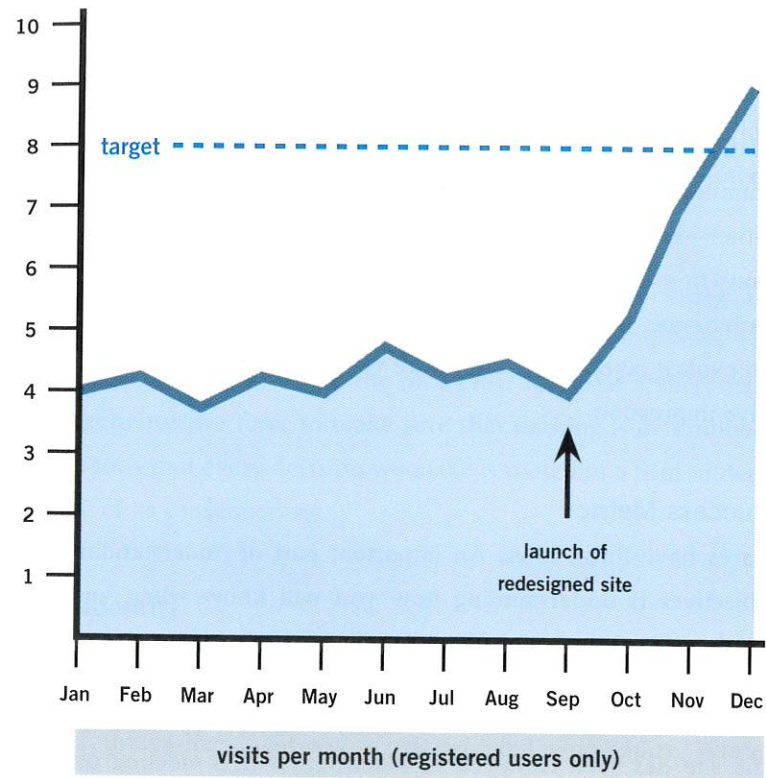
Success Metrics

Races have finish lines. An important part of understanding your objectives is understanding how you will know when you have reached them.

These are known as **success metrics**: indicators we can track after the site has been deployed to see whether it is meeting our own objectives and our users' needs. Good success metrics not only influence decisions made over the course of the project, they also provide concrete evidence of the value of user experience efforts if you find yourself facing a skeptical audience when seeking budget approval for your next user experience project.



Success metrics are concrete indicators of how effectively the user experience is meeting strategic goals. In this example, measuring the number of visits per registered user per month indicates how valuable the site is to its core audience.



Sometimes these metrics are related to the site itself and how it is used. How much time does the average user spend on your site during each visit? (Your server logs can help you determine this.) If you want to encourage your users to feel comfortable with the site, hang out, and explore what you have to offer, you'll want to see the time per visit increase. On the other hand, if you want to provide quick, get-in-get-out access to information and functionality, you may want to decrease the time per visit.

For sites that depend on advertising revenue, page views—the number of times each day a page on your site is requested—is an incredibly important metric. But you have to be careful to balance your objectives and the needs of your users. Adding several layers of navigational pages between the home page and the content users want will definitely increase your page views, but is it serving user needs? Probably not. And in the long run, it will show: as your users get frustrated and decide not to come back, your page views will drop from that initial high and will probably end up lower than they were when you started.

Not all success metrics have to be derived directly from your site. You can measure the indirect effects of the site as well. If your site provides solutions to common problems people encounter with your product, the number of phone calls coming into your customer support lines should go down. An effective intranet can provide ready access to tools and resources that can cut down on the time it takes for your salespeople to close a sale—which, in turn, translates directly into increased revenue.

The metrics that work best are those for which a change can be directly attributed to the user experience of the site. Of course, when a redesign launches and daily revenue from online transactions jumps 40 percent, it's easy to see the relationship between cause and effect. But for changes that happen over a longer period of time, it can be difficult to identify whether those changes stem from the user experience or from other factors.

For example, the user experience of your site can't do much by itself to bring new users to your site—you'll have to rely upon word-of-mouth or your marketing efforts for that. But the user experience has a whole lot of influence on whether those visitors bother to come back. Measuring return visits can be a great way to assess whether you're meeting user needs, but be careful: Sometimes those users don't come back because your competitor launched a gigantic advertising campaign or because your company just got some bad press. Any metric viewed in isolation can be misleading; be sure to take a step back and look at what's going on beyond the Web site to make sure you're getting the whole story.

User Needs

It can be easy to fall into the trap of thinking that we are designing our site for one idealized user—someone exactly like us. But we aren't designing for ourselves; we're designing for other people, and if those other people are going to like and use what we create, we need to understand who they are and what they need. By spending time researching those needs, we can break out of our own limited perspective and see the site from the point of view of the users.

Identifying user needs can be complicated because users can be quite diverse. Even if we're creating a site for use inside our organization, we still may have to address a wide range of needs. If our site is intended for a consumer audience, the possibilities increase exponentially.

User Segmentation

We can break this mass of user needs down into manageable chunks through **user segmentation**. We divide our audience into smaller groups (or segments) consisting of users with certain key characteristics in common. There are nearly as many ways to segment user groups as there are types of users, but here are a couple of the most common approaches.

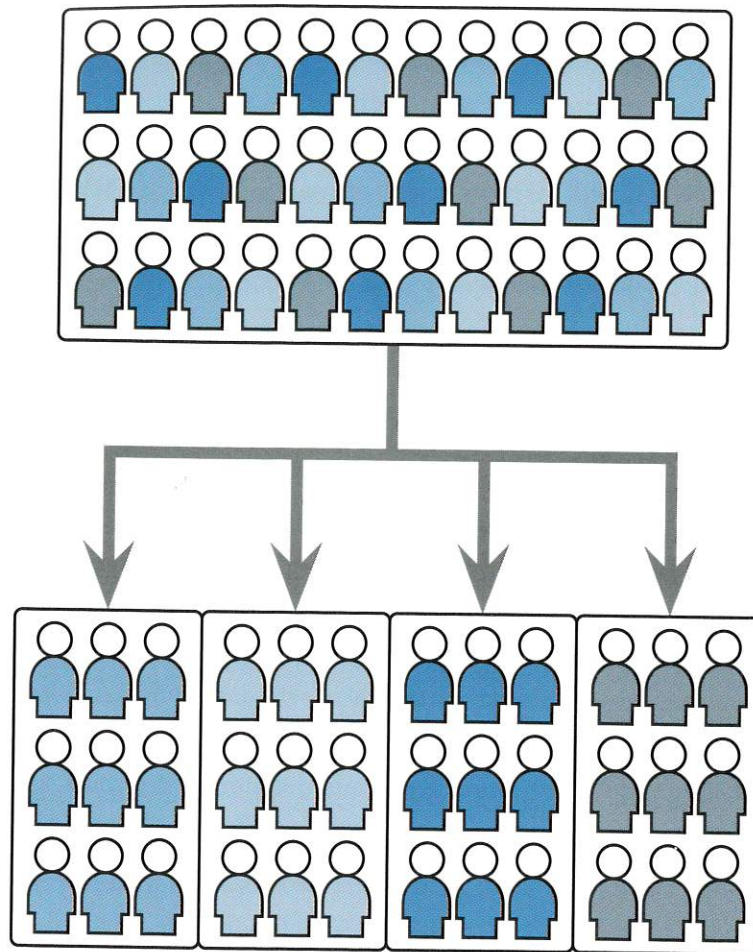
Market researchers commonly create audience segments based on **demographic** criteria: gender, age, education level, marital status, income, and so on. These demographic profiles can be quite general (men 18–49) or very specific (unmarried, college-educated women 25–34 making over \$50,000 a year).

Demographics aren't the only way you can look at your users. **Psychographic** profiles describe the attitudes and perceptions that your users have about the world or about the subject matter of your site in particular. Psychographics often correlate strongly with demographics: People in the same age group, location, and income level often have similar attitudes. Still, documenting the psychographics of your users can give you insights you can't get from demographics.

When developing Web sites, there's another very important set of attitudes to consider: the users' attitudes toward technology and the Web itself. How much time do your users spend using the Web every week? Are computers a part of their daily lives? Do they like working with technology? Do they always have the latest and greatest hardware, or do they only buy a computer every five years? Technophobes and power users approach Web sites in very different ways, and our designs need to accommodate them. Answers to questions like these can help us do just that.



User segmentation helps us understand user needs better by dividing the entire audience into smaller groups of people with shared needs.



In addition to understanding our users' familiarity and comfort level with technology, we need to understand what and how much they know about the subject matter of our site. Selling cookware to people just learning their way around a kitchen must be handled very differently from selling to professional cooks. Similarly, a stock-trading application used by those unfamiliar with the stock market will require a different approach from one for seasoned investors.

The way people use information often depends on their social or professional role. The information needs of the parents of a student applying for college are different from those of the student herself. Identifying the different roles of the users who come to your site can help you separate and analyze their different needs.

After you've conducted some research on your user groups, you might need to revise the segments you are working with. For example, if you're researching 25–34 year old, college-educated women, you might find that the needs of the 30–34 year olds differ from those of the 25–29 age group. If the difference is great enough, you might want to treat these as separate groups, rather than the single 25–34 group above. On the other hand, if the 18–24 group seems pretty similar to the 25–34 group, maybe you can combine them. Creating user segments is just a means to the end of uncovering user needs. You really only need as many different segments as you have different sets of user needs.

There's another important reason to create user segments. Not only will different groups of users have different needs, but sometimes those needs will be in direct opposition. Take the preceding example of the stock-trading application. The novices would probably be best served by an application that broke the process down into a sequence of simple steps. For the experts, however, such a sequence would be a hindrance. The experts need a unified interface that provides rapid access to a wide range of functions.

Obviously, we can't meet both sets of user needs with a single solution. Our options at this point are to focus on one user segment to the exclusion of the other, or to provide two separate ways for users to approach the same task. Whichever course we choose, this strategic decision will have consequences for every additional choice we make about the user experience.

Usability and User Research

If you've done any reading about Web design at all, you've probably come across the word **usability**. The concept means different things to different people. Some people use it to refer to the practice of testing designs with representative users. For others, it means adopting a very specific development methodology. But every approach to usability seeks to make products easier to use.

Many different definitions and lists of rules intend to codify what constitutes a usable Web site design. Some of them even agree with each other. But they all have the same principle at their core: Users need usable products. It's really the most universal user need of all.



To understand what our users need, we first have to get a sense of who they are. The field of **user research** is devoted to collecting the data needed to develop that understanding.

Some research tools—such as surveys, interviews, or focus groups—are best suited for gathering information about the general attitudes and perceptions of your users.

Other research tools—such as user tests or field studies—are more appropriate for understanding specific aspects of user behavior and interaction with the site.

Generally, the more time you spend with each individual user, the more detailed the information you will get from the research study. At the same time, that additional time spent with each user necessarily means you won't be able to include as many users in the study (if only because the site has to launch eventually).

Market research methods like surveys and focus groups can be valuable sources of general information about your users. These methods are most effective when you clearly articulate for yourself what information you're trying to get out of them. Do you want to find out what your users are doing when they use a particular feature of your site? Or maybe you already know that, but you need to know why they're doing it. The more clearly you can describe what you want, the more narrowly and effectively you can formulate the questions you ask to ensure that you get the right information.



Contextual inquiry refers to a whole set of methods that, collectively, form the most powerful and comprehensive toolkit for understanding your users in the *context* of their everyday lives (hence the name). These techniques are derived from the methods used by anthropologists to study cultures and societies. Applied on a smaller scale, the same methods used to examine, for example, how a nomadic tribe functions, can also be used to examine how people who buy aircraft parts function. The only downside is that contextual inquiry can sometimes be very time-consuming and very expensive. But if you have the resources, and your problem requires a deeper understanding of your users, contextual inquiry can reveal subtleties of user behavior that can't be discovered through other methods.

One method closely related to contextual inquiry is **task analysis**. The idea behind task analysis is that every user's interaction with a Web site takes place in the context of some task that user is performing. Sometimes the task is very focused (such as buying movie tickets) and sometimes it's broader (such as learning about international commerce regulations). Task analysis is a method of closely examining the precise steps users go through in accomplishing those tasks. This examination can be done either through interviews in which you get users to tell you stories about their experiences or through direct observation in the field, observing the users in their "natural habitat."

User testing is the most commonly employed form of user research. User testing is not about testing your users; instead, it's about getting your users to test what you've produced. Sometimes user tests work with a finished site, either in preparation for a redesign or to root out any usability issues before launch. In other cases, users can test a work in progress or even a rough prototype of the finished site.



Tests with a fully operational Web site can be very broad or very narrow in scope. As with surveys or focus groups, it's best if you have a clear sense of what you want to investigate before you sit down with users. That doesn't mean, however, that a user test has to be strictly limited to assessing how successfully users complete a narrowly defined task. User testing can also investigate broader, less concrete issues. For example, a user test could be used to find out whether modifications to the site design reinforce or undermine the company's brand message.

Another approach to user testing is to have users work with prototypes. These can take a variety of forms, from rough sketches on paper, to "lo-fi" mockups using stripped-down HTML pages, to "click-through" prototypes that create the illusion of a finished site. Larger-scale projects employ different kinds of prototypes at different stages to gather user input all the way through the development process.

Sometimes user tests don't involve the site at all. You can recruit users to perform a variety of different exercises that can give you insights into how they approach the subject matter of your site. **Card sorting** is one method used to explore how users categorize or group information elements. The user is given a stack of index cards, each of which has the name, description, or image of a piece or type of content on it. The user then sorts the cards into piles according to the groups or categories that feel most natural. Analyzing the results of card sorts conducted with several users can help us understand how they think about the information our site provides.




Collecting all sorts of data about your users can be incredibly valuable, but sometimes you can lose sight of the real people behind all the statistics. You can make your users more real by turning them into **personas** (sometimes called *user models* or *user profiles*). A persona is a fictional character constructed to represent the needs of a whole range of real users. By putting a face and a name on the disconnected bits of data from your user research and segmentation work, personas can help ensure that you keep the users in mind during the design process.

Let's look at an example. Suppose our site is designed to provide information for people who are starting their own businesses. We know from our research that our audience mostly falls in the 30–45 age range. Our users tend to be fairly comfortable with the Web and computer technology in general. Some of them have a lot of experience in the business world; for others, this is their first exposure to all of the issues involved in running a business.

In this case, it might be appropriate to create two personas. We'll call the first one Janet. She's 42 years old, she's married, and she has two kids. She's spent the last couple of years as a vice president at a large accounting firm. She's become frustrated with working for other people, and now she wants to build a company of her own.

The second persona is Frank. He's 37 years old and married with one child. Woodworking has been a weekend hobby of Frank's for many years. Some friends of his were impressed by some furniture he made, so he's been thinking he could go into business for himself selling his work. He's not sure if he'll have to quit his job as a school bus driver in order to launch his new business.

Personas are fictional characters drawn from user research who serve as example cases during user experience development.



Janet


"I don't have time to sort through a lot of information. I need quick answers."

Janet is frustrated with working in a corporate environment and wants to start her own accounting practice.


Age: 42
Occupation: Accounting firm vice president
Family: Married, two children
Household income: \$140,000/year

Technical profile: Fairly comfortable with technology; Dell laptop (about one year old) running Windows XP; DSL Internet connection; 8-10 hours/week online
Internet use: 75% at home; news and information, shopping


Favorite sites:



WSJ.com




Salon.com



Travelocity.com

Frank




"This stuff is all new to me. I want a site that will explain everything."

Frank is interested in learning how he can turn his hobby of making furniture into a business.


Age: 37
Occupation: School bus driver
Family: Married, one child
Household income: \$60,000/year

Technical profile: Somewhat uncomfortable with technology; Apple iMac (about two years old) running Mac OS 9; dial-up modem Internet connection; 4-6 hours/week online
Internet use: 100% at home; entertainment, shopping


Favorite sites:



ESPN.com



moviefone.com



eBay.com

Where did all this information come from? Well, for the most part, we made it up. We want our personas to be consistent with what we know about the users from our research, but the specific details of our personas are fictional inventions, used to breathe life into these characters who will stand in for our real live users.

Janet and Frank represent the range of user needs we'll have to keep in mind as we're making decisions about the user experience of our site. To help us remember them and their needs, we'll grab a couple of stock photos to give Janet and Frank a little more identity, and combine those photos with the information about them we've put together. These profiles can be printed out and posted around the office so that when we have decisions to make we can ask ourselves, "Would that work for Janet? How would Frank react to it?" The personas help us keep our users in mind every step of the way.

Team Roles and Process

Strategic issues affect everyone involved in the user experience development process. But despite this fact (or perhaps because of it), responsibility for formulating these objectives often falls through the cracks. Consulting firms will sometimes employ **strategists** on client projects to manage these issues—but because such rarefied expertise tends to be expensive, and because strategists aren't directly responsible for building any piece of the site itself, this line item is often one of the first to be cut from a project budget.



Strategists will talk to many people throughout the organization to get as many perspectives as possible on the questions of site objectives and user needs. **Stakeholders** are senior decision-makers who are responsible for parts of the organization that will be affected by the site strategy. For example, in the case of a site designed to provide customers with access to product support information, stakeholders might include representatives from marketing communications and customer service as well as product managers. It depends on the formal decision-making structure (and the informal political realities) of the organization.

One group often neglected in formulating a strategy are the rank and file—the people responsible for keeping the organization running on a day-to-day basis. But these people often have a better sense of what works and what doesn't than their managers do. They can inform the strategy in ways senior decision-makers can't—especially when it comes to user needs. No one knows what customers are having trouble with better than the people who talk to those customers every day. I am often surprised at how infrequently customer feedback finds its way to the product development teams who need it.

Site objectives and user needs are often defined in a formal **strategy document** or *vision document*. User needs are sometimes documented in a separate user research report (though there are certain advantages to having all your information in one place). This document isn't just a list of objectives—it provides an analysis of the relationship among the various objectives and of how those objectives fit into the larger context of the organization. The objectives and their analysis are often supported by direct quotes from stakeholders, rank-and-file employees, and users themselves. These quotes vividly illustrate the strategic issues involved in the project.



chapter **4**

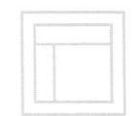
The Scope Plane

Functional Specifications and Content Requirements

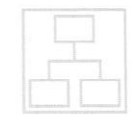
With a clear sense of what we want and what our users want, we can figure out how to satisfy all those strategic objectives. Strategy becomes scope when you translate user needs and site objectives into specific requirements for what content and functionality the Web site will offer to users.



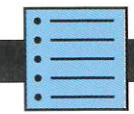
Surface



Skeleton



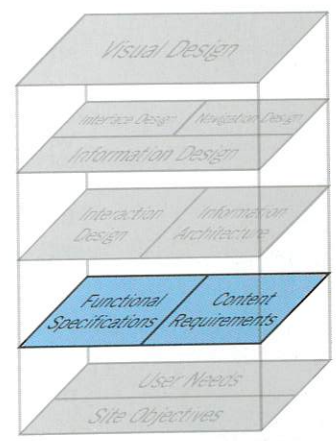
Structure



Scope



Strategy



Defining the Scope

We do some things because there's value in the process, like jogging or practicing scales on the piano. We do other things because there's value in the product, like making a cheesecake or fixing a car. Defining the scope of your project is both: a valuable process that results in a valuable product.

The *process* is valuable because it forces you to address potential conflicts and rough spots in the product while the whole thing is still hypothetical. We can identify what we can tackle now and what will have to wait until later.

The *product* is valuable because it gives the entire team a reference point for all the work to be done throughout the project and a common language for talking about that work. Defining your requirements drives ambiguity out of the development process.

I once worked on a Web application that seemed to be in a state of perpetual beta: almost, but not quite ready to roll out to actual users. A lot of things were wrong with our approach—the technology was shaky, we didn't seem to know anything about our users, and I was the only person in the whole company who had any experience with developing for the Web at all.

But none of this explains why we couldn't get the product to launch. The big stumbling block was an unwillingness to document requirements. After all, it was a lot of hassle to write everything down when we all worked in the same office anyway, and besides, the product manager needed to focus his energy on coming up with new features.

The result was a product that was an ever-changing mishmash of features in various stages of completeness. Every new article somebody read or every new thought that came along while somebody was playing with the product inspired another feature for consideration. There was a constant flow of work going on, but there was no schedule, there were no milestones, and there was no end in sight. Because no one knew the scope of the project, how could anyone know when we were finished?

There are two main reasons to go to the trouble of documenting requirements.

Reason #1: So You Know What You're Building

This seems kind of obvious, but it came as a surprise to the team building that Web application. If you write down a description of exactly what you're setting out to build, everyone will know what the project's goals are and when they've been reached. The final product stops being an amorphous picture in the product manager's head, and it becomes something concrete that everyone at every level of the organization, from top executives to entry-level engineers, can work with.

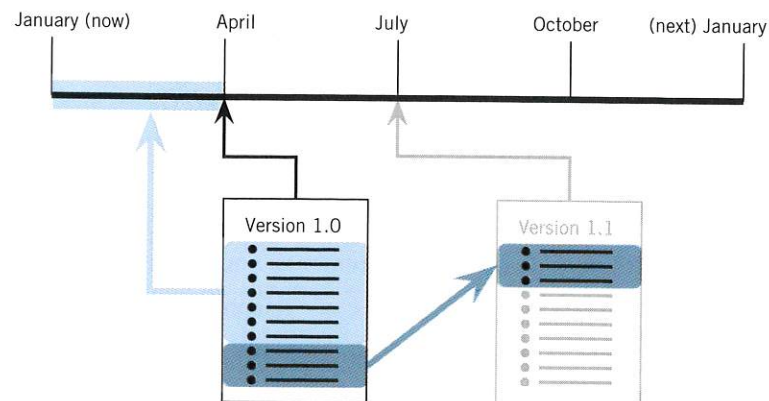
In the absence of documented requirements, your project will probably turn out like a schoolyard game of "Telephone"—each person on the team gets an impression of the product via word of mouth, and everyone's description ends up slightly different. Or even worse, everyone assumes someone else is managing some crucial aspect of the project, but in fact no one is.

Having a defined set of requirements allows you to parcel out responsibility for the work more efficiently. Seeing the entire scope mapped out enables you to see connections between individual requirements that might not otherwise be apparent. The support documentation and the product spec sheets may have seemed like separate content features in early discussions, but defining them as requirements might make it apparent that there's a lot of overlap and that the same group should be responsible for both.

Reason #2: So You Know What You're Not Building

Lots of features sound like good ideas, but they don't necessarily align with the strategic objectives of the project. Additionally, all sorts of possibilities for features emerge after the project is well underway. Having documented requirements provides you with a framework for evaluating those ideas as they come along.

Requirements that can't be met in the current schedule can form the basis for the next milestone in your development cycle.

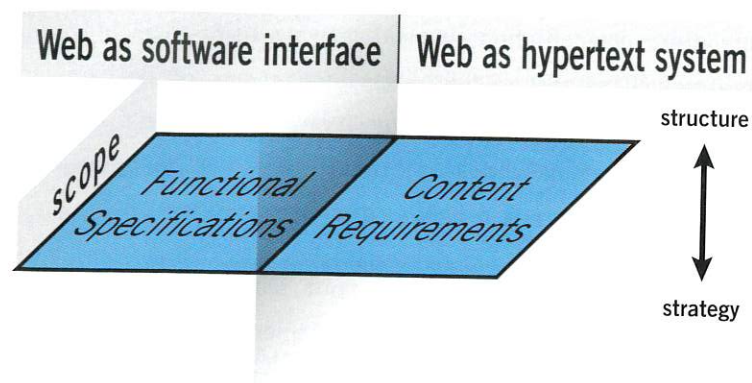


Knowing what you're not building also means knowing what you're not building *right now*. The real value in collecting all those great ideas comes from finding appropriate ways to fit them into your long-term plans. By establishing concrete sets of development requirements and stockpiling any requests that don't fit those requirements as possibilities for future releases, you can manage the entire process in a more deliberate and conscious way.

If you don't consciously manage your requirements, you'll get caught in the dreaded "scope creep." The image this always brings to mind for me is the snowball that rolls forward only an inch—and then another—picking up a little extra snow with each turn until it is careening down the hill, getting bigger and harder to stop all the way down. Likewise, each additional requirement may not seem like that much extra work. But put them all together, and you've got a project rolling away out of control, blowing past deadlines and budget estimates on its way toward an inevitable final crash.

Functionality and Content

On the scope plane, we turn from the abstract question of "Why are we making this site?" that we dealt with in the strategy plane and build upon it with a new question: "What are we going to make?"



The split between the Web as a software interface and the Web as a hypertext system starts coming into play on the scope plane. On the software side, we're concerned with functionality—what would be considered the “feature set” of the software product. On the hypertext side, we're dealing with content, the traditional domain of editorial and marketing communications groups.

Content and functionality seem just about as different as two things could be, but when it comes to defining scope, they can be addressed in very similar ways. Throughout this chapter, I'll use the term “feature” to refer to both software functions and content offerings.

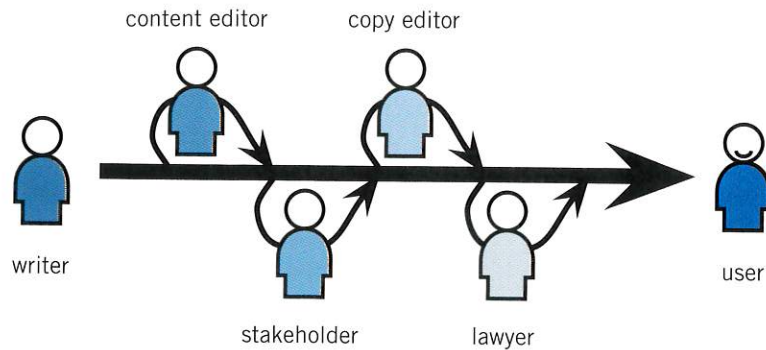
In software development, the scope is defined in the functional requirements or **functional specifications** documents. Some organizations use these terms to mean two different documents: requirements at the beginning of the project to describe what the system should do, and specifications at the end to describe what it

actually does do. In other cases, the specifications are developed soon after the requirements, filling in details of implementation. But most of the time, these terms are interchangeable—in fact, some people use the term “functional requirements specification” just to make sure they've covered all the bases. I'll use “functional specifications” to refer to the document itself, and “requirements” to refer to its contents.

The language of this chapter is mostly the language of software development. But the concepts here apply equally to content. Content development often does not involve as formal a requirements-gathering process as software does, but the underlying principles are the same. A content developer will sit down and talk with people or pore over source material, whether that be a database or a drawer full of news clippings, in order to determine what information needs to be included in the content she's developing. This less formal process for gathering **content requirements** is actually not all that different from the technologist brainstorming features with stakeholders and reviewing existing documentation. The purposes and approaches are the same.

Content requirements often have functional implications. These days, content is usually handled by a **content management system (CMS)**. These systems come in all shapes and sizes, from very large and complex systems that dynamically generate pages from a dozen different data sources to lightweight tools optimized for managing one specific type of content feature in the most efficient way. You might decide to purchase a proprietary content management system, use one of the many open-source alternatives, or even build one from scratch. In any case, it will take some tinkering to tailor the system to your organization and your content.

A content management system can automate the workflow required to produce and deliver content to users.



The functionality you will need in your content management system will depend on the nature of the content you'll be managing. Will you be maintaining content in multiple languages or data formats? The CMS will need to be able to handle all those kinds of content elements. Does every press release need to be approved by six executive vice presidents and a lawyer? The CMS will need to support that kind of approval process in its workflow. Will content elements be dynamically recombined according to the preferences of each user? The CMS will need to be able to accomplish that level of complex delivery.

Similarly, functional requirements have content implications. Will there be instructions on the preferences configuration screen? How about error messages? Somebody has to write those. Every time I see an error message on a Web site like "Null input field exception," I know some engineer's placeholder message made it into the final product because nobody made that error message a content requirement. Countless allegedly technical projects could have been improved immeasurably if the developers had simply taken the time to have someone look at the application with an eye toward content.

Gathering Requirements

Some requirements apply to the site as a whole. Branding requirements are one common example of this; certain technical requirements, such as supported browsers and operating systems, are another.

Other requirements apply only to a specific feature. Most of the time when people refer to a requirement, they are thinking of a short description of a single feature the product is required to have.

The most productive source for requirements will always be your users themselves. The best way to find out what people want is simply *to ask them*. The user research techniques outlined in Chapter 3 can all be used to help you get a better understanding of the kinds of features users want to see on your site.

Whether you are gathering requirements from stakeholders inside your organization or getting them directly from users, the requirements that come out of the process will fall into three general categories. First, and most obvious, are the things people say they want. Some of these are very clearly good ideas and will find their way into the final product.

Sometimes the things people say they want are not really good ideas, but they represent a path to the next type of requirement: things they *actually* want. It's not uncommon for anyone, when they encounter some difficulty with a process or a product, to imagine a solution that will alleviate that difficulty. Sometimes that solution is unworkable, or it addresses a symptom rather than the underlying disease. By exploring these suggestions, you can sometimes arrive at completely different requirements that solve the real problem.

The third type of requirement to come out of this process is the feature people don't know they want. When you get people talking about new requirements and strategic objectives, sometimes they'll hit upon great ideas that simply hadn't occurred to anyone during the ongoing maintenance of the site. These often come out of brainstorming exercises, when participants have had a chance to talk through and explore the possibilities for the project.

Ironically, sometimes the people least able to imagine new directions for a site are those most deeply involved in creating and working with it. For this reason, group brainstorming sessions that bring together people from diverse parts of the organization or represent diverse user groups can be very effective tools in opening the minds of participants to possibilities they wouldn't have considered before.

Getting an engineer, a customer service agent, and a marketing person in a room together to talk about the same Web site can be enlightening for everyone. Hearing perspectives on the site other than those they are familiar with—and having the opportunity to respond to them—encourages people to think in broader terms about both the problems involved in developing the site and the possible solutions.

Generating requirements is often a matter of finding ways to remove impediments. Assume that you have a user who has already decided to make a purchase—they just haven't decided if your product is the one they will buy. What can your site do to make this process—first selecting your product, and then buying your product—easier for them?

In Chapter 3, we looked at the technique of creating fictional characters called personas to help us better understand user needs. In determining requirements, we can use those personas again by putting our fictional characters into little stories called **scenarios**. A scenario is a short, simple narrative describing how a persona might go about trying to fulfill one of those user needs. By imagining the process our users might go through, we can come up with potential requirements that will meet their needs.

We can also look to our competitors for inspiration. Anyone else in the same business is almost certainly trying to meet the same user needs and is probably trying to accomplish similar site objectives as well. Has a competitor found a particularly effective feature for reaching one of these strategic goals? How have they addressed the tradeoffs and compromises we face?

Even sites that aren't direct competitors can serve as fertile sources for possible requirements. Most corporate sites, for example, offer information about employment opportunities. By looking at how companies outside our own industry have handled this type of content, we may find an approach that gives us an advantage over our direct competition.

The level of detail in your requirements will often depend on the specific scope of the project. If the goal of the project is to implement one very complex subsystem, a very high level of detail might be needed, even though the scope of the project relative to the larger site might be quite small. Conversely, a very large-scale content project might involve such a homogeneous base of content that the content requirements can only be very general.

Functional Specifications

Functional specifications have something of a bad reputation in certain quarters. Programmers often hate specs because they tend to be terribly dull, and the time spent reading them is time taken away from producing code. As a result, specs go unread, which in turn reinforces the impression that producing them is a waste of time.

One complaint about functional specifications is that they don't reflect the actual product. Things change during implementation. Everybody understands this—it's the nature of working with technology. Sometimes something you thought would work didn't, or more likely didn't quite work the way you thought it would. This, however, is not a reason to abandon writing specs as a lost cause. Instead, it highlights the importance of keeping specs and keeping them up-to-date. When things change during implementation, the answer is not to throw up your hands and declare the futility of writing specs. The answer is to be vigilant about keeping the spec in sync with development.

But no matter how large or complex the project may be, a few general rules apply to writing any kind of requirements.

Be positive. Instead of describing a bad thing the system shouldn't do, describe what it will do to prevent that bad thing. For example, instead of this:

The system will not allow the user to purchase a kite without kite string.

This would be better:

The system will direct the user to the kite string page if the user tries to buy a kite without string.

Be specific. Leaving as little as possible open to interpretation is the only way we can determine whether a requirement has been fulfilled.

Compare these examples:

1. *The site will be accessible to disabled people.*
2. *The site will comply with Section 508 of the Rehabilitation Act.*

The first example seems to identify a clear requirement, but it does not take much investigation to start poking holes in it. What counts as "accessible?" If the site provides text descriptions of all images, is that sufficient? And who counts as a disabled person? If the site doesn't rely on audio, it must therefore be accessible to the deaf—is that sufficient?

Fortunately, somebody else has worked out all these definitions and distinctions for us: the U.S. Congress. The second example refers us to the specific legal document that defines our goal in specific detail. By removing the possibility of differing interpretations, the second requirement neatly skirts the kinds of arguments likely to crop up during or after implementation.

Avoid subjective language. This is really just another way of being specific and removing ambiguity—and therefore the possibility for misinterpretation—from the requirements.

Here's a highly subjective requirement:

The site will have a hip, flashy style.

Requirements must be falsifiable—that is, it must be possible to demonstrate when a requirement has not been met. It's difficult to demonstrate whether subjective qualities like “hip” and “flashy” have been fulfilled. My idea of hipness probably doesn't match yours, and most likely the CEO has another idea entirely.

This doesn't mean you can't require that your site be hip. You just have to find ways to specify which criteria will be applied:

The site will meet the hipness expectations of Wayne, the mail clerk.

Wayne normally wouldn't have any say about the project, but our project sponsor clearly respects his sense of hipness. Hopefully it's the same sense our users have. But the requirement is still rather arbitrary because we're relying on Wayne's sense of style instead of criteria that can be more objectively defined. So perhaps this requirement would be best of all:

The look of the site will conform to the company branding guidelines document.

The whole concept of hipness has now disappeared entirely from the requirement. Instead, we have a clear, unambiguous reference to established guidelines. To make sure the branding guidelines are sufficiently hip, the VP of marketing may consult Wayne the mail clerk, or she may consult her teenage daughter, or she may even consult some user research findings. It's up to her. But now we can say definitively whether the requirement has been met.

We can also eliminate subjectivity by defining some requirements in quantitative terms. Just as success metrics make strategic goals quantifiable, defining a requirement in quantitative terms can help us clearly identify whether we've met the requirement. For example, instead of requiring that the system have “a high level of performance,” we can require that the system be designed to support at least 1,000 simultaneous users. If the final product only has a three-digit user number field, we can tell the requirement hasn't been met.

Content Requirements

Much of the time, when we talk about content, we're referring to text. But we should remember that images, audio, and video are also types of content. These different content types can also work together to fulfill a single requirement. For example, a content feature covering a sporting event might have an article accompanied by photographs and video clips. Identifying all the different content types associated with a given feature can help you determine what resources will be needed to produce the content (or whether it can be produced at all).

Don't get confused between the *format* of a piece of content and its *purpose*. When discussing content requirements with stakeholders, one of the first things I usually hear is something like, "We should have FAQs on the site." But the term "FAQ" really only refers to a content format: a simple series of questions and answers. The real value a FAQ provides to users is that it provides ready access to commonly needed information. Other content requirements can fulfill that same purpose; but when the focus is on the format, the purpose itself can be forgotten. More often than not, FAQs neglect the "frequently" part of the equation, offering instead answers to whatever questions the content provider could think of to satisfy the FAQ requirement.

The expected size of each of your content features has a huge influence on the user experience decisions you will have to make. Your content requirements should provide rough estimates of the size of each feature: word count for text features, pixel dimensions for images, and file sizes for downloadable, stand-alone content elements like PDF documents, or for features like audio or video. These size estimates don't have to be precise—approximations will suffice. We only have to collect the essential information we need to design an appropriate site around that content. Designing a site to provide access to small thumbnail images is different from designing a site to provide access to full-screen photographs; knowing in advance the size of the content elements we have to accommodate allows us to make smart, informed decisions along the way.

It's important to identify who will be responsible for each content element as early as possible. Once it has been validated against our strategic objectives, any content feature inevitably sounds like a really good idea—as long as someone else is responsible for creating and maintaining it. If we get too deep into the development process without identifying who will be responsible for every required content feature, we're likely to end up with gaping holes in our site because those features everybody loved when they were still hypothetical turned out to be too much work for anyone to actually take on.

And that's what people often forget when developing requirements: content is hard work. You might be able to hire on contract resources (or, more likely, stick someone down in marketing with the job) to create the content in time for the initial launch, but who will keep it up to date? Content—well, effective content, anyway—requires constant maintenance. Approaching content as if you can post it and forget it leads to a site that, over time, does an increasingly poor job of meeting user needs.

This is why, for every content feature, you should identify how frequently it will be updated. The frequency of updates should be derived from your strategic goals for the site: Based on your site objectives, how often do you want users to come back? Based on the needs of your users, how often do they expect updated information? However, keep in mind that the ideal frequency of updates for your users ("I want to know everything instantly, 24 hours a day!") may not be practical for your organization. You'll have to arrive at a frequency that represents a reasonable compromise between the expectations of your users and your available resources.

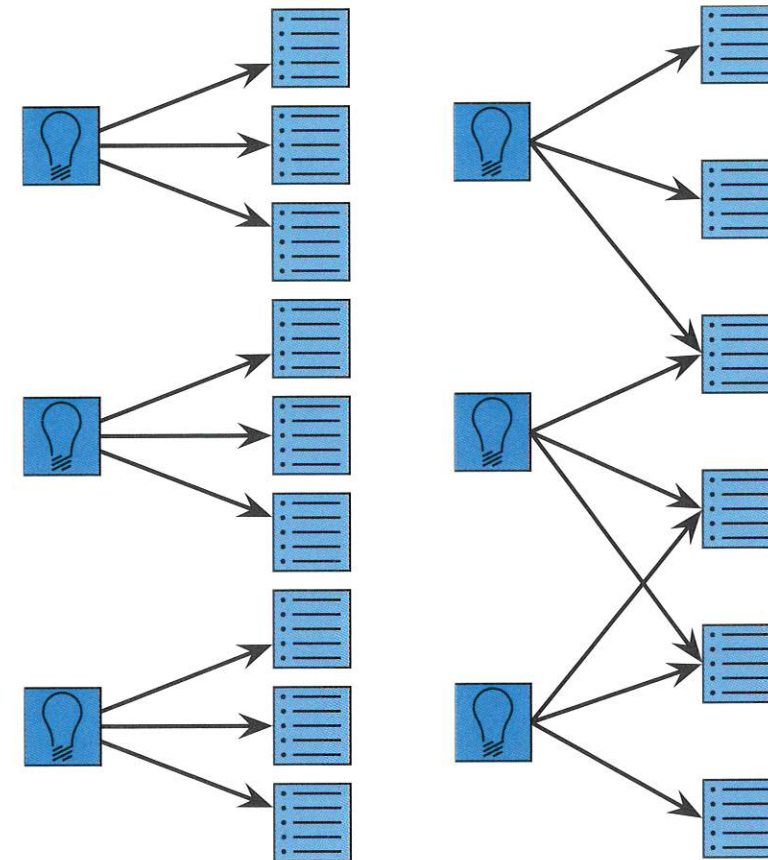
If your site has to serve multiple audiences, it can be useful to identify which audience a content feature is intended for. Particularly in cases where audiences have divergent needs, knowing which audience a piece of content is intended for can help us make better decisions about how to present that content. Information intended for children requires a different approach from information intended for their parents; information for both of them needs yet a third treatment.

For projects that involve working with a lot of existing content (rather than content that has to be created from scratch), much of this information about content is recorded in a **content inventory**. Taking an inventory of all the content on your existing site may seem like a tedious process—and it usually is. But having the inventory (which usually takes the form of a simple, albeit very large, spreadsheet) is important for the same reason that having concrete requirements is important: so everyone on the team knows exactly what they have to work with in creating the user experience.

Prioritizing Requirements

Collecting ideas for possible requirements is not hard. Almost everyone who regularly comes in contact with a product—whether they are inside the organization or outside—will have at least one idea for a feature that could be added. The tricky part is sorting out what features should be included in the scope for your project.

It's actually fairly rare that you see a simple one-to-one correlation between your strategic objectives and your requirements. Sometimes one requirement can be applied toward multiple strategic objectives. Similarly, one objective will often be associated with several different requirements.



Sometimes a strategic objective will result in multiple requirements (left). In other cases, one requirement can serve multiple strategic objectives (right).

Because the scope is built upon the strategy, we'll need to evaluate possible requirements based on whether they fulfill our strategic goals (both site objectives and user needs). In addition to those two considerations, defining the scope adds a third: How feasible will it be to actually make this stuff?

Some features can't be implemented because they're technically impossible—for example, there's just no way to allow users to smell products over the Web yet, no matter how badly they might want that ability. Other features (particularly in the case of content) aren't feasible because they would just demand more resources—whether human or financial—than we have at our disposal. In other cases, it's just a matter of time: the feature would take three months to implement, but we have an executive requirement to launch in two.

In the case of time constraints, you can push features out to a later release or project milestone. For resource constraints, technological or organizational changes can sometimes—but, importantly, not always—reduce the resource burden, enabling a feature to be implemented. (However, impossible things will remain impossible. Sorry.)

Few features exist in a vacuum. Even content features on a Web site rely upon the features around them to support and inform the user on how best to use the content provided. This inevitably leads to conflicts between features. Some features will require tradeoffs with others in order to produce a coherent, consistent whole. For example, some users may want a one-step order submission process—but the tangle of legacy databases the site needs to work with can't

accommodate all the data at once. Is it preferable to go with a multiple-step process, or should you rework the database system? It depends on your strategic objectives.

Keep an eye out for feature suggestions that indicate possible shifts in strategy that weren't apparent during the development of the vision document. Any feature suggestion not in line with the project strategy is, by definition, out of scope. But if a suggested feature that falls outside the scope doesn't fit any of the types of constraints above and still sounds like a good idea, you may need to re-examine some of your strategic objectives. If you find yourself revisiting strategy, however, you've probably jumped into gathering requirements too soon.

If your strategy or vision document identifies a clear hierarchy of priorities among your strategic objectives, these priorities should be the primary factors in determining the relative priority of suggested features. Sometimes, however, the relative importance of two different strategic objectives isn't clear. In these cases, whether features end up in the project scope all too often comes down to corporate politics.

When stakeholders talk about strategy, they usually start out with feature ideas, and then have to be coaxed back to the underlying strategic factors. Because stakeholders often have trouble separating features from strategy, certain features will often have champions during the requirements process. Thus the requirements gathering process becomes a matter of negotiation between motivated stakeholders.

Managing this negotiation process can be difficult. The best approach to resolving a conflict between stakeholders is to appeal to the defined strategy. Focus on strategic goals, not proposed means of accomplishing them. If you can assure a stakeholder with her heart set on a particular feature that the strategic goal the feature is intended to fulfill can be addressed in some other way, she won't feel the needs of her constituents are being neglected. Admittedly, this is often easier said than done. Demonstrating empathy with the needs of stakeholders is essential to resolving feature conflicts. Who says tech workers don't need people skills?

Further Reading

Wieggers, Karl E. *Software Requirements*. Microsoft Press, 1999.

Robertson, Suzanne and James Robertson. *Mastering the Requirements Process*. Addison Wesley, 1999.

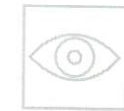
Web resources: www.jjg.net/elements/resources/.

chapter **5**

The Structure Plane

Interaction Design and Information Architecture

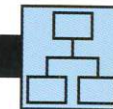
After the requirements have been collected and prioritized, we have a clear picture of what will be included in the final product. The requirements, however, don't describe how the pieces fit together to form a cohesive whole. This is the next level up from scope: developing a conceptual structure for the site.



Surface



Skeleton



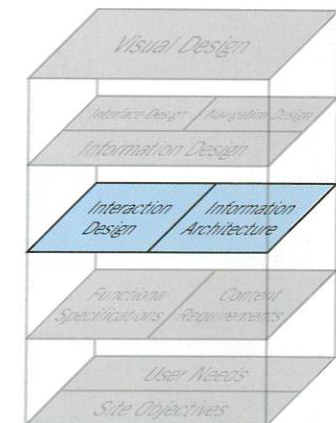
Structure



Scope



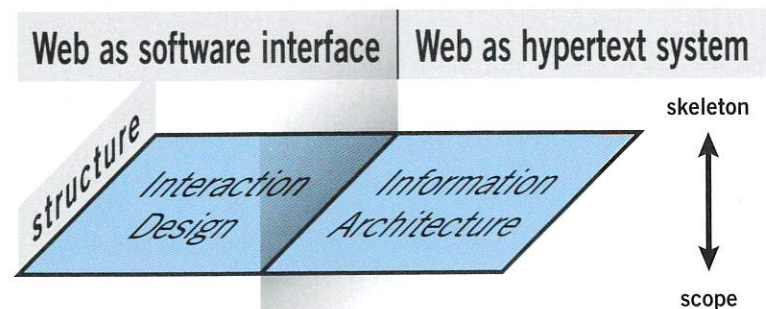
Strategy



Defining the Structure

The realm of structure is the third of the five planes, and appropriately it is the point at which our concerns shift from the more abstract issues of strategy and scope to the concrete factors that will determine what users finally experience. But the line between abstract and concrete can sometimes become blurry—although much of what we decide here will have a noticeable, tangible influence on the final site, the decisions themselves still involve largely conceptual matters.

In traditional software development, the discipline involved in creating a structured experience for the user is known as **interaction design**. It used to be lumped under the heading of “interface design,” but in recent years (due partly to the advent of Web applications and partly to the dedicated evangelism of its practitioners) interaction design has come into its own as a unique discipline.



In content development, structuring the user experience is a question of **information architecture**. This field draws on a number of disciplines that historically have been concerned with the organization, grouping, ordering, and presentation of content: library science, journalism, and technical communication, among others.

Interaction design and information architecture share an emphasis on defining patterns and sequences in which options will be presented to users. Interaction design concerns the options involved in performing and completing tasks. Information architecture deals with the options involved in conveying information to a user.

Interaction design and information architecture sound like esoteric, highly technical areas, but these disciplines aren't really about technology at all. They're about *understanding people, the way they work, and the way they think*. By building this understanding into the structure of our product, we help ensure a successful experience for those who have to use it.

Interaction Design

Interaction design is concerned with describing possible user behavior and defining how the system will accommodate and respond to that behavior. Anytime a person uses a computer, a sort of dance goes on between the user and the machine. The user moves around, and the system responds. The user moves in response to the system, and so the dance goes on. But the typical way that software has been designed doesn't really acknowledge this dance. The thinking seems to have been that if every application danced a little bit differently anyway, it wasn't unreasonable to expect the user to adapt. The system could just do its thing, and if some toes got

stepped on, well, that was part of the learning process. But every dancer will tell you that for the dance to really work, each participant must anticipate the moves of the other.

Programmers have traditionally focused on and cared most about two aspects of software: what it does and how it does it. There's a good reason for this—it is precisely their passion for these details that makes programmers good at what they do. But this focus meant that programmers could go ahead and build the system in the way that was most technically efficient without regard to what worked best for users. Especially back when computing power was a limited resource, the best approach was the one that got the job done within those system limitations.

The approach that works best for the computer is almost never the approach that works best for the person who has to use it. Thus, computer software acquired the reputation that has haunted it for most of its existence: Software is complicated, confusing, and hard to use. Only 10 years ago, “computer literacy”—teaching people about the inner workings of computers—was widely considered to be the only way to make users and software get along.

It took a long time, but as computers became more powerful and we learned more about how people used them, eventually we started catching on to the idea that, instead of designing software to work in the fashion that works best for the machine, we could design software to work in the fashion that works best for the people who use it, thereby skipping this whole business of sending file clerks to programming classes to improve their computer literacy. The new discipline that arose to help software developers do this is called interaction design.

Conceptual Models

Users' impressions of how the interactive components we create will behave are known as **conceptual models**. For example, is it a content element, a place the user visits, or an object the user acquires? Different sites take different approaches. Knowing your conceptual model allows you to make consistent design decisions. It doesn't matter whether the content element is a place or an object; what matters is that the site behaves consistently, instead of treating the element as a place sometimes and an object at other times.

For example, the conceptual model for the “shopping cart” component of a typical e-commerce site is that of a container. This metaphorical concept influences both the design of the component and the language we use in the interface. A container holds objects; as a result, we “put things into” and “take things out of” the “cart,” and the system must provide functions to accomplish these tasks.

Suppose the conceptual model for the component were a different real-world analogue, such as a catalog order form. The system might provide an “edit” function that would replace both the “add” and “remove” functions of the traditional cart, and instead of using a “checkout” metaphor to complete the process, users might “send” their orders in.

Both the retail store model and the catalog model seem perfectly suitable for allowing users to place orders over the Web. Which to choose? The retail store model is so widely used on the Web that it's taken on the status of a **convention**. If your users do a lot of shopping on other Web sites, you'll probably want to stick to that convention. Using conceptual models people are familiar with makes it

easier for them to adapt to an unfamiliar site. Of course, there's nothing wrong with breaking from convention either—as long as you have a good reason for doing so and have an alternate conceptual model that will meet your users' needs.

A conceptual model can refer to just one component of a system or to the system as a whole. When the news and commentary site Slate launched, its conceptual model was a real-world magazine: The site had a front and a back, and every page had both a page number and interface elements allowing the user to “turn the page.” As it turns out, the magazine conceptual model doesn't translate very effectively to the Web, and Slate eventually dropped the magazine concept.

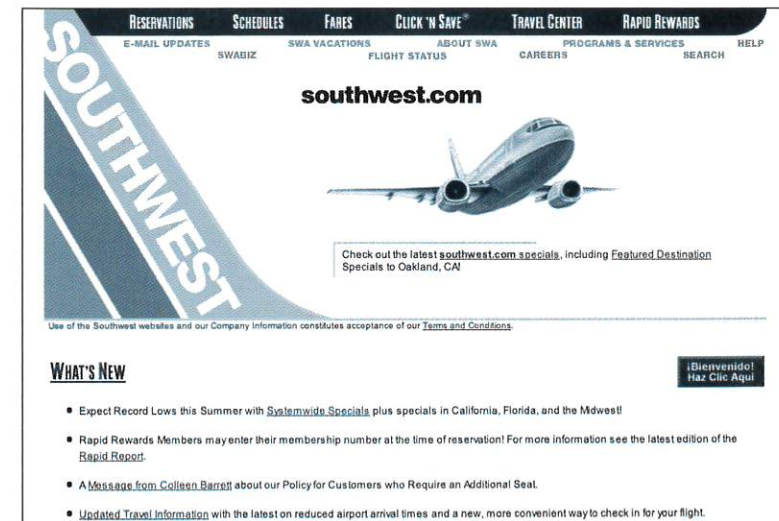
We don't have to communicate our conceptual models to our users explicitly—in fact, sometimes this only confuses users instead of helping them. It's more important that conceptual models are used consistently throughout the development of the interaction design. Understanding the models users themselves bring to the site (Does it work like a retail store? Does it work like a catalog?) helps us choose conceptual models that will work most effectively. Ideally, the users won't have to be told what conceptual model we're following; they'll pick up on it intuitively as they use the site because the behavior of the site will match their expectations.

Basing our conceptual models on metaphors involving real-world analogues to system functions can be valuable, but it's important not to take our metaphors too literally. The home page of the site for Southwest Airlines used to consist solely of a picture of a customer service desk, with a stack of brochures to one side, a telephone to the other side, and so on. For years, the site was held up as an example of a conceptual model gone too far—placing a reservation

may be analogous to making a phone call, but that doesn't mean the reservation system should actually be represented by a telephone. Southwest must have gotten tired of being used as a bad example; today their site is light on metaphor and considerably more functional.



The old Southwest Airlines site is a classic example of conceptual models being tied too closely to real-world counterparts.



The redesigned Southwest Airlines site makes content and functionality more directly apparent.

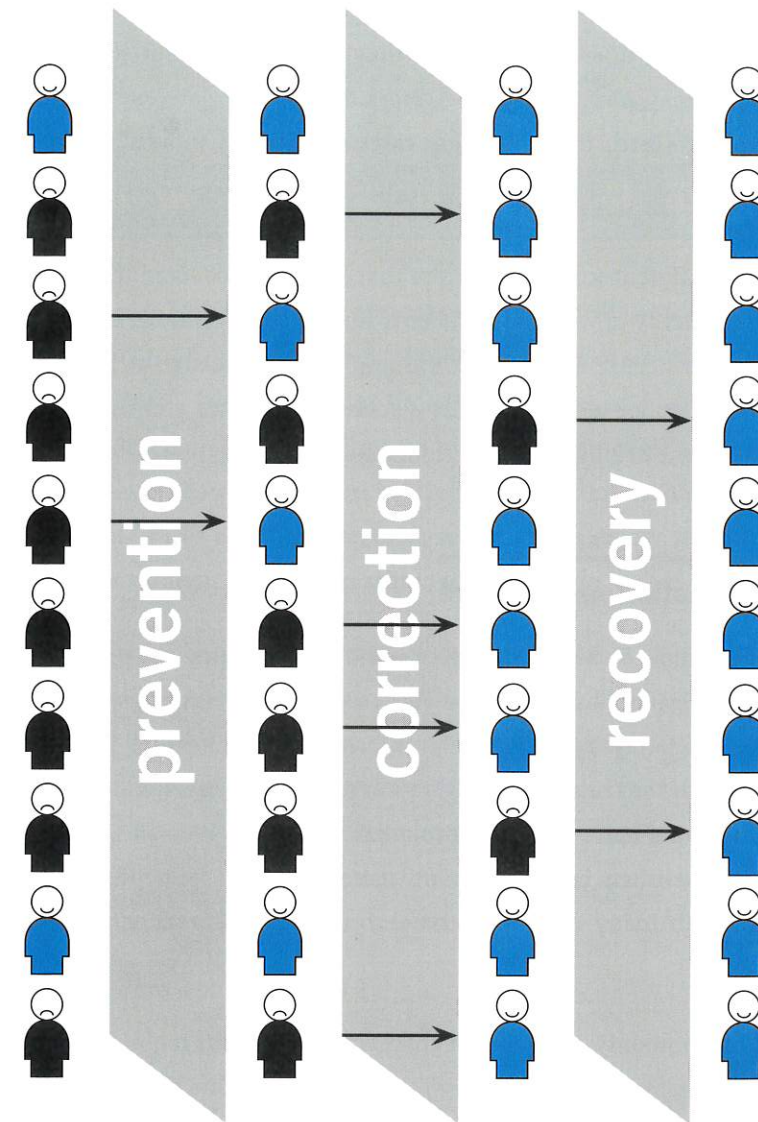
Error Handling

A huge part of any interaction design project involves dealing with “user error”—what does the system do when people make mistakes, and what can the system do to prevent those mistakes from happening in the first place?

The first and best defense against errors is to design the system so that errors are simply impossible. A good example of this type of defense can be seen in any car with an automatic transmission. Starting the car while the transmission is engaged can damage the sensitive and complex transmission mechanism; moreover, the car doesn’t actually start, but instead lurches forward abruptly. Bad for the car, bad for the driver, and possibly bad for an innocent bystander who happens to be in the path of the lurching car.

To prevent this, any car with an automatic transmission is designed so the starter won’t engage unless the transmission is disengaged. Because it’s impossible to start the car with the transmission engaged, the error never happens. Unfortunately, it’s not quite so easy to make most user errors impossible in this way.

The next best thing to making errors impossible is to make them merely difficult. But even with such measures in place, some errors are bound to happen. At this point, the system should do what it can to help the user figure out the error and fix it. In some cases, the system can even fix the error on the user’s behalf. But be careful—some of the most irritating behavior of software products results from well-intentioned efforts to correct user errors. (If you’ve ever used Microsoft Word, you know exactly what I’m talking about. Word offers numerous features intended to correct common errors; invariably, I find myself switching them off so I can get some work done and stop correcting the corrections.)



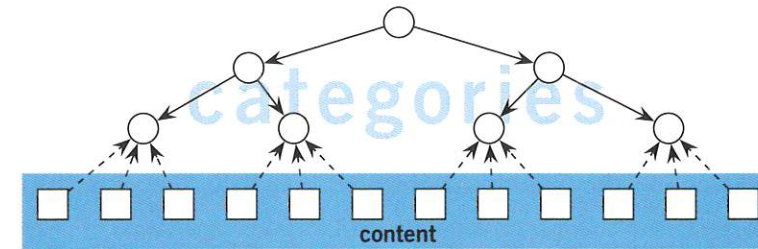
Each layer of error handling in your interaction design ensures that a higher percentage of users will have positive experiences.

Helpful error messages and well-designed interfaces can help users catch many kinds of errors after they've happened. But some user actions may not appear to be errors until it's too late even for the system to catch them. In these cases, the system should provide a way for users to recover from the error. The best-known example of this is the famous "undo" function, but error recovery can take many different forms. For errors that can't be recovered from, providing plenty of warning is the only means of prevention the system can provide. Of course, this warning is only effective when users actually notice it, which is why a succession of "Are you sure?" dialog boxes often annoys more users than it helps.

Information Architecture

Information architecture is concerned with creating organizational and navigational schemes that allow users to move through site content efficiently and effectively. Information architecture is closely related to the concept of information retrieval: the design of systems that enable users to find information easily. But Web site architectures are often called on to do more than just help people find things; in many cases, they have to educate, inform, or persuade users.

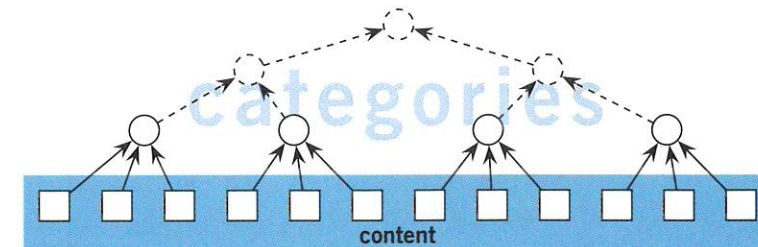
Most commonly, information architecture problems require creating categorization schemes that will correspond to our own objectives for the site, the user needs we intend to meet, and the content that will be incorporated in the site. We can tackle creating such a categorization scheme in two ways: from the top down, or from the bottom up.



Top-down architectural approach.

A **top-down approach** to information architecture involves creating the architecture directly from an understanding of site objectives and user needs. Starting with the broadest categories of possible content and functionality needed to accomplish these strategic goals, we then break the categories down into logical subsections. This hierarchy of categories and sub-categories serves as the empty shell into which the content and functionality will be slotted.

A **bottom-up approach** to information architecture also derives categories and sub-categories, but it does so based on an analysis of the content and functional requirements. Starting with the source material that exists (or that will exist by the time the site launches), we group items together into low-level categories and then group those into higher-level categories, building toward a structure that reflects our site objectives and user needs.



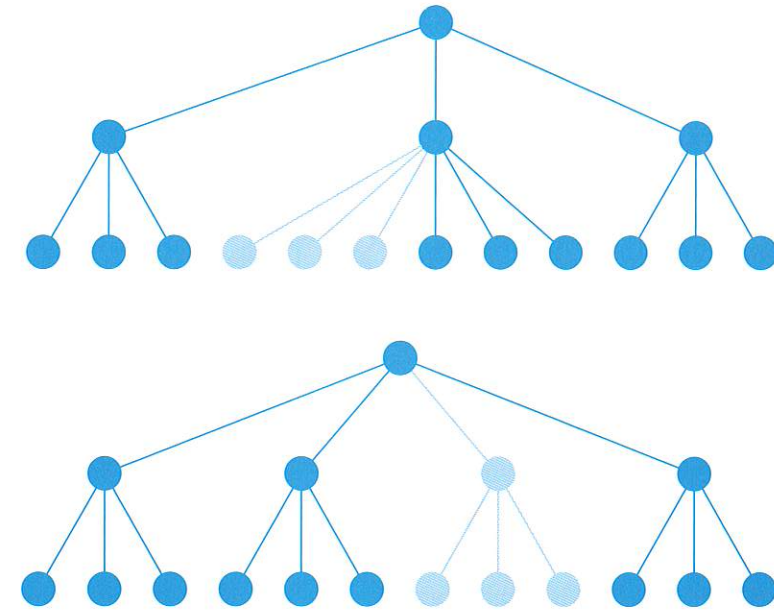
Bottom-up architectural approach.



Neither approach is better than the other. Approaching the architecture from the top down can sometimes cause important details about the content itself to be overlooked. On the other hand, a bottom-up approach can sometimes result in an architecture so precisely tuned and fitted to the existing content that it isn't flexible enough to accommodate changes or additions. Striking a balance between top-down and bottom-up thinking is the only way to make sure the final result can avoid these pitfalls.

Web sites are living entities. They require constant care and feeding. Inevitably, they grow and change over time. In most cases, a few new requirements acquired along the way shouldn't require rethinking the overall structure of the site. One trait of an effective structure is its ability to accommodate growth and adapt to change. But the accumulation of new content will eventually require a re-examination of the organizing principles employed on the site. For example, the architecture that enabled users to page through press releases day by day might have been fine when you had only a few months' worth, but organizing them by topic might be more practical after a few years.

It's not necessary to adhere to a particular number of categories at any level or in any section of the architecture. The categories just have to be the right ones for your users and their needs. Some people favor counting the number of steps it takes to complete a task or the number of clicks it takes for a user to reach a particular destination as a way to evaluate the quality of a site structure. The most important sign of quality, however, is not how many steps the process took, but whether each step made sense to the user and whether it followed naturally from the previous step. Users will invariably favor a clearly defined seven-step process over a confusingly compressed three-step alternative.



An adaptable architecture can accommodate the addition of new content within a section (top) as well as entire new sections (bottom).

The entire user experience, including the structure of the site, is built on an understanding of your objectives and the needs of your users. If what you want to accomplish with the site is redefined or the needs you intend the site to meet change, be prepared to rework the structure of your site accordingly. The need for such structural changes rarely announces itself in advance, though; often, by the time you can tell that you need to rework the architecture, your users are already suffering.

Architectural Approaches

The basic unit of information structures is the **node**. A node can correspond to any piece or group of information—it can be as small as a single number (like the price of a product) or as large as an entire library. By dealing with nodes rather than with pages, documents, or components, we can apply a common language and a common set of structural concepts to a diverse range of problems.

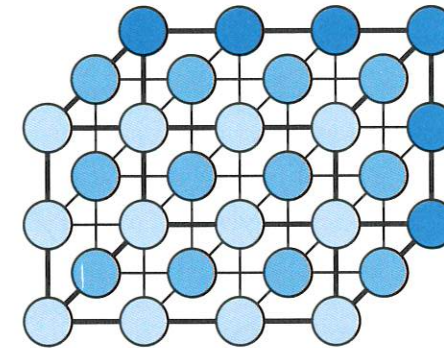
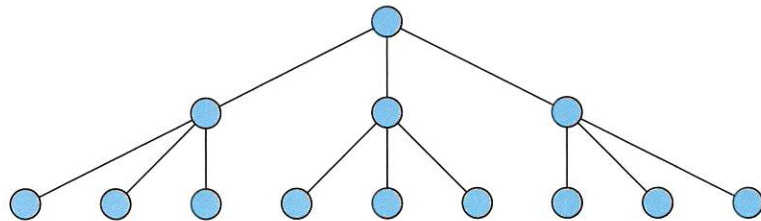


The abstraction of nodes also allows us to explicitly set the level of detail we will be concerned with. Most Web site architecture projects are only concerned with the arrangement of pages on the site; by identifying the page as our base-level node, we make it explicit that we won't be dealing with anything smaller. If the page itself is too small for the project at hand, we can have each node correspond to an entire section of the site.

These nodes can be arranged in many different ways, but these structures really fall into just a few general classes.

In a **hierarchical** structure—sometimes called a *tree* or *hub-and-spoke* structure—nodes have parent/child relationships with other related nodes. Child nodes represent narrower concepts within the broader category represented by the parent node. Not every node has children, but every node has a parent, leading all the way up to the parent node of the entire structure (or the “root” of the “tree,” if you prefer). Because the concept of hierarchical relationships is well understood by users and because computers tend to work in hierarchies anyway, this type of structure is far and away the most common.

Hierarchical structure.

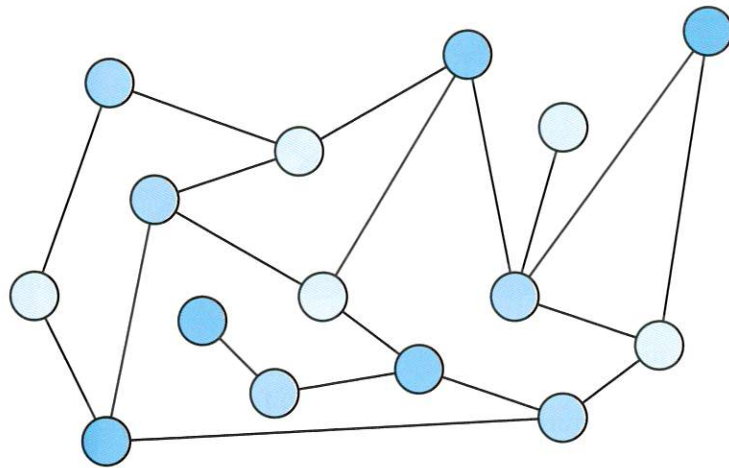


Matrix structure.

A **matrix** structure allows the user to move from node to node along two or more “dimensions.” Matrix structures are often useful for enabling users with different needs to navigate through the same content, because each user need can be associated with one “axis” of the matrix. For example, if some of your users really want to browse products by color, but others need to browse by size, a matrix can accommodate both groups. A matrix of more than three dimensions can cause problems, however, if you expect users to rely on it as their primary navigational tool. The human brain simply isn't very well equipped to visualize movement in four or more dimensions.

Organic structures don't attempt to follow any consistent pattern. Nodes are connected together on a case-by-case basis, and the architecture has no strong concept of “sections.” Organic structures are good for exploring a set of topics whose relationship is unclear or evolving. But organic structures don't provide users with a strong sense of where they are in the architecture. If you want to encourage a feeling of free-form exploration, such as on some entertainment or educational sites, an organic structure can be a good choice; however, it can present a challenge if your users need to reliably find their way back to the same piece of content again.

Organic structure.



Sequential structure.



Sequential structures are the ones you are most familiar with from offline media—in fact, you’re experiencing one right now. The sequential flow of language is the most basic type of information architecture there is, and the facilities needed to process it are built right into our brains. Books, articles, audio, and video are all designed to be experienced in a sequential fashion. Sequential structures on the Web are used most often for smaller-scale structures such as individual articles or sections; large-scale sequential structures tend to be limited to applications in which the order of content presentation is essential to meeting user needs, such as in instructional material.

Organizing Principles

Nodes in an information structure are arranged according to **organizing principles**. At its most basic level, the organizing principle is the criterion by which we determine which nodes are grouped together and which are kept separate. Different organizing principles will be applied in different areas and at different levels of the site.

For example, in the case of a corporate information site, we might have categories near the top of our tree, such as “Consumer,” “Business,” and “Investor.” At this level, the organizing principle is the audience for which the content is intended. Another site might have top-level categories like “North America,” “Europe,” and “Africa.” Using geography as an organizing principle is one approach to meeting the needs of a global audience of users.

Generally, the organizing principles you employ at the highest levels of your site are closely tied to the site objectives and user needs. At lower levels in the architecture, issues specific to the content and functional requirements begin to have a greater influence on the organizing principles that should be used.

For example, a site with news-oriented content will often have chronology as its most prominent organizing principle. Timeliness is the single most important factor for users (who, after all, look to news sites for information on current events, not history) as well as for the creators of the site (who must emphasize the timeliness of their content in order to remain competitive).



At the next level in the architecture, other factors more closely tied to content come into play. For a sports news site, the content might be divided into categories such as “Baseball,” “Tennis,” and “Hockey”; a more general-interest site might have categories like “International News,” “National News,” and “Local News.”

Any collection of information—whether it consists of two items, 200, or 2,000—has an inherent conceptual structure. In fact, it usually has more than one. That’s part of the problem we have to solve. The challenge isn’t creating a structure, but creating the *right* structure for our objectives and the needs of our users.

For example, suppose our site contained a repository of information about cars. One possible organizing principle would arrange the information according to the weight of the car in question. So the first thing the user would see would be information about the heaviest car in our database, then the second heaviest, and on down to the lightest.

For a consumer information site, this is probably the wrong way to organize the information. Most people, most of the time, aren’t concerned with the weight of a car. Organizing the information according to make, model, or type of car would probably be more appropriate for this audience. On the other hand, if our users are professionals who deal on a daily basis with the business of shipping cars overseas, weight becomes a very important factor. For these people, qualities like fuel economy and engine type are considerably less important, if they matter at all.

These attributes, in the language of library science, are known as **facets**, and they can provide a simple, flexible set of organizing principles for almost any content. But as the preceding example shows, using the wrong facets can be worse than using no facets at all. One common response to this problem is to position every conceivable facet as an organizing principle and let the users pick the one that’s important to them.

Unfortunately, unless you’re dealing with very simple information consisting of only a few facets, this approach soon turns the architecture into an unwieldy mess. The users have so many options to sort through that no one can find anything. The burden shouldn’t be on the user to sort through all the attributes and pick out what’s important—the burden is on us. The strategy tells us what the users need, and the scope tells us what information will meet those needs. In creating the structure, we identify the specific aspects of that information that will be foremost in the users’ minds. A successful user experience is one in which the user’s expectations are anticipated in advance.

Language and Metadata

Even if the structure is a perfectly accurate representation of the way users think about your subject matter, they won’t be able to find their way around the architecture if they can’t understand your **nomenclature**: the descriptions, labels, and other terminology the site uses. For this reason, it’s essential to use the language of your users and to do so in a consistent fashion. The tool we use to enforce that consistency is called a **controlled vocabulary**.



A controlled vocabulary is nothing more than a set of standard terms for use on the site. This is another area in which user research is essential. Talking to users and understanding how they communicate is the most effective way to develop a system of nomenclature that will feel natural to them. Creating a controlled vocabulary that reflects the language of your users (and adhering to it) is the best way to prevent your organization's internal jargon from creeping onto the site, where it will only confuse your users.

Controlled vocabularies also help create consistency across all your content. Whether the people responsible for producing the content sit right next to each other or in offices on separate continents, the controlled vocabulary provides a definitive resource to ensure that everyone is speaking the user's language.

A more sophisticated approach to controlling vocabulary is to create a **thesaurus**. Unlike a simple list of approved terms, a thesaurus will also document alternative terms that are commonly used but not approved for use on the site. With a thesaurus, you can map internal jargon, shorthand, slang terms, or acronyms to their approved counterparts. A thesaurus might also include other types of relationships among the terms, providing recommendations for broader, narrower, or related terms. Documenting these relationships can give you a more complete picture of the entire range of concepts found in your content, which in turn can suggest additional architectural approaches.

Having a controlled vocabulary or thesaurus can be especially helpful if you decide to build a system that includes **metadata**. The term metadata means simply "information about information." It refers to a structured approach to describing a given piece of content.

Suppose we were dealing with an article about how your latest product is being used by volunteer fire departments. Some of the metadata for that article might include:

- ▶ The name of the author
- ▶ The date the piece was posted
- ▶ The type of piece (for example, a case study or article)
- ▶ The name of the product
- ▶ The type of product
- ▶ The customer's industry (for example, volunteer fire department)
- ▶ Other relevant information (for example, municipal agencies or emergency services)

Having this information allows us to consider a range of possible architectural approaches that would be difficult (if not downright impossible) to implement without it. In short, the more detailed information you have about your content, the more flexibility you have in structuring it. If emergency services suddenly shows potential as a lucrative new market for the company to expand into, having this metadata will allow us to rapidly create a new section to meet the needs of these users with the content we already have.



But creating technical systems to collect and track all this metadata won't help us if the data itself isn't consistent. That's where controlled vocabularies come in. By using only one term for each unique concept in your content, you can rely on automation to help define the connections between your content elements. Your site could dynamically link together all the pages on a specific topic without anyone having to do anything more than use the same term consistently in their metadata.

In addition, good metadata can provide a faster and more reliable way for your users to find information on your site than a basic full-text search engine can provide. Search engines can be powerful, but in general they're very, very dumb—you give them a string of characters, and they pretty much go looking for exactly that string of characters. They don't understand what any of it means.

Connecting your search engine with a thesaurus and providing metadata for your content can help make the engine smarter. The search engine can use the thesaurus to map a search for a disallowed term to a preferred term; then it can check the metadata for that preferred term. Instead of getting no results at all, the user gets highly targeted, relevant results—and maybe even some recommendations for other related subjects that might be of interest.

Team Roles and Process

The documents needed to describe the structure of a site—from the specific details of nomenclature and metadata to the big picture of overall information architecture and interaction design—can vary substantially depending on the complexity of the project. For projects involving a lot of content in a hierarchical structure, simple text



outlines can be an effective way to document the architecture. In some cases, tools like spreadsheets and databases will be pressed into service to help capture the nuances of a complex architecture.

But the major documentation tool for information architecture or interaction design is the diagram. Representing the structure visually is the most efficient way for us to communicate the branches, groups, and interrelationships among the components of our site. Web site structures are inherently complicated things; trying to convey this complexity in words pretty much guarantees that no one will read them.

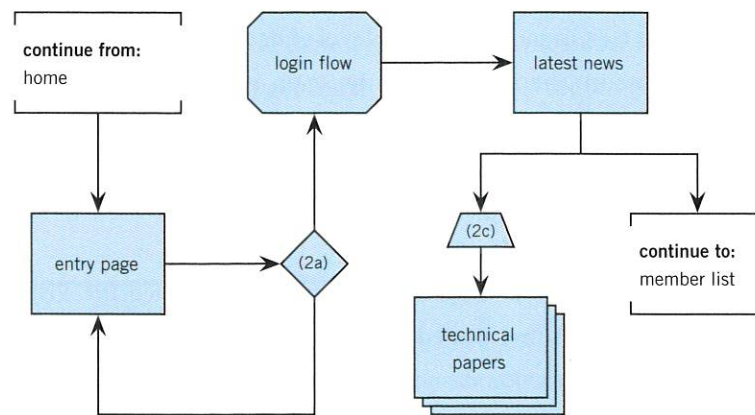
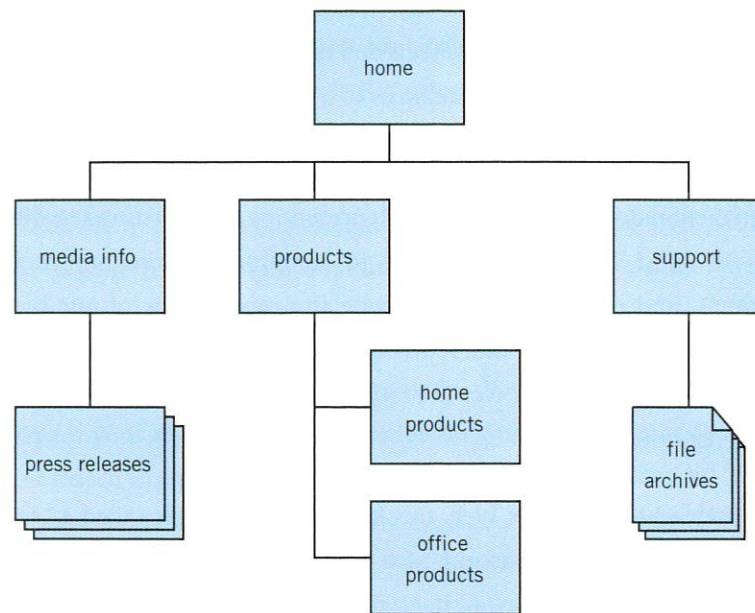
In the early days of the Web, this kind of diagram was called a “site map”; but because site map is also a term used for a particular kind of navigational tool on a site (which you'll read more about in Chapter 6), **architecture diagram** is now the favored term for the tool we use internally to describe site structure.

The diagram doesn't have to document every link on every page in your site. In fact, in most cases that level of detail only serves to confuse and obscure the information the team really needs. It's more important to document conceptual relationships: Which categories go together, and which remain separate? How do the steps in a given interaction sequence fit together?

The system I created to diagram site structures is called the Visual Vocabulary. Since I first posted it to the Web in 2000, information architects and interaction designers all over the world have adopted it. You can learn more about the Visual Vocabulary, see sample diagrams, and download tools for using it at my Web site: www.jjg.net/ia/visvocab/.



The Visual Vocabulary is a system for diagramming architectures ranging from the very simple (top) to the very complex (bottom). See www.jjg.net/ia/visvocab/ for more details.



Because both interaction design and information architecture are relatively recent arrivals on the user experience landscape, it's still very common for Web development teams to have no one explicitly charged with responsibility for these areas. In light of this, it might not be all that surprising that so few Web sites display evidence that their structures were planned in advance.

Responsibility for structure often lands in someone's lap by default rather than through conscious planning. Who ends up responsible for structure often depends on the culture of the organization or the nature of the project. Early in the history of the Web, sites were usually created and managed by the organization's existing technical staff; in organizations where change happens slowly (or resources are very limited), this is likely to still be the case today.

For content-heavy sites or in organizations in which creating a presence on the Web was initially seen as a marketing activity, the responsibility for determining the structure of the site has resided within content development, editorial, or marketing communications groups. If the organization has historically been led by technical people or had a technology-oriented internal culture, responsibility for structure has commonly fallen to the technical project manager working on the Web site.

Every project can benefit from having a full-time specialist dedicated to structural issues. Sometimes this person goes by the job title “interaction designer,” but more often they are likely to be referred to as an “information architect.” Don’t let the title confuse you—although it’s true that some information architects specialize exclusively in creating organizational schemes and navigational structures for content sites, more often than not, an information architect will have some degree of experience with interaction design issues. In fact, some people who have held the title of information architect are really much closer to being interaction design specialists.

Your organization might not have the volume of ongoing work to warrant hiring a full-time information architect as a permanent member of your staff. If your Web development efforts are mostly limited to keeping the content you have up-to-date and you don’t do much new development between site-wide redesign projects every couple of years, having a staff architect probably isn’t a good way to spend your money. But if you have a steady stream of new content and functionality being added to your site, having an information architect on hand can help you manage that process in the way that will be most effective for meeting the needs of your users and for meeting your own strategic objectives.

Whether you have a specialist to address structural concerns isn’t important, but it is important that those concerns are addressed by someone. Your site will have a structure whether you plan it out or not. The sites that are built according to a defined, explicit structural plan tend to be the ones that require less frequent overhauls, produce concrete results for their owners, and satisfy the needs of their users.

Further Reading

Cooper, Alan. *The Inmates Are Running the Asylum: Why High-Tech Products Drive Us Crazy and How to Restore the Sanity*. Sams, 1999.

Norman, Donald A. *The Design of Everyday Things*. Revised edition. Currency/Doubleday, 1990.

Rosenfeld, Louis and Peter Morville. *Information Architecture for the World Wide Web*. 2nd edition. O’Reilly, 2002.

Web resources: www.jjg.net/elements/resources/.

