

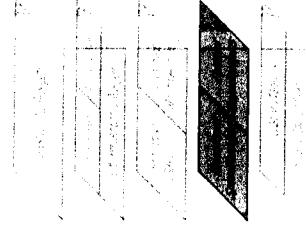
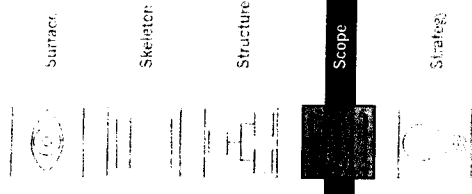
4

CHAPTER

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.

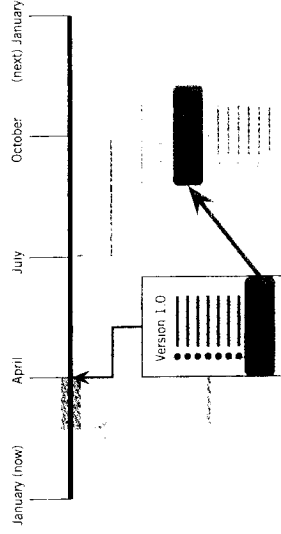


Having a defined set of requirements allows you to parcel out responsibility for the work more efficiently. Seeing the entire scope mapped out enables you to see connections between individual requirements that might not otherwise be apparent. The support documentation and the product spec sheets may 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
n't be met in the
rrent schedule can
m the basis for the
xt milestone in your
velopment 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.

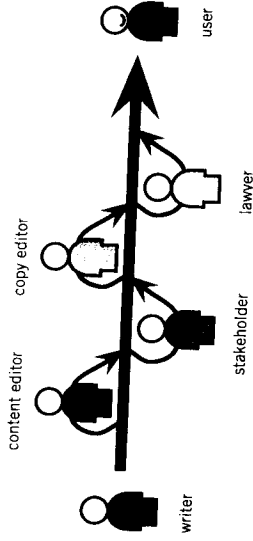
In software development, the scope is defined in the functional requirements or **functional specifications** documents. Some organizations use these terms to mean two different documents: requirements at the beginning of the project to describe what the system should do, and specifications at the end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

Content management can automate workflow required to 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.

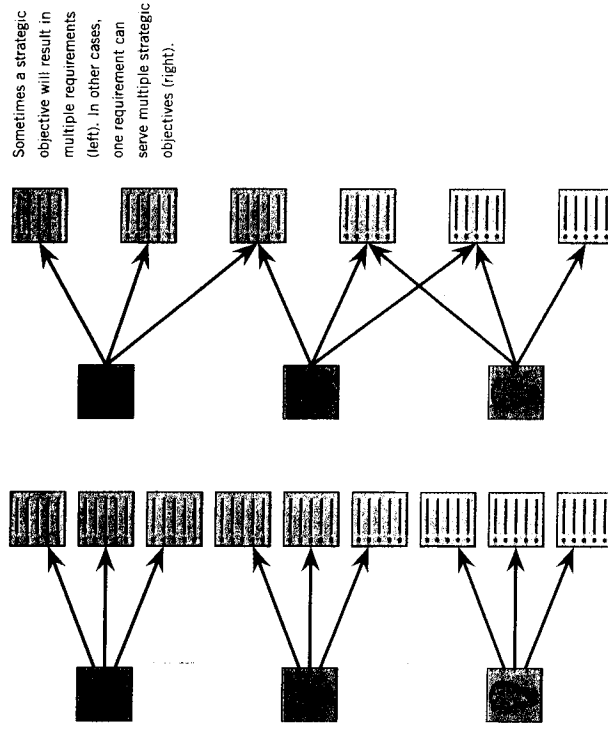
If your site has to serve multiple audiences, it can be useful to identify which audience a content feature is intended for. Particularly in cases where audiences have divergent needs, knowing which audience a piece of content is intended for can help us make better decisions about how to present that 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.



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

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

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

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



THE ELEMENTS OF USER EXPERIENCE

Smart organizations recognize that Web design is more than just creating clean code and sharp graphics. A site that really works fulfills your strategic objectives while meeting the needs of your users. Even the best content and the most sophisticated technology won't help you balance those goals without a cohesive, consistent user experience to support it.

But creating the user experience can seem overwhelmingly complex. With so many issues involved—usability, brand identity, information architecture, interaction design—it can seem as if the only way to build a successful site is to spend a fortune on specialists who understand all the details.

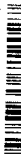
The Elements of User Experience cuts through the complexity of user-centered design for the Web with clear explanations and vivid illustrations that focus on ideas rather than tools or techniques. Jesse James Garrett gives readers the big picture of Web user experience development, from strategy and requirements to information architecture and visual design.

This accessible introduction helps any Web development team, large or small, to create a successful user experience.

Jesse James Garrett is one of the founders of Adaptive Path, a user experience agency based in San Francisco. Since it was first released in March 2000, "Elements of User Experience" model has been downloaded more than 10,000 times. Jesse's Web experience includes projects for AT&T, Intel, Motorola, Hewlett-Packard, and National Public Radio. His other contributions to the field of user experience include the Visual Vocabulary notation system for information architecture documentation that is used by organizations around the world. His personal site at www.jjg.net is one of the Web's most popular destinations for information architecture resources.



ISBN 0-7357-1202-6



529999

infobn.bn

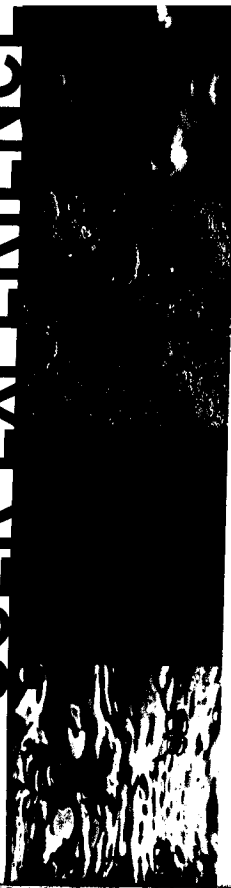
\$29.99 USA / \$36.99 CAN / £23.50 Net UK

www.jjg.net
www.newriders.com

Category: Web Design



THE ELEMENTS OF USER EXPERIENCE



USER-CENTERED DESIGN FOR THE WEB

Jesse James Garrett

THE ELEMENTS OF USER EXPERIENCE

Garre

"Jesse James Garrett brings incisive clarity to the complex process of providing a high-quality experience to the people who use your Web site. Deconstructing and modeling both the human and conceptual issues, he exposes the essence of a problem usually obscured by thick layers of technical camouflage."

Alan Cooper
author of *About Face* and *The Inmates Are Running the Asylum*

"Jesse James Garrett has finally expanded his famous diagram into a book that clarifies the entire jumbled field of user experience design. And because he's a very smart fellow, he's kept it very short so there's a useful insight on almost every page."

Steve Krug
author of *Don't Make Me Think!*

"Finally, a concise explanation of User Experience that synthesizes its many disparate parts. Clear-headed, readable, and necessary."

Louis Rosenfeld
co-author of *Information Architecture for the World Wide Web*