

The Elements of User Experience: User-Centered Design for the Web and Beyond, Second Edition

Jesse James Garrett

New Riders 1249 Eighth Street
Berkeley, CA 94710
510/524-2178
510/524-2221 (fax)

Find us on the Web at: www.newriders.com

To report errors, please send a note to errata@peachpit.com

New Riders is an imprint of Peachpit, a division of Pearson Education.

Copyright © 2011 by Jesse James Garrett

Project Editor: Michael J. Nolan

Development Editor: Rose Weisburd

Production Editor: Tracey Croom

Copyeditor: Doug Adrianson

Proofreader: Gretchen Dykstra

Indexer: Valerie Perry

Cover Designer: Aren Howell Straiger

Interior Designer: Kim Scott

Compositor: Kim Scott

Notice of Rights

All rights reserved. No part of this book may be reproduced or transmitted in any form by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher. For information on getting permission for reprints and excerpts, contact permissions@peachpit.com.

Notice of Liability

The information in this book is distributed on an “As Is” basis without warranty. While every precaution has been taken in the preparation of the book, neither the author nor Peachpit shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the instructions contained in this book or by the computer software and hardware products described in it.

Trademarks

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and Peachpit was aware of a trademark claim, the designations appear as requested by the owner of the trademark. All other product names and services identified throughout this book are used in editorial fashion only and for the benefit of such companies with no intention of infringement of the trademark. No such use, or the use of any trade name, is intended to convey endorsement or other affiliation with this book.

ISBN 13: 978-0-321-68368-7

ISBN 10: 0-321-68368-4

9 8 7 6 5 4 3 2 1

Printed and bound in the United States of America

*For my wife, Rebecca Blood Garrett,
who makes all things possible.*



Table of Contents



CHAPTER 1

User Experience and Why It Matters	2
Everyday Miseries	3
Introducing User Experience	4
From Product Design to User Experience Design	7
Designing (for) Experience: Use Matters	8
User Experience and the Web	9
Good User Experience Is Good Business	12
Minding Your Users	17

CHAPTER 2

Meet the Elements	18
The Five Planes	19
The Surface Plane	20
The Skeleton Plane	20
The Structure Plane	20
The Scope Plane	21
The Strategy Plane	21
Building from Bottom to Top	21
A Basic Duality	25
The Elements of User Experience	28
The Strategy Plane	28
The Scope Plane	29
The Structure Plane	30
The Skeleton Plane	30
The Surface Plane	30
Using the Elements	31

CHAPTER 3

The Strategy Plane

Product Objectives and User Needs	34
Defining the Strategy	36
Product Objectives	37
Business Goals	37
Brand Identity	38
Success Metrics	39
User Needs	42
User Segmentation	42
Usability and User Research	46
Creating Personas	49
Team Roles and Process	52



CHAPTER 4

The Scope Plane

Functional Specifications and Content Requirements	56
Defining the Scope	58
Reason #1: So You Know What You're Building	59
Reason #2: So You Know What You're Not Building	60
Functionality and Content	61
Defining Requirements	65
Functional Specifications	68
Writing It Down	69
Content Requirements	71
Prioritizing Requirements	74





CHAPTER 5

The Structure Plane

Interaction Design and Information Architecture	78
Defining the Structure	80
Interaction Design	81
Conceptual Models	83
Error Handling	86
Information Architecture	88
Structuring Content	89
Architectural Approaches	92
Organizing Principles	96
Language and Metadata	98
Team Roles and Process	101



CHAPTER 6

The Skeleton Plane

Interface Design, Navigation Design, and Information Design	106
Defining the Skeleton	108
Convention and Metaphor	110
Interface Design	114
Navigation Design	118
Information Design	124
Wayfinding	127
Wireframes	128

CHAPTER 7

The Surface Plane

Sensory Design	132
Defining the Surface	134
Making Sense of the Senses	135
Smell and Taste	135
Touch	135
Hearing	136
Vision	136
Follow the Eye	137
Contrast and Uniformity	139
Internal and External Consistency	143
Color Palettes and Typography	145
Design Comps and Style Guides	148



CHAPTER 8

The Elements Applied	152
Asking the Right Questions	157
The Marathon and the Sprint	159
Index	164

This page intentionally left blank

Introduction to the Second Edition

Let's cut to the chase: It's the second edition. What's different?

The main difference between this edition and the first is that this book is no longer just about Web sites. Yes, most of the examples are still Web-related, but overall, the themes, concepts, and principles apply to products and services of all kinds.

There are two reasons for this, both having to do with what's happened over the last ten years. One is what's happened to *Elements*, and one is what's happened to user experience itself.

Over the years, I've heard from (or heard about) people who have applied the *Elements* model to products that have nothing to do with the Web. In some cases they were Web designers asked to take on something new, like a mobile application. In other cases, they were designers of other kinds of products who somehow came across *Elements* and saw a connection to their own work.

Meanwhile, the field of user experience has broadened its horizons. Practitioners now regularly talk about the impact and value of user experience design in areas far beyond the limited context of the Web or even screen-based interactive applications that dominated the conversation back when this book was first written.

This new edition of the book takes a similarly broad view. The Web is still central to the book, if only to acknowledge the model's roots in that medium. But this book doesn't require an insider's knowledge of how Web development happens—so even if you don't create Web sites, you should be able to see how to apply these ideas in your own work.

Despite all this, those of you who have read the first edition should rest assured: This is not a radical reinvention. It's a honing and refinement of the familiar Elements model you know (and hopefully love), with the same core ideas and philosophy intact. The little things change, but the big ones really don't

I remain gratified and humbled by where people have taken *Elements*. I can't wait to see what happens next!

Jesse James Garrett

November 2010

Introduction to the First Edition

This is not a how-to book. There are many, many books out there that explain how Web sites get made. This is not one of them.

This is not a book about technology. There is not a single line of code to be found between these covers.

This is not a book of answers. Instead, this book is about asking the right questions.

This book will tell you what you need to know before you go read those other books. If you need the big picture, if you need to understand the context for the decisions that user experience practitioners make, this book is for you.

This book is designed to be read easily in just a few hours. If you're a newcomer to the world of user experience—maybe you're an executive responsible for hiring a user experience team, or maybe you're a writer or designer just finding your way into this field—this book will give you the foundation you need. If you're already familiar with the methods and concerns of the field of user experience,

this book will help you communicate them more effectively to the people you work with.

The Story Behind the Book

Because I get asked about it a lot, here is the story of how *The Elements of User Experience* came to be.

In late 1999, I became the first information architect hired into a long-established Web design consultancy. In many ways, I was responsible for defining my position and educating people both about what I did, and how it fit in with what they did. Initially, they were perhaps cautious and a bit wary, but soon they came to recognize that I was there to make their jobs easier, not harder, and that my presence did not mean their authority was diminished.

Simultaneously, I was compiling a personal collection of online material related to my work. (This would eventually find its way onto the Web as my information architecture resources page at [www.jjg.net/ia/.](http://www.jjg.net/ia/)) While I was doing this research, I was continually frustrated by the seemingly arbitrary and random use of different terms for the basic concepts in the field. What one source called information design appeared to be the same as what another called information architecture. A third rolled everything together under interface design.

Over the course of late 1999 and January 2000, I struggled to arrive at a self-consistent set of definitions for these concerns and to find a way to express the relationships between them. But I was busy with actual paying work as well, and the model I was trying to formulate wasn't really working out anyway; so by the end of January I had given up on the whole idea.

That March I traveled to Austin, Texas, for the annual South by Southwest Interactive Festival. It was an engaging and thought-provoking week during which I didn't get much sleep—the conference's schedule of day and night activities begins to resemble a marathon after a couple of days.

At the end of that week, as I walked through the terminal of the airport in Austin preparing to board the plane back to San Francisco, it abruptly popped into my head: a three-dimensional matrix that captured all of my ideas. I waited patiently until we boarded the plane. As soon as I reached my seat, I pulled out a notebook and sketched it all out.

Upon my return to San Francisco, I was almost immediately laid up with an enervating head cold. I spent about a week sliding in and out of a fevered delirium. When I felt particularly lucid, I worked on turning my notebook sketch into a finished diagram that would fit neatly onto a letter-size piece of paper. I called it "The Elements of User Experience." Later I would hear about how, for many people, that title evoked memories of periodic tables and Strunk and White. Unfortunately, none of these associations was in my mind when I chose that title—I chose *elements* out of a thesaurus to replace the more awkward and technical-sounding *components*.

On March 30, I posted the final product on the Web. (It's still there; you can find the original diagram at www.jjg.net/ia/elements.pdf.) The diagram started getting some attention, first from Peter Merholz and Jeffrey Veen, who would later become my partners in Adaptive Path. Soon after, I spoke with more people about it at the first Information Architecture Summit. Eventually I started hearing from people all over the world about how they had used the

diagram to educate their co-workers and to give their organizations a common vocabulary for discussing these issues.

In the year after it was first released, “The Elements of User Experience” was downloaded from my site more than 20,000 times. I began to hear about how it was being used in large organizations and tiny Web development groups to help them work and communicate more effectively. By this time, I was beginning to formulate the idea for a book that would address this need better than a single sheet of paper could.

Another March rolled around, and again I found myself in Austin for South by Southwest. There I met Michael Nolan of New Riders Publishing and told him my idea. He was enthusiastic about it, and fortunately, his bosses turned out to be as well.

Thus, as much by luck as by intent, this book found its way into your hands. I hope that what you do with the ideas presented here is as enlightening and rewarding for you as putting them together in this book has been for me.

Jesse James Garrett

July 2002

This page intentionally left blank

This page intentionally left blank

chapter 4

The Scope Plane

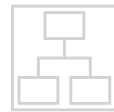
Functional Specifications and
Content Requirements



Surface



Skeleton



Structure

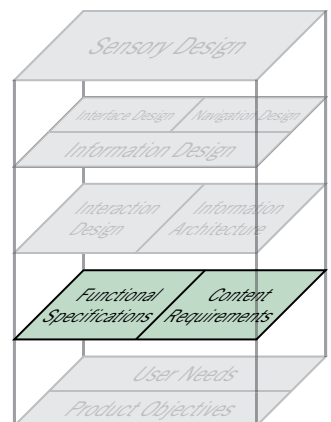


Scope



Strategy

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 product objectives into specific requirements for what content and functionality the product will offer to users.



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 design 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 at all with developing for the Web.

But none of this explains why we couldn't get the product to launch. The big stumbling block was an unwillingness to define 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 bother to define 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 clearly articulate 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 clear 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 the design and development of some crucial aspect of the product, when in fact no one is.



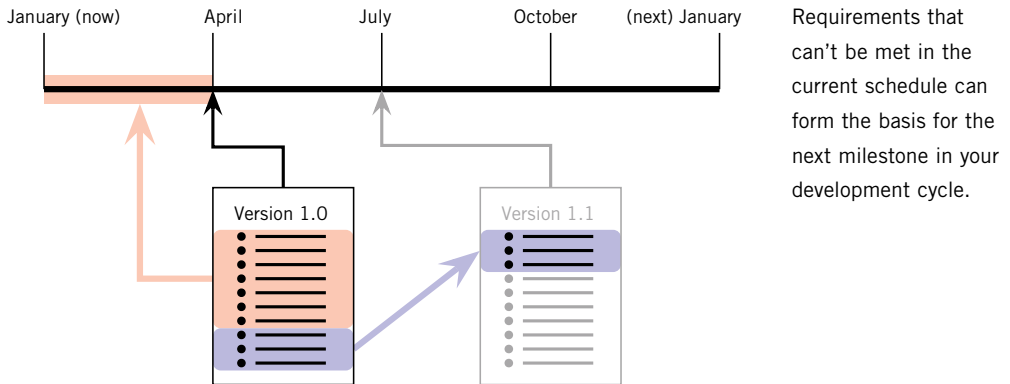
Having a defined set of requirements allows you to parcel out responsibility for the work more efficiently. Seeing the entire scope mapped out enables you to see connections between individual requirements that might not otherwise be apparent. For example, in early discussions, the support documentation and the product spec sheets may have seemed like separate content features, 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 clearly identified requirements provides you with a framework for evaluating those ideas as they come along, helping you understand how (or if) they fit into what you've already committed to build.

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 requirements, and stockpiling requests that don't fit as possibilities for future releases, you can manage the entire process in a more deliberate way.



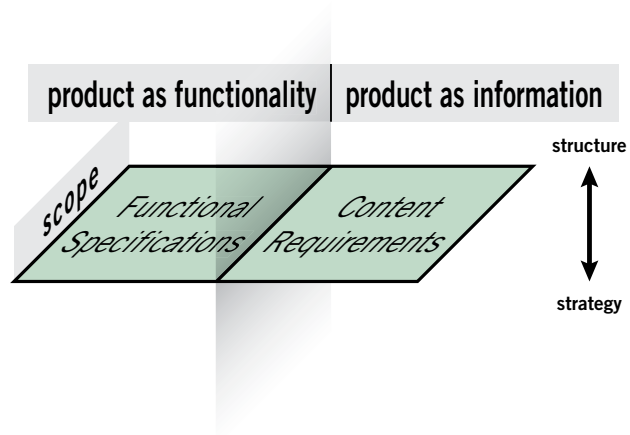


If you don't consciously manage your requirements, you'll get caught in the dreaded "scope creep." The image this always brings to mind for me is the snowball that rolls forward an inch—and then another—picking up a little extra snow with each turn until it is barreling down the hill, getting bigger 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, crushing deadlines and budget estimates on its way toward an inevitable final crash.

Functionality and Content

On the scope plane, we start from the abstract question of "Why are we making this product?" 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 vehicle for functionality and the Web as an information medium starts coming into play on the scope plane. On the functionality side, we're concerned with what would be considered the feature set of the software product. On the information 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 through functional requirements or **functional specifications**. 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. In other cases, the specifications are developed soon after the



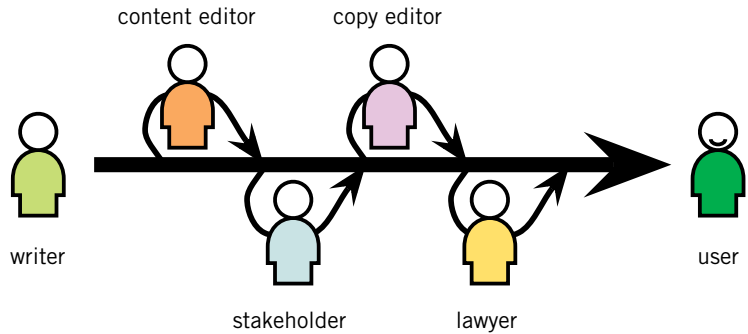
requirements, filling in details of implementation. But most of the time, these terms are interchangeable—in fact, some people use the term *functional requirements specification* just to make sure they’ve covered all the bases. I’ll use *functional specifications* to refer to the document itself, and *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 involves a less formal requirements-definition process than 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 process for defining **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, pure content sites are usually handled through 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 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, or the device they are using? The CMS will need to be able to accomplish that level of complex delivery.

Similarly, the functional requirements of any technology product 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.



Defining Requirements

Some requirements apply to the product 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 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 (such as a large number of functionally identical PDFs of product manuals) that the content requirements can only be very general.

The most productive source for requirements will always be your users themselves. But more often, your requirements will come from stakeholders, people in your organization who have some say over what goes into your product.

In either case, 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 in your product.

Whether you are defining requirements with help from stakeholders inside your organization or working directly with 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 the 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. Sometimes that solution is unworkable, or it addresses a symptom rather than the underlying cause of the problem. By exploring these suggestions, you can sometimes arrive at completely different requirements that solve the real problem.

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

Ironically, sometimes the people most deeply involved in creating and working with a product are the ones least able to imagine new directions for it. When you spend all your time immersed in maintaining an existing product, you can often forget which of your constraints are real, and which are simply products of historical choices. 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 unfamiliar perspectives—and having the opportunity to respond to them—encourages people to think in broader terms about both the problems involved in developing the product and the possible solutions.

Whatever device we are designing for—or if we are designing the device itself—our feature set will need to take into account hardware requirements, too. Does the device have a camera? GPS? Gyroscopic position sensors? These considerations will inform and constrain your functional possibilities.

Generating requirements is often a matter of finding ways to remove impediments. For example, assume that you have a user who has already decided to purchase a product—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 to help 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 product objectives as well. Has a competitor found a particularly effective feature to meet one of these strategic objectives? How have they addressed the same trade-offs and compromises we face?



Even products that aren't direct competitors can serve as fertile sources for possible requirements. Some gaming platforms, for example, offer users the ability to create social groups of fellow players. Adopting or building on their approach when scoping a similar feature for our digital video recorder may give us an advantage over our direct competition.

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—because it is! A bad approach to specs becomes a self-fulfilling prophecy.

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 specs that actually work. 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 make the process of defining specifications lightweight enough that the spec doesn't become a project separate from developing the product itself.



In other words, documentation won't solve your problems. Definition will. It's not about volume or detail. It's about clarity and accuracy. Specs don't have to embody every aspect of the product—just the ones that need definition to avoid confusion in the design and development process. And specs don't need to capture some idealized future state for the product—just the decisions that have been made in the course of creating it.

Writing It Down

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 most popular videos will be highlighted.*
2. *Videos with the most views in the last week will appear at the top of the list.*



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 popular? Videos with the most comments? The ones with the most “like” votes? And what constitutes highlighting them?

The second example defines our goal in specific detail, defining what we mean by popular and describing a mechanism for highlighting. 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 approval 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 quantitatively. Just as success metrics make strategic goals quantifiable, defining a requirement in quantitative terms can help us 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 allows three-digit user numbers, 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 images, audio, and video can be more important than the accompanying text. 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 content types associated with a 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, "We should have FAQs." But the term *FAQ* really only refers to a content format: a simple series of questions and answers. The real value of an FAQ 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 or video, and file sizes for downloadable, stand-alone content elements like audio files or PDF documents. These size estimates don't have to be precise—approximations are fine. We only have to collect the essential information needed to design an appropriate vehicle for 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 enables 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 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 product 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.

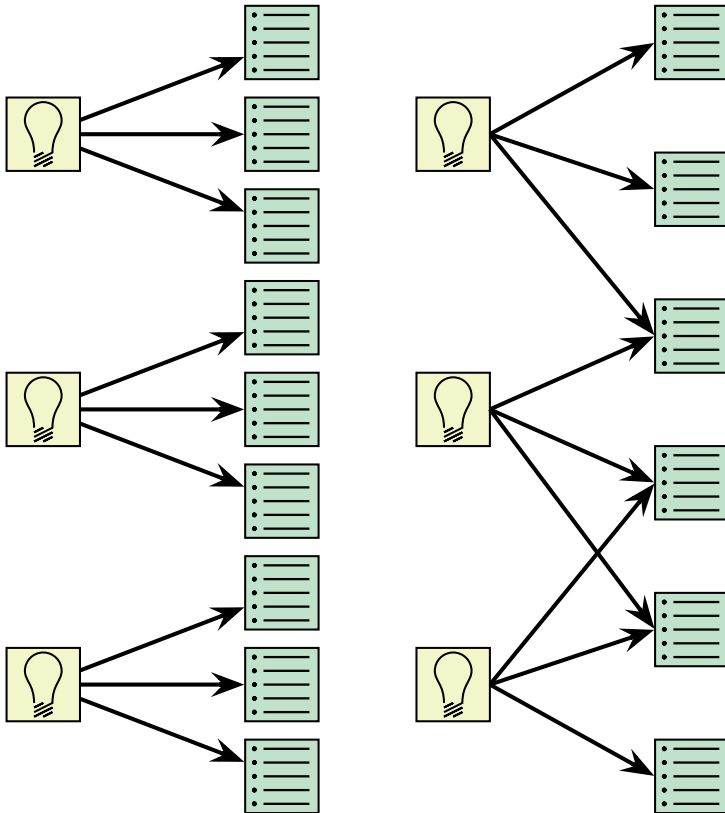
If your site has to serve multiple audiences with divergent needs, knowing which audience a piece of content is intended for can help you make better decisions about how to present that content. Information intended for children requires a different approach from information intended for their parents; information for both of them needs yet a third approach.

For projects that involve working with a lot of existing content, much of the information that will feed your requirements 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.





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

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 product 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 demand more resources—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 on the features around them to inform the user on how best to use the content provided. This inevitably leads to conflicts between features. Some features will require trade-offs with others in order to produce a coherent, consistent whole. For example, users may want a one-step order submission process—but the tangle of legacy databases the site uses 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 want to reexamine some of your strategic objectives. If you find yourself revisiting many aspects of your strategy, however, you've probably jumped into defining 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 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. Thus the requirements definition 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?



Index

A

- action buttons, described, 117
- architectural approaches.
See also information architecture
 - hierarchical structure, 93
 - hub-and-spoke structure, 93
 - matrix structure, 93–94
 - organic structure, 94–95
 - sequential structure, 95
 - tree structure, 93
- attitudes of users,
considering, 44
- audience, considering content
for, 74
- audience segments, basing on
demographics, 43–44

B

- brand identity, considering,
38–39
- branding requirements,
considering, 65
- business goals, defining,
37–38

C

- card sorting, 49
- checkboxes, described, 116,
118
- CMS (content management
system), 63–64
- color palettes, considering in
sensory design, 145–148
- communication, importance
of, 12–13
- comp, defined, 148
- competitors, getting
inspiration from, 67–68
- conceptual models, 83–85,
97, 111–112
- consistency
 - considering in sensory
design, 143–144
 - internal versus external,
143–144
- content
 - considering audience for,
74
 - defined, 12
 - format versus purpose, 72
 - impact on user experience,
32
 - structuring, 89–92
 - versus technology, 32

content features, updating, 73–74
 content inventory, taking, 74
 content requirements, 71–74. *See also* project requirements; requirements considering, 29 considering on scope plane, 61–64 defining, 63
 contextual inquiry, 47
 contextual navigation, 122
 contrast and uniformity, considering in sensory design, 139–143
 controlled vocabulary, using, 98–99
 convention and metaphor, 110–113
 conventions considering, 84 developing, 116
 conversion rate, measuring, 13–15
 courtesy navigation, 122–123
 customer loyalty, 13

D

demographics, applying to audience segments, 43–44
 design. *See also* product design; user-centered design by default, 156 by fiat, 156 by mimicry, 156
 design comps, considering in sensory design, 148–151
 design consistency, enforcing, 144

design systems, documenting, 150
 documenting. *See also* wireframes design systems, 150 functional specifications, 69–71
 dropdown lists, described, 117–118

E

efficiency, improving, 15–16
 Elements model. *See* planes
 error handling, 86–88 correction, 86–87 prevention, 86–87 recovery, 86–87 undo function, 88
 error messages, occurrence of, 64
 eyetracking, considering in sensory design, 137–139

F

facets benefits of, 121 defined, 98
 FAQs, considering, 72
 features conflicts between, 76 drawing analogies to experiences, 113 implementation of, 76 suggestions, 76–77 time constraints on, 76 use of terminology, 62
 five planes. *See also* scope plane; skeleton plane; strategy plane; structure

plane; surface plane; user experience
 building from bottom to top, 162
 choices related to, 23
 considering, 155
 decisions related to, 24
 dependencies of, 22–23
 failures on, 162–163
 scope, 21, 29
 skeleton, 20, 30
 strategy, 21, 28
 structure, 20, 30
 surface, 20, 30
 using elements of, 31–33
 working on, 24
 functional specifications, 29, 62–64, 68–71

G

global navigation, 120–121
 grid-based layout technique, 141–142

H

hearing experience, 136
 hierarchical structure, 93
 hub-and-spoke structure, 93
 hyperlinks, underutilization of, 122

I

impressions, defined, 40
 indexes, using for navigation, 123
 information architecture, 81, 88–89. *See also* architectural approaches

adaptability of, 91–92
 approaches, 92–95
 bottom-up approach, 90
 categories in, 90–91
 conceptual structure of, 97
 considering, 30
 controlled vocabulary, 98–99
 diagramming, 101–105
 documenting, 101–105
 flow of language in, 95
 language, 98–101
 metadata, 98–101
 nodes in, 92–93
 organizing principles, 96–98
 structuring content, 89–92
 top-down approach, 89–90
 use of thesaurus, 99
 Visual Vocabulary, 102–103
 information design, 108–109.
See also interface design
 considering, 30, 45
 organizing principles, 124–126
 role in interface design, 126
 visual type of, 124
 wayfinding, 127
 inline navigation, 122
 interaction design, 81–82
 conceptual models, 83–85
 considering, 30
 consistency of, 111–112
 convention and metaphor, 110
 conventions, 84
 error handling, 86–88
 interface conventions, changes in, 116

interface design, 108–109. *See also* information design
 considering, 30
 success of, 114
 interface elements
 action buttons, 117
 checkboxes, 116, 118
 dropdown lists, 117–118
 list boxes, 117
 radio buttons, 116, 118
 text fields, 116

L

language and metadata,
 98–101
 list boxes, described, 117
 local navigation, 121

M

marathon and sprint, 159–163
 market research methods, 46
 matrix structure, 93–94
 metadata and language,
 98–101
 metaphor and convention,
 110–113

N

navigation design, 108–109.
 See also wayfinding
 considering, 30
 goals of, 118–119
 importance of, 119–120
 navigation systems, 120
 contextual, 122
 courtesy, 122–123
 global, 120–121

indexes, 123
 inline, 122
 local, 121
 persistent elements, 120
 site maps, 123
 supplementary, 121
 navigation tools, 123
 nodes
 child and parent, 93
 in hierarchical structure,
 93
 in matrix structure, 93–94
 in organic structure, 94
 organizing principles of,
 96
 role in information
 structures, 92–93
 nomenclature, 98

O

objectives. *See* product
 objectives
 operating systems, interface
 elements, 116–118
 Orbitz color palette, 146
 organic structure, 94–95
 organizing principles,
 applying, 96–98

P

page layout, 128–131
 persistent navigation
 elements, 120
 personas
 creating, 49–51
 including in requirements,
 67
 phones, conventions related
 to, 110–111

planes. *See also* scope
plane; skeleton plane;
strategy plane; structure
plane; surface plane; user
experience
 building from bottom to
 top, 162
 choices related to, 23
 considering, 155
 decisions related to, 24
 dependencies of, 22–23
 failures on, 162–163
 scope, 21, 29
 skeleton, 20, 30
 strategy, 21, 28
 structure, 20, 30
 surface, 20, 30
 using elements of, 31–33
 working on, 24
problem solving, 157–159
product design, concept of,
 7–8. *See also* design; user-
 centered design
product goals, clarifying,
 61–64
product objectives
 brand identity, 38–39
 business goals, 37–38
 considering, 28, 36, 91
 success metrics, 39–41
products
 describing feature sets of,
 29
 as functionality, 27–29,
 161
 as information, 27–29, 161
 planning, 59–60
 success of, 12–13

project requirements. *See*
 also content requirements;
 requirements
 clarifying, 59–61
 defining, 65–68
 generating, 67
 including personas in, 67
 managing, 61
projects, planning, 24
prototypes, 48–49
psychographic profiles, 44

Q

questions, posing to users,
 158–159

R

radio buttons, described, 116,
 118
requirements. *See also* content
 requirements; project
 requirements
 collecting ideas for, 74
 developing, 73
 evaluating, 75
 prioritizing, 74–77
 strategic objectives for, 75
research tools
 card sorting, 49
 contextual inquiry, 47
 market research methods,
 46
 prototypes, 48–49
 task analysis, 47
 user testing, 47–48
return visits, measuring, 41
revenue, increasing, 14–15
ROI (return on investment),
 13

S

- scope, defining, 58–59
- scope plane, 21, 29. *See also* planes
 - content, 61–64
 - content requirements, 29, 71–74
 - defining content requirements, 63
 - defining requirements, 65–68
 - functional specifications, 29, 62, 68–71
 - functionality, 61–64
 - prioritizing requirements, 74–77
 - role in bottom-up architecture, 90
 - strategy component of, 75
- sensory design, 134
 - color palettes, 145–148
 - consistency, 143–144
 - contrast and uniformity, 139–143
 - design comps, 148–151
 - eyetracking, 137–139
 - hearing, 136
 - smell and taste, 135
 - style guides, 148–151
 - touch, 135–136
 - typography, 145–148
 - vision, 136–137
- sensory experience, considering, 30
- sequential structure, 95
- site, use of terminology, 9–10
- site maps, use of, 102, 123
- skeleton, defining, 108–109
- skeleton plane, 20, 30. *See also* planes
 - convention, 110–113
 - information design, 30, 108–109, 124–127
 - interface design, 30, 108–109, 114–118
 - metaphor, 110–113
 - navigation design, 30, 108–109, 118–123
 - wireframes, 128–131
- smell and taste, considering in sensory design, 135
- software, focus of, 82–83
- solutions, finding for problems, 157–159
- Southwest Airlines site, 85
- sprint
 - defined, 159
 - strategy, 160
- stakeholders
 - concerns about strategy, 77
 - defined, 52
 - resolving conflicts between, 77
- strategic goals, meeting, 40
- strategic objectives, requirements for, 75
- strategists, employment of, 52
- strategy document
 - contents of, 53–54
 - hierarchy of, 77
- strategy plane, 21, 28. *See also* planes
 - product objectives, 28
 - role in top-down architecture, 90
 - user needs, 28

structure plane, 20, 30. *See also* planes
information architecture, 30, 81, 88–101
interaction design, 30, 80–88
team roles and process, 101–105

structures
defining, 80–81
hierarchical structure, 93
hub-and-spoke structure, 93
matrix structure, 93–94
organic structure, 94–95
sequential structure, 95
tree structure, 93

style guides, considering in sensory design, 148–151

success metrics, 39–41

supplementary navigation, 121

surface, defining, 134

surface plane, 20, 30. *See also* planes
color palettes, 145–148
consistency, 143–144
contrast and uniformity, 139–143
design comps, 148–151
eyetracking, 137–139
hearing, 136
senses, 135–137
sensory experience, 30
smell and taste, 135
style guides, 148–151
touch, 135–136
typography, 145–148
vision, 136–137

T

task analysis, 47

taste and smell, considering in sensory design, 135

team roles and process
for strategy plane, 52–54
for structure plane, 101–105

technology tools, constraints on, 115

technology versus content, 32

telephones, conventions related to, 110–111

text elements, considering, 147

text fields, described, 116

thesaurus, using, 99

touch experience, 135–136

tree structure, 93

typography, considering in sensory design, 147–148

U

undo function, using in error handling, 88

uniformity and contrast, considering in sensory design, 139–143

usability, defined, 48

user acceptance testing, 158

user behavior. *See* interaction design

user experience. *See also* planes
choices made about, 156
defined, 6
delegating responsibility for, 31
design, 7–8

user experience (*continued*)

- design by default, 156
- design by fiat, 156
- design by mimicry, 156
- designing for, 8–9
- details of, 6
- drawing analogies to
 - features, 113
- formation of community, 26
- impact of content on, 32
- impact on business, 12–17
- including in development process, 11
- metaphor for, 159
- quality of, 12–17
- successful design of, 154
- on Web, 9–12
- user models, creating, 49–51
- user needs
 - considering, 28, 36
 - creating personas, 49–51
 - identifying, 42
 - psychographic profiles, 44
 - research tools related to, 46
 - understanding, 46–49
 - usability, 46–49
- user profiles, creating, 49–51
- user research, 46–49, 51
- user segments, 42–46
 - importance of, 45
 - revising after research, 45
- user testing, 47–48
- user-centered design, 17. *See also* design; product design
- users
 - choices presented to, 10
 - posing questions to, 158–159

V

- vision document
 - contents of, 53–54
 - hierarchy of, 77
- vision experience, 136–137
- visual designs, matching to wireframes, 149
- visual neutral layout, 140
- Visual Vocabulary, 102–103

W

- wayfinding, 127. *See also* navigation design
- Web
 - commercial interests on, 26
 - content requirements, 62
 - evolution of, 25–26
 - functional specifications, 62
 - user experience on, 9–12
- Web elements
 - functionality, 27–28
 - information medium, 27–28
- Web sites, failure of, 36
- wireframes. *See also* documenting
 - matching visual designs to, 149
 - skeleton plane, 128–131