THE ELEMENTS OF USER EXPERIENCE

SECOND EDITION

USER-CENTERED DESIGN FOR THE WEB AND BEYOND

Jesse James Garrett

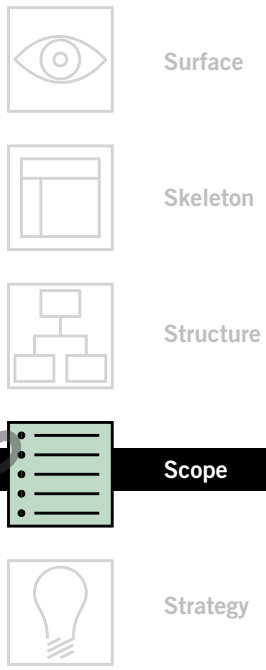
Table of Contents

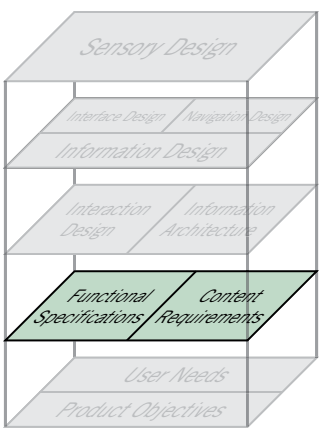
chapter 4

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 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.

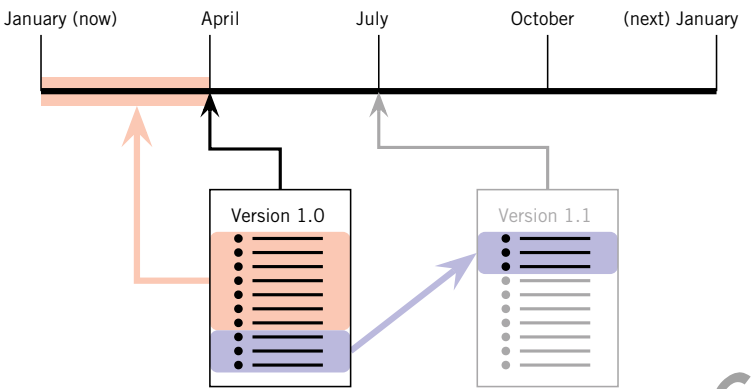
In the absence of clear requirements, your project will probably turn out like a schoolyard game of "Telephone"—each person on the team gets an impression of the product via word of mouth, and everyone's description ends up slightly different. Or even worse, everyone assumes someone else is managing 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.

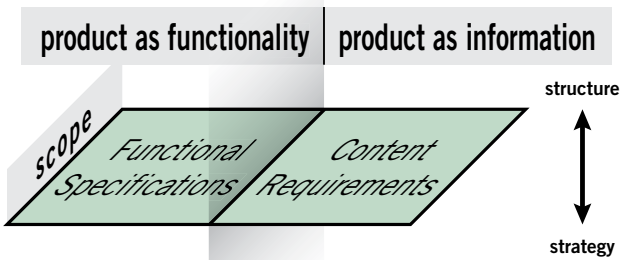


Requirements that can't be met in the current schedule can form the basis for the next milestone in your development cycle.

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.

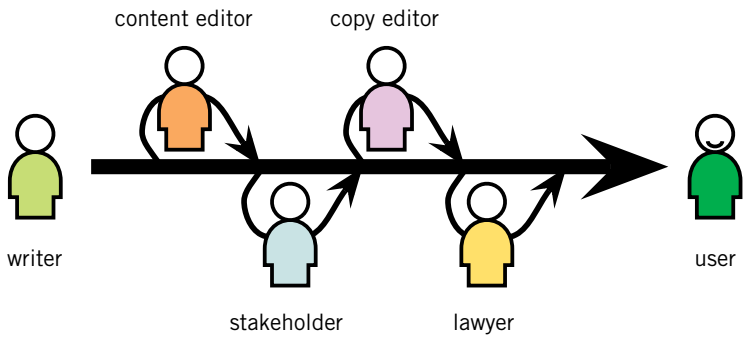
Content and functionality seem just about as different as two things could be, but when it comes to defining scope, they can be addressed in very similar ways. Throughout this chapter, I'll use the term *feature* to refer to both software functions and content offerings.

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