59: There's Nothing Functional about a Functional Spec

Don’t write a functional specifications document

These blueprint docs usually wind up having almost nothing to do with the finished product. Here’s why:

Functional specs are fantasies

They don’t reflect reality. An app is not real until builders are building it, designers are designing it, and people are using it. Functional specs are just words on paper.

Functional specs are about appeasement

They’re about making everyone feel involved and happy which, while warm and fuzzy, isn’t all that helpful. They’re never about making tough choices and exposing costs, things that need to happen to build a great app.

Functional specs only lead to an illusion of agreement

A bunch of people agreeing on paragraphs of text isn’t a true agreement. Everyone may be reading the same thing but they’re thinking something different. This inevitably comes out later on: “Wait, that’s not what I had in mind.” “Huh? That’s not how we described it.” “Yes it was and we all agreed on it — you even signed off on it.” You know the drill.

Functional specs force you to make the most important decisions when you have the least information

You know the least about something when you begin to build it. The more you build it, the more you use it, the more you know it. That’s when you should be making decisions — when you have more information, not less.

Functional specs lead to feature overload

There's no pushback during spec phase. There's no cost to writing something down and adding another bullet point. You can please someone who's a pain by adding their pet feature. And then you wind up designing to those bullet points, not to humans. And that's how you wind up with an overloaded site that's got 30 tabs across the top of the screen.

Functional specs don't let you evolve, change,and reassess

A feature is signed off and agreed on. Even if you realize during development that it's a bad idea, you're stuck with it. Specs don't deal with the reality that once you start building something, everything changes.

So what should you do in place of a spec? Go with a briefer alternative that moves you toward something real. Write a one page story about what the app needs to do. Use plain language and make it quick. If it takes more than a page to explain it, then it's too complex. This process shouldn't take more than one day.

Then begin building the interface — the interface will be the alternative to the functional spec. Draw some quick and simple paper sketches. Then start coding it into html. Unlike paragraphs of text that are open to alternate interpretations, interface designs are common ground that everyone can agree on.

Confusion disappears when everyone starts using the same screens. Build an interface everyone can start looking at, using, clicking through, and "feeling" before you start worrying about back-end code. Get yourself in front of the customer experience as much as possible.

Forget about locked-in specs. They force you to make big, key decisions too early in the process. Bypass the spec phase and you'll keep change cheap and stay flexible.

Useless Specs

> *A "spec" is close to useless. I have never seen a spec that was both big enough to be useful and accurate.*
>
> *And I have seen lots of total crap work that was based on specs. It's the single worst way to write software, because it by definition means that the software was written to match theory, not reality.*

—Linus Torvalds, creator of Linux (from: Linux: Linus On Specifications)

Fight the blockers

> *I found the people insisting on extensive requirements documents before starting any design were really 'blockers' just trying to slow the process down (and usually people with nothing to contribute on design or innovative thinking).*
>
> *All our best work was done with a few concepts in our heads about improving a site, doing a quick prototype (static), changing the design a bit and then building a live prototype with real data. After kicking the tires on this prototype, we usually had a real project in motion and good result.*

—Mark Gallagher, corporate intranet developer (from Signal vs. Noise)

60: Don't Do Dead Documents

Eliminate unnecessary paperwork

Avoiding functional specs is a good start but don't stop there; Prevent excess paperwork everywhere. Unless a document is actually going to morph into something real, don't produce it.

Build, don't write. If you need to explain something, try mocking it up and prototyping it rather than writing a longwinded document. An actual interface or prototype is on its way to becoming a real product. A piece of paper, on the other hand, is only on its way to the garbage can.

Here's an example: If a wireframe document is destined to stop and never directly become the actual design, don't bother doing it. If the wireframe starts as a wireframe and then morphs into the actual design, go for it.

Documents that live separately from your application are worthless. They don't get you anywhere. Everything you do should evolve into the real thing. If a document stops before it turns real, it's dead.

No One's Going to Read It

> *I can't even count how many multi-page product specifications or business requirement documents that have languished, unread, gathering dust nearby my dev team while we coded away, discussing problems, asking questions and user testing as we went. I've even worked with developers who've spent hours writing long, descriptive emails or coding standards documents that also went unread.*
>
> *Webapps don't move forward with copious documentation. Software development is a constantly shifting, iterative process that involves interaction, snap decisions, and impossible-to-predict issues that crop up along the way. None of this can or should be captured on paper.*
>
> *Don't waste your time typing up that long visionary tome; no one's going to read it. Take consolation in the fact that if you give your product enough room to grow itself, in the end it won't resemble anything you wrote about anyway.*

—Gina Trapani, web developer and editor of Lifehacker, the productivity and software guide

61: Tell Me a Quick Story

Write stories, not details

If you do find yourself requiring words to explain a new feature or concept, write a brief story about it. Don't get into the technical or design details, just tell a quick story. Do it in a human way, like you would in normal conversation.

It doesn't need to be an essay. Just give the flow of what happens. And if you can include the brief story in context with screens you are developing, all the better.

Stick to the experience instead of getting hung up on the details. Think strategy, not tactics. The tactics will fall into place once you begin building that part of your app. Right now you just want to get a story going that will initiate conversation and get you on the right track.

62: Use Real Words

Insert actual text instead of lorem ipsum

Lorem ipsum dolor is a trusted friend of designers. Dummy text helps people get what the design will look like once it's fleshed out. But dummy text can be dangerous too.

Lorem ipsum changes the way copy is viewed. It reduces text-based content to a visual design element — a shape of text — instead of what it should be: valuable information someone is going to have to enter and/or read. Dummy text means you won't see the inevitable variations that show up once real information is entered. It means you won't know what it's like to fill out forms on your site. Dummy text is a veil between you and reality.

You need real copy to know how long certain fields should be. You need real copy to see how tables will expand or contract. You need real copy to know what your app truly looks like.

As soon as you can, use real and relevant words. If your site or application requires data input, enter the real deal. And actually type in the text — don't just paste it in from another source. If it's a name, type a real name. If it's a city, type a real city. If it's a password, and it's repeated twice, type it twice.

Sure, it's easier to just run down the forms and fill the fields with garbage ("asdsadklja" "123usadfjasld" "snaxn2q9e7") in order to plow through them quickly. But that's not real. That's not what your customers are going to do. Is it really smart to take a shortcut when customers are forced to take the long road? When you just enter fake copy in rapid-fire fashion, you don't know what it really feels like to fill out that form.

Do as your customers do and you'll understand them better. When you understand them better, and feel what they feel, you'll build a better interface.

Lorem Ipsum Garbage

> *By not having the imagination to imagine what the content "might" be, a design consideration is lost. Meaning becomes obfuscated because "it's just text", understandability gets compromised because nobody realized that this text stuff was actually meant to be read. Opportunities get lost because the lorem ipsum garbage that you used instead of real content didn't suggest opportunities. The text then gets made really small, because, it's not meant to be used, we might as well create loads of that lovely white space.*

—Tom Smith, designer and developer (from I hate Lorem Ipsum and Lorem Ipsum Users)

63: Personify Your Product

What is your product's personality type?

Think of your product as a person. What type of person do you want it to be? Polite? Stern? Forgiving? Strict? Funny? Deadpan? Serious? Loose? Do you want to come off as paranoid or trusting? As a know-it-all? Or modest and likable?

Once you decide, always keep those personality traits in mind as the product is built. Use them to guide the copywriting, the interface, and the feature set. Whenever you make a change, ask yourself if that change fits your app's personality.

Your product has a voice — and it's talking to your customers 24 hours a day.