Joel on Software

Painless Functional Specifications - Part 4: Tips

by Joel Spolsky

Sunday, October 15, 2000

OK, we've talked about why you need a spec, what a spec has in it, and who should write them. In this fourth and final part of the series I'll share some of my advice for writing good specs.

The biggest complaint you'll hear from teams that *do* write specs is that "nobody reads them." When nobody reads specs, the people who write them tend to get a little bit cynical. It's like the old Dilbert cartoon in which engineers use stacks of 4-inch thick specs to build extensions to their cubicles. At your typical big, bureaucratic company, everybody spends months and months writing boring specs. Once the spec is done, it goes up on the shelf, never to be taken down again, and the product is implemented from scratch without any regard to what the spec said, because nobody read the spec, because it was *so dang mind-numbing*. The very process of writing the spec might have been a good exercise, because it forced everyone, at least, to think over the issues. But the fact that the spec was shelved (unread and unloved) when it was completed makes people feel like it was all a bunch of work for naught.

Also, if your spec never gets read, you get a lot of arguments when the finished product is delivered. Somebody (management, marketing, or a customer) says: "wait a minute! You promised me that there would be a Clam Steamer! Where's the clam steamer?" And the programmers say, "no, actually, if you look on the spec on chapter 3, subchapter 4, paragraph 2.3.0.1, you'll see it says quite explicitly 'no clam steamer.'" But that doesn't satisfy the customer, who is always right, so the grumpy programmers have to go retrofit a clam steamer into the thing (making them even more cynical about specs). Or a manager says, "hey, all the wording on this dialog is too verbose, and there should be an advertisement at the top of every dialog box." And the programmers say, in frustration, "but you *approved the spec* which *precisely* listed the layout and contents of every dialog box!" But of course, the manager hadn't actually *read* the spec, because when he tried, his brain started seeping out through his eye sockets, and anyway, it was interfering with his Tuesday golf game.

So. Specs are good, but not if nobody reads them. As a spec-writer, you have to *trick people* into reading your stuff, and you should also

probably make an effort not to cause any already-too-small brains to leak out through eye-sockets.

Tricking people into reading your stuff is usually just a matter of good writing. But it's not fair of me to just say "be a good writer" and leave it at that. Here are four easy rules that you absolutely *must* follow to make specs that get read.

Rule 1: Be Funny

Yep, rule number one in tricking people into reading your spec is to make the experience enjoyable. Don't tell me you weren't born funny, I don't buy it. Everybody has funny ideas all the time, they just self-censor them because they think that it's "unprofessional." Feh. Sometimes you have to break the rules.

If you read the volumes of garbage I've written on this web site, you'll notice that there are a few lame attempts at being funny scattered throughout. Just four paragraphs ago I was making a gross body-fluid joke and making fun of managers for playing golf. Even though I'm not really that funny, I still try pretty hard, and even the act of flailing around *trying* to be funny is in itself amusing, in a sad-clown sort of way. When you're writing a spec, an easy place to be funny is in the examples. Every time you need to tell a story about how a feature works, instead of saying:

- The user types Ctrl+N to create a new Employee table and starts entering the names of the employees.

write something like:

- Miss Piggy, poking at the keyboard with a eyeliner stick because her chubby little fingers are too fat to press individual keys, types Ctrl+N to create a new Boyfriend table and types in the single record "Kermit."

If you read a lot of Dave Barry, you'll discover that one of the easiest ways to be funny is to be *specific* when it's not called for. "Scrappy pugs" are funnier than "dogs." "Miss Piggy" is funnier than "the user". Instead of saying "special interests," say "left-handed avocado farmers." Instead of saying "People who refuse to clean up after their dogs should be punished," say that they should be "sent to prisons so lonely that the inmates have to pay spiders for sex."

Oh, and, by the way, if you think that it's unprofessional to be funny, then I'm sorry, but you just don't have a sense of humor. (Don't deny it. People without senses of humors always deny it. You can't fool me.) And if you work in a company where people will respect you less because your specs are breezy, funny, and enjoyable to read, then go find another company to work for, because life is just *too damn short* to spend your daylight hours in such a stern and miserable place.

Rule 2: Writing a spec is like writing code for a brain to execute

Here's why I think that programmers have trouble writing good specs.

When you write *code*, your primary audience is *the compiler*. Yeah, I know, people have to read code, too, but it's generally very hard for them. For most programmers it's hard enough to get the code into a state where the compiler reads it and correctly interprets it; worrying about making human-readable code is a luxury. Whether you write:

```
void print_count( FILE* a, char  *  b, int c ){
    fprintf(a, "there are %d %s\n", c, b);}

main(){ int n; n =
10; print_count(stdout, "employees", n) /* code
deliberately obfuscated */ }
```

or

```
printf("there are 10 employees\n");
```

you *get the same output*. Which is why, if you think about it, you tend to get programmers who write things like:

> Assume a function AddressOf(*x*) which is defined as the mapping from a user *x*, to the RFC-822 compliant email address of that user, an ANSI string. Let us assume user A and user B, where A wants to send an email to user B. So user A initiates a new message using any (but not all) of the techniques defined elsewhere, and types AddressOf(B) in the To: editbox.

This could also have been speced as:

> Miss Piggy wants to go to lunch, so she starts a new email and types Kermit's address in the "To:" box.
>
> **Technical note:** *the address must be a standard Internet address (RFC-822 compliant.)*

They both "mean" the same thing, *theoretically*, except that the first example is impossible to understand unless you carefully decode it, and the second example is easy to understand. Programmers often try to write specs which look like dense academic papers. They think that a "correct" spec needs to be "technically" correct and then they are off the hook.

The mistake is that when you write a spec, in addition to being correct, it has to be *understandable*, which, in programming terms, means that it needs to be written so that the human brain can "compile" it. One of the big differences between computers and human brains is that computers are willing to sit there patiently while you define the terms that you want to use later. But humans won't understand what you're talking about unless you motivate it first. Humans don't want to have to *decode* something, they just want to read it in order and understand it. For humans, you have to provide the big picture and *then* fill in the details. With computer programs, you start at the top and work your way to the bottom, with full details throughout. A computer doesn't care if your variable names are meaningful. A human brain understands things much better if you can paint a vivid picture in their mind by telling a story, even if it's just a fragment of a story, because our brains have evolved to understand stories.

If you show a chess board, in the middle of a real game of chess, to an experienced chess player for even a second or two, they will instantly be able to memorize the position of every piece. But if you move around a couple of pieces in nonsensical ways that couldn't happen in normal play (for example, put some pawns on the first row, or put both black bishops on black squares), it becomes much, much harder for them to memorize the board. This is different from the way computers think. A computer program that could memorize a chess board could memorize both possible and impossible layouts with equal ease. The way the human brain works is *not* random access; pathways tend to be strengthened in our brains and some things are just easier to understand than other things because they are more common.

So, when you're writing a spec, try to imagine the person you are addressing it to, and try to imagine what you're asking them to understand at every step. Sentence by sentence, ask yourself if the person reading this sentence will *understand it* at a deep level, in the context of what you've already told them. If some members of your target audience don't know what RFC-822 is, you either have to define it, or, at the very least, bury the mention of RFC-822 in a technical note, so that the executive-types who read the spec won't give up and stop reading the first time they see a lot of technical jargon.

Rule 3: Write as simply as possible

Don't use stilted, formal language because you think it's unprofessional to write in simple sentences. Use the simplest language you can.

People use words like "utilize" because they think that "use" looks unprofessional. (There's that word "unprofessional" again. Any time somebody tells you that you shouldn't do something because it's "unprofessional," you know that they've run out of *real* arguments.) In fact I think that many people think that clear writing means that something is wrong.

Break things down to short sentences. If you're having trouble writing a sentence clearly, break it into two or three shorter sentences. Avoid walls of text: entire pages with just text. People get scared and don't read them. When was the last time you noticed a popular magazine or newspaper with entire pages of text? Magazines will go so far as to take a quote from the article and print it, in the middle of the page, in a giant font, just to avoid the appearance of a full page of text. Use numbered or bulleted lists, pictures, charts, tables, and lots of whitespace so that the reading "looks" fluffier.

"Magazines will go so far as to take a quote from the article and print it, in the middle of the page, in a giant font, just to avoid the appearance of a full page of text."

Nothing improves a spec more than lots and lots of screenshots. A picture can be worth a thousand words. Anyone who writes specs for Windows software should invest in a copy of Visual Basic, and learn to use it *at least* well enough to create mockups of the screens. (For the Mac, use REAL Basic; for Web pages, use Front Page or Dreamweaver). Then capture these screenshots (Ctrl+PrtSc) and paste them into your spec.

Rule 4: Review and reread several times

Um, well, I was originally planning to have a lengthy exegesis of this rule here, but this rule is just too simple and obvious. Review and reread your spec several times, OK? When you find a sentence that isn't *super* easy to understand, rewrite it.

I've saved so much time by not explaining Rule 4 that I'm going to add another rule.

Rule 5: Templates considered harmful

Avoid the temptation to make a standard template for specs. At first you might just think that it's important that "every spec look the same." Hint: it's not. What difference does it make? Does every book on your bookshelf at home look exactly the same? Would you want them to?

Worse, if you have templates, what tends to happen is that you add a bunch of sections to the template for things that you think are important for every feature. Example: Big Bill decrees that from here on forward, every Microsquish product shall have an Internet component. So the spec template now has a section that says "Internet Component." Whenever somebody writes a spec, no matter how trivial, they have to fill in that section that says "Internet Component",

even if they're just creating the spec for the Microsquish Keyboard. (And you wondered why those useless Internet shopping buttons started cropping up like mushrooms on keyboards).

As these sections accumulate, the template gets pretty large. (Here is an example of a very, very bad template for specifications. Who needs a *bibliography* in a spec, for heaven's sake? Or a glossary?) The trouble with such a large template is that it scares people away from writing specs because it looks like such a daunting task.

A spec is a document that you want people to read. In that way, it is no different than an essay in *The New Yorker* or a college paper. Have you ever heard of a professor passing out templates for students to write their college papers? Have you ever read two good essays that could be fit into a template? Just drop the idea.

Next: Painless Bug Tracking