Joel on Software

Painless Functional Specifications - Part 3: But... How?

by Joel Spolsky

Wednesday, October 04, 2000

Now that you've read all about why you need a spec and what a spec has in it, let's talk about who should write them.

Who writes specs?

Let me give you a little Microsoft history here. When Microsoft started growing seriously in the 1980s, everybody there had read The Mythical Man-Month, one of the classics of software management. (If you haven't read it, I highly recommend it.) The main point of that book was that when you add more programmers to a late project, it gets even later. That's because when you have *n* programmers on a team, the number of communication paths is n(n-1)/2, which grows at $O(n^2)$.

So the programmers at Microsoft were worried about how to write bigger and bigger programs, when the prevailing wisdom of the day was that adding programmers just makes things worse.

Charles Simonyi, Microsoft's long time "chief architect", suggested the concept of *master programmers*. The idea was basically that one master programmer would be responsible for writing all the code, but he or she would rely on a team of junior programmers as "code slaves". Instead of worrying about debugging every function, the master programmer would basically just prototype each function, creating the bare outline, and then throw it to one of the junior programmers to implement. (Of course, Simonyi would be the Master Master Programmer.) The term "Master Programmer" was a bit too medieval, so Microsoft went with "Program Manager."

Theoretically, this was supposed to solve the Mythical Man-Month problem, because nobody has to talk to anyone else -- every junior programmer only talks to the one program manager, and so communication grows at O(n) instead of $O(n^2)$.

Well, Simonyi may know Hungarian Notation, but he doesn't know Peopleware. Nobody wants to be a code slave. The system didn't work at all. Eventually, Microsoft discovered that despite the alleged

Mythical Man Month, you can still add smart people to a team and get increased output, although at decreasing marginal values. The Excel team had 50 programmers when I was there, and it was marginally more productive than a team of 25 would have been -- but not *twice* as productive.

The idea of master/slave programming was discredited, but Microsoft still had these people called program managers bouncing around. A smart man named Jabe Blumenthal basically reinvented the position of program manager. Henceforth, the program manager would own the *design* and the *spec* for products.

Since then, program managers at Microsoft gather requirements, figure out what the code is supposed to do, and **write the specs**. There are usually about 5 programmers for every program manager; these programmers are responsible for implementing in code what the program manager has implemented in the form of a spec. A program manager also needs to coordinate marketing, documentation, testing, localization, and all the other annoying details that programmers shouldn't spend time on. Finally, program managers at Microsoft are supposed to have the "big picture" of the company in mind, while programmers are free to concentrate on getting their bits of code exactly right.

Program managers are invaluable. If you've ever complained about how programmers are more concerned with technical elegance than with marketability, you need a program manager. If you've ever complained about how people who can write good code never do a good job of writing good English, you need a program manager. If you've ever complained about how your product seems to drift without any clear direction, you need a program manager.

How do you hire a program manager?

Most companies don't even have the concept of program manager. I think that's too bad. In my time, the groups at Microsoft with strong program managers had very successful products: Excel, Windows 95, and Access come to mind. But other groups (such as MSN 1.0 and Windows NT 1.0) were run by developers who generally ignored the program managers (who weren't very good anyway, and probably deserved to be ignored), and their products were not as successful.

Here are three things to avoid.

1. Don't promote a coder to be a program manager. The skills for being a good program manager (writing clear English, diplomacy, market awareness, user empathy, and good UI design) are very rarely the skills for being a good coder. Sure, some people can do both, but they are rare. Rewarding good coders by promoting them to a *different position*, one that involves writing English, not C++, is a classic case of the Peter Principle: people tend to be promoted to their level of incompetence.

2. Don't let the marketing people be program managers. No offense, but I think my readers will agree that good marketing people rarely have a good enough grasp of the technology issues to design products.

Basically, program management is a separate career path. All program managers need to be very technical, but they don't have to be good coders. Program managers study UI, meet customers, and *write specs*. They need to get along with a wide variety of people -- from "moron" customers, to irritating hermit programmers who come to work in Star Trek uniforms, to pompous sales guys in $2000 suits. In some ways, program managers are the glue of software teams. Charisma is crucial.

3. Don't have coders *report to* the program manager. This is a

subtle mistake. As a program manager at Microsoft, I designed the Visual Basic (VBA) strategy for Excel and completely speced out, to the smallest detail, how VBA should be implemented in Excel. My spec ran to about 500 pages. At the height of development for Excel 5.0, I estimated that every morning, 250 people came to work and basically worked off of that huge spec I wrote. I had no idea who all these people were, but there were about a dozen people on the Visual Basic team alone just *writing documentation* for this thing (not to mention the team writing documentation from the Excel side, or the full time person who was responsible for hyperlinks in the help file.) The weird thing was that I was at the "bottom" of the reporting tree. That's right. *NOBODY* reported to me. If I wanted people to do something, I had to convince them that it was the right thing to do. When Ben Waldman, the lead developer, didn't want to do something I had speced out, he just didn't do it. When the testers complained that something I had speced was impossible to test completely, I had to simplify it. **If any of these people had reported to me, the product wouldn't have been as good.** Some of them would have thought that it's inappropriate to second-guess a superior. Other times, I would have just put my foot down and *ordered* them to do it my way, out of conceit or nearsightedness. As it was, I had no choice but to build consensus. This form of decision making was the best way to get *the right thing* done.

The final article in my series on specs talks about how to write good specs that people want to read.

Next: Painless Functional Specifications - Part 4: Tips

Want to know more? You're reading Joel on Software, stuffed with years and years of completely raving mad articles about software development, managing software teams, designing user interfaces, running successful software companies, and rubber duckies.

About the author. I'm Joel Spolsky, co-founder of Trello and Fog Creek Software, and CEO of Stack Exchange. More about me.