Joel on Software

The Development Abstraction Layer

by Joel Spolsky

Tuesday, April 11, 2006

A young man comes to town. He is reasonably good looking, has a little money in his pocket. He finds it easy to talk to women.

He doesn't speak much about his past, but it is clear that he spent a lot of time in a soulless big company.

He is naturally friendly and outgoing, and quietly confident without being arrogant. So he finds it easy to pick up small gigs from the job board at the local Programmer's Cafe. But he rapidly loses interest in insurance database projects, vanity web pages for housewives, and financial calculation engines.

After a year, he calculates that he has saved up enough money to pay his modest expenses for a year. So, after consulting with his faithful Alsatian, he sets up a computer in a sunfilled room in his rented apartment above the grocery store and installs a carefully-chosen selection of tools.

One by one, he calls his friends and warns them that if he seems remote over the next months, it is only because he is hard at work.

And he sits down to spin code.

And what code it is. Flawless, artistic, elegant, bug free. The user interface so perfectly mimics a users' thought process that the people he shows it to at the Programmer's Cafe hardly notice that there *is* a user interface. It's a brilliant piece of work.

Encouraged by the feedback of his peers, he sets up in business and prepares to take orders.

His modesty precludes any pretensions, but after a month, the situation in his bank account is not looking encouraging. So far only three orders have been taken: one from his mother, one from an anonymous benefactor at the Programmer's Cafe, and the one he submitted himself to test the commerce system.

In the second month, no more orders come in.

This surprises him and leaves him feeling melancholy. At the big company, new products were created on a regular basis, and even if they were inelegant and homely, they still sold in reasonable quantities. One product he worked on there went on to be a big hit.

After a few more months pass, his financial situation starts to look a little bit precarious. His dog looks at him sadly, not quite certain what is wrong, but aware that his face is looking a little bit gaunter than usual, and he seems to be unable to get up the energy to go out with friends, or go shopping to restock the dangerously low larder, or even to bathe.

One Tuesday morning, the local grocer has refused to extend him any more credit, and his banker has long since refused to return his calls.

The big company is not vindictive. They recognize talent, and are happy to hire him back, at a higher salary. Soon he is looking better, he has some new clothes, and he's got his old confidence back. But something, somewhere, is missing. A spark in his eye. The hope that he might become the master of his own destiny is gone.

Why did he fail? He's pretty sure he knows. "Marketing," he says. Like many young technicians, he is apt to say things like, "Microsoft has worse products but better marketing."

When uttered by a software developer, the term "marketing" simply stands in for all that business stuff: everything they don't actually understand about creating software and selling it.

This, actually, is not really what "marketing" means. Actually Microsoft has pretty terrible marketing. Can you imagine those dinosaur ads actually making someone want to buy

Microsoft Office?

Software is a conversation, between the software developer and the user. But for that conversation to happen requires a lot of work beyond the software development. It takes marketing, yes, but also sales, and public relations, and an office, and a network, and infrastructure, and air conditioning in the office, and customer service, and accounting, and a bunch of other support tasks.

But what do software developers do? They design and write code, they layout screens, they debug, they integrate, and they check things into the source code control repository.

The level a programmer works at (say, Emacs) is too abstract to support a business. Developers working at the developer abstraction layer need an implementation layer -- an organization that takes their code and turns it into products. Dolly Parton, working at the "singing a nice song" layer, needs a huge implementation layer too, to make the records and book the concert halls and take the tickets and set up the audio gear and promote the records and collect the royalties.

Any successful software company is going to consist of a thin layer of developers, creating software, spread across the top of a big abstract administrative organization.

The abstraction exists solely to create the illusion that the daily activities of a programmer (design and writing code, checking in code, debugging, etc.) are all that it takes to create software products and bring them to market. Which gets me to the most important point of this essay:

Your first priority as the manager of a software team is building the development abstraction layer.

Most new software managers miss this point. They keep thinking of the traditional, Command-and-Conquer model of management that they learned from Hollywood movies.

According to Command-and-Conquer, managers-slash-leaders figure out where the business is going to go, and then issue the appropriate orders to their lieutenants to move the business in that direction. Their lieutenants in turn divide up the tasks into smaller chunks and command their reports to implement them. This continues down the org-chart until eventually someone at the bottom actually does some work. In this model, a programmer is a cog in the machine: a typist who carries out one part of management's orders.

Some businesses actually run this way. You can always tell when you are dealing with such a business, because the

person you are talking to is doing something infuriating and senseless, and they know it, and they might even care, but there's nothing they can do about it. It's the airline that loses a million mile customer forever because they refuse to change his non-refundable ticket so he can fly home for a family emergency. It's the ISP whose service is down more often than it's up, and when you cancel your account, they keep billing you, and billing you, and billing you, but when you call to complain, you have to call a toll number and wait on hold for an hour, and then they still refuse to refund you, until you start a blog about how badly they suck. It's the Detroit automaker that long since forgot how to design cars that people might want to buy and instead lurches from marketing strategy to marketing strategy, as if the only reason we don't buy their crappy cars is because the rebate wasn't big enough.

Enough.

Forget it. The command-hierarchy system of management has been tried, and it seemed to work for a while in the 1920s, competing against peddlers pushing carts, but it's not good enough for the 21st century. For software companies, you need to use a different model.

With a software company, the *first priority* of management needs to be creating that abstraction for the programmers.

If a programmer somewhere is worrying about a broken chair, or waiting on hold with Dell to order a new computer, the abstraction has sprung a leak.

Think of your development abstraction layer as a big, beautiful yacht with insanely powerful motors. It's impeccably maintained. Gourmet meals are served like clockwork. The staterooms have twice-daily maid service. The navigation maps are always up to date. The GPS and the radar always work and if they break there's a spare below deck. Standing on the bridge, you have programmers who really only think about speed, direction, and whether to have Tuna or Salmon for lunch. Meanwhile a large team of professionals in starched white uniforms tiptoes around quietly below deck, keeping everything running, filling the gas tanks, scraping off barnacles, ironing the napkins for lunch. The support staff knows what to do but they take their cues from a salty old fart who nods ever so slightly in certain directions to coordinate the whole symphony so that the programmers can abstract away everything about the yacht except speed, direction, and what they want for lunch.

Management, in a software company, is primarily responsible for creating abstractions for programmers. We build the yacht, we service the yacht, we *are* the yacht, but we don't steer the yacht. Everything we do comes down to

providing a non-leaky abstraction for the programmers so that they can create great code and that code can get into the hands of customers who benefit from it.

Programmers need a Subversion repository. Getting a Subversion repository means you need a network, and a server, which has to be bought, installed, backed up, and provisioned with uninterruptible power, and that server generates a lot of heat, which means it need to be in a room with an extra air conditioner, and that air conditioner needs access to the outside of the building, which means installing an 80 pound fan unit on the wall outside the building, which makes the building owners nervous, so they need to bring their engineer around, to negotiate where the air conditioner unit will go (decision: on the outside wall, up here on the 18th floor, at the most inconvenient place possible), and the building gets their lawyers involved, because we're going to have to sign away our firstborn to be allowed to do this, and then the air conditioning installer guys show up with rigging gear that wouldn't be out of place in a Barbie play-set, which makes our construction foreman nervous, and he doesn't allow them to climb out of the 18th floor window in a Mattel harness made out of 1/2" pink plastic, I swear to God it could be Disco Barbie's belt, and somebody has to call the building agent again and see why the hell they suddenly realized, 12 weeks into a construction project, that another contract amendment is going to be needed for this goddamned air conditioner that they knew about *before Christmas* and they only just figured it out, and *if your programmers even spend one minute thinking about this* that's one minute too many.

To the software developers on your team, this all needs to be abstracted away as typing **svn commit** on the command line.

That's why you *have* management.

It's for the kind of stuff that no company can avoid, but if you have your programmers worrying about it, well, management has failed, the same way as a 100 foot yacht has failed if the millionaire owner has to go down into the engine room and, um, build the engine.

You've got your typical company started by ex-software salesmen, where everything is Sales Sales Sales and we all exist to drive more sales. These companies can be identified in the wild because they build version 1.0 of the software (somehow) and then completely lose interest in developing new software. Their development team is starved or nonexistent because it never occurred to anyone to build version 2.0... all that management knows how to do is drive more sales.

On the other extreme you have typical software companies built by ex-programmers. These companies are harder to find because in most circumstances they keep quietly to themselves, polishing code in a garret somewhere, which nobody ever finds, and so they fade quietly into oblivion right after the Great Ruby Rewrite, their earth-changing

refactoring-code code somehow unappreciated by The People.

Both of these companies can easily be wiped out by a company that's *driven* by programmers and *organized* to put programmers in the driver's seat, but which have an excellent *abstraction* that does all the hard work to convert code into products below the decks.

A programmer is most productive with a quiet private office, a great computer, unlimited beverages, an ambient temperature between 68 and 72 degrees (F), no glare on the screen, a chair that's so comfortable you don't feel it, an administrator that brings them their mail and orders manuals and books, a system administrator who makes the Internet as available as oxygen, a tester to find the bugs they just can't see, a graphic designer to make their screens beautiful, a team of marketing people to make the masses want their products, a team of sales people to make sure the masses can get these products, some patient tech support saints who help customers get the product working and help the programmers understand what problems are generating the tech support calls, and about a dozen other support and administrative functions which, in a typical company, add up to about 80% of the payroll. It is not a coincidence that the Roman army had a ratio of four servants for every soldier. This was not decadence. Modern armies probably run 7:1. (Here's something Pradeep Singh taught me today: if only 20% of your staff is programmers, and you can save 50% on salary by outsourcing programmers to India, well, how much of a competitive advantage are you *really* going to get out of that 10% savings?)

Management's primary responsibility to create the illusion that a software company can be run by writing code, because that's what programmers do. And while it would be great to have programmers who are also great at sales, graphic design, system administration, and cooking, it's unrealistic. Like teaching a pig to sing, it wastes your time and it annoys the pig.

Microsoft does such a good job at creating this abstraction that Microsoft alumni have a notoriously hard time starting companies. They simply can't believe how much went on below decks and they have no idea how to reproduce it.

Nobody expects Dolly Parton to know how to plug in a microphone. There's an incredible infrastructure of managers, musicians, recording technicians, record companies, roadies, hairdressers, and publicists behind her who exist to create the abstraction that when she sings, that's all it takes for millions of people to hear her song. All the support staff and management that make Dolly Parton possible can do their jobs best by providing the most perfect abstraction: the most perfect illusion that Dolly sings for us. It is *her* song. When you're listening to her on your iPod, there's a huge infrastructure that makes that possible, but the very best thing that infrastructure can do is *disappear completely*. Provide a *leakproof abstraction* that Dolly Parton is singing, privately, to us.

Next: FogBugz 4½ and Subjective Well-Being

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 Fog Creek Software, a New York company that proves that you can treat programmers well and still be highly profitable. Programmers get private offices, free lunch, and work 40 hours a week. Customers only pay for software if they're delighted. We make Trello, easy web-based collaboration software, FogBugz, an enlightened bug tracking and software development tool, and Kiln, a distributed source control system that will blow your socks off. I'm also the co-founder and CEO of Stack Exchange. More about me.