Column: Editor's Gossip Column

Throwing It Over The Wall

Whil Hentzen

Last month I suggested that maybe eating our children was a viable alternative to the difficulties involved in finding people to staff a custom software development shop. Well, not really, but the thought did spin through my mind briefly.

My favorite book over the past two months (well, besides Teenage Cheerleader Beach Mechanics from Outer Space) has been The One Best Way by Robert Kanigel (Viking, ISBN 0-670-86402-1.) In it, he tells that M.P Higgins told his colleagues that "200 young men suitable for <position> could be placed at once. Nothing is more difficult than to find men capable of filling such positions." Sound familiar? Unfortunately, as I've said before, those who do not study history are doomed to repeat it. (Or was it "Those who do study history will find that we can't learn from it?")

Higgins was addressing American Society of Mechanical Engineers members in an 1899 edition of the Society's journal, and the position to which he was referred was "foundry foreman." Fred Taylor said in the same year, "It is impossible to obtain, for any sum of money, enough qualified foremen and superintendents who are capable of performing all the functions demanded of them."

The solution back then, as I've proposed as well, was to break down the job into smaller, discrete pieces that can be handled by individuals with lesser skill sets. (That's French for "actually available in the real world.")

Having had a month or better to reflect, a couple of things have come up. This month's lecture involves the immediate danger in separating the parts of the job into divisions like "Analyst", "Programmer" and "Technician." This demarcation is fairly commonplace, and while on paper, it sounds good, remember that on paper, Dewey cleaned Truman's clock.

Let me tell you a story.

There's a very large consulting company out there that's being sued for a bazillion dollars. Why? Not because they screwed up a $15 million system for a customer. That happens all the time. (Just check out the $4 BILLION project that the IRS recently pulled the plug on.) The reason they're being sued is because the 28 year old hotshots in $1000 suits used the clients email system to play company politics, some of which consisted of lambasting each other over how incompetent and badly managed the project was. "Since we lost all of our C++ programmers in the last three months, don't you think we should tell the client that the project is going to be seriously late?" and "Don't tell (our manager) anything. He'll only make more stupid decisions." And (to a manager) "You want me to give you an estimate for a module despite the fact that I haven't ever used this software and there's no definition about what the module is to do. I can't estimate something I know nothing about."

You guessed it - the idiots left all their email on the client's system after they were fired. The client has, shall we say, more ammunition than was used in WWII. Naturally, the partner in charge was quoted as saying "This is a unique and isolated incident and we are confident that we will prevail in court." Of course, he *had* to say that, else provide more fodder for the lawsuit. But you and I know it's a complete fabrication. I can cite, myself, a half dozen similar second-hand stories with the same company.

But I bring this story up because I've also first hand knowledge - I've provided technical training at this company, and have witnessed how they manage software projects – precisely the way I've suggested – including the separation of analysts and programmers. But the problem is that the analysts are writing specs that can't be used to program from – and the programmers are being told to complete modules with languages that they've never seen before and given fixed timeframes in which to do it.

Training this group was truly rewarding, feeding good information to these hungry minds. But around noon on the second day, I started hearing the inside scoop. This group of programmers had been told to use FoxPro for the system, natch. But as we looked over the specifications from the analyst, it was clear that the analyst either didn't know what language was to be used, or wasn't aware of what a GUI was. Imagine getting specs for a Windows application from an analyst whom had been working with 80x25 character mode mainframe applications all their life! It was hysterical, or would have been, if it hadn't been true.

I also started getting tired of hearing "Boy, if we only knew this six months ago." It turns out that this training was for a specific project, but their manager didn't want to approve it in a certain budget period

because it would make his numbers look bad. So he stalled for about nine months before approving the funds (We're talking less than $5000 here, folks.) By that time, the project had already been delivered. *I was providing training for a project that had already been finished.* Standard Operating Procedure, according to my students.

So how does the management expect projects to be done? The same way I've suggested, I think. The analyst figures out what needs to be done and how it's to be done, writes it all up, throws the spec over the wall to the programmer (with a bag of Doritos and a sixer of Jolt) and then doesn't answer their phone for the next two weeks.

These aren't evil people, really, just under a lot of pressure to deliver the impossible (kind of like you and me, right?) We've done this type of separation in our shop to a small extent, and, to a small extent, it has worked out. We have detailed specs, use cases put together by our QA folk, write out manufacturing instructions, and turn over pieces to programmers.

But while it's been working out OK, it doesn't feel like it's going to scale – while a 1500 hour project might be handled by this technique, my gut tells me that a 15,000 or 150,000 hour project is too complex. So I suspect that although this idea of separating the functional tasks of software development is, on paper, a good idea, in the real world it probably isn't going to work.

The moral of the story – I suspect that there is some way to improve our software development manufacturing process, along the lines of Mr. Taylor's ideas. That manufacturing instructions and the division of functional tasks to better map to available skill sets are steps in that direction. But I'm still looking for that silver bullet.