

Nickieben Bourbaki Is a Customer on an Agile Project. Boy, does he have problems...

Something about the problem	Something about solutions
<p>Nickieben was originally consumed with fear. The project seemed far too much work to complete in the time allowed, and <i>he</i> would be responsible when it failed.</p>	<p>Time was the solution. As iterations delivered visible business value, he showed some of it to his Lords and Masters. They were pleased with the progress, so he grew calmer. As the business environment shifted, they changed the product direction, and Nickieben and the team showed they could change with it, which further pleased the L&M's.</p> <p>It would have helped Nickieben a lot if he had had a support group of other Customers who could tell him what being a Customer was like, but he didn't.</p>
<p>Planning meetings lasted way too long. Many people were uninvolved for big chunks, and the meetings seemed to drain the energy out of the whole team.</p> <p>And, for all that, the resulting estimates were not very good.</p>	<p>Nick started having "preplanning" meetings the iteration before. In them, he, a tester, and a programmer would discuss a story, write some test sketches, and make an initial estimate. People came to the planning meeting prepared for a short, focused discussion that informed the rest of the team and asked them to look for errors in the estimate.</p> <p>Nickieben's since discovered that other teams also do preplanning. The meetings vary in form, membership, the thoroughness of the discussion, etc. For example, one team had some analysts who spent the iteration ahead of the programmers predigesting the requirements, learning how to explain the domain (which was very complex), and writing tests. Nickieben doesn't think there's one right way to do it, but he does now have a motto: "Meetings must be snappy".</p>
<p>Early on, it seemed that the programmers focused much more on the technical tasks that made up a story than they did on the story itself. It seemed that the tasks were therefore inflated: the programmers did what "a complete implementation of task X" meant, rather than just enough to make the story work.</p>	<p>Nickieben kept harping on the stories as the thing that mattered to him, not tasks. He learned to pare stories down into small "slices" that stretched from the GUI, through the business logic, down to the database. As the programmers got used to making one slice at a time work, they learned that they didn't have to write lots of infrastructure up front.</p>
<p>At first, Nickieben was indecisive about prioritizing stories. He couldn't decide among the different stories that might go into the next iteration.</p>	<p>Short iterations helped after one of the programmers pointed out that any scheduling mistake he made could be corrected in less than two weeks. So the cost of getting something wrong wasn't too big.</p> <p>He forced himself to prioritize by writing down the cash benefit of each feature. Now he didn't have to decide</p>

	<p>which of two features was worth more; instead, he independently decided on worth, then used the cash benefit to pick.</p> <p>He'd started out using a spreadsheet to track the backlog of stories, only writing them on cards when he'd decided on what should go in the iteration. Later, he switched to writing everything on cards. When it came time to thinking about planning, he'd spread the cards out on a table and push them around. Important cards went "up" (farthest from him), and the lesser cards went down. He clumped related cards together, and sometimes a batch of cards made a theme for the iteration. He also found that he could sequence cards so that an iteration's set of stories all supported a particular business process.</p>
<p>Early in the project, Nickieben often found himself frustrated that "finished" stories weren't what he thought he was going to get. It was hard to think of everything he needed to tell the programmers; so much of what he did automatically had to be remembered and put into words. And he'd explain things, and the programmers would think they understood, and <i>he'd</i> think they understood, but it would turn out they hadn't.</p>	<p>He sketched tests up front. Instead of just explaining in words, he found himself writing more and more concrete examples on the whiteboard. Discussing those seemed to prompt him to remember steps or issued he'd otherwise forget.</p> <p>During the iteration, he also spent more time checking in with the programmers, instead of waiting for them to come to him with questions. He especially spent more time with the "GUI guy", talking about what he wanted the GUI to do, and how it did it, and sketching out examples of usage as tests.</p>
<p>As he moved toward more examples, Nickieben started making the examples too complicated. He produced one example that illustrated all the inherent complexities of its story's bit of the business.</p>	<p>He learned to start with the simplest possible example. Then he added one scenario or business rule at a time. In a way, he used the examples to progressively teach the programmers, and they used them to progressively teach the code.</p>
<p>He was sometimes surprised by the technical implications of his ideas. Once, a simple "let's put a Cancel button on the progress bar" led to all sorts of scary talk about transactions and undoing. He was uncomfortable not knowing whether something would be simple or hard.</p>	<p>For a time, he got the help of an analyst who bridged the business and technical worlds. That person helped him understand how big a decision was. But more: her technical knowledge and experience with similar applications allowed her to suggest considerations he would never have thought of.</p> <p>He also enlisted the programmers for lightweight training. He had short conversations about what they had to do to implement a story. (Some of the programmers were much better at this explaining than others.) Over time, those short conversations added up to a decent enough high-level understanding of the system.</p> <p>The programmers also got better at coping with</p>

	<p>change. As they worked more with the system, it got more pliable, so the "internal bigness" of the change more often - but not always! - corresponded to its "external bigness." Programmers also learned more about the business domain, so they could say, "Are you going to need X, Y, or Z? Cause if you do, it would probably be better to schedule those things early."</p> <p>Eventually, the team didn't need the analyst any more. All of them were analysts, a little.</p>
<p>There was a time when Nickieben felt cleanup was taking over control of the project. Parts of the system were old legacy code. When he started giving stories for that, it seemed like every story led to some technical task that was more than an iteration long. Everything seemed to lead to a huge refactoring.</p>	<p>Nickieben learned how to write stories in small slices, about one day's work or so each. And the programmers learned how to do the big refactoring one slice at a time, such that each story led to somewhat better code and enough stories would lead to really good code.</p> <p>They also made information radiators to track "technical debt". Sometimes the programmers couldn't see a way to make an improvement in the time they had - even with their greater experience, it seemed like the refactoring had to be a big chunk. Whenever a programmer left the code worse than she thought she should, she wrote it up on a card and put it on the Refactoring Board. At some point, Nickieben would start getting nervous that the messiness would start slowing the team down, so he would sanction some specific cleanup time. Nevertheless, they tried to tie each refactoring to something useful, like a small feature or a bug fix.</p> <p>The programmers' editor also let them visually track the number of "todo" items they'd left in the code, which was another stimulus to clean up.</p>
<p>In the end, Nickieben's project was a big success. The date did slip a bit, and the Lords and Masters didn't get everything they'd wanted from the release. But they'd changed business direction right in the middle, and the team had coped well and still produced a solid, salable product. Looking back, Nickieben is amazed at the difference between him then and him now. He'd started out floundering, practically on the edge of a nervous breakdown. While he still wouldn't call his job <i>easy</i>, he knows he can do it. The only problem is that he knows there are people just like he was nine months ago. And just like he had no support group, they still don't. So they get to learn it all again, the painful way.</p>	<p>Maybe this page will help.</p>
<p>One last thing: Nickieben has to serve multiple masters: there are different interest groups who care about what the</p>	<p>He isn't really sure what to do about it. He thinks that linking each interest group to a persona (as used in</p>

product does. There are two different classes of users, one very demanding buyer, operations, customer support, and so on. He has a lawyer friend who says it's common knowledge among lawyers that **someone trying to represent multiple interest groups usually gets trapped by one or two** and under-represents the others. Nickieben worries that he's doing that.

some styles of user-centered design) might help. He imagines putting big pictures of the personas up in the bullpen would keep them in his (and everyone's mind).

But he wishes he had better ideas.

Jennitta Andrea, Richard P. Gabriel, Brian Marick, and Geoff Sobering