Article

User Story Estimation Techniques

Posted by Jay Fields on Jun 30, 2008 09:57 AM

Community Agile Topics Customers & Requirements , Agile Techniques Tags Planning , User Stories

One of the great things about working as a consultant is the ability to try out many different ideas and adapting your personal favorite process to include things that work. This article gives the details about user story estimation techniques that I've found effective.

RelatedVendorContent

The Agile Business Analyst: Skills and Techniques needed for Agile

Agile Development: A Manager's Roadmap for Success

Agile Projects: Five Ways to Fail When You Scale

Related Sponsor

Powers of two

Originally I estimated stories as one, two, three, four or as small, medium, large, extra-large. It was always meant to be understood that a medium was twice the size of a small and a large was twice the size of a medium (and so on), but that never seemed to translate well when it came to planning. Then someone recommended to me that I try powers of two. Suddenly we were speaking a language that the business could understand. They knew that an 8 was significantly bigger than a 1. I believe the sizes one, two, four, eight are also much more appropriate. As stories get larger they almost always contain more unknown and risk. A powers of two scale emphasizes the risk associated with large stories.

Use four values

I was once on a project that started with 1, 2, 4, 8 as their estimation values. After the first two estimation sessions less than 5% of the stories were ones and about 30% of the stories were twos. The project manager decided to get rid of the one value because it made his life easier. An interesting thing happened at each subsequent estimation meeting, suddenly only 5% of the stories were twos and many more stories had become fours. I don't think that the developers consciously changed their scale, but developers are conditioned to be skeptical. Few developers are willing to say with certainty that any given story will be as easy as the scale allows. After witnessing this type of behavior on a few different projects, I prefer a minimum of four point values. I also prefer a maximum of 4 point values. After all, it's nothing more than an estimate. If you try to give too much precision to an estimate you'll end up having to account for why you missed the mark. The idea is to get a rough idea, not a rigid plan to live off of.

No averages or numbers not on the scale

Four values allow you to get a rough estimate without spending unnecessary time focusing on precision. Sometimes a story feels larger than a two but smaller than a four. The story should not be estimated as a three. There's really no reason to use a three. The story carries enough risk or unknowns that it is not a two; therefore, it's very likely that it will actually be a four. Using an average or an off scale number can briefly (and unnecessarily) confuse a team member or stakeholder. Also, in the big picture of the project, the occasional uncommon estimate isn't likely to make much of a difference. Keep it simple, stick to the scale.

Vote independently

It's human nature to be influenced by other people. If a technical leader says a story is a two, it's likely that the rest of the team will follow his lead. For this reason I prefer an estimation process that lets each team member vote independently. This can be done by using sheets of paper that no one reveals until everyone is ready. Another option (that I prefer) is to give your estimation rock-paper-scissors (RPS) style. In our estimation meetings we talk about a story until we are ready to estimate then we all "throw" our estimations the same as you would "throw" rock, paper, or scissors. What I mean by "throw our estimation" is that if we think it's a 1 we point 1 finger. Likewise a 2 is two fingers and 4 is four fingers. If you need to throw an 8, you can use both hands.

Take the largest estimate

Even when reminded, developers seem to have a hard time estimating with a team in mind. If a developer thinks they can do the story in 1 day, they throw 1 finger. Unfortunately, that developer may not be available to do the story, and then some other team member is stuck working on a story that they thought was a 2 or even a 4. I prefer to always take the largest estimate thrown by any team member. You may consider this to be sandbagging, but in reality it's likely that each team member has identified different risks and the team member with the largest estimate has probably correctly identified that there is more risk than the other members have thought of.

Taking the largest estimate has additional benefits. If you must agree on a lower estimate then the team member with the larger estimate will need to discuss why they chose a larger value. This discussion can be uncomfortable for developers who are less senior on the team. They may not know how to do something as quickly, based on limited experience with the language or tools. Their concerns are often justified by their skill level, and it would be unfortunate if they felt uncomfortable giving their true estimate because they were afraid to discuss why it was higher.

Any discussion around taking a higher or lower value may lead to the entire team raising their value, or it may lead to that developer uncomfortably lowering their estimate. Either way, you'll need to spend more time talking and you wont have gained--in the end it's consistency that matters. You always know how many stories you expect to get done in an iteration by tracking velocity*. Therefore, even if your estimates are "bloated" so will your velocity be, thus it has no effect on planning.

Finally, taking the largest estimate can help save time in an estimation meeting. If any member of the team believes the story is an 8 he can speak up at any time while discussing the story and announce that he is going to throw an eight. Unless someone else believes that there is a large estimation gap among team members, there's no reason to continue talking about the story since it will ultimately become an eight anyway.

Large estimate gaps

When estimating it's usually the exception that the entire team agrees on the size of a story. Like I previously said, I like to handle the mismatch by always taking the larger estimate. However, sometimes a large gap represents a misunderstanding. For this reason any time there is a two value gap in estimation, additional conversation always occurs (e.g. if a team member throws a 1 and another throws a 4, some clarification needs to occur). Discussing large gaps also ensures that taking the largest estimate has less chance of being abused.

Insufficient information

On occasion a story may need leave the meeting unestimated. It's better to ask for more information than to give an estimate that you are uncomfortable with. An estimate of 8 implies that it's a large story, but you expect it to take twice as long as a 4. Therefore, don't simply estimate ill-defined stories as eights, because you will likely be expected to get it done in the same amount of time as it takes to get two stories estimated as fours completed. The goal of an estimation meeting isn't to estimate all the stories, it's to provide estimates on the stories that provide sufficient information.

Required involvement

No one enjoys estimation meetings (okay, no one I know). In my past projects the fastest reader would read the story aloud, the developers would ask the domain experts questions, and then they would estimate. When the developers weren't asking the domain experts questions, the domain experts usually did other things on their laptops. At first glance I thought this was a good use of their time, but things got missed. Later I joined a project where the manager insisted that we go around the room and make everyone read a story when it was their turn. Suddenly the domain experts were engaged because they were worried about looking silly when it was their turn to read. The meetings became much more valuable due to everyone's involvement.

Pigs and Chickens

In a ham-and-eggs restaurant, the pig is committed but the chicken is simply involved.

I often hear that the business shouldn't influence developer estimates because developers are pigs and the business is full of chickens. I actually think this is a bad analogy. It's more likely that a bad product will get the business team fired than the technology team. I'm sure the business feels just as committed as the developers. However, it is a conflict of interest to let the business interfere with estimates.

It's as simple as this, the business wants to know what functionality they can get in the next iteration. To know what to expect they need estimates. Since the business will not be writing the code, they cannot contribute proper estimates. The more they are involved (in the actual estimation), the less likely it is that they will receive realistic estimates. The best domain experts answer questions in meetings, but never assert in any way the level of effort it will take to complete any given story.

Estimation group size

Teams come in many different sizes. On smaller teams (6 or less) I suggest the entire team attend the estimation session. The many points of view are likely to solidify vision and positively contribute to an estimate. However, I believe there is a point of diminishing returns. Not everyone on a large team needs to be part of estimating each story. Additionally, it's an estimate, 6 people should be just as accurate as 15 people would be. If your team is larger than 6 people I suggest breaking into smaller groups for estimation. In general I like to get at least 3 people to estimate any given story, but no more than 6.

New stories

New stories come in two forms: new feature requests and stories that split. I generally wait to estimate new stories based on their priority. If a story needs to be done in the next iteration, it generally requires an immediate estimate. However, if the new stories aren't going to be played for several iterations it can make sense to hold off until you have enough stories to justify an estimation meeting. I find estimates from estimation meetings to be more reliable, since they come from an environment where everyone is focused solely on estimation. Stories resulting from a split provide an additional complication: they likely already have an estimate. I strongly suggest that the new stories be estimated without taking into consideration the previous estimate. If a story carried enough risk or uncertainty that it required splitting, it's not likely that the estimate is realistic--ignore the original estimate.

No laptops

At least no laptops for developers. Print the story list for everyone, or project the list on the screen, but don't ask the developers to read the story list from their laptops. Laptops almost always find ways to distract developers, thus taking away from the goal of the meeting: Getting valuable estimates.

Required participation

This suggestion is a very important one. In theory, no developer from outside the team should be attending an estimation session. That means that every developer that attends an estimation session will potentially be tasked with working on a story that's being estimated. If a developer is not comfortable estimating a story, then I'm not comfortable with them working on the story. Of course, there are exceptions. I generally give new team members one week to come up to speed before I ask that they participate in an estimation session. But, in general, a developer who refuses to participate in estimation should signal that there's a bigger issue that needs to be resolved.

Stale estimations

Teams change, projects change, and random events occur. Whatever the reason, estimations can get stale. Stale estimations don't help anyone. The development team feels pressure to deliver to stale estimates and the business expects stories to be completed according to projected velocity. It doesn't matter why estimates get stale, what matters is that the estimates are no longer realistic and the plan is no longer reliable. I've never been part of a project where the estimates didn't go stale within 12-24 weeks. It's better to admit that an estimation is stale than it is to plan with inaccurate information. For this reason, I suggest revisiting any estimation that was given more than 12 weeks ago. The estimation will hopefully still be good, but giving the developers an opportunity to speak up given new information is nothing but helpful to the business.

Bribes

This is the easiest suggestion of all: Bring high quality snacks to all estimation meetings. Sugar has been scientifically linked to happiness, and happiness leads to collaboration. It's the simplest and cheapest possible way to make an estimation meeting something to look forward to. Keep in mind though, high quality is the key. If you bring the same snacks that are already sitting in the team room, it's not very exciting. On my last project I went to the