Story Points Revisited

© May 23, 2019 • [Agile-Related, Dark-Agile, Dark-Scrum, SAFe, success]

> *I like to say that I may have invented story points, and if I did, I'm sorry now. Let's explore my current thinking on story points. At least one of us is interested in what I think.*

Stories, of course, are an XP idea, not a Scrum idea. Somehow, Scrum practitioners have adopted the idea. Even though the official Scrum Guide refers to backlog items, having backlog items be User Stories is a common Scrum practice.

At least to the limited extent that they get it right. I've written elsewhere about the general use of stories as she should be done. Here we'll talk about "Story Points".

In XP, stories were originally estimated in time: the time it would take to implement the story. We quickly went to what we called "Ideal Days", which was informally described as how long it would take a pair to do it if the bastards would just leave you alone. We multiplied Ideal Days by a "load factor" to convert to actual implementation time. Load factor tended to be about three: three real days to get an Ideal Day's work done.

We spoke of our estimates in days, usually leaving "ideal" out. The result was that our stakeholders were often confused by how it could keep taking three days to get a day's work done, or, looking at the other side of the coin, why we couldn't do 50 "days" of work in three weeks.

So, as I recall it, we started calling our "ideal days" just "points". So a story would be estimated at three points, which meant it would take about nine days to complete. And we really only used the points to decide how much work to take into an iteration anyway, so if we said it was about 20 points, no one really objected.

I may have made the name-changing suggestion. If I did, I'm sorry now. Here are some of my current thoughts on the topic, from an email to my colleague Simon, who asked:

> *Do you really regret they were invented, or do you simply deplore their misuse when relative sizing is not properly understood?*

I replied that

- I certainly deplore their misuse;
- I think using them to predict "when we'll be done" is at best a weak idea;
- I think tracking how actuals compare with estimates is at best wasteful;
- I think comparing teams on quality of estimates or velocity is harmful.

Let's look a bit more deeply.

Some approaches to "Agile" actually recommend normalizing story points across teams, in the name of easier planning. While this seems sensible enough on the surface, it's too easy to fall into the trap of comparing teams, and, too often, organizations do that.

Comparing

First of all, even if they "look the same", each team has its own skills and works in its own environment. So if they look at two stories that seem the same, and one team says it's a two and the other one says it's a six, that's just not very interesting, and it's certainly not a useful way to compare teams.

Now, you and I, seeing that situation, would approach it with curiosity, first whether the situations were similar enough to compare, and then to explore whether the team with the higher estimate needed some kind of help that we could provide. That would be good. Implicitly or explicitly concluding that the "slower" team was inferior or messing up in some way – that would be very bad, but it is unfortunately common.

I think given two teams producing things, it's an irresistible temptation, for many managers, to compare them. I think it's irresistible enough that I'd drop the notion of story points, and even the notion of estimating stories at all, where possible. We'll come back to the question of how to work with fewer estimates, and there are other articles here addressing that as well.

Tracking

To many managers, the existence of an estimate implies the existence of an "actual", and means that you should compare estimates to actuals, and make sure that estimates and actuals match up. When they don't, that means people should learn to estimate better.

To me, the important thing in Real Agile is to pick the next few things to do, and do them promptly. The key question is to find the most valuable things to do, and to do them quickly. Doing them quickly comes down to doing small slices of high value, and iterating rapidly. Story cost estimation doesn't help much with that, if at all.

So if the existence of an estimate causes management to take their eye off the ball of value and instead focus on improving estimates, it takes attention from the central purpose, which is to deliver real value quickly.

This makes me think that estimation, be it in points or time, is to be avoided.

Pressure

Related to the estimate / actuals concern is the natural pressure of management to want "more". However much the team is getting done, it's not enough. More, more, more.

The best way to deliver value isn't more, more, more, it's to do small valuable things frequently. If instead of estimating stories, we slice them down to "small enough", we can come to a smooth flow of value, delivering all the time.

The focus on more gets in the way of increasing value. Increasing pressure to do more almost inevitably has a bad result: the team tries to go faster, and wind up skimping on code quality and on tests. They soon begin shipping more defects, slowing down because of the increased rework to fix the defects, and slowing down even more because the code quality rapidly declines. Things get worse and worse, pressure increases, and it becomes a race to disaster.

Because estimates are at least implicated in the application of undue pressure, I'd prefer to avoid them.

I'll go further: I'd prefer to avoid iteration or Sprint planning entirely. We wouldn't work to fill up a budget for the next few weeks: we'd work to have a list of the few most important next things to do.

Predicting Done

It is common practice to make a list of essential features, think about them for a while, and then decide that they define the next release of our product. The next question, of course, is "when will all this be done?"

The answer is that no one knows. We could do a lot of work to improve our not knowing, and in some areas and at some times some of that is worth doing, such as when there's a large contract waiting to be bid. But when we're in the business of developing solutions for internal or external customers, we do best to provide small amounts of value frequently, not wait for Big Bang releases that seem often to recede indefinitely into the future.

It's far better to pick a close-in date for the next release to customers, and pick as much good stuff into that release as possible. Estimating, be it in story points or gummi bears or even time, gets in the way of this. Where possible, in my opinion, it's best avoided.

Slicing

So the question comes up, if you don't like story estimation, what do you like? Well, I like story slicing, which is the practice of taking larger stories and slicing them down into smaller ones, each of as high value as possible, but requiring very little time to get done, ideally less than a day, maybe just a couple of hours.

Now, I don't care to quibble with you about whether there must be some kind of estimation going on in slicing. If you or your team estimate in your heads and never tell anyone, then the problems with estimates, whether they are story points or time, aren't likely to arise. And certainly, knowing the difference between "probably small enough" and "probably not small enough" isn't the same as knowing the difference between "three days" and "one day".

Plus, there's a trick. I mentioned it in Getting Small Stories and Slicing, Estimating, Trimming. I learned it from Neil Killick: slice stories down until they just need a single acceptance test. With a little practice that gets things right down to a good size.

And of course there are other articles on the subject of estimation, just click the link at the top of the page for more than you ever wanted to know.

Predicting the future

But … isn't there some legitimate need to know how long a product release will take, and aren't estimates necessary for that? Well, perhaps, but perhaps not story estimates. You probably won't even *have* your requirements down to the story level, and if you do, they are likely too bulky and largely waste.

Of course, if you have to do it, go ahead and do it. Whatever I'd do or my theories about what you should do, are just ideas. In the end you have to do whatever it takes to succeed in your situation. But there is something that I think is better.

First, think about one or a few important capabilities for the next release. Talk about what the problems are that they solve, and what software might help solve them. Talk about the simplest capabilities that might help a bit. We don't have to solve everything: often if we can give a bit of help, that's enough to get things rolling.

Second, think about a close-in deadline such that you feel you could get some good capabilities built by then. Set the deadline and get to work.

Third, slice off thin slices of the important capabilities and do them. You should be able to get them down to a day or less pretty easily. Work only on the most important next bits: don't try to fill out the first capability all the way to the tiniest frill. You're trying to get into a frame of mind where you think "If we just did this one little thing, Customer Jack could actually use this". Then, do that little thing and let Customer Jack try it. We want to move as quickly as we can to continuous delivery of value.

We want to make the value of what we're doing so visible that our Product Owner and other stakeholders can't wait to get it out there. Then … we'll be doing the right thing, with, or without, story estimates.

Summing up

Well, if I did invent story points, I'm probably a little sorry now, but not very sorry. I do think that they are frequently misused and that we can avoid many pitfalls by not using story estimates at all. If they're not providing great value to your team or company, I'd advise dropping them on the grounds that they are waste. If, on the other hand, you just love them, well, carry on!

Categories: • [Agile-Related, Dark-Agile, Dark-Scrum, SAFe, success]

For deeper interaction with me and other people who are interested in these ideas, please join AgileMentoring.com. The fee is small and shows a bit of support for what I do! Thanks!