Should we stop using story points and velocity?

Prasanna Vaste
Thoughtworks India
9 Jan 2014

Abstract

Most of the Agile teams break down their work into a couple of Iterations at a time, and often estimate that work. They usually get done most of what they predict they'll get done in that iteration.
A common scenario runs like this:

- Developers are asked for estimates for upcoming work.
- Team goes into estimation session which runs for almost an hour or two. People are optimists, so these estimates tend to be too low, even without pressure to make them low (and there's usually at least some implicit pressure)
- These tasks and estimates are turned into release plans tracked with burn-up/burn-down charts
- Time and effort spent into monitoring progress against these plans. Everyone is upset when actuals end up being more than estimates. In effort to increase pace with the estimates, developers are told to sacrifice quality, which makes things worse.

In short, Agile teams estimate so that,

- Track velocity
- Decide scope for the Iteration
- Help Prioritize stories
- Help Release planning

I am sure most of the Agile teams went through this exercise during their Agile Journey.
Efforts put into estimation is kind of waste as it is a guess also known as guesstimate. Project Managers/product owners tend to relate these estimates to number of days it will take to complete the story, in some teams estimate is equal to deadline. Most of the teams which use story points to estimate the work face these issues. This results in lack of confidence on development team when stories are taking more time to complete.
In Mike Cohn's book "User Stories Applied" you will find set of claims around story points.

Claim 1: The use of Story points allows us to change our mind whenever we have new information about a story

If a story changes the size slightly there's no impact on the Story Point estimate, but what if the story changes size drastically? Well, at this time you would probably have another estimation session, or you would break down that story into some smaller granularity stories to have a better picture of it's actual size

and impact on the project.
Comment: If we were to use a simple metric like the number of stories completed we would be able to immediately assess the impact of the new items in the progress for the project. Story Points offer no advantage over just simply counting the number of items left to be Done.

Claim 2: The use of Story points works for both epics and smaller stories.

Allowing for large estimates for items in the backlog (say a 100SP) does help to account in some way for the uncertainty that large pieces of work represent.
Comment: But the fact is that we don't really know if an Epic is really equivalent to a similar size aggregate of User Stories (say 100 times 1 Story Point story).
There is no significant added information by classifying a story in a 100 Story Points.

Claim 3: The use of Story points doesn't take a lot of time

Estimating one or two user stories may not take much time. It will be easier to estimate smaller set of stories.
Comment: But the fact is, as anybody that has tried a non-trivial project knows it can take days of work to estimate the initial backlog for a reasonable size project. Most of the agile teams spend hours estimating project backlog. Most of these estimates change over a period of time.

Claim 4: The use of Story points provides useful information about our progress and the work remaining.

This claim holds true **if, and only if** you have estimated all of your stories in the Backlog and go through the same process for each new story added to the Backlog. Even the stories that will only be developed a few months or even a year later must be estimated! This approach is not very efficient.
Comment: Instead progress assessment on the Number of Items completed in each Sprint is faster to calculate (No. of stories in backlog/No. of stories completed in each sprint=No of Sprints left)

Claim 5: The use of Story points is tolerant of imprecision in the estimates.

Even though I agree that this is true but there's no data to justify the belief that Story Points do this better than merely counting the number of Stories Done.
Comment: We can argue that counting the number of stories is even more tolerant of imprecisions.

Claim 6: The use of Story points can be used to plan releases.

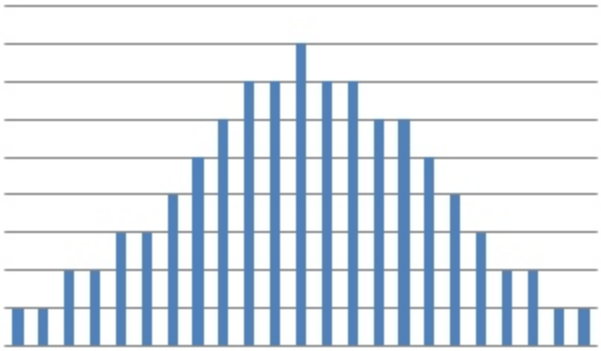
My argument to this will be, we can use any estimation technique to do this, so how would Story Points be better in this particular claim than any other estimation technique?
Comment: We will see how counting the number of Stories Done (and left to be Done) is a very effective way to plan a release.

Real Life Use of Simple Metric for Project Progress Measurement:

I was working for car insurance company to build Mobile web app to allow users to buy car insurance on the go. We did 1 week workshop to identify the project vision, product backlog, MVP Scope. Product

owner was asking for rough estimate of how much time it will take to complete the MVP scope. Instead of estimating each individual stories in product backlog, we asked question to development team can this story be completed in 1 week Iteration by a dev pair? If not then break the story down.

From past experience of working on Agile projects, one can say that by continuously estimating the size of the Stories/Epics you are creating a distribution of the sizes around the median:



Assuming a normal distribution of the size of the stories means that you can assume that for the purposes of looking at the long term estimation/progress of the project, one can assume that all stories are the same size, and can therefore measure progress by measuring the number of items completed per Sprint. The size differences got evened down over the period of time.

So how does this help in release planning? Well, by counting number of story cards we knew that we have to develope around 100 story cards to complete MVP scope. We looked at the features that can be developed in parallel. Team realized that we can only work on 3 parallel streams of work in sprint. So in a way we would need 3 pairs to work in parallel.

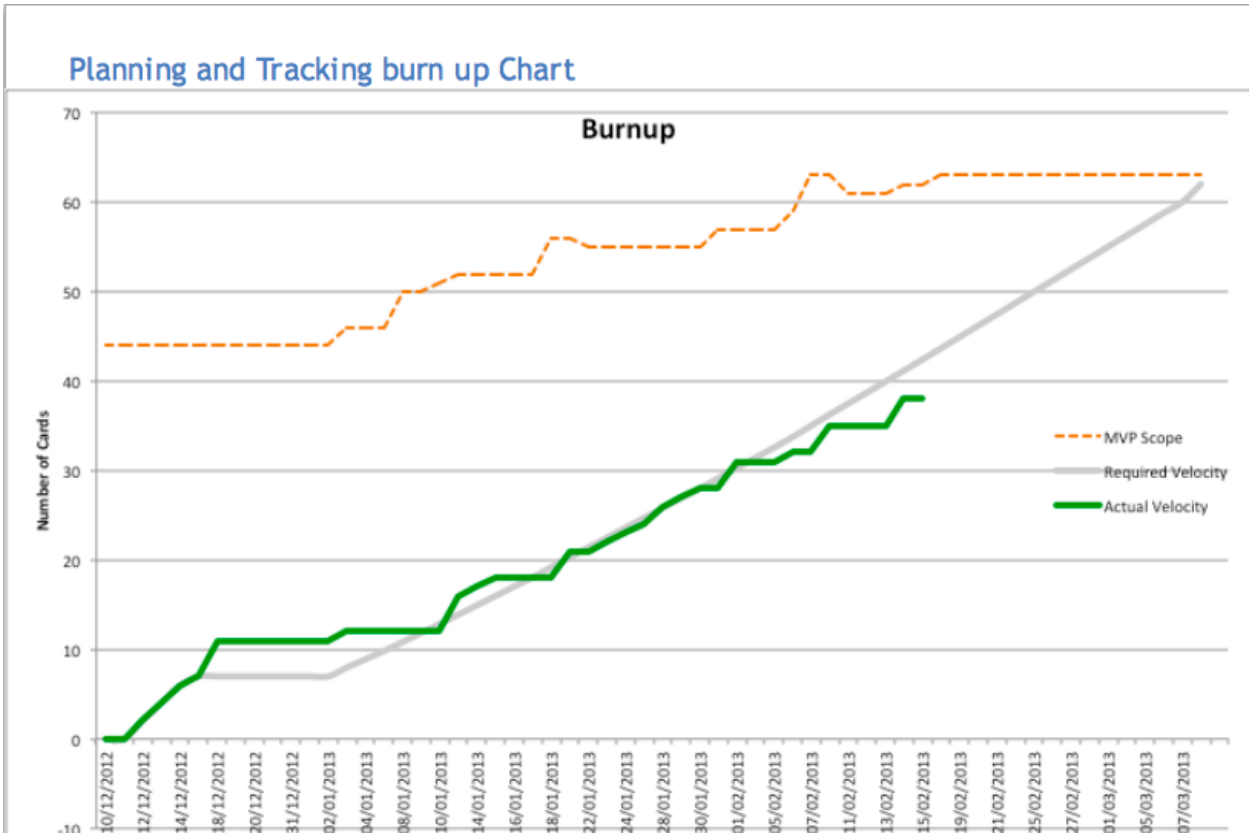
As usual team played velocity game to come with no. of cards a pair of dev can complete in 1 week sprint. So we started with assumption that this team will deliver X stories end of sprint 1. Team then calculated

Number of stories in backlog / Number of stories completed in a sprint = Number of sprints to complete the MVP scope

This information was then shared with Product owner. This helped us in identifying how many approximate sprints we need to complete MVP scope.

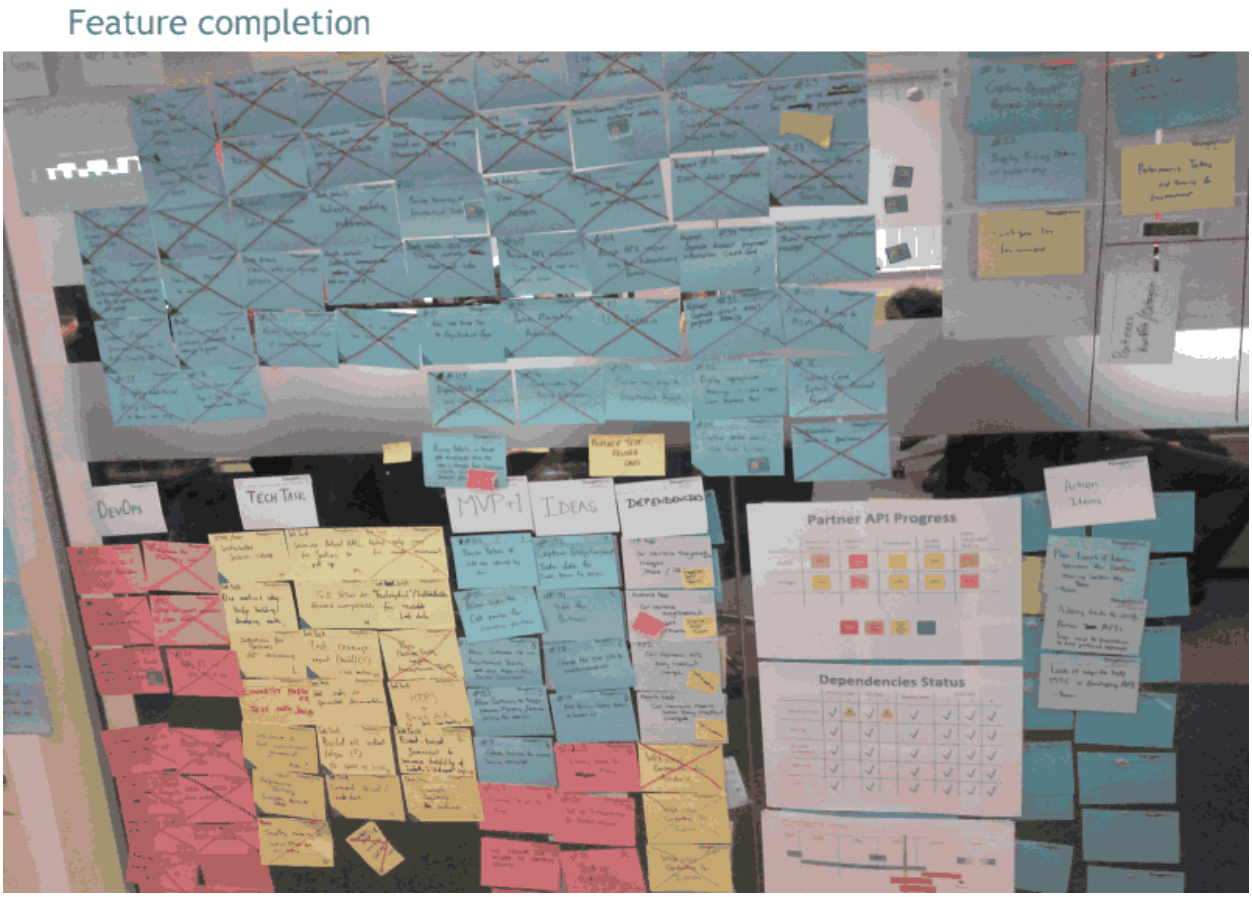
After the first 2 Iterations we relooked at the team velocity and adjusted the planned velocity accordingly. Using no of story cards completed in 1 week scrum helped us to get to the velocity number for the team.

We used burn up chart (based on story count) to track team progress.



As you can see in the below picture, crossed stories show completed scope. It was easier for product owner to view the completed/uncompleted scope at any given time. We also used something called backlog wall to show project progress. Instead of Burn up one can use this to show project progress.

Feature Completion Tracker



At the start of the scrum, during the scrum planning meeting team looked at past velocity (story count) and signed up for more or less same number of stories for upcoming Scrum. More focus was on discussing story value and business priority instead of estimation.

We also used Cycle time to help team to improve and streamline process. Cycle time is a simple but powerful metric. It is the measure of the elapsed time from the moment you start working on an item (story, task, bug, feature, etc.) until it is done. Different teams use different definitions for “start” and “done”. Teams often mark “start” as the time when the team starts working on an item including analysis and “done” when it is signed off by the stakeholder, or pushed to production.

How does cycle time help?

When looked at in aggregate and across time, cycle time reveals how smoothly work is flowing through your development process, helps you spot bottlenecks and see the effects they have on your delivery. It provides the insight you need to make improvements and deliver faster.

In case of story point, there is this natural tendency of changing the story effort (from 1 SP to 2 SP or 2 SP to 4 SP) when story scope changes. In this case, any scope changes or missing requirement resulted in new story and thus helped team to track changes in scope in a easier way. Product owner was

more involved in discussing business value rather than getting into discussion around story point. Continuous prioritization of planned scope helped in working on most important or valuable stories first rather than working on stories which give quick gain in story points. I have seen cases where story point is used to track team progress there is this tendency to add up story points to show better velocity. Using story count definitely helps in avoiding such situations.

Product owner and Team were happy with this change as

There are significant benefits we get from this move:

Fewer metrics, more conversations:

In Scrum planning meetings, we shifted focus from the number to a collaborative conversation. This provided a better platform for our team to discuss and eventually establish a shared understanding about what to build and how. We noticed that subsequent development work became much smoother after these conversations.

Less math, more effective planning:

In planning meetings: Earlier we used to translate our scope to points, scratched our heads to figure out the exact number of points to put in or take out. Freed up from these calculations, we focused more on business value and being more responsive to ad-hoc requirements.

Better way to track change in scope:

Any new requirement/scope resulted in new story instead of adding up story points on existing story. Helped team to track scope changes in effective manner.

As Martin Fowler said: *"So whenever you're thinking of asking for an estimate, you should always clarify what decision that estimate is informing. If you can't find one, or the decision isn't very significant, then that's a signal that an estimate is wasteful."* -- PurposeOfEstimation

This is an experiment we just started, so would like to use this on other Agile projects as well. This works best for projects with duration more than 3 months and where there is lot of trust between Product owner and developement team. Maybe you need to estimate your first release to build trust. You can then evolve your estimation process from "Ideal Days => Relative sizing => Counting cards => Cycle time"

In the end I will say stop wasting time trying to estimate a never ending Backlog. There's no proof that that will help you predict the future any better than just counting the number of stories "Done".