Agile Planning and Estimation

Guidelines for Agile Estimating and Planning

- Story Points

Whole Team Works together

Task Boards

Burn down charts

Velocity

Story Maps Helps to prioritize the Backlog

Retrospective to improve team work

Techniques for Estimation: Estimates Shared, Estimation Scale, Derive Estimation, Planning Poker

Guidelines for Agile Estimating and Planning

With all of these reasons in mind, the following is a list of a dozen guidelines for successful agile estimating and planning:

1. Involve the whole team. Primary responsibility for certain activities may fall to one person or group, as prioritizing requirements is primarily the responsibility of the product owner. However, the whole team needs to be involved and committed to the pursuit of the highest-value project possible. We see this, for example, in the advice that estimating is best done by the whole team even though it may be apparent that only one or two specific team members will work on the story or task being estimated. The more responsibilities are shared by the team, the more success the team will have to share.

2. Plan at different levels. Do not make the mistake of thinking that a release plan makes an iteration plan unnecessary, or the other way around. The release, iteration, and daily plans each cover a different time horizon with a different level of precision and each serves a unique purpose.

Guidelines for Agile Estimating and Planning

With all of these reasons in mind, the following is a list of a dozen guidelines for successful agile estimating and planning:

3. Keep estimates of size and duration separate by using different units. The best way to maintain a clear distinction between an estimate of size and one of duration is to use separate units that cannot be confused.

Estimating size in story points and translating size into duration using velocity is an excellent way of doing this.

4. Express uncertainty in either the functionality or the date. No plan is certain. Be sure to include an expression of uncertainty in any release plan you produce. If the amount of new functionality is fixed, state your uncertainty as a date range ("We'll finish in the third quarter" or "we'll finish in between 7 and 10 iterations").

5. Re-plan often. Take advantage of the start of each new iteration to assess the relevancy of the current release plan. If the release plan is based on outdated information or on assumptions that are now false, update it.

Use re-planning opportunities to ensure that the project is always targeted at delivering the greatest value to the organization.

6. Track and communicate progress. Many of a project's stakeholders will have a very strong interest in the progress of the project.

Keep them informed by regularly publishing simple, very understandable indicators of the team's progress. Burn down charts and other at-a-glance indicators of project progress are best.

Guidelines for Agile Estimation and Planning

7. Acknowledge the importance of learning. Because a project is as much about generating new knowledge as it is about adding new capabilities to a product, plans must be updated to include this new knowledge. As we learn more about our customers' needs, new features are added to the project.

As we learn more about the technologies we are using or about how well we are working as a team, we adjust expectations about our rate of progress and our desired approach.

8. Plan features of the right size. Functionality that will be added in the near future (within the next few iterations) should be decomposed into relatively small user stories, typically items that will take from one or two days up to no more than 10 days. We are best at estimating work that is all within one order of magnitude in size.

Working with user stories that fall within these ranges will provide the best combination of effort and accuracy. It will also provide stories that are small enough to be completed during one iteration for most teams.

Of course, working with small user stories can become quite an effort on longer projects. To balance this, if you are creating a release plan that will look more than two or three months into the future, either write some larger stories (called epics) or estimate the more distant work at the theme level to avoid decomposing large stories into small ones too far in advance.

9. Prioritize features. Work on features in the order that optimizes the total value of the project.

In addition to the value and cost of features when prioritizing consider also the learning that will occur and the risk that will be reduced by developing the feature.

Early elimination of a significant risk can often justify developing a feature early.

Similarly, if developing a particular feature early will allow the team to gain significant knowledge about the product or their effort to develop it, they should consider developing that feature early.

10. Base estimates and plans on facts.

Whenever possible, ground your estimates and plans in reality. However, whenever possible estimates and plans should be based on real, observed values.

This goes, too, for an estimate of how much of a feature is complete. It's easy to tell when a feature is 0% done (we haven't started it) and it's relatively easy to tell when we're 100% done (all tests pass for all of the product owner's conditions of satisfaction). It's hard to measure anywhere in between—is this task 50% done or 60% done? Because that question is so hard, stick with what you can know, 0% and 100%.

11. Leave some slack. Especially when planning an iteration, do not plan on using100% of every team member's time. Just as a highway experiences gridlock when filled to 100% capacity, so will a development team slow down when every person's time is planned to be used.

12. Coordinate teams through look ahead planning. On a project involving multiple teams, coordinate their work through rolling look ahead planning. By looking ahead and allocating specific features to specific upcoming iterations, inter-team dependencies can be planned and accommodated.

Portfolio Backlog Item Estimates

Although the portfolio backlog is not formally a part of Scrum, many organizations maintain one that contains a prioritized list of all of the products (or projects) that need to be built.

To properly prioritize a portfolio backlog item we need to know the approximate cost of each item.

We typically won't have a complete, detailed set of requirements at the time when this cost number is initially requested, so we can't use the standard technique of estimating each individual, detailed requirement and then summing those estimates to get an aggregate estimate of the total cost.

Instead, to estimate portfolio backlog items, many organizations choose to use rough, relative size estimates like T-shirt sizes (such as small, medium, large, extralarge, and so on).

Product Backlog Estimates

Once a product or project is approved and we start adding more detail to its product backlog items, however, we need to estimate differently.

When PBIs have risen in priority and been groomed to include more detail, most teams prefer to put numeric size estimates on them, using either story points or ideal days.

Typically, PBI estimation occurs in “estimation meetings,” the first of which likely coincides with initial release planning. The product owner might also call additional estimation meetings during a sprint if any new PBIs need to be estimated.

Not all Scrum practitioners believe that PBI size estimation is a necessary activity.

Their experience has shown that when Scrum teams become good enough, they are able to create PBIs that are small and of roughly the same size.

Such practitioners have determined that it is wasteful to estimate small, similarly sized items. Instead, they just count the number of PBIs.

They still use the concept of velocity, but it is measured as the number of PBIs that are completed in a sprint, instead of the sum of the sizes of the PBIs that are completed in a sprint.

I understand the “no-estimates-required” argument, but I still prefer to estimate PBIs for a few reasons:

not all PBIs will be at the same size at the same time, so there will be some larger PBIs in the backlog even if we do have a collection of smaller, similarly sized items toward the top.

It can take some time for teams to acquire the skills to break down PBIs to be roughly the same size.

Teams might have to split stories at unnatural points to achieve the same-size goal.

Finally, and most importantly, one of the primary values of estimation is the learning that happens during the estimation conversations.

Nothing promotes a healthy debate like asking people to put a number on something, which will immediately surface any disagreements and force assumptions to be exposed. If we were to do away with estimation, we would need to substitute an equally effective way of promoting these healthy discussions.

Task Estimates

At the most detailed level we have the tasks that reside in the sprint backlog. Most teams choose to size their tasks during sprint planning so that they can acquire confidence that the commitments they are considering are reasonable.

Tasks are sized in ideal hours (also referred to as effort-hours, man-hours, or person-hours).

The team estimates that the UI task will take five effort-hours to complete. That doesn't mean it will take five elapsed hours. It might take one person a couple of days to code the UI, or it could take a couple of people working together less than a day.

The estimate simply states how much of the team's effort is expected to complete the task. I will describe the use of task estimates in more detail when I describe the details of sprint planning.

ESTIMATE AS A TEAM

In many traditional organizations the project manager, product manager, architect, or lead developer might do the initial size estimation.

Other team members might get a chance to review and comment on those estimates at a later time. In Scrum, we follow a simple rule: The people who will do the work collectively provide the estimates.

To be clear, when I say people who will do the work, I mean the development team that will do the hands-on work to design, build, and test the PBIs.

The product owner and Scrum Master don't provide estimates. Both of these roles are present when the PBIs are being estimated, but they don't do any hands-on estimation.

The product owner's role is to describe the PBIs and to answer clarifying questions that the team might ask. The product owner should not guide or "anchor" the team toward a desired estimate.

The Scrum Master's role is to help coach and facilitate the estimation activity.

The goal is for the development team to determine the size of each PBI from its collective perspective. Because everyone sees a story from a different point of view, depending on his area of expertise, it is important that all members of the development team participate during estimation.

Estimates Are Not Commitments

Estimates are not commitments, and it is important that we not treat them as such. That statement typically concerns managers.

“What do you mean we’re not asking the team to commit to its estimates? How are we going to get precise estimates unless they do?”

When this topic comes up in my classes, I do a simple visual demonstration to make the point.

I hold up a sticky note and say, “Imagine that I ask you to size this story and you tell me it’s this big.

Accuracy versus Precision

Our estimates should be accurate without being overly precise. We have all been involved with products where the estimates were at a ridiculous level of precision.

You know, the one where the estimate was 10,275 man-hours or the other one where the projected cost was $132,865.87.

Generating these wrong, overly precise estimates is wasteful. First, there is the wasted effort of coming up with the estimate, which can be considerable. Second, there is the waste that occurs when we deceive ourselves by thinking we understand something that we don't, and then make important, wrong, and costly business decisions based on this deception.

We should invest enough effort to get a good-enough, roughly right estimate. When estimating, there will always be a point of diminishing returns, beyond which for every additional unit of effort we invest we don't get a corresponding increase in the accuracy of the estimate.

Beyond that point we are just wasting our time and probably starting to negatively affect the estimate's accuracy by considering an increasing amount of lower-value data.

Relative Size Estimation

We should estimate PBIs using relative sizes, not absolute sizes.

We compare items to determine how large an item is relative to the others.

My personal observations have convinced me that people are much better at relative size estimation than absolute size estimation.

PBI Estimation Units

Although there is no standard unit for PBI size estimates, by far the two most common units are story points and ideal days. There isn't a right or wrong choice when deciding between these two. I'd say 70% of the organizations I work with use story points and the other 30% use ideal days.

Story Points

Story points measure the bigness or magnitude of a PBI. We expect story points to be influenced by several factors, such as complexity and physical size.

Something doesn't have to be physically large to be big. The story might represent the development of a complex business algorithm.

The end result won't be very large, but the effort required to develop it might be.

On the other hand, a story might be physically quite big but not complex.

Let’s say we have to update every cell in a 60,000-cell spreadsheet. None of the individual updates is difficult, but the updates can’t be automated. How much of this work can we get done in a sprint?

Though not complex, this would be a large story.

Story points combine factors like complexity and physical size into one relative size measure.

The goal is to be able to compare stories and say things like “Well, if the create-a-ticket story is a 2, then the search-for-a-ticket story is an 8,” implying that the searching story is roughly four times the size of the creation story.

In the example at the beginning of this chapter, the approach was to estimate the PBI sizes and then derive the duration by dividing the sum of the sizes by the average velocity.

Because size measures like story points are ultimately used to calculate time (duration), story points must reflect the effort associated with the story from the development team’s perspective.

Ideal Days

An alternative approach for estimating PBIs is to use ideal days.

Ideal days are a familiar unit—they represent the number of effort-days or person-days needed to complete a story. Ideal time is not the same thing as elapsed time.

Ideally the American football game has four quarters that are each 15 minutes long (so the game is played in one ideal hour). However, it takes more like three to three and a half hours to actually play the game.

I stated earlier that there isn't a right or wrong answer when choosing between story points and ideal days. However, an important factor against ideal time is the risk of misinterpretation.

There are other differences between story points and ideal time, but misinterpretation is one of the bigger issues. A student in one of my classes summed up her preference between the two when she said to her colleagues, "Look, we've been using ideal time for the past 15 years that I've been here and it has never worked. Honestly, I'd just like to try something different."

Planning Poker

Planning Poker is a technique for sizing PBIs that was first described by James Grenning (Grenning 2002) and then popularized by Mike Cohn (Cohn 2006).

Planning Poker is based on a few important concepts

Planning Poker Concepts:

- Consensus based
- Expert opinion
- Intense discussion
- Relative sizing
- Accurate grouping/binning
- Leverage estimating history
- Planning Poker is a consensus-based technique for estimating effort. Knowledgeable people (the experts) slated to work on a PBI engage in an intense discussion to expose assumptions, acquire a shared understanding, and size the PBI.
- Planning Poker yields relative size estimates by accurately grouping or binning together items of similar size. The team leverages its established PBI estimation history to more easily estimate the next set of PBIs.

Estimation Scale

To perform Planning Poker, the team must decide which scale or sequence of numbers it will use for assigning estimates. Because our goal is to be accurate and not overly precise, we prefer to not use all of the numbers.

Instead, we favor a scale of sizes with more numbers at the small end of the range and fewer, more widely spaced numbers at the large end of the range.

The most frequently used scale is the one proposed by Mike Cohn, based in part on a modified Fibonacci sequence: 1, 2, 3, 5, 8, 13, 20, 40, and 100.

An alternative scale that some teams use is based on powers of 2: 1, 2, 4, 8, 16, 32, ...

When using this type of scale, we group or bin together like-size PBIs and assign them the same number on the scale. To illustrate this concept, let's say we work at the post office and we need to group packages of similar size together in the same bin.

When we receive a package, we need to decide which bin to place the package in. Now, not all packages in the same bin are or will be identically the same physical shape, size, or weight, so we need to examine the packages that are currently in the bins so that we can find the best-fit bin for the package we are estimating.

Once we find the closest matching bin, we put the package in the bin and move on to the next package.

Obviously, the more packages we put into the bins, the easier it should be to size and bin future packages because we'll have more points of comparison.

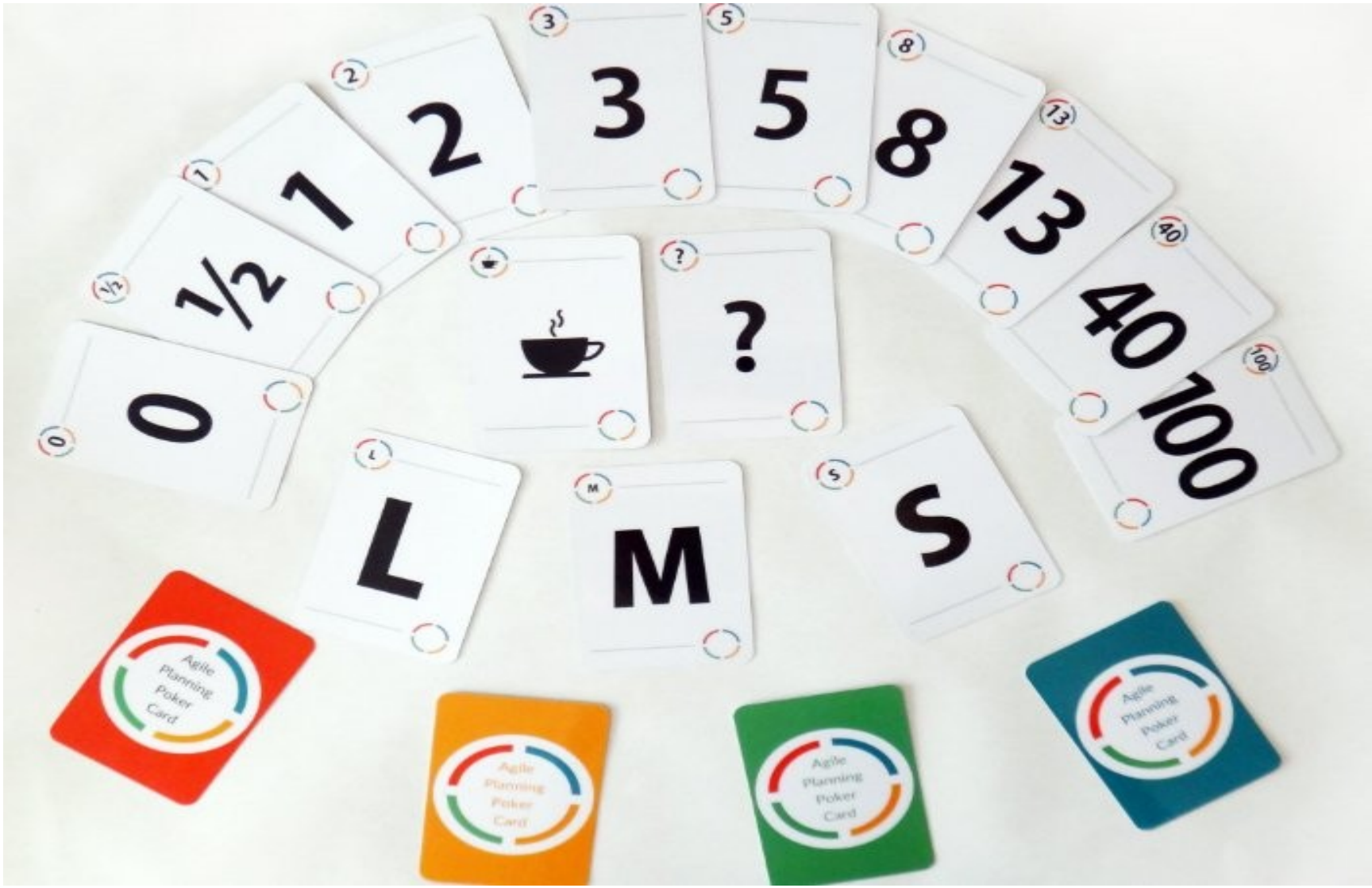
To avoid being overly precise, we don't have a "4 bin" (if we're using a scale based on the Fibonacci sequence). So, when we get a package that we feel is larger than a 2 but smaller than an 8, we need to put it in either the "3 bin" or the "5 bin."

How to Play

The full Scrum team participates when performing Planning Poker. During the session, the product owner presents, describes, and clarifies PBIs.

The Scrum Master coaches the team to help it better apply planning poker. The Scrum Master is also looking for people who , by their body language or by their silence , seem to disagree and helping them engage. And the development team is collaboratively generating the estimates.

Each development team member is provided with a set of Planning Poker cards.


0
½
1
2
3
5
8
13
40
100
?
L
M
S
Agile Planning Poker Card

Card	Interpretation
0	the item is already completed or it is so small that it doesn't make sense to even give it a size number.
1/2	Used to size tiny items.
1, 2, 3	Used to size small items.
5, 8, 13	Used to size medium items. For many teams, an item of size 13 would be the largest they would schedule into a sprint. They would break any item larger than 13 into a set of smaller items.
20, 40	Used to size large items (for example, feature- or theme-level stories).
100	Either a very large feature or an epic.
∞ (infinity)	Used to indicate that the item is so large it doesn't even make sense to put a number on it.

Card	Interpretation
?(Question Mark)	Indicates that a team member doesn't understand the item and is asking the product owner to provide additional clarification. Some team members also use the question mark as a way of recusing themselves from the estimation of the current item—typically because the person is so far removed from the item he has no idea how to estimate it. Although it is acceptable not to estimate, it is unacceptable not to participate! So, just because someone doesn't feel comfortable offering up an estimate, that doesn't allow him to disengage from the conversation or responsibility of helping the team find a consensus estimate.
π (pi)	π (pi) In this context, π doesn't mean 3.1415926! Instead, the pi card is used when a team member wants to say, "I'm tired and hungry and I want to get some pie!" Some Planning Poker decks use a coffee cup image instead of pi. In either case, this card emphasizes an important point. The team members can engage in an intense estimation discussion for only a limited period of time (perhaps an hour or two). At that point, they really do need a break or the enthusiasm for the discussion will turn into an effort to figure out how to quickly get the estimates done, regardless of their accuracy or the learning that takes place. If people are playing the pi card, the team needs to take a break.

The rules of Planning Poker are as follows:

1. The product owner selects a PBI to be estimated and reads the item to the team.
2. Development team members discuss the item and ask clarifying questions to the product owner, who answers the questions.
3. Each estimator privately selects a card representing his estimate.
4. Once each estimator has made a private selection, all private estimates are simultaneously exposed to all estimators.
5. If everyone selects the same card, we have consensus, and that consensus number becomes the PBI estimate.
6. If the estimates are not the same, the team members engage in a focused discussion to expose assumptions and misunderstandings. Typically we start by asking the high and low estimators to explain or justify their estimates.
7. After the discussion, we return to step 3 and repeat until consensus is reached.

In Planning Poker we don’t take averages or use any number not on the scale/cards.

The goal is not to compromise, but instead for the development team to reach a consensus about the estimate of the story’s overall size (effort) from the team perspective. Usually this consensus can be achieved within two or three rounds of voting, during which the team members’ focused discussion helps obtain a shared understanding of the story.

Benefits

Planning Poker brings together the diverse team of people who will do the work and allows them to reach consensus on an accurate estimate that is frequently much better than any one individual could produce.

As I mentioned earlier, there are some in the agile community who believe that estimating PBIs is not worthwhile. The intense discussion of the PBIs fostered by Planning Poker, however, is incredibly valuable.

In my experience, you really motivate people to think about the details of the PBIs and expose any assumptions when you ask them to put a size number on them.

The majority of the value associated with Planning Poker is the discussion and better understanding that team members will share about the PBIs.

I hope they also get size estimates on the PBIs; however, I am more concerned that they learn about the PBIs. If they do, they have gotten a good return on the team’s investment.

Velocity is the amount of work completed each sprint. It is measured by adding the sizes of the PBIs that are completed by the end of the sprint. A PBI is either done or it's not done.

The product owner doesn't get any value from undone items, so velocity does not include the size numbers of partially completed PBIs.

Velocity measures output (the size of what was delivered), not outcome (the value of what was delivered). We assume that if the product owner has agreed that the team should work on a PBI, it must have some value to him. However, completing a PBI of size 8 doesn't necessarily deliver more business value than completing a PBI of size 3.

Perhaps the PBI of size 3 is high value and therefore we work on it early (because it is high value and low cost), and we work on the PBI of size 8 later (because it is lower value and higher cost).

Velocity is used for two important purposes. First, it is an essential concept for Scrum planning.

For release-level planning, we divide the size of the release by the team's average velocity to calculate the number of sprints necessary to complete the release.

Additionally, at sprint planning, a team's velocity is used as one input to help determine its capacity to commit to work during the upcoming sprint of the release by the team's average velocity to calculate the number of sprints necessary to complete the release.

Velocity is also a diagnostic metric that the team can use to evaluate and improve its use of Scrum to deliver customer value.

By observing its own velocity overtime, the team can gain insight into how specific process changes affect the delivery of measurable customer value.

Calculate a Velocity Range

For planning purposes, velocity is most useful when expressed as a range, such as “The team is typically able to complete between 25 and 30 points each sprint.” Using a range allows us to be accurate without being overly precise.

With a velocity range we can more accurately provide answers to questions like “When will we be done?” “How many items can we complete?” or “How much will all this cost?” Because most of these questions get asked early on in a product development effort, when we have the least information about the product, it’s impossible to give a very precise answer. By using a range, we can communicate our uncertainty.

One common way to forecast a team's velocity is to have the team perform sprint planning to determine what PBIs it could commit to delivering during a single sprint.
If the commitment seems reasonable, we would simply add the sizes of the committed PBIs and use that as the team's forecasted velocity.
Because what we really want is a velocity *range, we could have the team plan two* sprints and use one estimated velocity number as the high and the other as the low(the two estimates would likely be different).

Alternatively, we could make some intuitive adjustments to one estimated velocity based on historical data for other teams, thereby converting the one estimate into a two-estimate range.

As soon as the team has performed a sprint and we have an actual velocity measurement, we should discard the forecast and use the actual. And as the team builds up a history of actual velocities, we should compute averages or apply other statistics to the data to extract a velocity range.

His reasoning was that if the team is constantly inspecting and adapting (continuously improving), its velocity should keep getting better and better.

I would expect a team that is aggressively trying to improve itself and is focused on delivering features in accordance with a robust definition of done and low technical debt to see an increase in velocity.

Well, at least an increase up to a certain point, at which time its velocity will likely plateau .

Just because a team’s velocity has leveled out doesn’t mean there is no more upward potential. There are a number of ways that the Scrum team and managers can help get velocity to the next plateau.

For example, introducing new tools or increasing training can have a positive effect on velocity.

Or managers can strategically change team composition with the hope that the change will eventually lead to a greater overall velocity. Of course, managers should be careful because haphazardly moving people on and off teams can and probably will cause velocity to decline.

Forecasting Velocity

In the previous examples I assumed that the team had historical velocity data that we could use to predict future velocity. Certainly one of the benefits of having long-lived teams is that they will acquire such useful historical data more detailed discussion of the benefits of long-lived teams). But how do we handle the situation where we have a new team whose members haven't worked together and therefore have no historical data? We'll have to forecast it.

One common way to forecast a team's velocity is to have the team perform sprint planning to determine what PBIs it could commit to delivering during a single sprint.

If the commitment seems reasonable, we would simply add the sizes of the committed PBIs and use that as the team's forecasted velocity.

Because what we really want is a velocity range, we could have the team plan two sprints and use one estimated velocity number as the high and the other as the low (the two estimates would likely be different). Alternatively, we could make some intuitive adjustments to one estimated velocity based on historical data for other teams, thereby converting the one estimate into a two-estimate range.

As soon as the team has performed a sprint and we have an actual velocity measurement, we should discard the forecast and use the actual.

And as the team builds up a history of actual velocities, we should compute averages or apply other statistics to the data to extract a velocity range.

Affecting Velocity

Do you believe that a team's velocity should constantly increase over time? An executive once said to me, "Last year my team's velocity averaged 30 points per sprint. This year I'm expecting the team to achieve 35 points per sprint."

His reasoning was that if the team is constantly inspecting and adapting (continuously improving), its velocity should keep getting better and better.

I would expect a team that is aggressively trying to improve itself and is focused on delivering features in accordance with a robust definition of done and low technical debt to see an increase in velocity. Well, at least an increase up to a certain point, at which time its velocity will likely plateau.

Just because a team's velocity has leveled out doesn't mean there is no more upward potential.

There are a number of ways that the Scrum team and managers can help get velocity to the next plateau. For example, introducing new tools or increasing training can have a positive effect on velocity.

Or managers can strategically change team composition with the hope that the change will eventually lead to a greater overall velocity. Of course, managers should be careful because haphazardly moving people on and off teams can and probably will cause velocity to decline.

Although introducing new tools, getting training, or changing team composition can have a positive effect on velocity, these actions usually cause a dip in velocity while the team absorbs and processes the change.

After this decline, there will probably be an increase to the point where the team establishes a new plateau until some other change causes yet another plateau to be achievable.

Of course, there is one obvious thing we could do to try to improve velocity: work longer hours. Working a lot of consecutive overtime might initially cause velocity to increase.

Access control issues, such as who is allowed to access particular information, is clearly a behavioral requirement although they are generally considered to be a security issue which falls into the non-behavioral category.

Loosen up a bit and don't allow yourself to get hung up on issues such as this. The critical thing is to identify and understand a given requirement, if you miscategorize the requirement who really cares?

That increase will almost certainly be followed by an aggressive decline in velocity along with a simultaneous decline in quality. Even after the overtime period ends, the team will need some amount of time to recover before returning to its reasonable baseline velocity. I have seen examples of where the trough (decreased velocity area) during the recovery period is larger than the crest (increased velocity area) during the overtime period.

The end result is that lots of overtime may provide some short-term benefits, but these are frequently far outweighed by the long-term consequences.

Misusing Velocity

Velocity is used as a planning tool and as a team diagnostic metric. It should not be used as a performance metric in an attempt to judge team productivity.

When misused in this way, velocity can motivate wasteful and dangerous behavior.

For example, say I have decided to give the largest bonus to the team that has the highest velocity. Superficially this idea might seem sensible; the team with the highest velocity must be getting the most work done each sprint, right? So, why not reward that behavior?

Well, if I’m comparing teams that aren’t sizing their PBIs using a common baseline (which is very likely true), comparing the numbers would make no sense.

Let’s say that team A assigns a value of 5 to a PBI, whereas team B assigns a value of 50 to the same PBI. Team A doesn’t really want me to compare its velocity against team B’s velocity. Team B’s velocity will be ten times that of team A, even if both teams actually get about the same quantity of work completed each sprint.

Once team A sees the problem, its members will start to game the system to ensure that their velocity numbers are higher. The easy way to do this is to just change the scale the team uses to estimate PBIs. So, team A now sizes the same item (the one it originally sized a 5) to be a 500.

I call this behavior point inflation, and it serves no purpose other than to align a team's behavior with a misguided measurement system.

Don't do this. Even if teams are using the same units to consistently size PBIs, if I set up the reward system to favor bigger numbers, that's exactly what I'll get—bigger numbers (point inflation).

Even worse than point inflation is when teams cut corners to get more "done" in an effort to achieve higher, more desirable velocities. Doing so leads to increasingly greater levels of technical debt.

At the end of the day, we should judge velocity on how well it assists us with performing accurate planning and how well it helps a team to internally improve itself.
Any other uses will likely promote the wrong behavior.