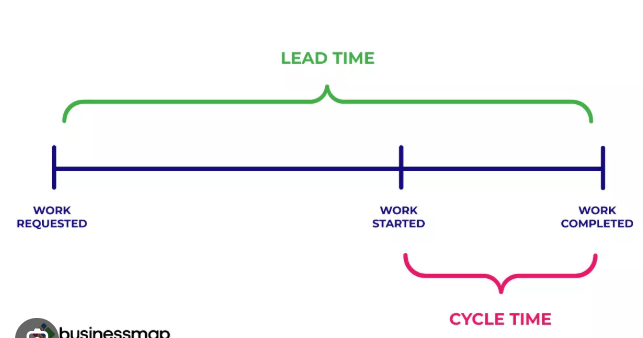
<https://www.vibhorchandel.com/p/13723218_metrics>

The term **"metrics"** refers to **measurable values** that help you track and assess the performance, progress, or quality of something.

Scrum metrics are specific data points that scrum teams track and use to improve efficiency and effectiveness. Scrum teams use metrics to inform decision-making and become more efficient in planning and execution, as well as set target goals and improvement plans.

Cycle Time Vs Lead Time

In a nutshell, cycle time measures the time it takes for a team to make a product, while lead time measures the time between the customer order and order fulfillment. Lead time is always longer than cycle time because cycle time fits into the timeline of lead time



What Is Cycle Time?

In simple terms, cycle time captures the actual time spent on a single unit of work. This metric is measured from the beginning of the first task to the end of the last task and accounts for both value-added time and non-value-added time.

“Actual time” is key here because cycle time is not a measure of the “expected time” to complete one unit of work. Let’s look at a simple example. If the first step for making a single plastic container starts at 7:30 AM, and the completed container is boxed and ready to ship at 9:30 AM, the cycle time for making the container is two hours.

Apart from being used for an entire process, cycle time can also measure specific sub-portions of a process. For instance, a plant can benchmark one cycle time for assembly, another for quality assurance, and a separate one for packaging.

## What Is Lead Time?

In a nutshell, lead time is the time it takes to go from the beginning of a process to its end. Lead time has applications in supply chain management, manufacturing, and project management. It is a useful metric during pre-processing, processing, and post-processing.

Numerous processes affect lead time, including production and inventory management. For instance, building all components of a finished product onsite may elongate the production process, while outsourcing the building of some components can shorten it. Another example of a factor affecting lead time is local labor’s use to speed production.

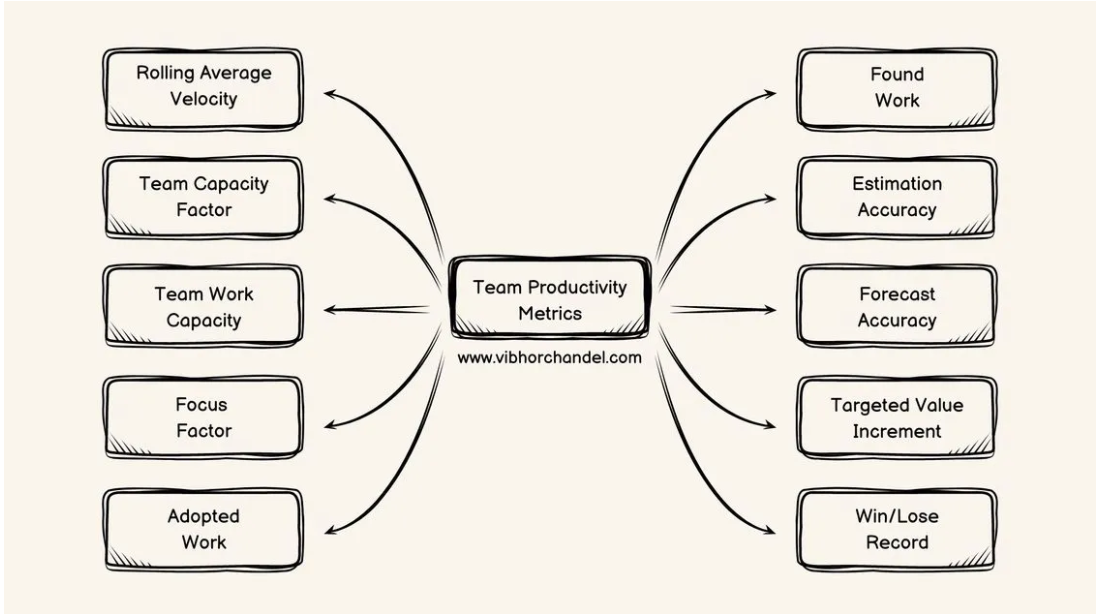
Comparing lead time against benchmarked metrics is an effective way to determine inefficiencies. Minimizing lead time helps enhance operations, boost productivity, increase output, and maximize profitability. Conversely, longer lead times can impact the manufacturing and sales processes negatively.

## How do you measure team productivity? Burndown charts and velocity are not reliable metrics. What other metrics would you suggest?

Thanks for the question!

This was the question that troubled me the most when I worked as a Scrum Master in large legacy organizations. Over time with the help of my colleagues, mentors and google, I collected 9 different metrics that, when used collectively as a system, provide a near-accurate measure of a scrum team’s productivity per sprint.

This can also be used to compare the performance of different scrum teams. The image below lists these 9+1 different metrics.



What is the difference between velocity and capacity?

Capacity is based on the team's expected or projected future availability. Velocity is based on actual points completed, which is typically an average of all previous sprints.

## 9 metrics to measure scrum team’s productivity

1. **Team Capacity Factor** - Team’s availability for the upcoming sprint
2. **Team’s Work Capacity** - Total number of story points “committed” for the Sprint
3. **Focus Factor** - How much of the work capacity is being used/ **how much of a team's total available time (or capacity) is actually spent on productive work**, especially **story points** in a sprint.
4. **Adopted Work (%)** - Amount of scope creep pushed
5. **Found Work (%)** - Amount of scope creep pulled
6. **Estimation Accuracy** **(%)**- The team's ability to correctly estimate the complexity of a piece of work during sprint planning
7. **Forecast Accuracy** **(%)**- The team's ability to correctly estimate the amount of work they can complete during a sprint
8. **Targeted Value Increment** - Targeted value increment is used to measure the improvement in team velocity over time. Whether increases or decreases, the goal is to measure the team’s ability to deliver more value over time. **Measure of a Scrum Team's improvement in their ability to deliver value**
9. **Win/Lose Record for Sprint** - Helps the team understand how successful they are at meeting their commitments.

## Let’s get into details

### Prerequisite - Team’s Rolling Average Velocity

**Purpose**: The rolling average smooths out spikes or drops in velocity from individual sprints and gives you a better idea of long-term trends.

Before we get into the details of calculating and using the above metrics, you need your team’s “Rolling Average Velocity.” It is your team’s average velocity in the last 4-5 sprints. If your team is new and doesn’t have any historical data available to calculate velocity, I would recommend watching the below video. It will guide you on how to calculate a new team’s “initial” velocity.

Once you have your team’s “rolling average velocity” or “initial velocity,” you can dive into the team’s performance metrics below.

For the sake of education, let’s imagine that your team’s **rolling average velocity** is **26.5**

1. **Flexibility in Window Size**:
   * The rolling average window size can be **any number of sprints** you want. It’s common to use 3, 5, or 10 sprints, but you can calculate it for the last 4 sprints, as you mentioned, or even more if you prefer.
   * A **shorter window** (e.g., 3 sprints) will be more responsive to recent changes in velocity.
   * A **longer window** (e.g., 5 or more sprints) can provide a smoother trend, reducing the impact of outliers or unusual sprint performance.
2. **Why Choose Different Windows?**:
   * A **smaller window** might be chosen if you want to track your team’s current performance more closely and adapt quickly to changes.
   * A **larger window** is good for spotting long-term trends and mitigating the effects of variations between individual sprints (like an unusually low or high-performing sprint).

### Example Using 4 Sprints:

If you wanted to use the **last 4 sprints** instead of 3, here’s how it would look:

| **Sprint #** | **Velocity (Story Points)** | **Rolling Average (4 Sprints)** |
| --- | --- | --- |
| 1 | 30 | - |
| 2 | 35 | - |
| 3 | 40 | - |
| 4 | 45 | (30 + 35 + 40 + 45) / 4 = 37.5 |
| 5 | 50 | (35 + 40 + 45 + 50) / 4 = 42.5 |
| 6 | 55 | (40 + 45 + 50 + 55) / 4 = 47.5 |

As you can see, the calculation method is the same—just change the number of sprints in the window.

### Best Practices:

* **Team Preferences**: Pick a window size that aligns with your team’s project pace. For a stable, predictable velocity, a larger window (e.g., 5 sprints) might be helpful. For teams working in a dynamic environment with more variability, a smaller window might provide quicker insights.
* **Historical Context**: If your team’s performance is improving (or declining) rapidly, you may want to adjust the rolling window size to reflect that.

<https://www.linkedin.com/pulse/calculating-velocity-new-agile-team-sarah-stewart>

### 1. Team Capacity Factor

**Definition:**

It is the availability of your team for the upcoming sprint. Calculated as a fraction of the team’s baseline capacity (ideal case scenario).

**Example Scenario:**

Let’s say your scrum team consists of 5 members (not including SM & PO), who work together in 2-week sprints.

**Baseline Capacity** = (number of working days in the Sprint) x (number of team members) x (available hours per day)

Example Baseline Capacity = 10 x 5 x 8 = 400 hours

Note: If some of your team members are regularly only available for less than 40 hours per week, then the sum total of individual regular hours available will become the baseline.

**Team Capacity** = (TeamMember#1 hours for 2 weeks) + (TeamMember#2 hours for 2 weeks) + (TeamMember#3 hours for 2 weeks) + (TeamMember#4 hours for 2 weeks) + (TeamMember#5 hours for 2 weeks)

Example Team Capacity = 45 + 65 + 80 + 80 + 25 = 295 hours

**Team Capacity Factor** = (Team Capacity)/(Baseline Capacity)

Example Team Capacity Factor = 295/400 = 0.74

You must calculate your team’s capacity factor before the team starts the new sprint. It should be based on a realistic assessment of the team's ability to deliver the work.

Don’t forget to take into account any leave, vacations, and things like meetings, planning sessions, and other non-work-related activities that may affect the team's availability during the Sprint.

### 1. **Determine the total available hours for the team**:

First, you need to calculate the total number of hours available for all team members, considering their different working hours. Here's how you can do it:

* For each team member, multiply their **daily working hours** by the number of **days** in the sprint to get their total available hours.
* Sum up the total hours for all team members.

Example:

* Member 1: 6 hours/day × 10 days = 60 hours
* Member 2: 7 hours/day × 10 days = 70 hours
* Member 3: 5 hours/day × 10 days = 50 hours
* Member 4: 6.5 hours/day × 10 days = 65 hours
* Member 5: 8 hours/day × 10 days = 80 hours

**Total hours = 60 + 70 + 50 + 65 + 80 = 325 hours**

### 2. **Apply the Team Capacity Factor (0.74)**:

Now, use the **team capacity factor** (0.74) to adjust the total capacity based on the actual productivity and efficiency of the team.

**Adjusted available hours** = Total hours × Team Capacity Factor

325 hours×0.74=240.5 hours325 \text{ hours} × 0.74 = 240.5 \text{ hours}325 hours×0.74=240.5 hours

### 3. **Convert the adjusted hours into story points**:

Next, if you know how many hours it typically takes for your team to complete **one story point**, you can convert the total available hours into the number of story points.

For example, let’s say your team completes **1 story point per 4 hours** on average.

Number of story points=Adjusted available hoursHours per story point\text{Number of story points} = \frac{\text{Adjusted available hours}}{\text{Hours per story point}}Number of story points=Hours per story pointAdjusted available hours​ 240.5 hours4 hours per story point=60.125 story points\frac{240.5 \text{ hours}}{4 \text{ hours per story point}} = 60.125 \text{ story points}4 hours per story point240.5 hours​=60.125 story points

### Result:

So, using the team capacity factor of **0.74**, the team can realistically complete around **60 story points** in the sprint.

### Notes:

* The number of story points depends on your team's average story point velocity (how many hours it takes to complete 1 story point).
* The capacity factor adjusts for the actual efficiency of the team, considering factors like different working hours, team experience, and potential interruptions.

### 2. Team’s Work Capacity

**Definition:**

The **team’s work capacity** is the total amount of work in **story points** that the team **commits** to for the upcoming Sprint. It helps the team understand how much work they can handle in the next sprint. Whether they are over-committing or under-committing. It also helps in identifying areas where the team can improve its efficiency.

Taking too much work can lead to burnout and decreased productivity. Taking too little can lead to underutilization of the total capacity available.

**How to calculate:**

To calculate your team’s work capacity, you need to estimate the amount of work (in story points) that can be completed during the next sprint. Use your team’s past performance, their knowledge of the work and other factors that might affect their ability to complete the work.

Here’s a simplified formula:

**Work Capacity** = (Total number of story points the team feels it can commit to) x (team capacity factor)

**Example**: Your Product Owner and the team together select the user stories that are important to deliver the sprint goal. Let’s say the total number of story points for the selected user stories is **45**. Then,

Work Capacity = 45 x .74 = **33.3**

This means your team can accept between **30 to 35** story points for the next sprint, taking the team’s availability into account. Where **0.74** is the **team capacity factor** (calculated above).

**Best practices to use work capacity:**

1. Review and adjust every week: Calculate work capacity before starting each sprint.
2. Communicate changes to stakeholders and keep them aligned for support.
3. Identify problems: Observe the fluctuation (increase or decrease) every sprint to pinpoint potential bottlenecks, impediments and over-commitment or underutilization of the team’s availability.

**Note**

The team’s average rolling velocity is **26.5**(shown in the prerequisite section above), less than the **team’s work capacity of 33.3**.

The question now arises, why is the team accepting more than its velocity?

This is completely normal as opposed to the common understanding.

Typically most Scrum Masters force the team to accept work equal to or below the average velocity. This, in most circumstances, proves to be the wrong approach as it underestimates the team’s ability to improve (the **Focus Factor**) as described below.

A better practice is to let the Product Owner and team select what is needed (realistically) to achieve the sprint goal.

### 3. Focus Factor

**Definition:**

Focus Factor measures how much of its available work capacity the team is using and whether the team is focusing on the most important work.

**Focus Factor** = (team’s rolling average velocity) / (team’s work capacity)

Why is it important?

It helps the team identify the areas where it can improve its productivity and efficiency.

**Example**: 26.5/33.5 = 0.794

The focus factor for a healthy team is usually around **0.8** (give or take a few decimals).

If the focus factor is much less (eg. .70), it indicates that the team is over-committing or not using the full team’s capacity.

If the focus factor is too high (e.g. >1.2), it usually means the team is under-committing or ignoring things they shouldn’t, like technical debt.

By monitoring your team’s **Focus Factor** over time, you can identify trends. This can help you make adjustments to the process (or whatever else) to improve your team’s performance.

**How to use Focus Factor?**

1. Use it as a continuous improvement tool
2. To set realistic expectations for the team. Asking the team to keep their focus factor around 0.8 can help them set realistic expectations
3. Regularly reviewing Focus Factor can help your team to stay on track and make adjustments as needed

### 4. % Adopted Work

**Definition:**

It measures the scope creep that happens in the middle of the Sprint. It makes the team understand how much work they are bringing in from future sprints, helping them identify planning or estimation issues.

**% Adopted Work** = (no. of adopted story points) / (original story points committed for the sprint) x 100

**Why do we need it?**

It helps the team measure the amount of scope creep that happens during a particular sprint. If Adopted work is > 20%, it indicates that the planning process must be improved or the team is unable to stick to their plan due to external pressure.

**How to use % Adopted Work?**

1. Track it over time to identify trends and make adjustments to your sprint planning as needed. For example, asking the team to spend more time on backlog refinement or improving communication with the Product Owner.
2. Share it with the stakeholders to make them understand the impact of scope creep.

### 5. % Found Work

**Definition:**

It measures the scope creep that happens due to unexpected but unavoidable work.

**% Found Work** = (no. of found story points) / (original story points committed for the sprint) x 100

**Why do we need it?**

It helps the team understand the amount of time that is being spent on unexpected work and how this unexpected work may affect the delivery of their planned work.

It also highlights that the team needs to improve their user story-splitting exercise.

**Best Practices:**

1. Keep track of all the unplanned work
2. Analyze the reason behind the found work
3. Review story splitting exercise

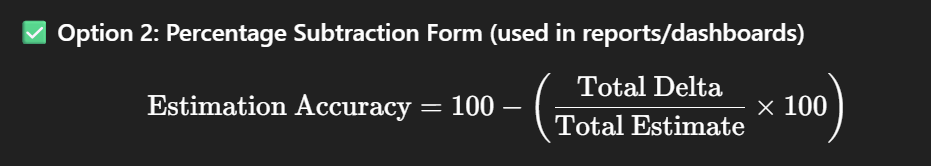
* **Adopted work** is intentionally added to the project scope (through stakeholder decision-making).
* **Found work** is unplanned or unforeseen tasks that arise naturally as the project progresses.

Both contribute to scope creep, but they differ in their origins and the level of control or planning involved.

### 6. Estimation Accuracy

**Definition:**

Estimation accuracy reflects how accurately the team can estimate the amount of work they can complete during the sprint.



**Estimation Accuracy** = 1 - ∑(Estimate deltas) / (original total story points committed for the sprint) x 100

**What are estimated deltas?**

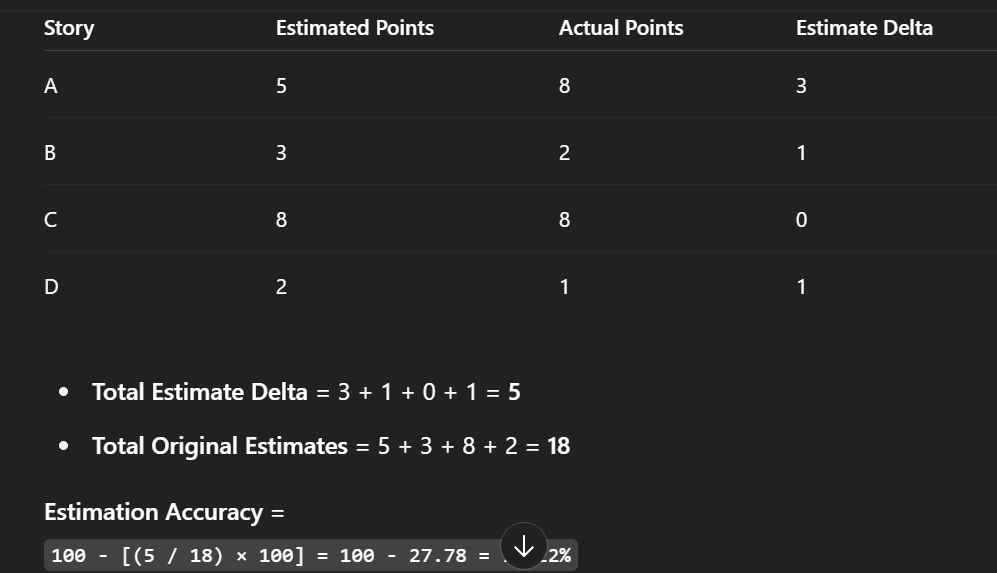
Suppose your team estimates a user story A to be 5 story points. But while working on it during the sprint, the team discovers it is more like an 8-pointer.

Then estimate delta for user story A is:

8 - 5 = 3

The estimate delta for user story A will remain the same if the original estimate is 8 and later found to be 5.

To calculate the accuracy of estimation, add all the estimate deltas for all user stories and use the formula above.



Ah, I see! You're asking about what constitutes a "good" or "bad" **Estimation Accuracy** percentage.

Here’s a general guideline for interpreting **Estimation Accuracy** percentages:

### **Good Estimation Accuracy**:

* **80% or higher**: A good rule of thumb is that if your team is achieving an estimation accuracy of 80% or more, you're doing well. This means that, on average, your estimates are within 20% of the actual work required. This is a strong indicator that your team is getting a good sense of the work involved and can plan sprints effectively.
  + **Why it’s good**: Estimation is an inherently imprecise process. External factors (new tech, interruptions, changes in scope, etc.) will often affect accuracy. If your team is consistently getting close to the original estimate, you’re striking a good balance between realistic forecasting and unforeseen challenges.

### **Fair Estimation Accuracy**:

* **70% to 80%**: This is still considered acceptable, but there's room for improvement. If your team’s estimation accuracy is in this range, it means that there’s a moderate gap between estimated and actual work. The team is somewhat effective at forecasting but may still need to improve in understanding the complexity and risks involved in user stories.
  + **Why it’s fair**: A 70-80% accuracy range indicates that there might be some recurring challenges in estimation, such as underestimating complexity or misjudging capacity. However, it’s not so far off that it severely hampers sprint planning or delivery.

### **Poor Estimation Accuracy**:

* **Below 70%**: Anything below 70% starts to raise concerns. If your team consistently misses the mark by over 30% in either direction (too high or too low), this suggests issues in the estimation process or miscommunication around story complexity and effort.
  + **Why it’s poor**: Low estimation accuracy can result in poor sprint planning. If the estimates are consistently off by such a large margin, your team may be committing to more work than they can handle or undercommitting and underdelivering. This can cause frustration, missed deadlines, or overburdened team members.

### What to Aim For:

* **Target Range**: Aim for an estimation accuracy of around **80-85%**. This allows for some flexibility and accounts for the natural uncertainties in the estimation process, while still maintaining a good level of predictability and efficiency.

### Key Points:

* **Overestimating** consistently (leading to a low estimation accuracy percentage) can result in **idle time** or **wasted capacity** in your sprints.
* **Underestimating** leads to **unrealistic sprint goals**, missed deadlines, and team stress.

### What to Do if Your Estimation Accuracy is Low:

* **Review Estimation Techniques**: Make sure your team is using consistent, effective methods (like Planning Poker or relative sizing).
* **Analyze Common Causes**: Look into specific areas where estimates tend to be inaccurate and understand the root causes.
* **Improve Sprint Planning**: Consider breaking large tasks into smaller, more manageable stories to make them easier to estimate accurately.
* **Adjust Team Practices**: Maybe your team is overcommitting or undercommitting. Regularly track and adjust your capacity based on real data.

In summary:

* **80% or higher** is generally **good**.
* **70%-80%** is **acceptable** but can improve.
* **Below 70%** is **poor** and needs attention.

**Why do we need it?**

Helps the team identify areas and ways to improve user story estimation.

**Best Practices:**

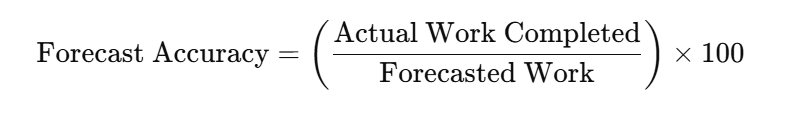
1. Ensure that the team has a good understanding of the complexity of the work involved
2. Break down work items into smaller, more manageable user stories
3. Should remain close to 80% in healthy teams
4. If over 90%, the team is spending more time than needed on estimation
5. if less than 70%, it usually means that the team is experiencing a lot of external pressure, user stories are not well understood, or PO is not available
6. It may also mean that some members of the team are hoarding knowledge

### Forecast Accuracy

**Forecast Accuracy** is another important metric that evaluates how well your team’s forecasts align with actual outcomes. It helps to measure how effectively the team can predict the work that will be completed in a given time frame (e.g., sprint, release cycle).

The formula for **Forecast Accuracy** is as follows:

### **Forecast Accuracy Formula:**



Where:

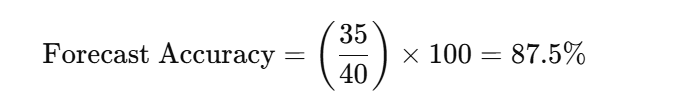
* **Actual Work Completed** is the actual number of story points, tasks, or units of work completed during the sprint (or time period being measured).
* **Forecasted Work** is the work your team initially predicted or committed to completing during that sprint or time period.

### **Interpreting Forecast Accuracy:**

* **100% Accuracy**: If the forecasted work exactly matches the actual work completed, your forecast accuracy is 100%. This means that your team predicted their capacity and output with perfect precision.
* **Greater Than 100%**: If you complete more than forecasted (say, 120% of the forecasted work), this may indicate that your team is underestimating their capacity or they are delivering more than initially expected.
* **Less Than 100%**: If the actual work completed is less than forecasted (for example, 80% of the forecast), it suggests that the team overestimated their capacity or there were unexpected challenges preventing full completion.

### **Example Calculation:**

Suppose your team forecasted that they would complete **40 story points** in a sprint. By the end of the sprint, they actually completed **35 story points**.



This means the team’s forecast accuracy is **87.5%**, which indicates that the team completed most of the forecasted work but didn't quite hit the mark.

### **Why is Forecast Accuracy Important?**

* **Improves Planning**: High forecast accuracy helps teams make more reliable future predictions and set achievable sprint goals.
* **Better Resource Allocation**: Teams with good forecast accuracy are better at managing their capacity and avoid over-committing or under-delivering.
* **Increases Stakeholder Confidence**: Consistently hitting forecast targets builds confidence with stakeholders, as they can rely on your team’s capacity and predictions.

### **How to Improve Forecast Accuracy**:

1. **Track Historical Data**: Use past sprint data to improve future forecasts. If your team consistently underestimates or overestimates, use this data to adjust your forecasting approach.
2. **Break Down Work into Smaller Tasks**: Smaller tasks tend to be more predictable, leading to more accurate forecasts.
3. **Iterate and Learn**: Regularly review and adjust your forecasting methods based on retrospective insights. Learn from what was overestimated or underestimated.
4. **Incorporate Buffer**: Allow for a small buffer in your forecasts to account for uncertainties, especially with complex tasks.

In short, **Forecast Accuracy** helps assess how closely the team’s predicted output matches the actual result, and achieving a high forecast accuracy rate enables better planning, expectations, and resource management.

**Definition:**

Forecast accuracy reflects how accurately the team can forecast the amount of work they can complete during the sprint.

**Estimation Accuracy** = (original total story points committed for the sprint) / [ (original total story points committed for the sprint) + (adopted story points) + (found story points)]

**Why do we need it?**

Helps the team identify how reliable their forecasts are.

**Best Practices:**

1. Ensure that the team has a good understanding of the complexity of the work involved
2. Break down work items into smaller, more manageable user stories
3. Healthy team’s forecast accuracy is around 80% (again)
4. 100% accuracy means that Goodhart’s law is in effect. The team is targeting velocity
5. Below 75% means the team needs protection from forced work

### 8. Targeted Value Increment

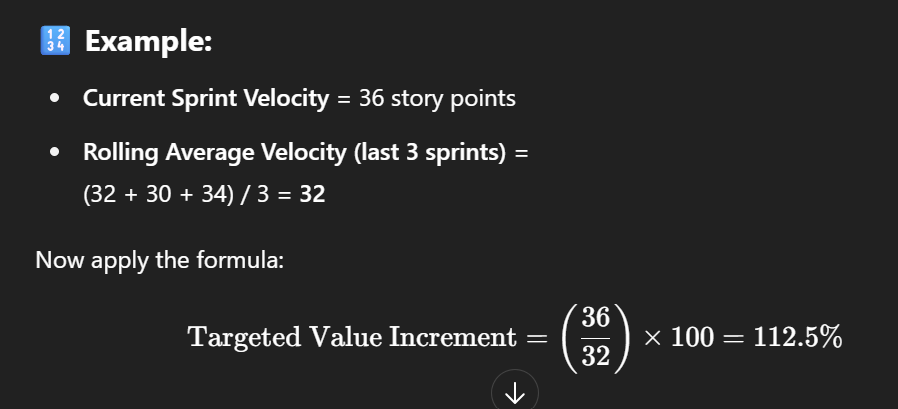
**Definition:**

Targeted value increment is used to measure the improvement in team velocity over time. Whether increases or decreases, the goal is to measure the team’s ability to deliver more value over time.

**Targeted Value Increment** = (current sprint’s velocity) / (rolling average velocity) x 100

**Best Practices:**

1. Make sure your team uses a consistent process to estimate user stories
2. Keep historical data handy to identify trends
3. Analyze the root cause of the change
4. Always use TVI in conjunction with other metrics



### Interpretation:

* **TVI > 100**: This means the team completed **more work** in the current sprint compared to their **average velocity** from previous sprints. Essentially, the team delivered more value than usual in the current sprint, which can be considered a good sign of increased productivity or efficiency.
* **TVI = 100**: This indicates the team completed **exactly as much work** as their average from previous sprints, meaning they’re on track and maintaining consistent productivity.
* **TVI < 100**: This means the team delivered **less work** than their rolling average velocity, which might indicate a dip in productivity, challenges faced during the sprint, or other factors like unexpected issues or blockers.

### Is >100 good?

* **Yes**, but with nuance:
  + **Short-term success**: A TVI greater than 100 indicates that the team was able to deliver more than expected. This could be due to better team collaboration, fewer blockers, or optimized processes in that sprint.
  + **Caution**: A significantly high TVI could also suggest that the team might have over-committed or worked more than usual, potentially leading to burnout or quality concerns. If this pattern continues over multiple sprints, it might signal that the team's average velocity was underestimated, or that they might be taking on more work than they can sustainably handle.

So, while a TVI above 100 is generally positive, it's essential to monitor if it's a sustainable pace and whether the quality of work is still maintained.

### Key Takeaway:

* A **TVI > 100** is a sign of increased productivity and could indicate that the team is improving or tackling tasks more efficiently.
* **Sustainability** is key: As long as the team can maintain quality and avoid burnout, a TVI > 100 is beneficial. However, if the increase is due to overcommitment or shortcuts, it may be a warning sign.

### 9. Win/Lose Record

**Definition:**

Win/Lose record helps the team measure its ability to meet the sprint goal. It helps track if the team is able to deliver the planned work and manage uncertainty or unplanned work.

**Win**= **80%** of committed work is **accepted** by the Product Owner, and the combined surprise work **(found + adopted)** remains at a level of **20%** or less of the original forecast

**Lose** = Less than 75% of work is accepted

**Progressing** = Between 80% and 75% accepted

**Best Practices:**

1. Use a consistent definition of win/lose
2. Analyze the root cause of losses
3. Use it in conjunction with other metrics
4. Focus on continuous improvement

This is it 🙏

This post contains a great deal of information, so don't worry about attempting to remember it all.

Consider it more of a reference guide or a source of inspiration for when you're generating ideas. And when you do come up with brilliant ideas, hit me up!

I am always open to suggestions and feedback. Pls, let me know if I missed something or if something seems odd.

🌟**Also, if you share parts of the framework on social media, I request you to please use proper credits. I won’t be as frequent in sharing this kind of content if I have to track copies of it floating around on social media**.🌟

<https://premieragile.com/burn-down-chart-vs-burn-up-chart/>

what are all the circumstances scope creep comes in a Sprint?

###### ChatGPT said:

ChatGPT

**Scope creep** in a sprint occurs when additional work or requirements are introduced beyond the agreed-upon scope of the sprint, often leading to delays or quality issues. This can be particularly problematic in Agile methodologies, as it affects both the team’s ability to meet their goals and the overall project timeline.

Here are the common circumstances under which **scope creep** can occur in a sprint:

### 1. **New Requirements or Features Introduced by Stakeholders**

* **Circumstance**: Stakeholders or clients request new features, functionalities, or changes after the sprint has already started.
* **Example**: A product owner or client asks for a new report feature to be added midway through the sprint, even though the team has already committed to a set of tasks.
* **Impact**: This introduces additional work that wasn’t planned for, leading to delays or the need to reallocate resources.

### 2. **Lack of Clear Requirements at Sprint Planning**

* **Circumstance**: The product backlog items (PBIs) or user stories aren’t well-defined at the start of the sprint.
* **Example**: A user story is written with vague or incomplete acceptance criteria, and as the team works through it, new requirements or interpretations emerge.
* **Impact**: As the team clarifies requirements during the sprint, new tasks or features can be added, increasing the scope.

### 3. **Unforeseen Dependencies**

* **Circumstance**: Unanticipated external or internal dependencies emerge that require additional work to resolve.
* **Example**: The team realizes that a feature they're working on requires integration with a third-party system or another team’s work that wasn't initially identified.
* **Impact**: Resolving dependencies or waiting for external teams can delay work and introduce new tasks or technical debt, pushing the sprint’s scope beyond its original plan.

### 4. **Product Owner or Team’s Misalignment on Priorities**

* **Circumstance**: There’s a miscommunication between the product owner and the development team regarding priorities.
* **Example**: Mid-sprint, the product owner re-prioritizes tasks or features based on new business goals or feedback, causing changes to what was originally planned.
* **Impact**: Reprioritizing tasks without proper planning may cause the team to start working on additional or more urgent items, disrupting the agreed scope of the sprint.

### 5. **Urgent Bugs or Issues That Need Immediate Attention**

* **Circumstance**: Critical bugs or issues are discovered during the sprint that must be addressed immediately, requiring time and resources that were not accounted for.
* **Example**: A major bug in the production environment forces the team to stop working on their sprint tasks and resolve the issue, increasing the scope of the sprint.
* **Impact**: The unplanned work of fixing urgent issues can push the team to either delay other planned work or exceed the sprint’s capacity.

### 6. **Over-Commitment During Sprint Planning**

* **Circumstance**: The team commits to too many items during sprint planning or underestimates the complexity of the tasks.
* **Example**: During sprint planning, the team accepts a large number of user stories, only to realize that some tasks are more complex than originally thought.
* **Impact**: As the team runs out of time, they may attempt to add unplanned work to meet sprint goals, thus expanding the scope.

### 7. **Uncontrolled Changes to the Backlog**

* **Circumstance**: The product backlog is constantly changing during the sprint without proper control or prioritization.
* **Example**: New user stories or changes are added to the sprint backlog after sprint planning, often because stakeholders change their minds or new ideas emerge during the sprint.
* **Impact**: Continuous changes to the backlog during a sprint lead to scope creep, as the team is constantly adjusting to new work instead of focusing on the committed tasks.

### 8. **Lack of Proper Sprint Review and Feedback Control**

* **Circumstance**: During the sprint review, stakeholders provide additional feedback that leads to unplanned changes to the features that were completed.
* **Example**: After seeing the work completed during the sprint, the stakeholders request additional tweaks or changes, thinking it’s a small change, but it requires significant additional effort.
* **Impact**: This unanticipated feedback forces the team to make changes to the scope of the sprint even after the work has been “done.”

### 9. **Team Misunderstanding of Sprint Goals**

* **Circumstance**: The team misunderstands the focus or priority of the sprint goals, leading them to work on non-essential tasks that were not part of the sprint commitment.
* **Example**: The team may decide to implement a nice-to-have feature they believe is important, even though it wasn’t part of the sprint plan.
* **Impact**: This additional work leads to scope creep and may reduce the team's ability to focus on delivering the most critical, high-priority items.

### 10. **External Pressures**

* **Circumstance**: External forces, like marketing teams, customer feedback, or business needs, exert pressure on the team to introduce additional features.
* **Example**: A marketing campaign is planned, and the product team is asked to implement a specific feature in the current sprint to meet the campaign’s timeline.
* **Impact**: The team is forced to adjust the scope of their sprint to accommodate external deadlines, leading to scope creep.

### How to Prevent Scope Creep in a Sprint:

1. **Clear Sprint Planning**: Ensure that user stories are well-defined, prioritized, and have clear acceptance criteria.
2. **Stakeholder Alignment**: Establish clear communication with stakeholders to avoid mid-sprint changes or additions.
3. **Timeboxed Sprints**: Stick to the sprint goal and timebox the work to prevent the temptation to add new tasks during the sprint.
4. **Manage Change Requests**: Any changes during a sprint should be carefully considered and discussed with the team, evaluating the impact on current commitments.
5. **Continuous Communication**: Hold daily standups to identify potential scope creep early and adjust as needed.
6. **Keep the Backlog Organized**: Ensure that the product backlog is regularly reviewed and that changes are introduced only during sprint planning or after the sprint review.

By being proactive and managing the above factors, teams can minimize the risk of scope creep and stay focused on delivering high-quality, value-driven work within each sprint.

You're right that **burndown** and **burnup** charts in Scrum both visually track progress, but they do so in different ways and offer distinct insights into the team's performance and scope changes during a sprint or project. Here's a breakdown of the main differences:

### 1. **Burndown Chart**:

* **What It Tracks**: The burndown chart tracks the **remaining work** in the sprint or project.
* **X-Axis (Horizontal)**: Represents time (usually in days or sprints).
* **Y-Axis (Vertical)**: Represents the remaining work (typically measured in story points, hours, or tasks).
* **How It Works**: As the team works through the sprint, the chart shows how much work remains to be completed. The goal is to see the line “burn down” to zero, meaning all work is completed by the end of the sprint.
* **What It Shows**:
  + **Remaining Work**: At the start, the chart shows the total planned work, and as tasks are completed, the remaining work decreases.
  + **Ideal Line**: There's usually a straight, diagonal line showing the ideal pace at which the work should be completed (this line represents the expected progress).
* **Purpose**: It helps the team track if they are on schedule and if they will finish the sprint on time with the planned amount of work.

**Example of Burndown**:

* If the team starts with 50 story points and completes tasks, the line will go down to zero as the remaining work is completed. Any deviation from the ideal line can indicate potential delays or obstacles.

### 2. **Burnup Chart**:

* **What It Tracks**: The burnup chart tracks **completed work** and the **total scope** of the project or sprint.
* **X-Axis (Horizontal)**: Represents time (usually in days or sprints).
* **Y-Axis (Vertical)**: Represents the total amount of work (usually measured in story points, hours, or tasks).
* **How It Works**: The burnup chart shows how much work has been completed over time, and it also tracks changes in the total scope of the project. The total scope line typically stays constant, but it can also increase if new work is added during the sprint (e.g., new requirements or features).
* **What It Shows**:
  + **Completed Work**: As the team completes work, the completed line moves upward, showing progress toward the sprint goal.
  + **Scope Line**: A line that represents the total scope of the project or sprint. If new features or tasks are added (scope creep), this line will increase.
* **Purpose**: It helps to visualize both the amount of work completed and how the total scope is changing (i.e., if new work is added during the sprint). It’s particularly useful for understanding **scope changes** and how they affect the team’s progress.

**Example of Burnup**:

* The team might start with 50 story points. As work is completed, the line moves upward, but if new tasks are added (e.g., client requests), the scope line will move upward as well. This helps identify the effect of scope creep.

### 

### **When to Use Each**:

* **Burndown Chart**: Ideal when the scope is fixed (i.e., no new tasks or features are being added during the sprint) and the goal is to see if the team is on track to complete all the work they committed to.
  + **Best For**: Focus on tracking how quickly the team is completing their original sprint commitments.
* **Burnup Chart**: Best when the scope is variable or there’s a risk of scope creep, as it shows both progress and how the total scope is changing. This makes it easier to understand if the team’s progress is being affected by new requirements being added during the sprint.
  + **Best For**: Projects with frequent changes in scope or where tracking progress against a changing set of deliverables is important.

### **Summary**:

* **Burndown**: Shows **remaining work** and helps assess whether the team will complete their tasks by the end of the sprint.
* **Burnup**: Shows both **completed work** and the **total scope** (which can change), making it more useful in situations where the scope might change mid-sprint.

In short, **burndown** is great for tracking how much work is left, while **burnup** gives a more complete picture of progress and scope, especially in projects where the scope can evolve.