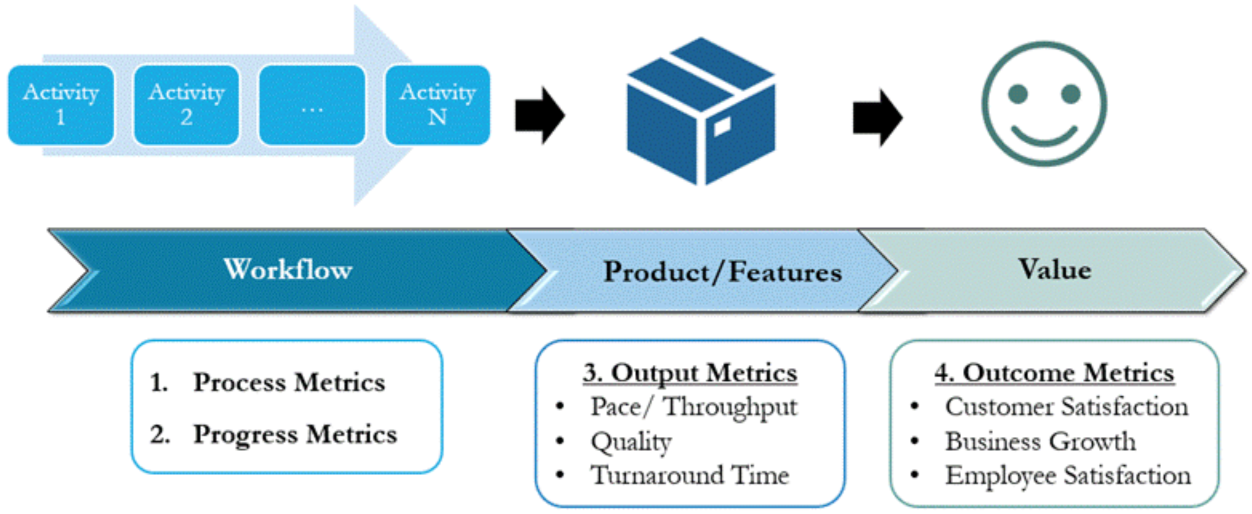
Accionlabs
Driving Outcomes Through Actions!

Metrics & KPI

Metrics and KPI’s

Types of Metrics

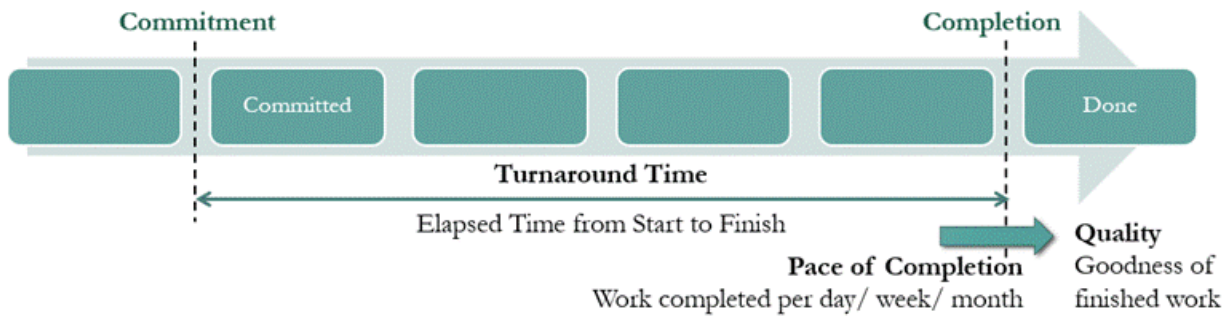


As depicted in the picture above, there are different categories of metrics relevant for end-to-end product development flow:

1. **Process Metrics:** These help us assess the usage and quality of recommended process and practices. Examples: Team size, team composition (Presence of specific roles, Dev-QA ratio), tools used, meetings conducted, etc.
2. **Progress Metrics:** These are time trends that often serve as leading indicators for team and flow performance. Examples: Burnup/Burndown charts and Cumulative flow diagram.
3. **Output or Flow Performance Metrics:** These metrics help us assess the performance of a single team and the product-level flow. Examples: Team Velocity, Cycle Time, Defect Count, Defect Density, etc.
4. **Outcome Metrics:** These metrics help us measure the ultimate outcomes that are important for customers, stakeholders, people and the organization. These are the most important metrics and help us assess the overall performance. Unfortunately, these metrics are often difficult to measure and are lagging indicators.

In this handbook, we will focus primarily on the Output and Progress metrics.

Output or Flow Performance Metrics


Commitment
Completion
Committed
Done
Turnaround Time
Elapsed Time from Start to Finish
Pace of Completion
Work completed per day/ week/ month
Quality
Goodness of finished work

There are three primary dimensions to assess the performance of end-to-end product development workflow:

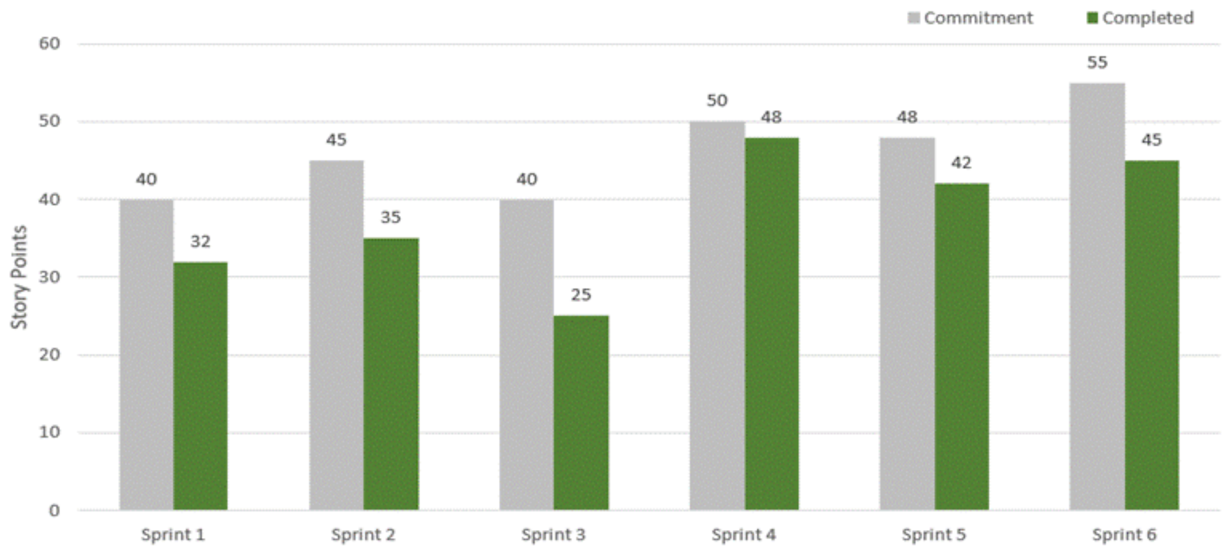
1. Pace of Completion - Measures 'How Much' work is completed in a specific time period (week, sprint, or month).
2. Quality – Measures 'How Well' the finished work looks, with respect to functional expectations and quality standards.
3. Turnaround Time – Measures 'How Fast' we finish work.

The table below lists some common examples of these three Output metrics at the team and product level (in a multi-team scenario):

	Pace	Quality	Turnaround Time
Team Level	Team Velocity, Throughput	Defect Count/Density (QA)	Cycle Time (Story), Lead Time (Story)
Product Level	Epic/Feature Throughput	Code Quality/Complexity, Code Coverage (UT), Test Automation Coverage, Defect Count/Density (UAT), Escaped defects (P1, P2), Change Failure Rate	Cycle Time (Epic), Lead Time (Epic), Release/Deployment Frequency

Team Velocity

Velocity measures the average amount of work a Scrum Team completes in a Sprint, usually expressed in story points (or backlog-item count). Tracking velocity over multiple Sprints helps teams assess their consistency and/or improvement in performance. Below is an example of Velocity chart:



To assess the quality of Team Velocity chart, we should look for the following:

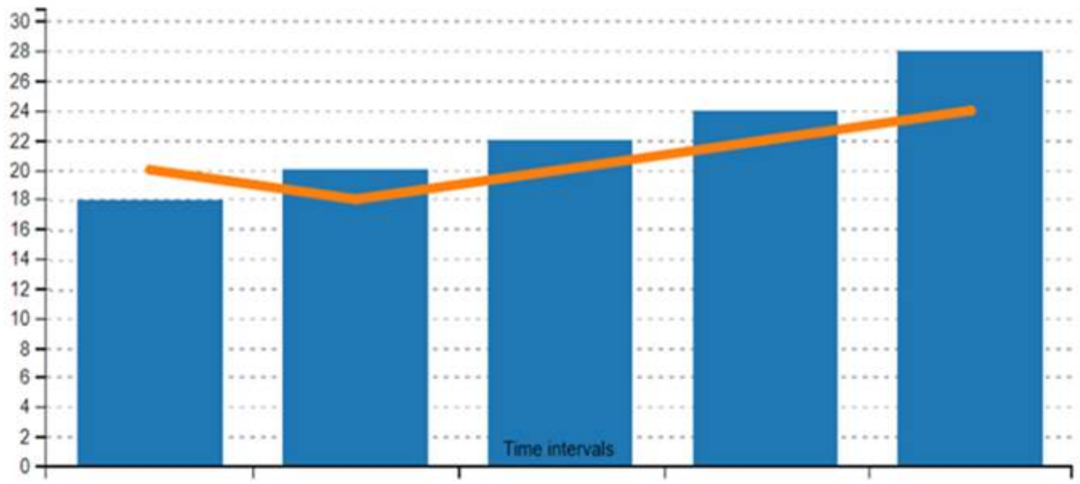
Interpretation Guidelines:

1. **Chart Reliability**
   - Are story points sized consistently across items?
   - Do "Committed" vs "Completed" bars accurately reflect scope at Sprint start and end?
2. **Say/Do Ratio**
   - Is actual completion ≥ 80% of planned work for most sprints?
3. **Velocity Trend**
   - Is velocity fairly steady or gently rising with time?

Team Throughput

Throughput measures the amount of work completed in a time interval (week or month). It is usually calculated as the number of items completed, but may be revised to count total story points completed (weighted throughput) if the team uses story point estimation.

Throughput offers a simple view of delivery rate and can reveal bottlenecks when combined with work-in-progress metrics. Plotting throughput trends can help teams smooth flow and set realistic expectations.



Interpretation Guidelines:

1. **Chart Reliability**
   - Are throughput numbers reliable? Does the team update items in a timely manner.
2. **Trend Analysis**
   - Is throughput stable, climbing, or dropping?
   - A downward trend and high variation imply that there are challenges that need to be addressed.

Epic/Feature Throughput

Epic or feature throughput counts the number of Epics (or large features) finished over time. This high-level metric shows how quickly major slices of functionality move from idea to done.

This is a product-level metric that is especially relevant in a multi-team context – it provides product-level pace across teams and can be used for forecasting release scope.

The interpretation guidelines are similar to the guidelines for Team Throughput – looking for consistency across the timeline. A common challenge is teams starting too many epics in parallel and finishing most of them towards the release date. A better approach is to limit Epics in progress (WIP) so they can be completed incrementally, thereby enabling faster feedback and reducing stress towards the end of release.

Defect Count (QA/UAT)

Defect count is the total number of reported bugs (often by severity) in a given period or release. Monitoring total defect count helps teams understand stability and user-impacting issues. Teams usually keep a separate count of different defects - QA, Integration, and UAT defects. Integration defects are optional – possibly relevant for a multi-team scenario where Epic level integration testing is done after teams are done with their Story level testing.

Interpretation Guidelines:

1. **Severity Mix**
   - What proportion are critical vs. minor?
   - High-severity spikes suggest an urgent need for retrospection.
2. **Trend Over Releases**
   - Are defect counts decreasing to indicate improving quality practices?
   - Plateau or rise suggest need to improve testing rigor and regression coverage.
3. **Origin Breakdown**
   - Defect count should decrease significantly with each stage – QA >> Integration >> UAT.

Defect Density (QA/UAT)

Defect density normalizes defect count by the volume of the work delivered - commonly measured as average defects per Story Point or per Story. This metric offers a more realistic comparison of quality across releases or time periods.

1. **Trend over Time**
   - Is density falling/increasing over time?
   - Rising density indicates need to invest in technical debt reduction.
2. **Module Comparison**
   1. Are some components/modules/epics have significantly higher defect density than others?
3. **Normalization Basis**
   - Ensure consistent size estimation for reliable comparison.

Escaped Defects (P1, P2)

Number of defects detected (Severity 1, Severity 2) in production during the first two weeks after the deployment.

Interpretation Guidelines:

1. **Severity Mix**
   a. What is the ratio of P1 to P2? How does it change over time?
2. **Trend Over Releases**
   a. Are defect counts decreasing to indicate improving quality practices?
   b. How does the count compare to QA/UAT defect count? Is there a positive or negative correlation?

Code Quality/Complexity

Code quality and complexity metrics (e.g., cyclomatic complexity, code smells) are drawn from static analysis tools. Lower complexity and fewer code smells generally correlate with easier maintenance and fewer bugs. Track average complexity per module and trend over time.

Interpretation Guidelines:

1. **Average Complexity**
   - Are most methods/functions under agreed complexity thresholds?
   - Outliers - refactor into smaller, clearer units.
2. **Trend of Smells**
   - Is the count of new code smells decreasing?
   - If increasing, schedule periodic cleanup sprints.
3. **Risky Modules**
   - Identify hotspots with both high complexity and defect density.

Code Coverage (Unit Tests)

Unit-test code coverage is the percentage of application code exercised by automated unit tests. High coverage (e.g., > 80%) indicates that most code paths have been validated, reducing the risk of escaped defects. However, coverage alone isn't a guarantee of quality—tests must also be meaningful (asserting correct behavior), not just executing lines.

Interpretation Guidelines:

1. **Coverage Targets**

- Are you meeting your team's minimum (e.g., 80%) consistently?
- In case of gaps, add tests around critical or complex code.

2. **Meaningful Tests**
   - Is coverage high but defect escape rate still high?
   - Review test quality - assert behaviour, not just execution.
3. **Trend Monitoring**
   - Is coverage slipping with new code?
   - Enforce threshold gates in CI/CD to prevent drop-offs.

Test Automation Coverage

Test automation coverage measures the proportion of overall tests cases (functional, integration, etc.) that have been automated. Increasing automation coverage speeds up regression runs, enables more frequent releases, and frees testers for exploratory testing.

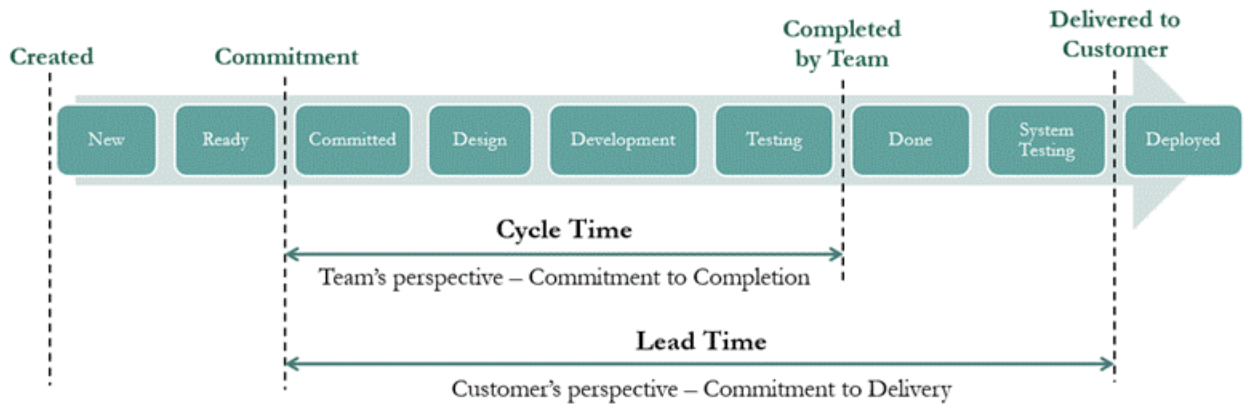
Interpretation Guidelines:

1. **Critical Path Coverage**
   - Are high-risk or core workflows fully automated?
2. **Pace of Automation**
   - Is automation coverage growing sprint over sprint?
   - New vs legacy code – are we able to improve test coverage for legacy code in addition to new code?
3. **Automation Latency**
   - How soon are we able to add automated tests for new code – Are we able to automate functional tests during Sprint, right after, or weeks later?

Cycle Time and Lead Time (Story)

Cycle Time and Lead Time are similar concepts with a slight difference – both measure turnaround time, but one from team's perspective and the other from Customer's perspective.

Cycle time is the elapsed time from the moment team commits an item to when it is marked as done. Lead Time is the elapsed time from the moment an item is committed to the customer/stakeholders to when it is delivered (deployed in Production).



Interpretation Guidelines:

1. **Metric Reliability**
   - Do we have the required workflow statuses in ALM tool to track Cycle and Lead time in a reliable manner?
   - Do team members update workflow status on time to ensure the commitment and completion times are accurately tracked?
2. **70% and 85% Percentile Numbers**
   - For cycle/lead time, percentile numbers are more useful for forecasting than the average numbers.
   - Do we complete most work items within the expected timelines (SLA)?
3. **Spread and Distribution Analysis**
   - What is the shape of the overall spread? Does it have a long-tail (outliers)?
   - Are there any common patterns that cause delays and push items to the tail of the distribution?
4. **Trend over Time**
   - Does the Cycle/Lead time trend show improvement over time - month to month, and one release to the next?
   - How does the Cycle/Lead time trend relate to variation in throughput and quality?

Epic Cycle Time and Lead Time

The definition of Epic Cycle time and Lead time will be similar to that for User Stories, but their timelines will be relatively longer as Epic start time will correspond to the start time for the first story and end time will correspond to the end time for the last story.

In a multi-team scenario where Epic is the deployable unit of work, Epic Lead Time will be more useful than Story Lead Time.

The interpretation guidelines for Epic Cycle/Lead time will also be similar to that for User Stories, but possibly more insightful as Epics take longer than User Stories, and hence, have more room for improvement.

Release/Deployment Frequency

Release frequency counts how often the team delivers shippable increments to production (or to end users) over a period (e.g., monthly, weekly). This metric is useful for teams/products that practice 'release on demand' or 'continuous delivery' rather than 'release on cadence'.

High release frequency enables faster feedback, lowers batch risk, and supports Continuous Delivery mindsets.

Interpretation Guidelines:

1. **Batch Size vs. Frequency**
   - Fewer, larger releases could mean higher risk.
   - Too many tiny releases could imply operational overhead.
   - Aim for a sweet spot balancing speed and stability.
2. **Release Success Rate**
   - What percent of releases are defect-free?
   - Low success → invest in automated pipelines and rollback plans.

Change Failure Rate

This measures the percentage of deployments that fail and need to be rolled back. This metric becomes more important when releases become shorter and more frequent.

Interpretation Guidelines:

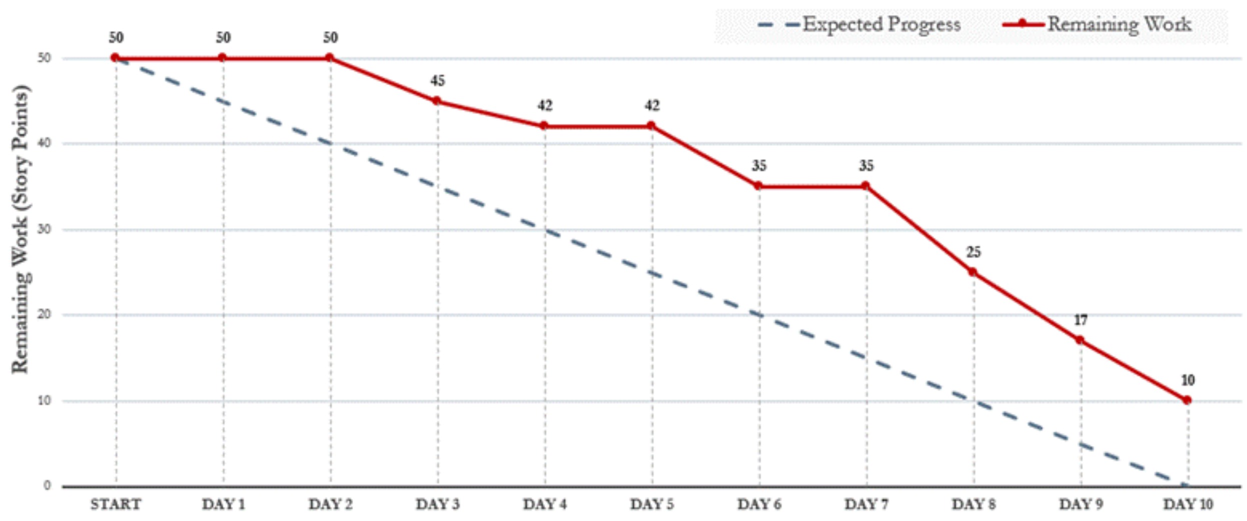
1. **Release confidence**
   a. Are we able to keep failure rate to low levels, thereby improving our release confidence?
2. **Comparison with Release Frequency**
   a. Is the change failure rate directly or inversely proportional to the release frequency?

Progress Metrics

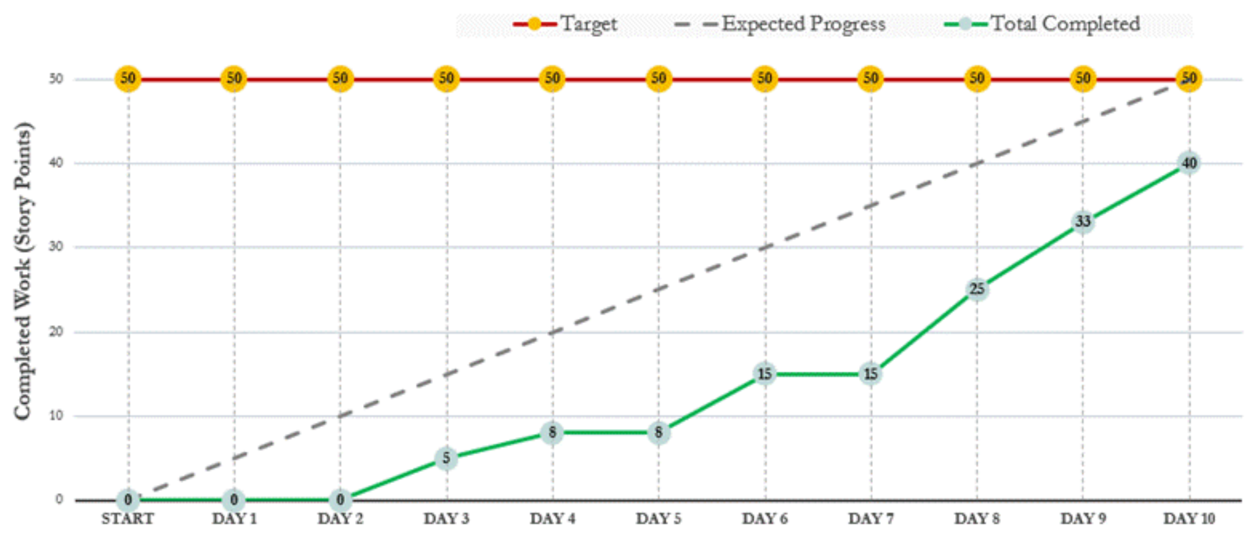
	Scrum Team	Kanban Team
Team Level	Sprint Burnup Sprint Burndown	Story-level Cumulative Flow Diagram (CFD)
Product Level	Release Burnup, Initiative Burnup Story CFD, Epic CFD	

Sprint Burndown/ Burndown Chart

Sprint Burndown chart shows the remaining work (story points or tasks) versus time. It's a classic tool for visualizing whether the team is on track to finish the committed work before Sprint end.



The Sprint Burnup chart plots cumulative work completed (e.g., story points or item count) against total sprint scope across the Sprint timeline. Unlike burndown, it shows both progress and scope changes.

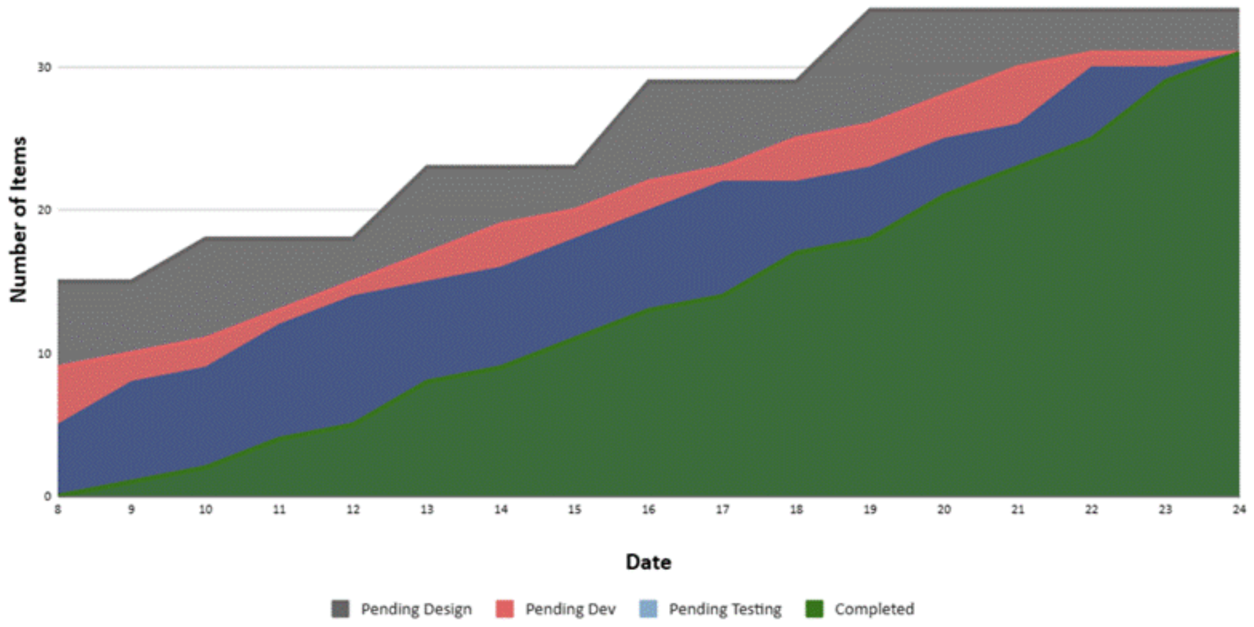


Interpretation Guidelines:

1. **Chart reliability** – Reliability is a big concern for Burndown/Burnup charts as new Scrum teams struggle with ensuring all team members update item status on time.
   - Does the progress line reflect on-ground reality?
   - Is the starting point of Scope line correct.
2. **End State**
   - How much work were we able to complete in the end?
3. **Shape across Sprint**
   - Is there incremental progress during the Sprint, or do we see a big progress spike in the end?
   - How close is the progress line to the ideal line?
   - Are there up-down movements that suggest closed items were reopened?
4. **Scope Changes (Burnup only)**
   - Is the team able to minimize scope changes during Sprint?

Story-Level Cumulative Flow Diagram (CFD)

A story-level CFD shows counts of stories in each workflow state (Backlog, In Progress, Review, Done, etc.) over time. The width of each coloured "band" represents WIP in that state.



Interpretation Guidelines:

1. **Band widths or gap between lines**
    - Are any bands steadily widening – indicating a possible bottleneck.
    - Are any bands too narrow – indicating possible starvation.
    - Ideal flows show gently sloping, parallel bands. Jagged or crossing bands mean unstable flow or frequent state changes.
2. **Slope of Lines**
    - Are all lines moving upwards regularly or are there flat periods indicating lack of progress?
    - Does the slope of bottommost band (Completed) reflect incremental completion of work items?
    - Is the pace of completion good, consistent and improving?

Epic-Level Cumulative Flow Diagram

An epic-level CFD tracks the progress of Epics across the defined workflow – from left to right. It will look similar to Story CFD but would usually have more stages. There may be a need to include discovery stages in Epic CFD or plot Epic discovery CFD separately, in addition to the CFD that covers delivery flow.

Interpretation Guidelines will be similar to the guidelines for Story CFD.

Release Burnup

The Release Burnup chart looks similar to the Sprint Burnup but plots cumulative work progress at a higher level:

- The Scope line plots the total release scope
- The progress line plots release level progress – across teams and across Sprints
- The chart may be plotted in Story points or Item Count

If we have multiple teams working on the release, we should plot Release Burnup in story count instead of story points – as the story point scale will likely differ across teams.

Interpretation Guidelines:

1. **Scope Change Tracking**
    - Is the scope-line stable?
2. **Pace of Progress**
    - Is there a regular, incremental progress or do we see a big spike towards the release date?
    - Does the completion-line's slope match the expected pace of completion (velocity/throughput)?
3. **Forecasting**
    - Based on the current trend, how confident are we to achieve the release goals?
    - Based on our confidence, is there a need to replan the remaining work?

Initiative Burnup

Initiative Burnup looks very similar to the Release Burnup but the scope is different – all work under a large, strategic collection of work (initiative) that could sometimes span across multiple releases.

Initiative Burnup is an optional metric, more useful when the components of an initiative (epics, stories) are worked on by multiple teams and there is a need to track initiative level progress. We usually plot the Initiative Burnup in story count instead of story points, as the story point scale will differ across teams.

The interpretation guidelines will also be similar to Release Burnup, with some changes:

- There could be more scope changes if the initiative spans across multiple releases

- We might see slower pace or flatter sections in Initiative Burnup if teams tend to context-switch regularly across initiatives.