Accionlabs
Driving Outcomes Through Actions!

Agile and Scrum Handbook

Agile Methodologies

Agile Fundamentals

What is Agile?

At its core, Agile is defined by the Agile Manifesto, which was created in 2001 by a group of 17 experts. The Manifesto lays out four values and twelve principles that define the mindset behind Agile ways of working. While it emphasizes team collaboration and adaptability, it doesn't prescribe specific implementation details like roles, practices, meetings, or metrics. For these, we rely on specific frameworks such as Scrum, Kanban, XP (eXtreme Programming), SAFe, LeSS, etc.

Read Agile Manifesto here… https://agilemanifesto.org

Agile is also used as an umbrella term for the various frameworks, models and practices that have evolved in the Agile community over the last three decades – through the collective effort of Agile practitioners trying to embrace Agile ways of working. In addition to popular Agile frameworks, they include practices such as User Stories, Story Point Estimation, Scrum of Scrum, Agile Metrics, etc.

When to use Agile?

Agile processes are particularly suited for complex knowledge work, such as software development, where there is significant uncertainty and variability in both requirements and technical approaches (refer Stacey's Matrix). This uncertainty demands flexibility in scope and adaptability in technical implementation.

Agile is generally not suitable for the manufacturing world, where product development is standardized — producing more and more units of the same, pre-decided design (Simple work as per Stacey's Matrix).

Benefits of Agile

Even in the face of dynamic requirements and evolving solutions, a well-implemented Agile process delivers the following benefits:

1. **Benefits to the Customer/ Stakeholders**
    a. Improved visibility into the work progress
    b. Maximized return on investment

   c. Faster time to market (or faster ROI)
   d. Responsiveness to change
   e. Enhanced Product Quality

2. **Benefits to the Organization/ Management**
   a. Better alignment between stakeholders and the development team
   b. Increased customer satisfaction
   c. Predictable and sustainable pace of development
   d. Motivated work environment (employee engagement)
   e. Continuously evolving process

Popular Agile Frameworks and Common Usage

Scrum

Scrum is Widely used in product development, especially software. It provides a structured approach with 3 roles (Scrum Master, Product Owner, Developers), five events (Sprint, Planning, Daily Scrum, Review, Retrospective), and three artifacts.

Scrum was created and is primarily used for the dynamic world of new product development where its Sprint-based iterative approach facilitates incremental development, fast feedback, quick learning and continuous adaptation to maximize business value delivery.

Kanban Method

Kanban Method (often called simply ‘Kanban’) uses continuous flow to adapt to changing requirements and business needs. It has a flexible process structure and often needs to be customized to suit the team context. It has six core practices, 3 core meetings (Replenishment, Daily Kanban, Retrospective) and utilizes powerful flow metrics to drive continuous improvement.

While it is often used for enhancements and support work, it can work equally well for new product development.

SAFe (Scaled Agile Framework)

SAFe is designed for larger organizations that need to scale Agile practices across multiple teams and departments. It uses Scrum and Kanban at the team level, and adds additional structure (visualization, roles, metrics, meetings, etc.) to support alignment and coordination across teams that work on the same product or a collection of related products (portfolio).

SAFe is an extensive framework and requires significant investment with slow learning curve. It can be a good choice for:

- Large enterprises (500+ people) with complex structures.
- Organizations who want structured guidance in their transition from traditional models to Agile.
- Organizations needing tight alignment with existing hierarchy, portfolio management, and governance.

LeSS (Large-Scale Scrum)/ Nexus/ Scrum@Scale

Similar to SAFe, these are scaling frameworks that are relevant for large-scale product development involving multiple Scrum teams. While the usage is similar to SAFe, they have some structural differences and are relatively simpler than SAFe. These are good options for:

- Mid-size organizations (a few hundred people), department-level scaling or scaling Agile at the product level (3 to 9 teams)
- Organizations that are already practicing Scrum well at the team level
- Organizations that value simplicity and lean thinking, and prefer scaling incrementally and/or bottom-up

Value Discovery Concepts

Before we start building products and features, it’s essential to understand **what** we want to build and **why**. Value discovery ensures that our time and effort go into creating meaningful, useful outcomes for our customers and users. It focuses on aligning product development with real needs, thereby maximizing return on investment and impact.

Product

A **product** is a solution that delivers value to users by solving a specific problem or fulfilling a particular need. It can be a digital product (mobile app or a web-based solution), a physical item (a wearable device), or even a service (consulting or counseling). In Agile, we treat the product as something that evolves over time through feedback and iteration, rather than being built in a single pass. A well-defined product has a clear user base, a unique value proposition, and measurable outcomes.

Product Vision, Goals and Roadmap

- **Product Vision**: A short, inspiring statement that defines the long-term purpose and aspiration for the product. It provides direction and motivation for the team. Example: “Help individuals manage their health and fitness with minimum time commitment.”
- **Goals**: Medium-term outcomes that contribute to the vision. These are specific, measurable, and time-bound objectives, like “Increase user conversion rate from X% to Y% by the end of <month>.”
- **Roadmap**: A high-level, time-oriented plan that outlines how the team will achieve the product goals. It includes major milestones, feature releases, and alignment with strategic priorities.

Product Backlog and Requirements Hierarchy

The Product Backlog (PBL) is an emergent, ordered list of what is needed to improve the product. It is the single source of work undertaken by the Scrum Team. The individual items in the PBL are called Product Backlog Items (PBI’s).

Most Agile teams use User Stories to define product feature requirements.

User Stories

User Stories are simple, structured descriptions of a feature from the perspective of the end user. They follow the format:

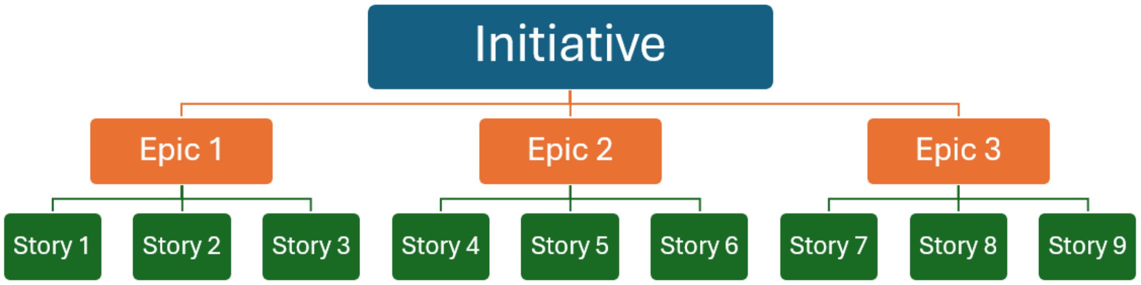
As a <user persona or role>, I want <feature/functionality> so that <value proposition>

For example... *As a high net-worth individual (HNI) I want to see a summary of my investments across different financial products so that I can balance/rebalance my portfolio regularly.*

Each story should include **Acceptance Criteria** – a set of conditions that must be true for the story to be considered complete.

A good user story is INVEST – *Independent, Negotiable (based on value and feasibility), Valuable (to the end user), Estimatable (by the team), Small enough (to be completed within a Sprint), Testable (validation for completion).*

While User Stories are sufficient for individual teams that work on small products, we often need a bigger requirement hierarchy for big products that need multiple teams to work together. The most common requirement hierarchy is Initiative => Epics => User Stories:



Epics/ Features

An Epic is a user facing functionality that is too big to be done by a single team in a single Sprint. In other words, an Epic will either take multiple Sprints to be completed or might require multiple teams to collaborate across one or more Sprints.

Epics are broken down into smaller user stories that may be implemented by one or more teams. Each story must be small enough to be completed by a single team within a single Sprint. While splitting an Epic into smaller User Stories, we should use **vertical slicing** – ensuring each slice delivers end-to-end value and can be validated independently.

<u>Epic Usage Guidelines</u>

i. Each Epic should aim to accomplish an end-to-end functionality, and should be defined using User Story format, with high level acceptance criteria.

ii. Epics must be testable - acceptance criteria will provide guidelines on how the Epic will be tested.
iii. Epics may be estimated based on their relative size, during or before the planning process.

Initiative

An **Initiative** is a strategic grouping of related epics that support a broader business objective or theme. Each initiative should be tied to a clear **business outcome**, such as improving user retention or entering a new market segment.

Initiatives can either be released as a whole or in parts, depending on dependencies and value delivery cadence. For initiatives that need to be released as a whole, we should aim to release them within 3 months or less, to ensure incremental value delivery and faster feedback. Large initiatives can be split into smaller, outcome-oriented sub-initiatives, each aiming to deliver a tangible business outcome.

Requirement Prioritization

Requirements prioritization is a 2-step process:

- Requirement selection (filtering) – to ensure we choose to build the right product and features.
- Prioritization (ordering) – to ensure we build them in an order that maximizes value delivery.

Requirement Selection Criteria

When evaluating potential requirements, consider:

- **Desirability** – Do users really want or need this?
- **Feasibility** – Can we build it with the time, technology, and skills available?
- **Viability** – Will this contribute to business success (e.g., ROI, market differentiation)?

Prioritization Criteria

When comparing and sequencing backlog items, consider:

- **Value** – What business or customer benefit will it deliver?

- **Cost** – How much effort, time, and resources do we require?
- **Risk** – What are the chances of failure and their possible impact?
- **Sequence/Order** – Are there dependencies or logical order among work items?

Prioritization Approaches

1. Eisenhower Matrix (Value vs Urgency)
This method segregates items based on:

- **High Value & High Urgency**: Do immediately
- **High Value & Low Urgency**: Schedule
- **Low Value & High Urgency**: Delegate or reassess
- **Low Value & Low Urgency**: Remove or defer

This helps teams focus on what truly matters instead of being reactive.

2. Kano Model
This approach categorizes features into:

- **Basic Needs** – Must-haves. Missing them causes dissatisfaction.
- **Performance Needs** – More is better. Directly correlated with satisfaction.
- **Delighters** – Unexpected features that delight users but aren't demanded.

The model helps balance effort between meeting expectations and surprising users.

3. Cost of Delay (CoD) & Weighted Shortest Job First (WSJF)
This method quantifies the **impact of delay** in delivering a feature:

- **Cost of Delay (CoD)** - User-business impact per unit of time delayed. Please note this is not the same as value of a feature; it is 'loss of value' with time.
- **WSJF** - CoD / Job Duration (or Effort)

Features with a **higher WSJF** score are prioritized to **maximize value delivered per unit of effort**.

4. MoSCoW Model
This is a qualitative prioritization approach:

- **Must have** – Critical for this release
- **Should have** – Important but not urgent

- **Could have** – Nice-to-have, if team capacity allows
- **Won’t have (now)** – Out of scope for the current release

This is useful in timeboxed environments like Sprints or quarterly releases.

Scrum Framework

Scrum is a lightweight framework that helps people, teams and organizations generate value through adaptive solutions for complex problems. It was developed in early 1990s for managing software product development and now is considered the most popular and most widely adopted framework to implement Agile.

At a very high level, Scrum can be broken down into the following components:

- Values – Commitment, Courage, Openness, Focus and Respect
- Sprint – A fixed timebox, one month or less, where team focuses on achieving a well-defined business goal.
- 3 Roles – Scrum Master, Product Owner, Developers (aka Dev Team)
- 5 events – Sprint, Sprint Planning, Daily Scrum, Sprint Review and Sprint Retrospective
- 3 Artifacts – Product Backlog, Sprint Backlog and Product Increment

Sprint-based Flow

The idea of Sprint is at the core of Scrum; it binds everything together. Let us try to understand the idea and the motivations behind it:

1. **Fixed timebox:** Sprint duration is a fixed timebox that is expected to stay the same from one sprint to another. While 2 weeks is the most common Sprint duration, a team must choose what works best for them (not more than a month).
2. **Sprint Goal:** The key idea of a timebox is to commit to a specific business goal and finish it before the Sprint ends. While the scope of work to achieve that goal may go through changes, the Sprint goal is expected to stay the same. Significant changes in Sprint goal during Sprint may call for a need to cancel the current Sprint and start a fresh one.
3. **Prioritize finishing:** A clear goal for a short, fixed timebox helps align entire team's efforts and brings a sense of urgency enabling faster completion of work items.
4. **Inspect and Adapt culture:** Sprint enforces a continuous improvement culture by providing multiple opportunities to inspect and adapt the product as well as the process.

Scrum Team

The fundamental unit of Scrum is a small team of people, a Scrum Team, that consists of one Scrum Master, one Product Owner, and Developers - it is a cohesive unit of professionals focused on one objective at a time, the Product Goal.

Scrum Teams are **cross-functional**, meaning the members have all the skills necessary to create value each Sprint.

They are also **self-managing**, meaning they internally decide who does what, when, and how. The Scrum Team is responsible for all product-related activities from stakeholder collaboration, verification, maintenance, operation, experimentation, research and development, and anything else that might be required. They are structured and empowered by the organization to manage their own work. Working in Sprints at a sustainable pace improves the Scrum Team's focus and consistency.

The entire Scrum Team is accountable for creating a valuable, useful Increment every Sprint.

Product Owner (PO)

The Product Owner is accountable for maximizing the value of the product resulting from the work of the Scrum Team.

Here are the key responsibilities of the Product Owner:

1. Develop and explicitly communicate the Product Goal and Roadmap
2. Work closely with stakeholders and manage their expectations
3. Manage Product Backlog:
    a. Create backlog items, with sufficient details
    b. Keep the Product Backlog items in priority order
    c. Ensure that the Product Backlog is transparent, visible and understood
4. Review completed work items and provide timely feedback
5. Maintain an open, positive and collaborative working relationship with Scrum Master and Developers
    a. Attend all Scrum events
    b. Be available to developers on priority basis to answer their questions/concerns

Scrum Master (SM)

The Scrum Master is accountable for the Scrum Team's effectiveness. They are expected to work as an enabler for the team – an enabler for sustainable pace, good quality and overall team growth.

1. Foster a positive and collaborative **team culture**:
    a. Educate/mentor team on Agile and Scrum practices

   b. Promote collaboration and collective work ownership
   c. Facilitate team meetings effectively - to harness collective intelligence
2. Enable and sustain high **team performance**:
   a. Assist team in removing impediments and unnecessary distractions
   b. Promote engineering practices to improve product quality
   c. Help team track their work – board, artefacts and metrics
   d. Promote continuous learning and improvement

Developers (Dev Team)

Developers include all team members who contribute in converting backlog items to product increment – UI designers, Architects, Developers, Testers, Business Analysts, DBA's, etc.

The Developers are responsible for:
1. Creating a plan for the Sprint, the Sprint Backlog
2. Completing work with good quality, by adhering to team's Definition of Done
3. Sharing information, knowledge and collaborating with each other
4. Adapting their plan each day toward the Sprint Goal
5. Holding each other accountable as professionals
6. Improving team processes through regular inspect and adapt

Scrum Values

1. **Commitment**
   Team members personally commit to achieving the goals of the Scrum Team. This includes commitment to quality, to the Sprint Goal, and to supporting each other. It fosters responsibility and ownership.
2. **Focus**
   With a clear Sprint Goal and a prioritized backlog, the team focuses on the most important work and avoids distractions. The timeboxed nature of Sprints encourages a sustained and sharp focus on delivering value.
3. **Openness**
   Scrum teams are open about their progress, challenges, and learning. They share feedback candidly during events like Sprint Reviews and Retrospectives. Openness builds trust and fosters continuous improvement.

4. **Respect**
   Scrum emphasizes mutual respect among team members. Everyone is valued for their unique perspectives and contributions. Respect enables better collaboration and reduces friction in decision-making.
5. **Courage**
   Scrum teams have the courage to take on tough challenges, speak up about impediments, and make necessary changes—even when uncomfortable. Courage supports innovation and adaptability in the face of uncertainty.

These values guide the behaviour and mindset of the team. When embraced fully, they help create a culture of trust, collaboration, and continuous learning—enabling Scrum to deliver its full potential.

Scrum in Action: Sprint Execution

In Scrum, value creation activities take place inside the 'Sprint' – each Sprint is an opportunity to create and deliver business value to stakeholders and customers. Below is the sequence of key activities in a Sprint (more details in the next section):

1. **Sprint Planning:** This is the first activity in the sprint where team defines a meaningful goal and plans necessary work (Sprint Backlog) to achieve the Sprint goal. The Sprint Backlog provides clarity on what needs to be done, by who, in roughly what order.
2. **Sprint Execution:** Soon after the planning, the team members start pulling individual tasks (one by one), with a focus on finishing stories in priority order. Once a work item is completed in all respects (coding, code review, testing, etc.), Product Owner reviews it and provides feedback to the team.
3. **Daily Scrum:** This is a daily event time-boxed to 15-minutes, usually held at the start of the day. Here, the team quickly inspects work done since last Daily Scrum and the overall progress towards the Sprint goal, and plans work for the next 24 hours.
4. **Sprint Review:** Held on the last day of the Sprint, here the Scrum team inspects the Product Increment with the stakeholders and adapts the Product Backlog if needed. This is an informal meeting to seek timely feedback from stakeholders and foster trust and collaboration.
5. **Sprint Retrospective:** This is an opportunity for the Scrum Team to continuously evolve their process by inspecting their experience in the current Sprint, learning from it and planning small improvements in the next one. If Sprint Review is an opportunity to review and enhance the Product, Retrospective is an opportunity to review and enhance the Team Process.

Backlog Refinement

Purpose: Help team develop a better understanding of prioritized business needs for the next 1-2 Sprints.

Participants: The whole Scrum team (PO, SM, Dev Team)	**Expected Duration:** 2 to 4 hours for a 2-week Sprint

Prerequisites:

i. Product roadmap for next 3-12 months
ii. Prioritized Product Backlog for 1-3 months
iii. User stories defined for next 1-2 Sprints - with acceptance criteria

Outcome:

Sufficient work items are “Ready” for execution (per team's Definition of Ready) for the upcoming Sprint:

i. Refined acceptance criteria: As agreed between PO and Dev Team
ii. Size estimates (relative): Story point estimation or some other approach

Agenda/ Process:

i. Big Picture: PO revisits the product roadmap, highlighting any recent updates since their previous discussion.
ii. Review and refine work items for next 1-2 Sprints, in priority order:
    a. PO shares details on work items one by one, with acceptance criteria
    b. Team reviews acceptance criteria and seeks clarifications where needed
    c. [Optional] Once the work item is well understood, team estimates the size of work item (relative size estimation)

Success Criteria:

i. Product Owner comes prepared to the session – acceptance criteria is defined for all work items.
ii. All team members are engaged in discussions, and each member speaks voluntarily multiple times.
iii. Team members seek clarifications on work items without hesitation.
iv. The difference in opinion about requirements or development approach is resolved through open discussion rather than authority.
v. The acceptance criteria are updated for multiple work items, right in front of the team (not later).
vi. Size estimates are decided based on whole team consensus:
    a. Size estimates are relative, not based on time estimates.
    b. Size estimates are not pushed or influenced by the PO, SM or specific individuals.
    c. Size estimates are not influenced by who will work on the work item.

Sprint Planning

Purpose: Define a valuable and achievable goal for the Sprint and plan work items to achieve the same.

Participants: The whole Scrum team (PO, SM, Dev Team)	**Expected Duration:** Up to 4 hours for a 2-week Sprint

Prerequisites:

i. There are enough "Ready" items prioritized for the Sprint at the top of the Product Backlog.
ii. Team has good understanding of the current state of Product Increment.
iii. Team members know how much capacity they have available in this Sprint.
iv. Previous Sprint has been closed in the ALM tool (Jira, ADO, etc.).

Outcome:

i. **Sprint Goal:** The theme/objective of this Sprint that binds (most of) the work items together.
ii. **Sprint Backlog:** The collection of work items and specific subtasks that will help Dev Team achieve the Sprint Goal.
iii. Updated Scrum Board: The team board in ALM tool has been updated to show the Sprint Backlog.

Agenda/ Process:

i. PO proposes the Sprint goal and shares the list of work items that align with that goal
ii. SM notes down the available team capacity - who is available for how much time.
iii. SM facilitates planning of Work Items - One by one, in order of priority:
    a. Team reviews if the work item is "Ready" to start development (per Definition of Ready).
    If the item is not ready, they review and refine to ensure it is ready. If there are challenges, they may decide to skip the item or put it on "Hold".
    b. Team discusses high-level design approach (open discussion)
    c. Once there is an agreement on approach, team identifies specific tasks that need to be done.
    d. Team members volunteer for tasks (Pull, not Push) and share their effort estimates (hours/days).
    e. Individual effort estimates are noted to revise available individual capacity (by individual or Scrum Master).
    f. If team has enough capacity left, continue to plan the next work item (repeat 'a' to 'e')
iv. Close the Sprint Planning
    a. Finalize Sprint Goal (based on items pulled into the Sprint)
    b. Review team confidence and address concerns, if any

c. Ensure all work items and subtasks are visible inside the ALM tool under the new Sprint
d. Start the Sprint in ALM tool

Success Criteria:

i. Readiness criteria (DoR) are checked before planning the work item.
ii. There is active participation from all team members during design discussions.
iii. Team members pull tasks voluntarily, without any push from others.
iv. Team members share their honest task estimates with the team, without adding extra buffer time.
v. Sprint forecast is based on team's available capacity and their confidence, rather than a goal/target set by PO, SM or the stakeholders.
vi. Team members display a sense of ownership about the Sprint Goal and Sprint Backlog.
vii. During confidence voting, most team members showed high confidence (> 80%) in achieving the Sprint goal.

Daily Scrum

Purpose: Inspect team progress towards Sprint Goal and adapt/plan next steps accordingly.

Participants: The whole Scrum Team, with Dev Team as mandatory participants.	**Expected Duration:** 15 min or less

Prerequisites:

i. The task board is up to date (reflects the correct status of all stories and subtasks)
ii. Team members are ready to share their progress/challenges/impediments, with relevant details

Outcome:

i. Updated team board, reflecting latest work status and assignments.
ii. Updated burnup and burndown charts.
iii. Updated impediment list and (possibly) action plan for each impediment.

Agenda/ Process:

The most common format: Standard three questions (it is okay to try different formats)

i. Ensure task board is up to date
ii. Each team member shares their updates with the whole team, answering three questions:
   a. What did I do yesterday that helped the Team progress towards the Sprint Goal?
   b. What will I do today to help the Team achieve the Sprint Goal?
   c. Do I see any impediments that could prevent me or the Team from achieving the Sprint Goal?

Success Criteria:

i. The whole team arrives on or before time.
ii. The board is up to date before the Daily Scrum starts.
iii. There is good energy - people are engaged/ tuned in:
    a. Members pay attention to other members' updates and concerns.
    b. There are multiple instances when members ask question, seek clarification from each other and/or offer help.
iv. Updates from team members indicate a sense of ownership, commitment and urgency.
v. The meeting is crisp and meaningful, and ends within the 15 min timebox.
vi. Long problem-solving conversations are taken offline right after Daily Scrum.
vii. The tone of the meeting is friendly and informal.

Sprint Review

Purpose: Inspect Product Increment and adapt the Product Backlog.

Participants: The whole Scrum Team, plus the stakeholders invited by PO.	**Expected Duration:** Up to 2 hours for a 2-week Sprint

Prerequisites:
i. Product Owner has reviewed Product backlog items (PBI's) and has agreed on the completion status with the team.
ii. Completed and approved Backlog items are ready for demonstration
iii. PO and Dev team have agreed on what PBI's will be presented, in what order and who will present them
iv. PO has invited and confirmed the presence of key stakeholders
v. PO has a prioritized list of work items expected to be picked up in the next sprint

Outcome:
i. Revised product increment based on review by stakeholder
ii. Revised product backlog with clarity on desired focus on the next sprint

Agenda/ Process:
i. PO Shares the sprint goal and work items picked up during the sprint
ii. PO shared high level summary of completed work in the sprint
iii. Dev Team demonstrates the completed items one by one
    a. Review the story and acceptance criteria
    b. Demo functionality
    c. Take feedback from stakeholders
iv. PO discusses readiness of the Product Increment for deployment (optional)
v. PO reviews the impact of product increment on the upcoming product release and the overall product goal/roadmap

Success Criteria:
i. PO sets the context by explaining how the work picked up for the Sprint aligns with the Product goal and the upcoming release ii. The demonstration of completed functionality is well articulated by team members - focusing on system capability and business value, not internal technical details iii. Stakeholders are engaged in the meeting and share their honest feedback iv. There is healthy and positive interaction among Product Owner, stakeholders and Dev Team members v. Feedback from stakeholders is used to update or reprioritize the Product Backlog. vi. Despite the presence of Stakeholders and other senior people, the tone of the meeting is informal, conversational, and inclusive. vii. Stakeholders are aligned with the Sprint Goal and share positive feedback about the progress made during Sprint.

Sprint Retrospective

Purpose: Inspect team's current process and discipline, and plan ways to improve overall team effectiveness and performance	
Participants: The whole Scrum Team (including Product Owner)	**Expected Duration:** 1 to 2 hours
Prerequisites: i. All work items for the sprint have been updated in the ALM tool. ii. The updated Sprint metrics are available and are reliable. iii. The team has updates on the progress on last Retrospective's action items.	
Outcome: A short list of process improvement action items with: - Owner - Expected due date	
Agenda/ Process: **i. Set Context:** SM reminds the team why we are here, sets a positive environment before engaging the team in deeper conversations. For a new team, sets the ground rules, such as: a. Share your honest thoughts, concerns and ideas b. Don't make it personal and don't take it personally c. Use respectful language and a positive tone d. Listen with an open mind; everyone's experience is valuable" **ii. Look Back:** How did the team perform?	

a. Opening up: How did the sprint go? Share rating on a scale of 5 and add thought to support
b. Data and Metrics: Bring objectivity to perspective; share Burnup, Velocity etc and gather team's thoughts on them
c. Previous Retrospective Items: Review the progress on retrospective items during the Sprint

iii. Generate Insight: Invite team members to share their opinions/ideas/insights:

a. Each team member writes up to 2-3 stickies (one idea per sticky) per section
b. Once all have shared their thoughts, all stickies on the wall are read out loud. Clarifications are addressed on need basis.

iv. Look Ahead: Engage the team in converging these ideas into actionable items

a. Group the ideas into themes
b. Select the most common theme/ area of concern via dot voting
c. Solutioning: Scrum Master helps the team discuss the top selected ideas/concerns and find action items – who needs to do what?

v. Wrapping up

a. SM shares the action plan with the team via email (or a shared location).
b. Some items may be added to the next Sprint backlog.

Success Criteria:

i. The entire Scrum team is present in the Retrospective, from start to finish.
ii. All team members are engaged in discussions - no one dominates the conversation, and no one stays silent throughout.
iii. It feels like a safe environment - Team members feel safe to speak honestly, including their mistakes and concerns.
iv. PO and SM participate as peers, prioritizing listening to team's inputs over pushing their own ideas.
v. Discussion includes a mix of perspectives - what went well, what didn’t, and what could be improved.
vi. The meeting is positive and productive - with a focus on solving problems rather than assigning blame.
vii. The team agrees on 2–5 clear action items - concrete, achievable, and within the team’s control.
viii. The meeting concluded with a short list of action items based on team consensus
ix. Format is varied over time - to add variety and to match team needs.

Succeeding with Scrum

Kanban Method

Quick introduction

When to use Kanban?

Core Practices

Meetings

Metrics

Scrum vs Kanban

Comparison

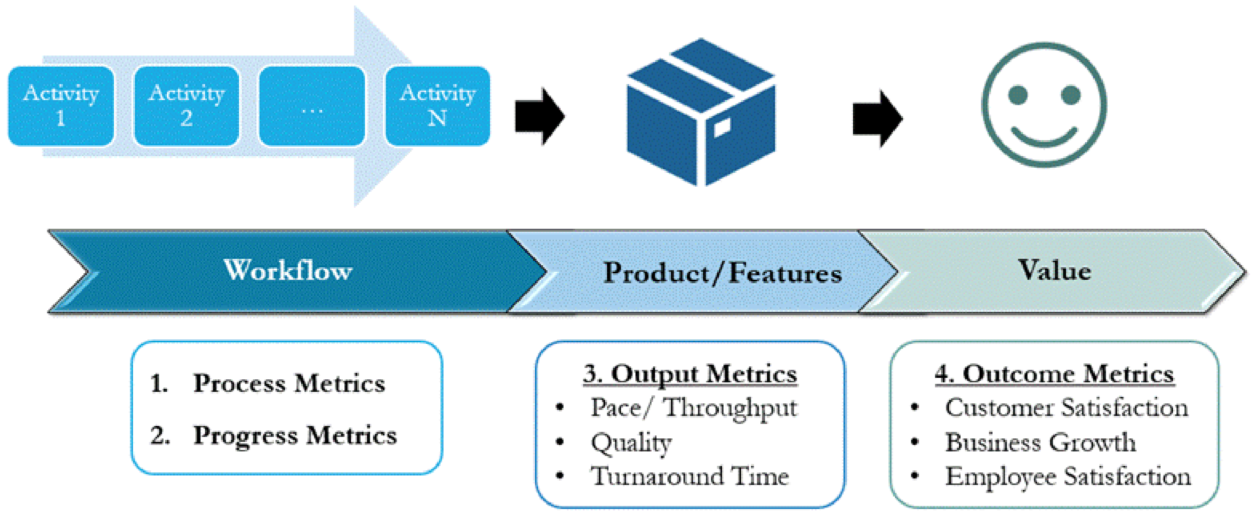
How to decide between Scrum and Kanban?

ScrumBan – how to combine Scrum and Kanban?

Succeeding with Kanban

Metrics and KPI's

Types of Metrics

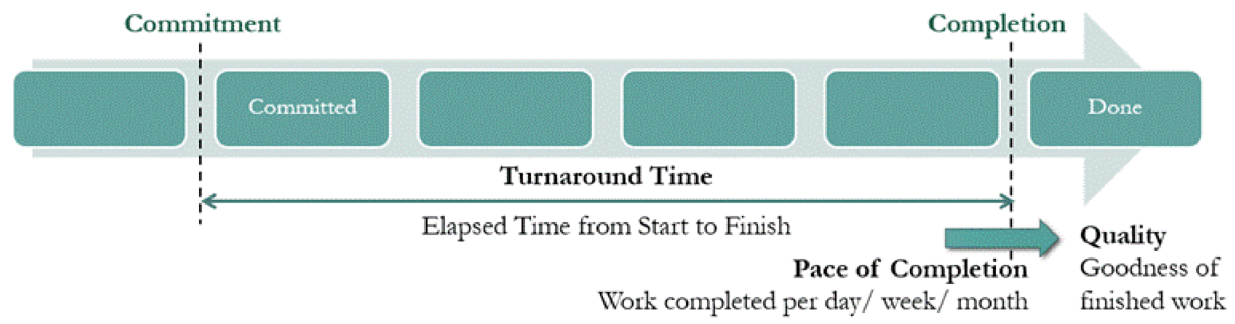


As depicted in the picture above, there are different categories of metrics relevant for end-to-end product development flow:

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:

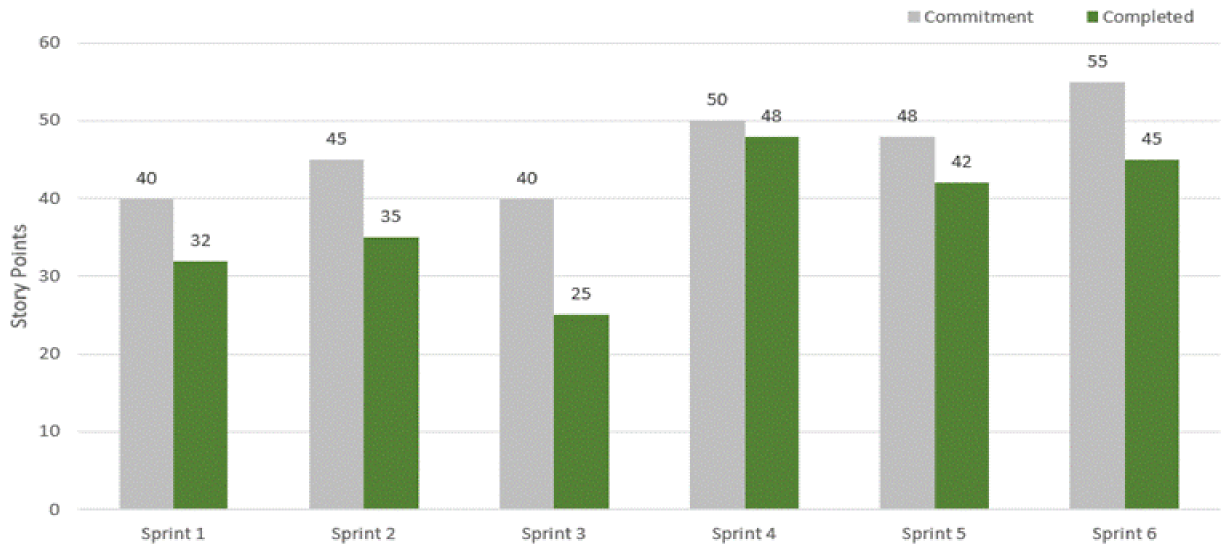
1. Pace of Completion - Measures 'How Much' work is completed in a specific time period (week, sprint, or month).
2. Quality – Measures 'How Well' the finished work looks, with respect to functional expectations and quality standards.
3. Turnaround Time – Measures 'How Fast' we finish work.

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:

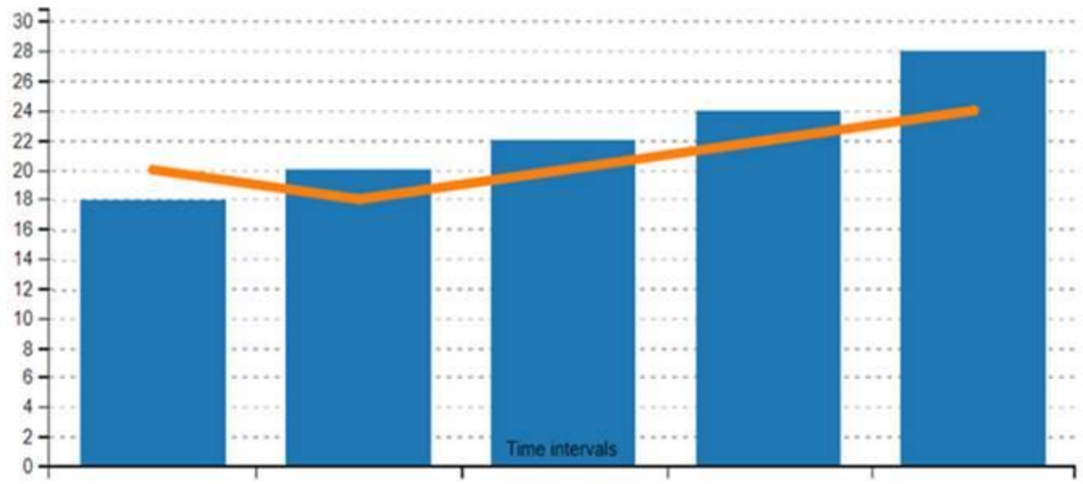
Interpretation Guidelines:

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.

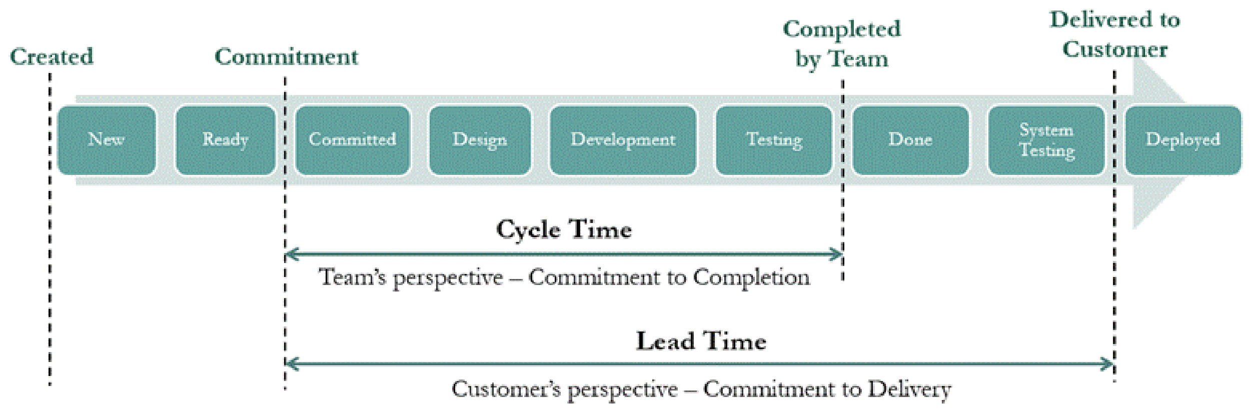
Interpretation Guidelines:

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:

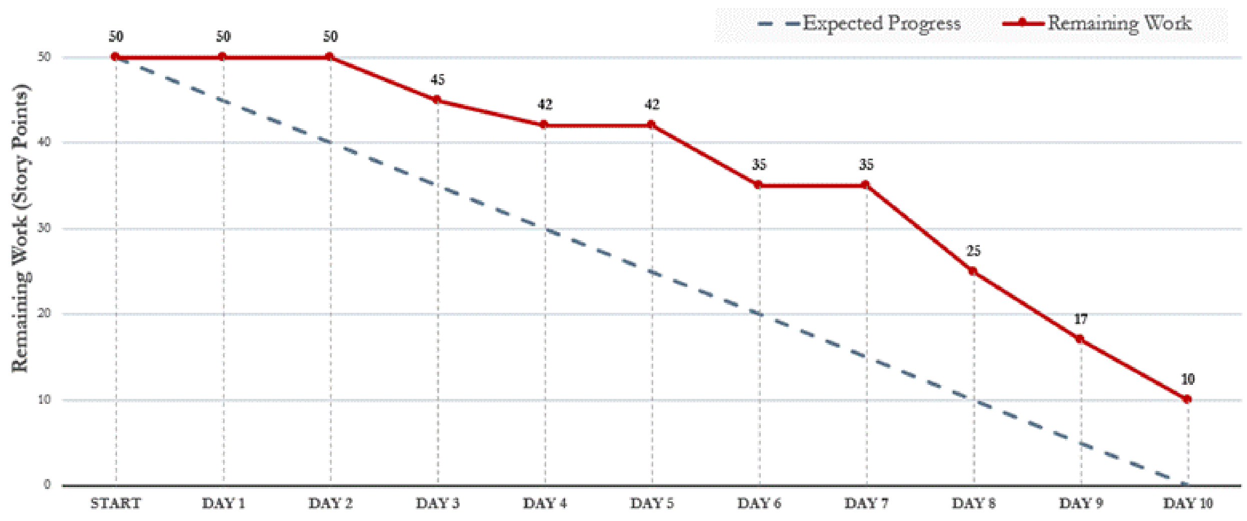
1. **Release confidence**
    a. Are we able to keep failure rate to low levels, thereby improving our release confidence?
2. **Comparison with Release Frequency**
    a. Is the change failure rate directly or inversely proportional to the release frequency?

Progress Metrics

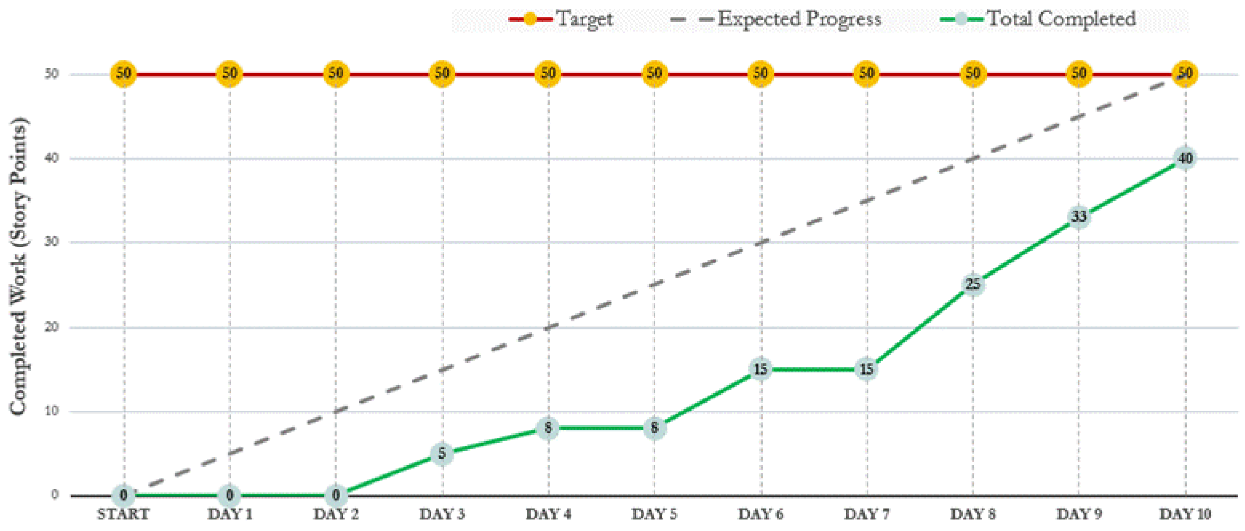
	Scrum Team	Kanban Team
Team Level	Sprint Burnup Sprint Burndown	Story-level Cumulative Flow Diagram (CFD)
Product Level	Release Burnup, Initiative Burnup Story CFD, Epic CFD	

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.

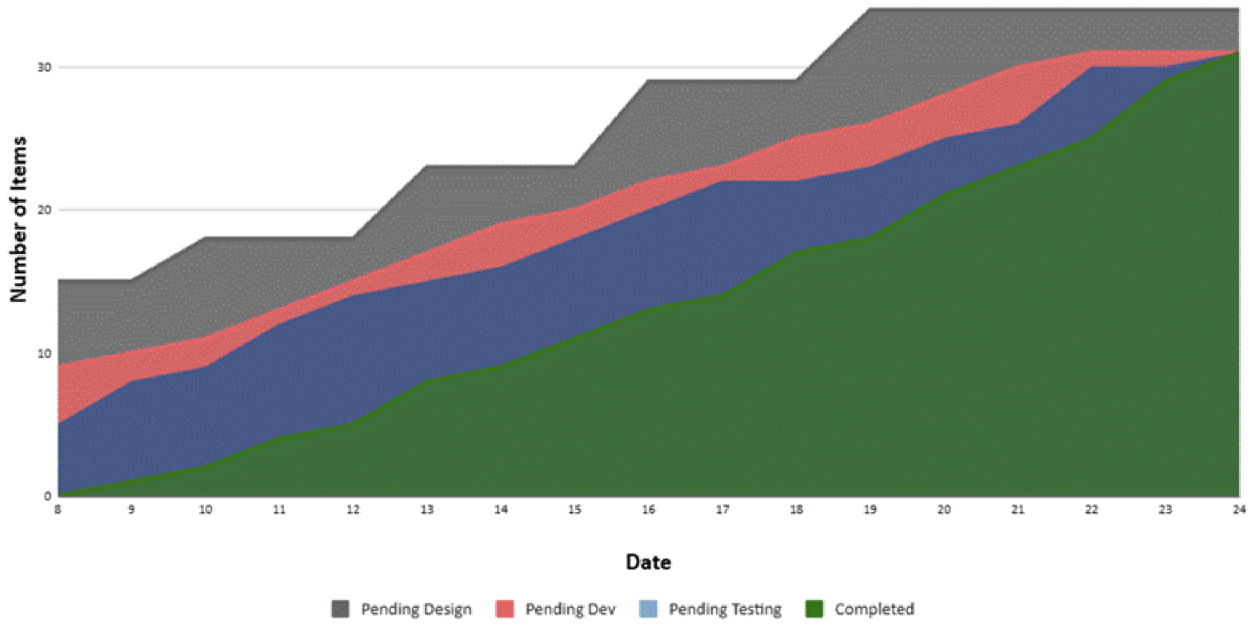


Interpretation Guidelines:

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.