AGILE PLAYBOOK

All the Basics of Agile for Everyone

Agile done right isn't complicated. This playbook covers the Scrum framework from ceremonies to estimation, written for teams at every experience level.

Contents

Introduction

This playbook is a practical guide to Agile software development. It covers how teams organize their work, run their ceremonies, define what "done" means, and improve sprint over sprint. Whether you are new to Agile or joining a team with an established practice, this playbook gives you a shared reference for how we work.

What is Agile?

Agile is a philosophy for developing software in a way that embraces change rather than resisting it. Instead of planning everything upfront and delivering at the end of a long cycle, Agile teams work in short iterations, deliver working software frequently, and adjust based on feedback.

The Agile philosophy is grounded in four core values, articulated in the Agile Manifesto (2001):

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

The Manifesto does not reject processes, documentation, contracts, or plans. It establishes what to prioritize when tradeoffs must be made.

Scrum

Scrum is a framework for putting Agile values into practice. It organizes work into fixed-length iterations called sprints, typically two weeks long. Each sprint produces working, potentially shippable software.

Scrum gives teams a clear set of roles, ceremonies, and artifacts. The roles define who does what. The ceremonies create a predictable rhythm. The artifacts make the work and its status visible to everyone. This playbook is built around the Scrum framework.

Kanban

Kanban is another widely used Agile framework, and it takes a different approach. Where Scrum organizes work into sprints with fixed commitments, Kanban uses a continuous flow model. Work items move through a board from start to finish as capacity allows, with no sprint boundaries and no sprint planning ceremony.

The key discipline in Kanban is limiting work in progress. By capping how many items can be in any given stage at once, teams expose bottlenecks and keep work moving rather than accumulating. Kanban is often a better fit for support, maintenance, or operational work where demand is unpredictable and incoming requests do not lend themselves to sprint-sized batches.

> **Scrum and Kanban are not mutually exclusive. Many teams borrow from both.**

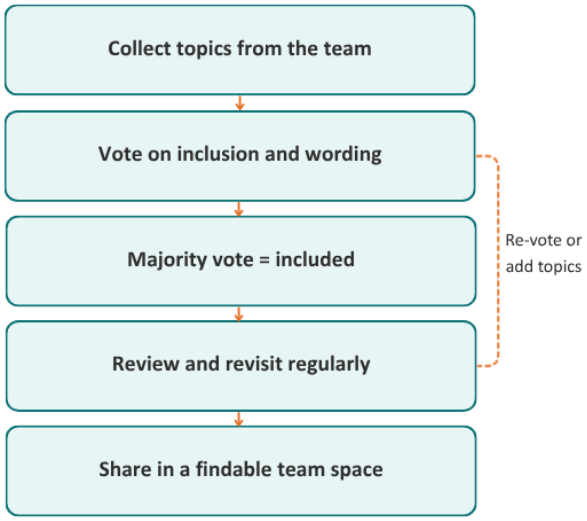
How to use this playbook

This playbook is organized into three parts. Part 1 covers the sprint ceremonies: the recurring events that give a sprint its structure. Part 2 covers the definitions that anchor how a team commits to and closes out work. Part 3 covers estimation. Part 4 covers work item types – the hierarchy from initiatives down to spikes.

Start with Part 1 if you are new to Scrum. Return to specific sections as questions come up in your day-to-day work.

Sprint Team Charter

The sprint team charter captures what the team agrees is necessary to work well together. Each member commits to it, which creates a shared foundation for how the team operates. Ideally, the charter is created during Sprint Zero, but it can be introduced or updated at any time with full team agreement. Charters are built from team input, not handed down.



Topics worth considering include the Definition of Ready, Definition of Done, team communication channels, sprint review roles (who facilitates, who takes notes), and guidelines for pulling in new stories while others are still in progress.

Scrum Roles

Scrum defines three roles. Each has a distinct set of responsibilities, and the health of the team depends on all three being well understood and respected.

Product Owner

The Product Owner is responsible for the what and the why. They define what gets built, in what order, and why it matters. The Product Owner owns the backlog: creating, prioritizing, and maintaining it so the team always has a clear, ordered view of upcoming work. They bring business context to refinement and planning, write or review acceptance criteria, and make decisions about scope.

> **The Product Owner is the voice of the customer and the business inside the team.**

Scrum Master

The Scrum Master is responsible for the how. They help the team understand and apply Scrum, remove impediments that block progress, and protect the team's ability to do its best work. The scrum master facilitates ceremonies, coaches the team on continuous improvement, and shields the team from outside interference.

> **The Scrum Master is not a manager or a project manager. They lead by serving.**

The Team

The Team is the group of people who do the work. In Scrum, the team is cross-functional: it includes everyone needed to design, build, test, and document a working increment of software. The team owns its commitment. It decides how much work to take on in a sprint, how to break stories into tasks, and how to get work done. No one outside the team assigns work to individuals during a sprint.

> **The Product Owner sets direction. The scrum master removes friction. The Team delivers.**

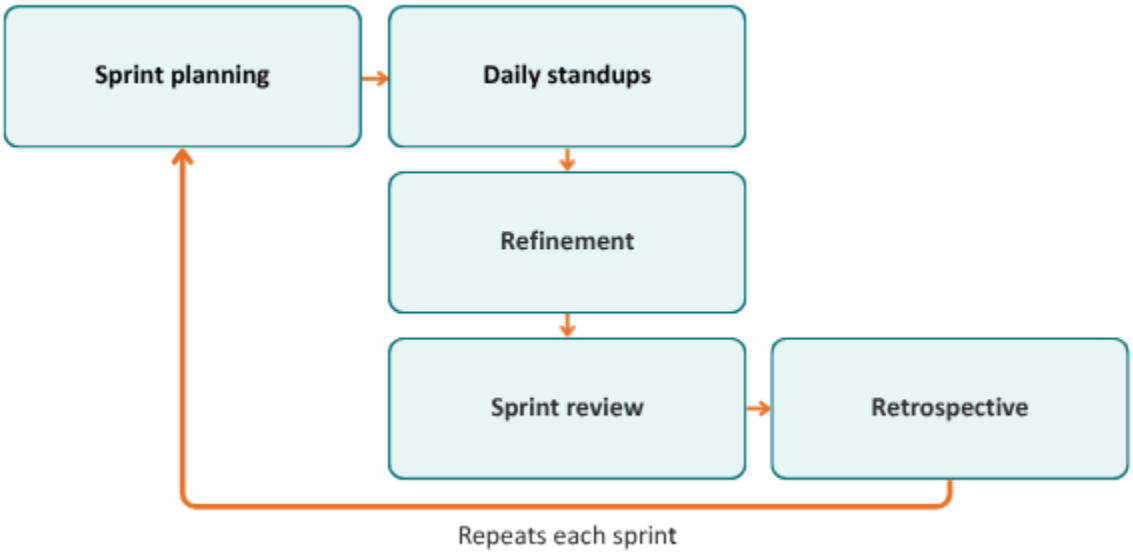
These three roles are complementary. When all three are working well together, the sprint runs itself.

Additional resources

- The Agile Manifesto: Is the Agile Manifesto still a thing?
- Scrum Guide: the authoritative reference for the Scrum framework
- Kanban Guide: a concise overview of the Kanban framework

Part 1: Sprint Ceremonies

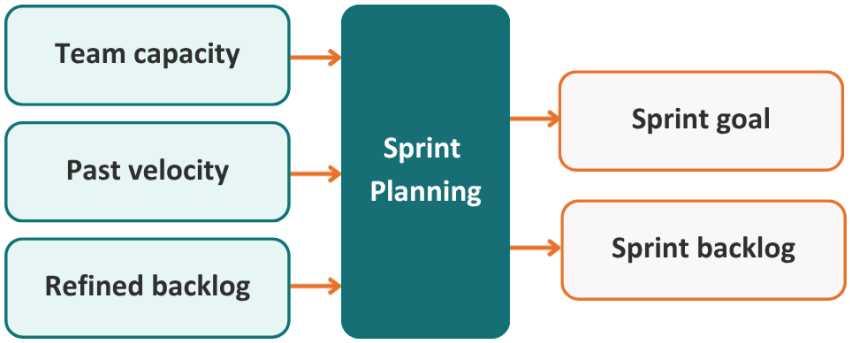
Sprint ceremonies are the recurring events that give a sprint its structure. Each one serves a distinct purpose in the lifecycle. The charter establishes how the team works together. Planning sets the commitment. Standups maintain daily alignment. Refinement keeps the backlog ready. The review closes the loop with stakeholders. The retrospective drives continuous improvement.



Done well, these ceremonies are not overhead. They are the engine of a healthy team.

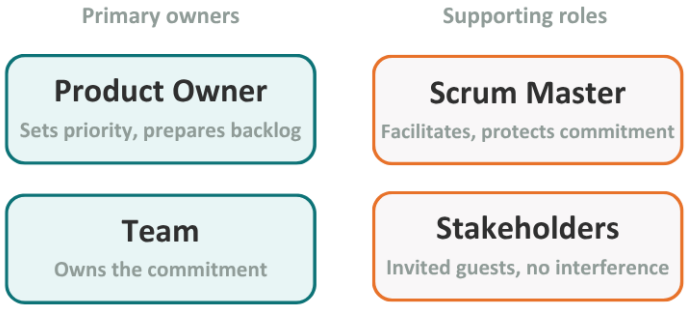
Sprint Planning

Sprint planning begins with the Product Owner presenting the highest-priority backlog items to the team. The team asks questions, works through the details, and makes a commitment for the sprint. When refinement has been done well, the team typically enters planning with more estimated stories than it can complete in a single sprint. This shifts planning from a discovery session into a capacity exercise: matching available work to available time and formalizing the commitment. The Product Owner should come prepared to discuss at least two sprints' worth of stories, or the top 10 priority items.



The two outputs of Sprint Planning are a sprint goal (a short statement describing the sprint objective) and a sprint backlog (the stories the team is committing to).

Before locking in the commitment, check in on team availability. Vacations, training, and other obligations affect capacity. Accounting for those gaps up front protects the team from overcommitting.



The team owns its commitment. The Product Owner and management set priorities. The team determines how much it can take on.

> **Use past velocity as a guide, not a mandate.**

Stories entering planning should be well defined and carry acceptance criteria. The team has standing to push back on anything vague or incomplete.

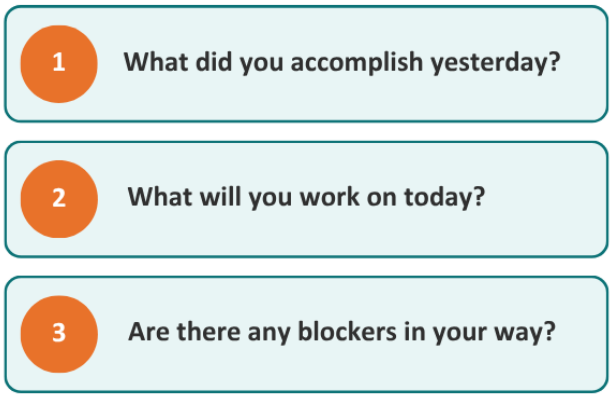
Take the time to get the sprint backlog right. Everyone should feel comfortable with what the team is signing up for before the meeting ends. At the end of the sprint, revisit the sprint goal during the review.

Additional Resources

- Sprint Planning Meeting: Mountain Goat Software
- Sprint Planning Meeting: Agile Alliance

Team Standups

The daily standup is built around coordination and shared commitment. It is a short, focused event for the entire team, with one output: a shared plan for the next 24 hours. Each person answers three questions:



Standups should run 15 minutes or less. Lengthy updates and deep dives on blockers belong in separate conversations. Meet in person when possible or have your camera on if the team is remote. Come with your three questions already thought through.

The standup belongs to the Team, not the Scrum Master.

If you are comfortable, make eye contact with teammates and give your update to them.

The Scrum Master participates the same as everyone else and does not need to run every standup.

Rotating that responsibility builds leadership across the team. Standups happen even when the Scrum Master is unavailable.

Give whoever is speaking your full attention. When blockers surface, note them and address them after the standup with only the people who need to be involved. Everyone else is free to go.
Managers are welcome as observers. They should hold questions and comments until after everyone has spoken. The standup is the team talking to itself, not reporting up the chain.

Additional Resources

- The Daily Standup: Scrum Alliance
- Daily Standups: Atlassian

Backlog Refinement

Backlog refinement prepares stories for sprint planning. The work typically includes adding new stories to epics, breaking larger work into sprint-sized pieces, and estimating effort.

Stories should be scoped to work the team can complete within a single sprint, with limitations, assumptions, acceptance criteria, and the definition of done clearly established before a story is considered ready.

The **Product Owner** should come to refinement with stories well understood and a draft of acceptance criteria in hand. The better prepared the backlog is going in, the more productive the session will be.

The Product Owner is responsible for clearly communicating what needs to be built and why. The team owns how it gets built.

Estimates made during refinement are not final until a story is accepted into a sprint. For significant risks or unknowns, assign action items and create spikes where appropriate to keep stories moving rather than stalling in an unresolved state.

Ask team members to review upcoming stories 24 hours before the session. Late-breaking stories happen. Try to minimize them but handle them without friction when they arise.

Some teams prefer to do refinement as part of planning. Both approaches are valid. Running multiple shorter refinement sessions across the sprint generally produces better results than a single long meeting. It allows for more discovery, more iteration, and less meeting fatigue.

The full Scrum Team should attend. Other key stakeholders may join occasionally.

Additional Resources

- Backlog Refinement: Agile Alliance

Sprint Review

The sprint review is held at the end of the sprint to showcase completed work and collect feedback. It is not a formal presentation, and it is not a sign-off meeting. The goal is conversation: the team, Product Owner, customers, and stakeholders working together to shape the direction of the backlog.

Keep it informal and encourage a conversational tone. A demo can be a useful tool for sparking discussion, but it is not required, and slide decks are not the point. The emphasis is on working software and honest dialogue, not polish. Actively seek feedback from everyone in the room.

> **An hour of prep is plenty. More than that signals the meeting is being treated as a performance rather than a conversation.**

The Product Owner should be an active participant, not just an observer. At minimum, the Product Owner should comment on velocity and give the team and stakeholders a realistic read on upcoming work. The review should not be the first time the Product Owner sees what the team built, and it is not the moment for formal acceptance.

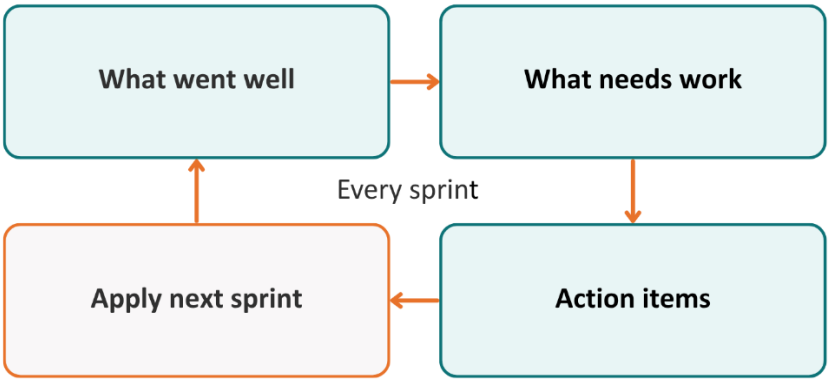
Product Owners benefit from attending reviews for teams other than their own. Cross-team visibility improves coordination and builds shared context across the organization.

Additional Resources

- Sprint Review Meeting: Mountain Goat Software

Sprint Retrospective

The retrospective is the team's dedicated time to examine how they work together. What is going well? What needs to change? The goal is a more effective team sprint over sprint.



A retrospective is a structured, psychologically safe meeting for surfacing observations and generating action items. It is the team looking honestly at its own process.

A retrospective is not a forum for blame, venting, or solving problems on the spot. Issues raised become action items to work on between sprints, not rabbit holes to disappear into during the meeting.

Retrospective Tips

Creating the right environment is the facilitator's first responsibility. People share honestly when they feel safe, and that safety is built deliberately over time. When negativity surfaces, redirect it without dismissing it. The focus is on where the team is going, not where it has been.

Come prepared. Bringing notes and observations from the sprint makes for a richer conversation. Revisit action items from previous retrospectives to track what changed and what did not. When something goes wrong during a sprint, the conversation should be about systems and processes, not individuals.

Use icebreakers when new teammates join. Getting comfortable early pays off in the meeting and beyond. Stay focused: the retrospective produces action items, not in-the-room solutions. Tangents that turn into problem-solving sessions belong elsewhere.

Do not skip retrospectives when things seem fine. **Continuous improvement is most valuable as a habit, not a reaction.** Hold the retrospective immediately after the sprint review, or as soon as possible after. Rotate the facilitator, the format, and where possible, the location. A periodic cross-team retrospective can surface shared improvement opportunities the same way a Scrum of Scrums surfaces cross-team blockers.

Additional Resources

- Retromat: a tool for finding and mixing retrospective formats
- Fun Retrospectives: similar, with a large format library
- Retrospectives: Agile Alliance

Part 2: Acceptance Criteria, Definition of Ready, & Definition of Done

Three definitions anchor how a team commits to and closes out work. Getting these right is one of the most valuable investments a team can make.

- **Acceptance Criteria (AC):** a set of requirements the Product Owner defines for a specific story. Each criterion has a clear pass/fail result. When all criteria are met, the story is done.
- **Definition of Ready (DoR):** a checklist the team uses to decide whether a story is ready to enter a sprint. If a story does not meet the Definition of Ready, it needs more refinement.
- **Definition of Done (DoD):** a checklist the team uses to verify that a story is truly complete. The simplest DoD is "acceptance criteria met," but a strong DoD also captures the good practices the team holds itself to regardless of the story.

These three definitions work together.

> **The AC defines what success looks like. The DoR confirms the team can start. The DoD confirms the team can close.**

Teams should not expect a single DoR or DoD to work across all teams. Each team should tune its own as part of its Agile process. A shared starting point is reasonable, but the team owns its definitions and should revisit them regularly.

Acceptance Criteria

Acceptance criteria are the specific conditions a story must meet to be considered complete. They are written by the Product Owner before or during refinement, and agreed upon by the full team before the story enters a sprint.

Think of acceptance criteria as the contract between the Product Owner and the team. The Product Owner defines what success looks like. The team builds to that definition. When all criteria are met, the story is done. Not when someone feels good about it, not when the code is merged, but when every criterion passes.

Good acceptance criteria share three qualities: written in plain language the whole team can understand, unambiguous about what is acceptable and what is not, and testable. Each criterion can be directly translated into one or more manual or automated test cases.

> **There is no partial acceptance. Either a criterion is met or it is not. A story that meets eight of ten criteria is not done.**

Acceptance criteria also protect the team from scope creep. When a stakeholder asks for something outside the original criteria, the team has a clear, agreed-upon reference for what was and wasn't committed to.

Definition of Ready (DoR)

Before a story enters a sprint, the team needs confidence it can be completed. The **INVEST** criteria give the team a shared language for making that call.

Independent: not dependent on incomplete work

- All dependencies are satisfied and no blockers exist
- All necessary resources are available
- All assumptions are clearly identified

Negotiable: not a fixed contract; open to revision until it enters a sprint

- Acceptance criteria are explicit in the description
- The team has reviewed and agreed on the acceptance criteria

Valuable: delivers something meaningful to the user or business

- The what and why are clearly stated in the description
- The team understands the value being delivered

Estimable: the team can assess its complexity with reasonable confidence

- Known risks have been identified and discussed
- The team has reached agreement on the complexity of the story

Small: fits within a single sprint

- Subtasks have been identified
- The team agrees the story is completable within a sprint

Testable: enough information exists to write test cases

- Enough information exists for QA to begin test preparation at the start of the sprint
- Any migration, installation, upgrade, or integration impacts are identified in the description

Acceptance criteria define the boundaries of a story and determine when it is complete. They must be expressed clearly, in plain language, without ambiguity about what is acceptable and what is not.

Each criterion must be testable: easily translated into one or more manual or automated test cases. There is no partial acceptance: either a criterion is met or it is not.

Definition of Done (DoD)

The Definition of Done is the team's shared standard for what "complete" actually means. Without it, done is whatever each individual thinks it is, which is a reliable path to half-finished work slipping into a release.

The DoD applies to every story and every sprint, without exception. It is not a per-story negotiation. It is the floor the team holds itself to regardless of time pressure, sprint goals, or stakeholder enthusiasm.

> **A story is not done when the code is written. It is done when every item on the DoD checklist is satisfied. The Product Owner confirms this before closing the story.**

The DoD should be created by the team, owned by the team, and revisited regularly. As the team matures, the DoD tends to get stronger, more thorough, and more precise. That is a sign of a healthy team, not an overly cautious one.

The checklist below is a starting point. Your team should adapt it to fit your stack, your workflow, and your definition of quality.

- Acceptance criteria met
- Code is properly formatted and passes all style checks
- All relevant unit and integration tests written and passing
- Code merged into the main branch
- Test coverage maintained at or above the team's agreed threshold
- QA has validated the feature against acceptance criteria
- Any UX changes validated by design
- User-facing documentation written, reviewed, and published
- API or technical documentation written and reviewed

Additional Resources

- Definition of Done vs. Acceptance Criteria: A complete guide
- Definition of Done, Definition of Ready and Acceptance Criteria are not the same darn thing
- Definition of Done vs. Definition of Ready: Scrum Alliance

Part 3: Estimating

Estimation is the process of assessing the expected effort to complete a story. Good estimates help the team make realistic sprint commitments and build more accurate roadmaps over time.

The entire team estimates, not just developers. Every team member, regardless of role, participates in estimating every story.

Estimation generates discussion, and that discussion surfaces assumptions, risks, and gaps that might otherwise go unnoticed until mid-sprint. A story estimate reflects the work of all team members, not just the people writing code.

Stories are estimated in story points. Story points measure size, which is an intentionally ambiguous term that folds in complexity, effort, and risk. **Story points are relative, not absolute. They mean nothing outside the context of the team that created them.**

A team's velocity is not comparable to another team's velocity.

How to Estimate

Estimation works best as a team ritual, not a solo exercise. When everyone estimates together, the process becomes a forcing function for shared understanding. Stories that seem obvious to one person often reveal hidden complexity when the whole team weighs in.

Baseline Story

All estimates are relative to a baseline story: a simple, well-understood piece of work the team agrees represents a known level of effort. The team assigns that story a point value, typically 2 or 3 points for something small and clear. Every subsequent estimate answers one question: how does this story compare to the baseline? Is it larger or smaller, and by how much?

The baseline story may need to change as the team's work evolves. Revisit it when the team's typical work shifts significantly in nature or complexity.

Scale

Teams should use a non-linear scale. The Fibonacci-based sequence works well: 0, 1, 2, 3, 5, 8, 13, 20, 40, 100.

The gaps between higher values are intentional. A story estimated at 40 points carries far more uncertainty than one estimated at 5. The scale reflects that reality - at high point values, the team does not have the precision to distinguish a 38 from a 40, so the scale does not ask them to.

When a story lands above 13, it is usually a signal to break it down further rather than carry a large, uncertain piece of work into a sprint.

Avoid linear scales like 1, 2, 3, 4, 5. They imply a precision that does not exist in estimation and mask the difference between well-understood work and complex, risky work.

Planning Poker

Estimation happens during refinement using Planning Poker. The process uses simultaneous reveal: everyone votes at the same time, which prevents anchoring bias and "following the loudest voice in the room." Here is how it works:

1. The Product Owner or Scrum Master presents a story.
2. Each team member privately selects an estimate.
3. Everyone reveals their estimate at the same time.
4. If the team reaches consensus, the story is assigned that value.
5. If estimates diverge, the team members with the highest and lowest values explain their reasoning. The discussion surfaces what each person knows or assumes about the story. The team votes again after discussion until consensus is reached.

The value of Planning Poker is not the number. It is the conversation that divergent estimates produce.

A developer who estimates 2 and a QA engineer who estimates 13 are not both wrong. They are looking at the same story from different angles, and both perspectives belong in the estimate.

Here is an example of that dynamic in practice: a developer estimates a story at 2 points because the core logic is already partially built. A QA engineer estimates 13 because the feature touches three integrated systems and the test matrix is substantial. The resulting conversation reveals scope neither had fully considered. The team lands on 8 and creates a spike to resolve the integration uncertainty before the story enters a sprint.

Bugs and Estimation

Bugs are timeboxed rather than estimated with story points. The uncertainty involved in bug resolution is typically too high for story points to be useful. Estimates will almost always be wrong, and wrong estimates damage sprint planning.

Instead, the team allocates a timebox to work on the bug during the sprint. If the timebox ends without resolution, a new timebox is set for the following sprint and the bug remains open. The debugging work done in the first timebox informs a better timebox estimate for the next sprint. Repeat until the bug is fixed.

Tools

Several digital Planning Poker tools are available that integrate with common project management platforms or run as standalone apps. When evaluating tools, look for simultaneous reveal, support for Fibonacci-based scales, and easy session management. The specific tool matters less than consistent use - pick one the team will use and stick with it.

Part 4: Work Item Types

Agile teams organize work into a hierarchy of item types. Understanding how they relate to each other helps everyone on the team communicate clearly and keep work at the right level of detail.

Initiatives

An initiative is the highest level of the hierarchy. It represents a broad organizational goal that typically spans multiple teams and multiple quarters. Initiatives are not sprint work — they are the strategic objectives that sprint work ultimately serves. Epics roll up to initiatives.

Epics

An epic is a large body of work that cannot be completed in a single sprint. It represents a significant feature, capability, or improvement that gets broken down into smaller, sprint-sized stories. Epics typically take several sprints to complete. They give the team a shared sense of what they are working toward across multiple cycles.

Stories

A story, short for user story, is the primary unit of work for a sprint team. Stories are written from the perspective of the person who will benefit from the work, and they should be small enough for the team to complete within a single sprint. Each story should have a clear goal, defined acceptance criteria, and enough detail for the team to estimate and execute it. Stories are the building blocks of epics.

Subtasks

Subtasks break a story into the individual work items needed to complete it. They give team members a way to divide and track the specific pieces of work within a story. A story is not complete until its subtasks are closed.

Bugs

A bug is a defect found in previously completed work. Bugs are tracked separately from stories and are not estimated with story points. Instead, they are timeboxed. See Part 3 for detail on how bugs are handled in estimation.

Spikes

A spike is a short, timeboxed investigation used to resolve uncertainty before a story can be properly estimated or started. If the team encounters a question they cannot answer without doing some exploratory work first, a spike is the right tool. Spikes produce an answer or a recommendation, not shippable software.

Glossary

Acceptance Criteria (AC): The specific conditions a story must meet to be considered complete. Defined by the Product Owner. Each criterion has a clear pass/fail result.

Backlog: The prioritized list of work waiting to be done. The Product Owner owns and maintains it.

Baseline Story: A simple, well-understood story the team uses as a reference point for all other estimates.

Bug: A defect found in previously completed work. Bugs are timeboxed rather than estimated with story points.

Definition of Done (DoD): A checklist the team uses to confirm a story is truly complete.

Definition of Ready (DoR): A checklist the team uses to confirm a story is ready to enter a sprint.

Epic: A large body of work broken down into multiple stories. Typically takes several sprints to complete.

Initiative: A broad organizational goal that multiple epics contribute to. Usually spans multiple teams and quarters.

INVEST: A set of criteria for a well-formed user story: Independent, Negotiable, Valuable, Estimable, Small, Testable.

Planning Poker: An estimation technique that uses simultaneous reveal to prevent anchoring bias and generate discussion.

Product Owner: The team member responsible for the backlog, priorities, and acceptance criteria.

Scrum Master: The team member responsible for facilitating Scrum, removing impediments, and coaching the team.

Spike: A short, timeboxed investigation used to resolve uncertainty before a story can be estimated or started.

Sprint: A fixed-length iteration, typically two weeks, during which the team commits to and delivers a set of stories.

Sprint Backlog: The set of stories the team commits to completing during a sprint.

Sprint Goal: A short statement describing the objective of a sprint.

Sprint Zero: A setup sprint used to establish team agreements, tools, and processes before regular sprint work begins.

Story (User Story): The primary unit of sprint work. Written from the perspective of the person who benefits from it. Should be completable within a single sprint.

Story Points: A relative measure of the size and complexity of a story. Meaningful only within the team that defined them.

Subtask: An individual work item that breaks a story into smaller, trackable pieces.

Timebox: A fixed period allocated to a piece of work. When the timebox ends, the work stops regardless of completion status.

Velocity: The amount of work a team completes in a sprint, measured in story points. Used as a guide for future sprint planning, not a performance target.