Learning AGILE

SOFTWARE DEVELOPMENT METHODOLOGY

Raj Vatnani

IT professional and enthusiast

eMensches
developing technology, changing lives

Contact: +91.94523.80498 | Web: www.emensches.com | Email: tasneembunglowala@yahoo.com

Learning Agile

1. What is Agile?

Agile is an iterative approach to project management and software development that helps teams deliver value to their customers faster and with fewer headaches. Instead of betting everything on a "big bang" launch, an agile team delivers work in small, but consumable, increments. Requirements, plans, and results are evaluated continuously so teams have a natural mechanism for responding to change quickly.

Whereas the traditional "waterfall" approach has one discipline contribute to the project, then "throw it over the wall" to the next contributor, agile calls for collaborative cross-functional teams. Open communication, collaboration, adaptation, and trust amongst team members are at the heart of agile. Although the project lead or product owner typically prioritizes the work to be delivered, the team takes the lead on deciding how the work will get done, self-organizing around granular tasks and assignments.

Agile isn't defined by a set of ceremonies or specific development techniques. Rather, agile is a group of methodologies that demonstrate a commitment to tight feedback cycles and continuous improvement.

2. Why choose Agile?

Teams choose agile so they can respond to changes in the marketplace or feedback from customers quickly without derailing a year's worth of plans. "Just enough" planning and shipping in small, frequent increments lets your team gather feedback on each change and integrate it into future plans at minimal cost.

But it's not just a numbers game—first and foremost, it's about people. As described by the Agile Manifesto, authentic human interactions are more important than rigid processes. Collaborating with customers and teammates is more important than predefined arrangements. And delivering a working solution to the customer's problem is more important than hyper-detailed documentation.

An agile team unites under a shared vision, then brings it to life the way they know is best. Each team sets their own standards for quality, usability, and completeness. Their "definition of done" then informs how fast they'll churn the work out. Although it can be scary at first, company leaders find that when they put their trust in an agile team, that team feels a greater sense of ownership and rises to meet (or exceed) management's expectations.

3. Agile yesterday, today, and tomorrow

The publication of the Agile Manifesto in 2001 marks the birth of agile as a methodology. Since then, many agile frameworks have emerged such as Scrum, Kanban, Lean, and Extreme Programming (XP). Each embodies the core principles of frequent iteration, continuous learning, and high quality in its own way. Scrum and XP are favored by software development teams, while Kanban is a darling among service-oriented teams like IT or human resources.
Today, many agile teams combine practices from a few different frameworks, spiced up with practices unique to the team. Some call this heresy. We call it practical. It's not about "Agile" – it's about agility.

The agile teams of tomorrow will value their own effectiveness over adherence to doctrine. Openness, trust, and autonomy are emerging as the cultural currency for companies who want to attract the best people and get the most out of them. Such companies are already proving that practices can vary across teams, as long as they're guided by the right principles.

4. Running agile programs

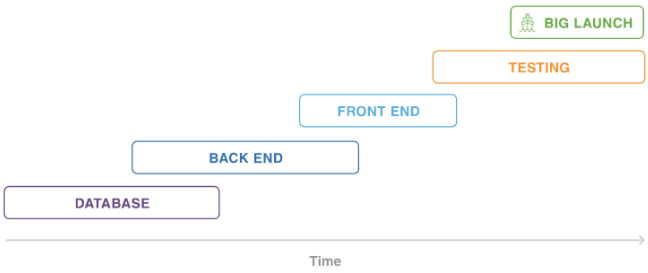
Early adopters of agile development were small, self-contained teams working on small, self-contained projects. They proved the agile model can work, to the joy and betterment of software makers around the world. More recently, larger organizations are scaling agile beyond single teams or projects, and seeking ways to apply it to whole programs.

This is not without its challenges. But that doesn't mean it can't be done!

Waterfall versus agile

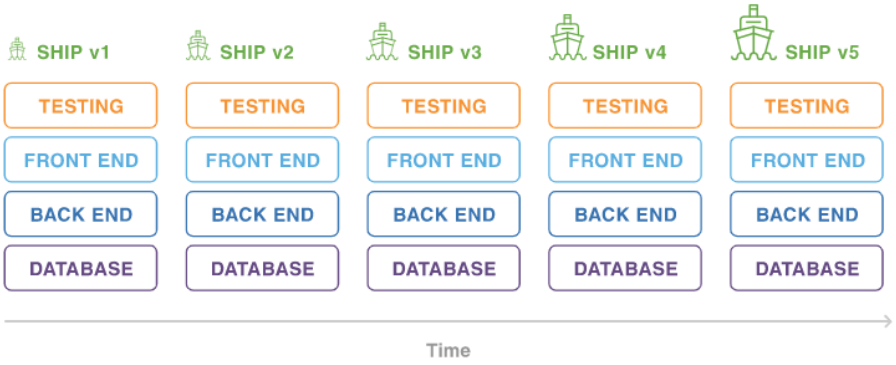
Let's start with the basics–like what makes agile different.

Traditional project management styles, like waterfall, build in phases. Below is an illustration of a standard waterfall project. This style of product development folds everything into a single, "big bang", high-risk release. Once a project passes one phase, it's painful to revisit it because teams are always pressing forward to the next stage.



Traditional project management styles often create "critical paths", where the project can't move forward until a blocking issue is resolved. To add insult to injury, the end customer can't interact with the product until it's fully complete. Thus, important issues in the product's design, and code, go undiscovered until release.

Let's contrast that with an agile project management style, which takes an iterative approach to development with regular feedback intervals. These iterations allow for the team to be diverted to (and productive in) another area of the project while a blocking issue is resolved.



Besides removing critical paths, iterations let you interact with the product during development.

This, in turn, gives the team constant opportunities to build, deliver, learn, and adjust. Market changes don't catch you flat-footed, and teams are prepared to adapt quickly to new requirements.

An even greater benefit is shared skill sets among the software team. The team's overlapping skill sets add flexibility to the work in all parts of the team's code base. This way, work and time isn't wasted if the project direction changes. (For more on that, see our article on building great agile teams.)

How to build a great agile program

When a program transitions from traditional project management to agile, the team and the stakeholders must embrace two important concepts:

The product owner's focus is to optimize the value of the development team's output. The development team relies on the product owner prioritizing the most important work first.

The development team can only accept work as it has capacity for it. The product owner doesn't push work to the team or commit them to arbitrary deadlines. The development team pulls work from the program's backlog as it can accept new work.

Let's explore the mechanisms agile programs use to organize, run, and structure work in an iterative way.

Roadmaps

A roadmap outlines how a product or solution develops over time. Roadmaps are composed of initiatives, which are large areas of functionality, and include timelines that communicate when a feature will be available. As the program develops, it's accepted that the roadmap will change–sometimes subtly, sometimes broadly. The goal is to keep the roadmap focused on current market conditions and long-term goals.

Requirements

Each initiative in the roadmap breaks down into a set of requirements. Agile requirements are lightweight descriptions of required functionality, rather than the 100-page documents associated with traditional projects. They evolve over time and capitalize on the team's shared understanding of the customer and the desired product. Agile requirements remain lean while everyone on the team develops a shared understanding via ongoing conversation and collaboration. Only when implementation is about to begin are they are fleshed out with full details.

Backlog

The backlog sets the priorities for the agile program. The team includes all work items in the backlog: new features, bugs, enhancements, technical or architectural tasks, etc. The product owner prioritizes the work on the backlog for the engineering team. The development team then uses the prioritized backlog as its single source of truth for what work needs to be done.

Agile delivery vehicles

Agile can be implemented using various frameworks (like scrum and kanban) to deliver software. Scrum teams use sprints to guide development, and kanban teams often work without fixed work intervals. Both frameworks, however, use large delivery vehicles like epics and versions to structure development for a synchronized release cadence out to production.

Agile metrics

Agile teams thrive on metrics. Work in progress (WIP) limits keep the team, and the business, focused on delivering the highest priority work. Graphs like burndown and control charts help the team predict their delivery cadence, and continuous flow diagrams help identify bottlenecks. These metrics and artefacts keep everyone focused on the big goals and boost confidence in the team's ability to deliver future work.

Agile runs on trust

An agile program cannot function without a high level of trust amongst team members. It requires candour to have difficult conversations regarding what's right for the program and the product. Because conversations happen at regular intervals, ideas and concerns are regularly expressed. That means team members also have to be confident in each other's ability (and willingness) to execute on the decisions made during those conversations.

In summary, agile development is a structured and iterative approach to making software. It gives you the ability to respond to change without going off the rails. And that's good news for any program.

5. Scrum Theory

Scrum is founded on empirical process control theory, or empiricism. Empiricism asserts that knowledge comes from experience and making decisions based on what is known. Scrum employs an iterative, incremental approach to optimize predictability and control risk. Three pillars uphold every implementation of empirical process control: transparency, inspection, and adaptation.

Transparency

Significant aspects of the process must be visible to those responsible for the outcome. Transparency requires those aspects be defined by a common standard so observers share a common understanding of what is being seen.

For example:

- A common language referring to the process must be shared by all participants; and,
- Those performing the work and those inspecting the resulting increment must share a common definition of "Done".

Inspection

Scrum users must frequently inspect Scrum artefacts and progress toward a Sprint Goal to detect undesirable variances. Their inspection should not be so frequent that inspection gets in the way of the work. Inspections are most beneficial when diligently performed by skilled inspectors at the point of work.

Adaptation

If an inspector determines that one or more aspects of a process deviate outside acceptable limits, and that the resulting product will be unacceptable, the process or the material being processed must be adjusted. An adjustment must be made as soon as possible to minimize further deviation.

6. Agile manifesto

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

7. Principles behind the Agile Manifesto

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity--the art of maximizing the amount of work not done--is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

8. Scrum

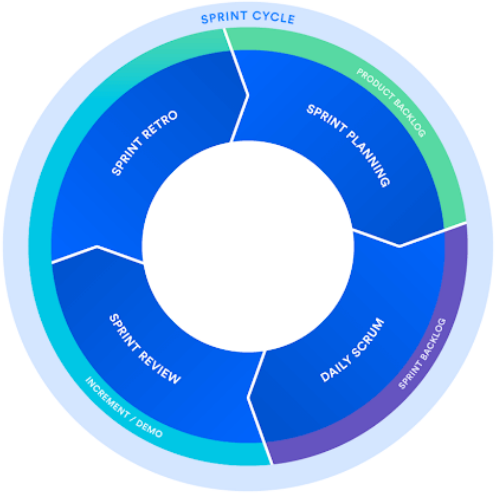
What is Scrum?

Scrum is a framework that helps teams work together. Much like a rugby team (where it gets its name) training for the big game, Scrum encourages teams to learn through experiences, self-organize while working on a problem, and reflect on their wins and losses to continuously improve.

While the Scrum I'm talking about is most frequently used by software development teams, its principles and lessons can be applied to all kinds of teamwork. This is one of the reasons Scrum is so popular. Often thought of as an agile project management framework, Scrum describes a set of meetings, tools, and roles that work in concert to help teams structure and manage their work.

The framework

People often think Scrum and agile are the same thing because Scrum is centered around continuous improvement, which is a core principle of agile. However, Scrum is a framework for getting work done, where agile is a mindset. You can't really "go agile", as it takes dedication from the whole team to change the way they think about delivering value to your customers. But you can use a framework like Scrum to help you start thinking that way and to practice building agile principles into your everyday communication and work.
The scrum framework is heuristic; it's based on continuous learning and adjustment to fluctuating factors. It acknowledges that the team doesn't know everything at the start of a project and will evolve through experience. Scrum is structured to help teams naturally adapt to changing conditions and user requirements, with re-prioritization built into the process and short release cycles so your team can constantly learn and improve.



9. Kanban

Kanban is a popular framework used to implement agile software development. It requires real-time communication of capacity and full transparency of work. Work items are represented visually on a kanban board, allowing team members to see the state of every piece of work at any time.

Kanban boards

The work of all kanban teams revolves around a kanban board, a tool used to visualize work and optimize the flow of the work among the team. While physical boards are popular among some teams, virtual boards are a crucial feature in any agile software development tool for their traceability, easier collaboration, and accessibility from multiple locations. Regardless of whether a team's board is physical or digital, their function is to ensure the team's work is visualized, their workflow is standardized, and all blockers and dependencies are immediately identified and resolved. A basic kanban board has a three-step workflow: To Do, In Progress, and Done. However, depending on a team's size, structure, and objectives, the workflow can be mapped to meet the unique process of any particular team.

The kanban methodology relies upon full transparency of work and real-time communication of capacity, therefore the kanban board should be seen as the single source of truth for the team's work.

The benefits of Kanban

Kanban is one of the most popular software development methodologies adopted by agile teams today. Kanban offers several additional advantages to task planning and throughput for teams of all sizes.

Planning flexibility

A kanban team is only focused on the work that's actively in progress. Once the team completes a work item, they pluck the next work item off the top of the backlog. The product owner is free to reprioritize work in the backlog without disrupting the team, because any changes outside the current work items don't impact the team. As long as the product owner keeps the most important work items on top of the backlog, the development team is assured they are delivering maximum value back to the business. So there's no need for the fixed-length iterations you find in scrum.

PRO TIP:

Savvy product owners always engage the development team when considering changes to the backlog. For example, if user stories 1-6 are in the backlog, user story 6's estimate may be based on the completion of user stories 1-5. It's always a good practice to confirm changes with the engineering team to ensure there are no surprises.

Shortened time cycles

Cycle time is a key metric for kanban teams. Cycle time is the amount of time it takes for a unit of work to travel through the team's workflow–from the moment work starts to the moment it ships. By optimizing cycle time, the team can confidently forecast the delivery of future work.

Overlapping skill sets lead to smaller cycle times. When only one person holds a skill set, that person becomes a bottleneck in the workflow. So teams employ basic best practices like code review and mentoring help to spread knowledge. Shared skills mean that team members can take on heterogeneous work, which further optimizes cycle time. It also means that if there is a backup of work, the entire team can swarm on it to get the process flowing smoothly again. For instance, testing isn't only done by QA engineers. Developers pitch in, too.

In a kanban framework, it's the entire team's responsibility to ensure work is moving smoothly through the process.

Fewer bottlenecks

Multitasking kills efficiency. The more work items in flight at any given time, the more context switching, which hinders their path to completion. That's why a key tenet of kanban is to limit the amount of work in progress (WIP). Work-in-progress limits highlight bottlenecks and backups in the team's process due to lack of focus, people, or skill sets.

For example, a typical software team might have four workflow states: To Do, In Progress, Code Review, and Done. They could choose to set a WIP limit of 2 for the code review state. That might seem like a low limit, but there's good reason for it: developers often prefer to write new code, rather than spend time reviewing someone else's work. A low limit encourages the team to pay special attention to issues in the review state, and to review others work before raising their own code reviews. This ultimately reduces the overall cycle time.

Scrum vs. kanban

Kanban and scrum share some of the same concepts but have very different approaches. They should not be confused with one another.

	SCRUM	KANBAN
Cadence	Regular fixed length sprints (ie, 2 weeks)	Continuous flow
Release methodology	At the end of each sprint if approved by the product owner	Continuous delivery or at the team's discretion

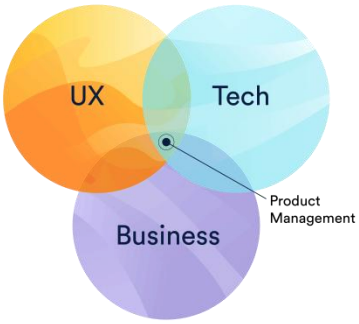
Roles	Product owner, scrum master, development team	No existing roles. Some teams enlist the help of an agile coach.
Key metrics	Velocity	Cycle time
Change philosophy	Teams should strive to not make changes to the sprint forecast during the sprint. Doing so compromises learnings around estimation.	Change can happen at any time

10. What is Product Management?

Product management is an organizational function that guides every step of a product's lifecycle: from development, to positioning and pricing, by focusing on the product and its customers first and foremost. To build the best possible product, product managers advocate for customers within the organization and make sure the voice of the market is heard and heeded.

Thanks to this focus on the customer, product teams routinely ship better-designed and higher-performing products. In tech, where entrenched products are quickly uprooted by newer and better solutions, there is more need than ever for an intimate understanding of customers and the ability to create tailored solutions for them. That's where product management comes in.

As a member of a product team myself, I work daily with product managers and have interviewed dozens more about their roles and responsibilities. Despite the advice here, I've learned that there is no one way to apply principles of product management. Every product has its own goals and challenges which require a unique and customized approach to product management. Martin Eriksson has famously described product management as the intersection of business, user experience, and technology.



- Business — Product management helps teams achieve their business objectives by bridging the communication gap between dev, design, the customer, and the business.
- UX — Product management focuses on the user experience, and represents the customer inside the organization. Great UX is how this focus manifests itself.

- Technology — Product management happens, day to day, in the engineering department. A thorough understanding of computer science is paramount.
- Three additional skills that every PM needs are storytelling, marketing, and empathy.

Storytelling

A product leader should be as inspirational as they are tactical, and storytelling is their tool of choice. Through customer interviews and market research product managers learn more about the customer than even the salespeople. They then use their storytelling skills to share that perspective with the rest of the company.

Marketing

Product Management's customer focus also informs marketing efforts. Instead of sticking to the brand and using established techniques, product management teams (often including Product Marketing Managers) integrate the language of their customers into the messaging of their product. Furthermore, knowledge of the competitive landscape and the ability to stand out and differentiate pays dividends in the long run. Understanding basic marketing and positioning concepts will help product managers ship products that people can find and relate to.

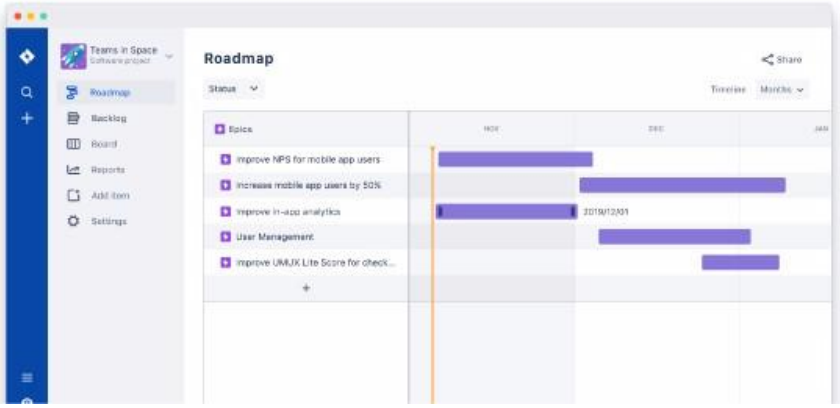
Empathy

Finally, product management is about empathy--Empathy for the developers and how they work, empathy for the customer and their pain points, and even empathy for upper management, who juggle aggressive goals and impossible schedules. This skill in empathy, one developed through immersion within and intimate understanding of each group and stakeholder, separates the product teams that can rally the organization around common goals from those who are incapable of doing so.

11. What is a product roadmap?

A product roadmap is a shared source of truth that outlines the vision, direction, priorities, and progress of a product over time. It's a plan of action that aligns the organization around short- and long-term goals for the product or project, and how they will be achieved.

A product roadmap is the key to communicating how short-term efforts match long-term business goals. Understanding the role of a roadmap—and how to create a great one—is key for keeping everyone on your team headed in the same direction.



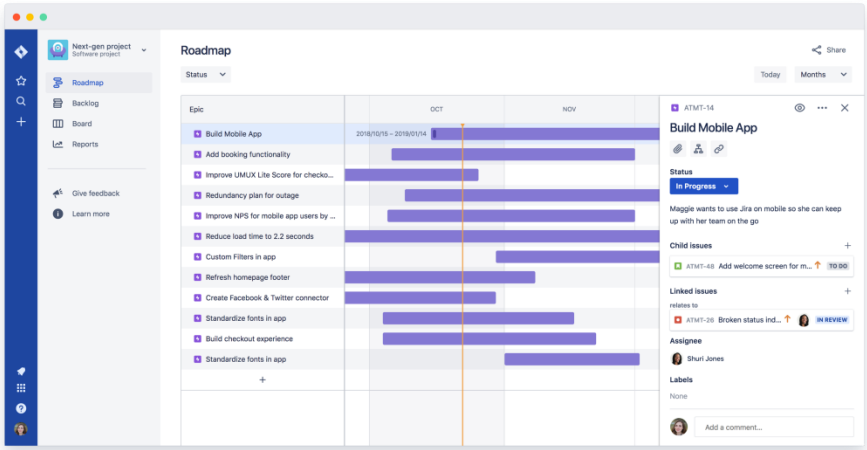
While it's common for the roadmap to show what you're building, it's just as important to show why. Items on the roadmap should be clearly linked to your product strategy, and your roadmap should be responsive to changes in customer feedback and the competitive landscape.

Product owners use roadmaps to collaborate with their teams and build consensus on how a product will grow and shift over time. Agile teams turn to the roadmap to keep everyone on the same page and gain context for their everyday work and future direction.

Who are roadmaps for?

Roadmaps come in several different forms and serve a variety of audiences:

Internal roadmap for development team: These roadmaps can be created in several ways, depending on how your team likes to work. Some common versions include the detail about the prioritized customer value to be delivered, target release dates and milestones. Since many development teams use agile methodologies, these roadmaps are often organized by sprints and show specific pieces of work and problem areas plotted on a timeline.



Internal roadmap for executives: These roadmaps emphasize how teams' work supports high-level company goals and metrics. They are often organized by month or by quarter to show progress over time towards these goals, and generally include less detail about detailed development stories and tasks.

Internal roadmap for sales: These roadmaps focus on new features and customer benefits in order to support sales conversations. An important note: avoid including hard dates in sales roadmaps to avoid tying internal teams to potentially unrealistic dates.

External roadmap: These roadmaps should excite customers about what's coming next. Make sure they are visually appealing and easy to read. They should provide a high-level, generalized view of new features and prioritized problem areas to get customers interested in the future direction of the product.

Why should I create a product roadmap?

The biggest benefit of the product roadmap is the strategic vision it illustrates to all stakeholders. The roadmap matches broader product and company goals with development efforts, which align the teams around common goals to create great products.

For organizational leadership, the roadmap provides updates on the status of work and "translates" developer tasks in Jira into non-technical terms and a format that's easily understood.

For product owners and managers, roadmaps unify teams working on high impact product enhancements and allow them to communicate priorities effectively with adjacent teams.

For the developers themselves, roadmaps provide a better understanding of the "big picture," which allows team members to focus on the most important tasks, avoid scope creep, and make fast, autonomous decisions.

How do I build a product roadmap?

To build a roadmap, product owners should consider market trajectories, customer value propositions, strategic goals, and effort constraints. Once these factors are understood, the product owner can work with their team to start prioritizing initiatives and epics on the roadmap.

The content of a roadmap will depend on its audience - a roadmap for the development team may cover only one product, while a roadmap for executives can cover multiple products. Depending on the size and structure of an organization, a

single roadmap may span multiple teams working on the same product. An external roadmap will often cover multiple products aligned with one point of emphasis or customer need.

The most important takeaway: create a roadmap that your audience can easily understand. Providing too much or too little detail on the roadmap can make it easy to gloss over, or worse, to too intimidating to read. A roadmap with just the right amount of detail and some visual appeal can earn the buy-in you need from key stakeholders.

Presenting the product roadmap

The product roadmap needs buy-in from two key groups: leadership and the agile development team. Presenting the roadmap is a great opportunity to demonstrate to key stakeholders that you understand the company's strategic objectives, the needs of your customer, and have a plan to meet them both.

As you move through the project, make sure to link your team's work back to the roadmap to build context. A tried-and-true method: break initiatives down into epics in the product backlog, then further decompose them into requirements and user stories. Establishing this issue hierarchy makes it easier for product owners and the development team to make decisions and understand how their work fits into the bigger picture.

Using and updating the roadmap

As the competitive landscape shifts, customers' preferences adjust, or planned features are modified, it's important to ensure the product roadmap continues to reflect the status of current work as well as long-term goals.

The roadmap should be updated as often as necessary - this could be every week or fortnightly - so that it can remain an accurate source of truth. As we've all experienced at one time or another, a roadmap is counter-productive if it isn't up to date. You'll know if your roadmap needs to be updated more frequently because your stakeholders will start calling you for updates instead of consulting your roadmap. These one-off requests reflect a distrust in your roadmap, and a huge potential time suck.

However, on the flip side, you don't want to spend more time updating the roadmap than is necessary to achieve alignment between stakeholders and within your team. Remember, the roadmap is a planning tool to think through how to build great products. If you're spending time updating your roadmap that you could (and should) be spending on execution, think about changing your roadmapping tool to something more simple.

Best practices for the best roadmaps

Building and maintaining product roadmaps is an ongoing process to embark upon with your team. There are a few simple ways to set yourself up for success:

- Only include as much detail as necessary for your audience
- Keep the roadmap evenly focused on short-term tactics and how these relate to long-term goals
- Review roadmaps on a regular basis and make adjustments when plans change
- Make sure everyone has access to the roadmap (and checks it on a regular basis)
- Stay connected with stakeholders at all levels to ensure alignment

12. The product backlog: your ultimate to-do list

A well-prioritized agile backlog not only makes release and iteration planning easier, it broadcasts all the things your team intends to spend time on—including internal work that the customer will never notice. This helps set expectations with stakeholders and other teams, especially when they bring additional work to you, and makes engineering time a fixed asset.

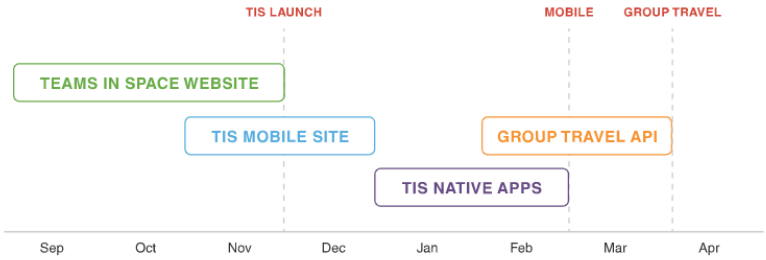
What is a product backlog?

A product backlog is a prioritized list of work for the development team that is derived from the roadmap and its requirements. The most important items are shown at the top of the product backlog so the team knows what to deliver first. The development team doesn't work through the backlog at the product owner's pace and the product owner isn't pushing work to the development team. Instead, the development team pulls work from the product backlog as there is capacity for it, either continually (kanban) or by iteration (scrum).

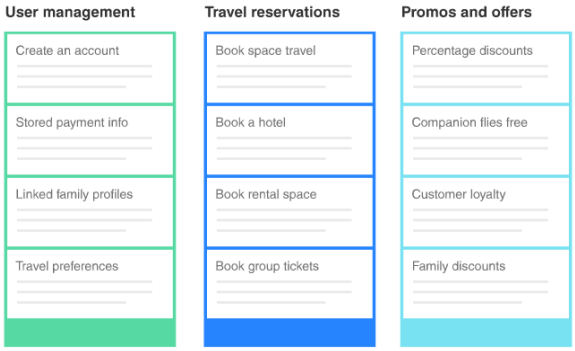
PRO TIP:
Keep everything in one issue tracker–don't use multiple systems to track bugs, requirements, and engineering work items. If it's work for the development team, keep it in a single backlog.

Start with the two "R"s

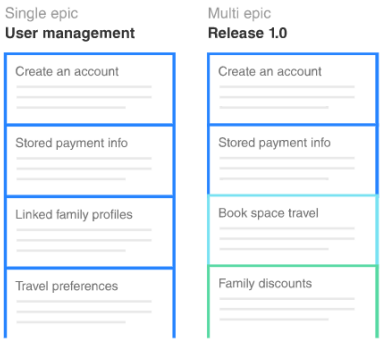
A team's roadmap and requirements provide the foundation for the product backlog. Roadmap initiatives break down into several epics, and each epic will have several requirements and user stories. Let's take a look at the roadmap for a fictitious product called Teams in Space.



Since the Teams in Space website is the first initiative in the roadmap, we'll want to break down that initiative into epics (shown here in green, blue, and teal) and user stories for each of those epics.



The product owner then organizes each of the user stories into a single list for the development team. The product owner may choose to deliver a complete epic first (left). Or, it may be more important to the program to test booking a discounted flight which requires stories from several epics (right). See both examples below.



What may influence a product owner's prioritization?

- Customer priority
- Urgency of getting feedback
- Relative implementation difficulty
- Symbiotic relationships between work items (e.g. B is easier if we do A first)

While the product owner is tasked with prioritizing the backlog, it's not done in a vacuum. Effective product owners seek input and feedback from customers, designers, and the development team to optimize everyone's workload and the product delivery.

Keeping the backlog healthy

Once the product backlog is built, it's important to regularly maintain it to keep pace with the program. Product owners should review the backlog before each iteration planning meeting to ensure prioritization is correct and feedback from the last iteration has been incorporated. Regular review of the backlog is often called "backlog grooming" in agile circles(some use the term backlog refinement).
Once the backlog gets larger, product owners need to group the backlog into near-term and long-term items. Near-term items need to be fully fleshed out before they are labeled as such. This means complete user stories have been drawn up, collaboration with design and development has been sorted out, and estimates from development have been made. Longer term items can remain a bit vague, though it's a good idea to get a rough estimate from the development team to help prioritize them. The key word here is "rough": estimates will change once the team fully understands and begins work on those longer term items.

The backlog serves as the connection between the product owner and the development team. The product owner is free to re-prioritize work in the backlog at any time due to customer feedback, refining estimates, and new requirements. Once work is in progress, though, keep changes to a minimum as they disrupt the development team and affect focus, flow, and morale.

PRO TIP:

Once the backlog grows beyond the team's long term capacity, it's okay to close issues the team will never get to. Flag those issues with a specific resolution like “out of scope” in the team's issue tracker to use for research later.

Anti-patterns to watch for

- The product owner prioritizes the backlog at the start of the project, but doesn't adjust it as feedback rolls in from developers and stakeholders.

- The team limits items on the backlog to those that are customer-facing.
- The backlog is kept as a document stored locally and shared infrequently, preventing interested parties from getting updates.

How do product backlogs keep the team agile?

Savvy product owners rigorously groom their program's product backlog, making it a reliable and sharable outline of the work items for a project.

Backlogs prompt debates and choices that keep a program healthy–not everything can be top priority.

Stakeholders will challenge priorities, and that's good. Fostering discussion around what's important gets everyone's priorities in sync. These discussions foster a culture of group prioritization ensuring everyone shares the same mindset on the program.

The product backlog also serves as the foundation for iteration planning. All work items should be included in the backlog: user stories, bugs, design changes, technical debt, customer requests, action items from the retrospective, etc. This ensures everyone's work items are included in the overall discussion for each iteration. Team members can then make trade-offs with the product owner before starting an iteration with complete knowledge of everything that needs to be done.

PRO TIP:

Product owners dictate the priority of work items in the backlog, while the development team dictates the velocity through the backlog. This can be a tenuous relationship for new product owners who want to "push" work to the team. Learn more in our article about work-in-progress limits and flow.

13. Agile Teams

Build your team

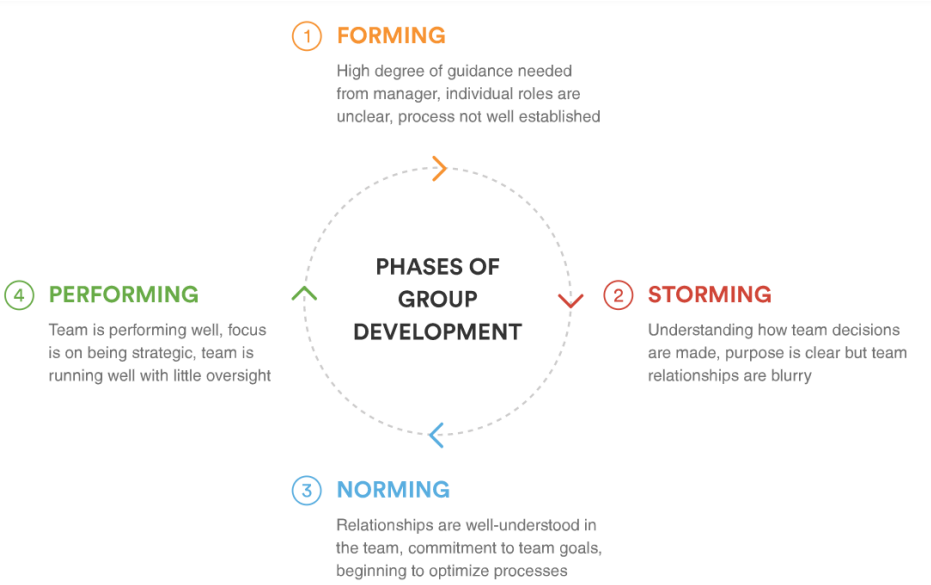
Agile visionaries believed that teamwork is essential to delivering great software and that great agile teams embody "we" rather than “I." Nothing is more rewarding than sharing the adventure of building something that truly matters with engaged teammates.

Despite sharing common values, there is no formula for the perfect agile team. Some implement scrum while others use kanban. Agile purists prefer co-located teams, but business realities sometimes necessitate distributing an agile team across geographies. Most agile teams possess all the required skills, but sometimes it's necessary to call on specialists for specific work. So how do you know whether your team is on the path to greatness?

Build upon a solid foundation

Once the team is in place, it's important to remember that agile teams are like individuals: they take time to grow. Agile theorists often quote Tuckman's "stages of group development.” Agile teams go through four key phases as they develop.

After a team reaches the performing stage, development truly becomes awesome. Members trust each other, understand one another's strengths, and use that understanding to optimize how they build software.



Keeping agile teams intact takes some organizational discipline, but it pays to protect the team–within reason, of course. When change is introduced (new hire, employee departure, etc.), the team reverts back to the forming stage as it absorbs the change.

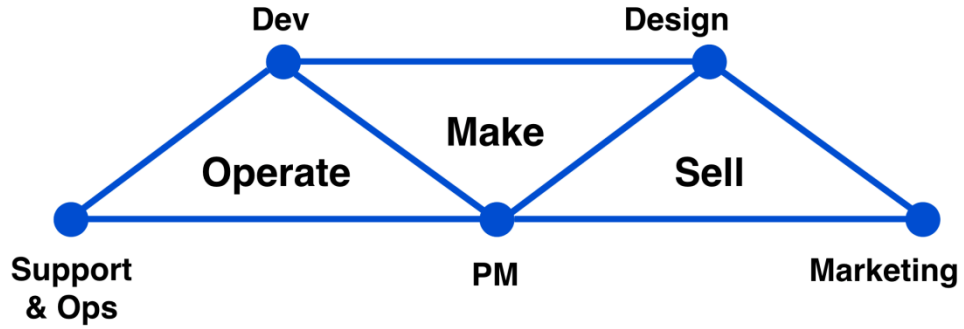
High-performing agile teams are also built on sound engineering practices like code reviews, task branching, continuous integration, and regular release cadences. We can't stress this enough: engineering fundamentals are crucial for building great teams. (Read more on those topics in our "Agile Developer" section.)

PRO TIP:

Agile teams aren't just for engineers. In larger software organizations, agile teams form in many parts of the business: marketing, HR, finance... you name it!
There are two other pillars of great agile teams: continuous mentoring and shared skill sets. One of the big benefits in working on a team is that colleagues learn from one another and mentor one another. Mentoring isn't just an activity for junior members to learn from senior members. Everyone on the team learns from one another so that the impact of the team as a whole is greater than the sum of the impact made by its individual members. Meanwhile, shared skill sets unlock the power of the team to tackle heterogeneous work. As engineers, it's always important to learn new skills because it makes us more valuable to the organization and better equipped to support each other's work. It also guards against someone becoming a critical path, which takes a load off everyone's mind.

How agile teams collaborate across departments

Today's software teams include product managers, designers, marketers, and operations as well as developers and testers. As an example, we focus our agile teams around three product phases: make, sell, and operate.



Each product phase is supported by three teams (ideally 5-7 members each), and forms a triad. Each triad is agile in its approach, because as the product develops, teams are continuously working on each phase and learning more about the product as well as the market. Below is a breakdown of each triad and the who, what, where, and why for each team within the larger software team.
Here's the catch: reaching the 'performing' stage is impossible if a team's make-up shifts a lot.

Regardless of which triad your team operates in, agile can make your team deliver faster and have more fun. Dig further into this section and learn how to focus and optimize agile teams.

Triad	Who	Focus
Make	Product Management	Understand the market, targeted customer personas, and good product design principles
	Design	Define the value proposition, product goals, and minimum viable product
	Development	Develop the product using sound, sustainable engineering practices
Sell	Product Management	Understand the product's competitive landscape and market evolutions
	Design	Create messaging that highlights the product's value propositions to each customer segment
	Marketing	Build collateral to support the product launch: web pages, announcement emails, blogs, videos, etc.
Operate	Product Management	Release software to customers with a regular cadence
	Development	Respond to customer issues
	Support & Ops	Relay customer feedback to the make triad (Dev, PM, Design) as input for future product development

14. Scrum roles

The Scrum Team

The Scrum Team consists of a Product Owner, the Development Team, and a Scrum Master. Scrum Teams are self-organizing and cross-functional. Self-organizing teams choose how best to accomplish their work, rather than being directed by others outside the team. Cross-functional teams have all competencies needed to accomplish the work without depending on others not part of the team. The team model in Scrum is designed to optimize flexibility, creativity, and productivity. The Scrum Team has proven itself to be increasingly effective for all the earlier stated uses, and any complex work.

Scrum Teams deliver products iteratively and incrementally, maximizing opportunities for feedback. Incremental deliveries of "Done" product ensure a potentially useful version of working product is always available.

The Product Owner

The Product Owner is responsible for maximizing the value of the product resulting from work of the Development Team. How this is done may vary widely across organizations, Scrum Teams, and individuals.

The Product Owner is the sole person responsible for managing the Product Backlog. Product Backlog management includes:

- Clearly expressing Product Backlog items;
- Ordering the items in the Product Backlog to best achieve goals and missions;
- Optimizing the value of the work the Development Team performs;
- Ensuring that the Product Backlog is visible, transparent, and clear to all, and shows what the Scrum Team will work on next; and,
- Ensuring the Development Team understands items in the Product Backlog to the level needed.

The Product Owner may do the above work, or have the Development Team do it. However, the Product Owner remains accountable.

The Product Owner is one person, not a committee. The Product Owner may represent the desires of a committee in the Product Backlog, but those wanting to change a Product Backlog item's priority must address the Product Owner.

For the Product Owner to succeed, the entire organization must respect his or her decisions. The Product Owner's decisions are visible in the content and ordering of the Product Backlog. No one can force the Development Team to work from a different set of requirements.

The Development Team

The Development Team consists of professionals who do the work of delivering a potentially releasable Increment of "Done" product at the end of each Sprint. A "Done" increment is required at the Sprint Review. Only members of the Development Team create the Increment.

Development Teams are structured and empowered by the organization to organize and manage their own work. The resulting synergy optimizes the Development Team's overall efficiency and effectiveness.

Development Teams have the following characteristics:

- They are self-organizing. No one (not even the Scrum Master) tells the Development Team how to turn Product Backlog into Increments of potentially releasable functionality;
- Development Teams are cross-functional, with all the skills as a team necessary to create a product Increment;
- Scrum recognizes no titles for Development Team members, regardless of the work being performed by the person;
- Scrum recognizes no sub-teams in the Development Team, regardless of domains that need to be addressed like testing, architecture, operations, or business analysis; and,
- Individual Development Team members may have specialized skills and areas of focus, but accountability belongs to the Development Team as a whole.

Development Team Size

Optimal Development Team size is small enough to remain nimble and large enough to complete significant work within a Sprint. Fewer than three Development Team members decrease interaction and results in smaller productivity gains. Smaller Development Teams may encounter skill constraints during the Sprint, causing the Development Team to be unable to deliver a potentially releasable Increment. Having more than nine members requires too much coordination. Large Development Teams generate too much complexity for an empirical process to be useful. The Product Owner and Scrum Master roles are not included in this count unless they are also executing the work of the Sprint Backlog.

The Scrum Master

The Scrum Master is responsible for promoting and supporting Scrum as defined in the Scrum Guide. Scrum Masters do this by helping everyone understand Scrum theory, practices, rules, and values.

The Scrum Master is a servant-leader for the Scrum Team. The Scrum Master helps those outside the Scrum Team understand which of their interactions with the Scrum Team are helpful and which aren't. The Scrum Master helps everyone change these interactions to maximize the value created by the Scrum Team.

Scrum Master Service to the Product Owner

The Scrum Master serves the Product Owner in several ways, including:

- Ensuring that goals, scope, and product domain are understood by everyone on the Scrum Team as well as possible;
- Finding techniques for effective Product Backlog management;
- Helping the Scrum Team understand the need for clear and concise Product Backlog items;
- Understanding product planning in an empirical environment;
- Ensuring the Product Owner knows how to arrange the Product Backlog to maximize value;
- Understanding and practicing agility; and,
- Facilitating Scrum events as requested or needed.

Scrum Master Service to the Development Team

The Scrum Master serves the Development Team in several ways, including:

- Coaching the Development Team in self-organization and cross-functionality;
- Helping the Development Team to create high-value products;
- Removing impediments to the Development Team's progress;
- Facilitating Scrum events as requested or needed; and,

- Coaching the Development Team in organizational environments in which Scrum is not yet fully adopted and understood.

Scrum Master Service to the Organization

The Scrum Master serves the organization in several ways, including:

- Leading and coaching the organization in its Scrum adoption;
- Planning Scrum implementations within the organization;
- Helping employees and stakeholders understand and enact Scrum and empirical product development;
- Causing change that increases the productivity of the Scrum Team; and,
- Working with other Scrum Masters to increase the effectiveness of the application of Scrum in the organization.

15. Scrum Events

Prescribed events are used in Scrum to create regularity and to minimize the need for meetings not defined in Scrum. All events are time-boxed events, such that every event has a maximum duration. Once a Sprint begins, its duration is fixed and cannot be shortened or lengthened. The remaining events may end whenever the purpose of the event is achieved, ensuring an appropriate amount of time is spent without allowing waste in the process.

Other than the Sprint itself, which is a container for all other events, each event in Scrum is a formal opportunity to inspect and adapt something. These events are specifically designed to enable critical transparency and inspection. Failure to include any of these events results in reduced transparency and is a lost opportunity to inspect and adapt.

The Sprint

The heart of Scrum is a Sprint, a time-box of one month or less during which a "Done", useable, and potentially releasable product Increment is created. Sprints have consistent durations throughout a development effort. A new Sprint starts immediately after the conclusion of the previous Sprint.

Sprints contain and consist of the Sprint Planning, Daily Scrums, the development work, the Sprint Review, and the Sprint Retrospective.

During the Sprint:

- No changes are made that would endanger the Sprint Goal;
- Quality goals do not decrease; and,
- Scope may be clarified and re-negotiated between the Product Owner and Development Team as more is learned.

Each Sprint may be considered a project with no more than a one-month horizon. Like projects, Sprints are used to accomplish something. Each Sprint has a goal of what is to be built, a design and flexible plan that will guide building it, the work, and the resultant product increment.

Sprints are limited to one calendar month. When a Sprint’s horizon is too long the definition of what is being built may change, complexity may rise, and risk may increase. Sprints enable predictability by ensuring inspection and adaptation of progress toward a Sprint Goal at least every calendar month. Sprints also limit risk to one calendar month of cost.

Cancelling a Sprint

A Sprint can be cancelled before the Sprint time-box is over. Only the Product Owner has the authority to cancel the Sprint, although he or she may do so under influence from the stakeholders, the Development Team, or the Scrum Master.

A Sprint would be cancelled if the Sprint Goal becomes obsolete. This might occur if the company changes direction or if market or technology conditions change. In general, a Sprint should be cancelled if it no longer makes sense given the circumstances. But, due to the short duration of Sprints, cancellation rarely makes sense.

When a Sprint is cancelled, any completed and "Done" Product Backlog items are reviewed. If part of the work is potentially releasable, the Product Owner typically accepts it. All incomplete Product Backlog Items are re-estimated and put back on the Product Backlog. The work done on them depreciates quickly and must be frequently re-estimated.

Sprint cancellations consume resources, since everyone regroups in another Sprint Planning to start another Sprint. Sprint cancellations are often traumatic to the Scrum Team, and are very uncommon.

Sprint Planning

The work to be performed in the Sprint is planned at the Sprint Planning. This plan is created by the collaborative work of the entire Scrum Team.

Sprint Planning is time-boxed to a maximum of eight hours for a one-month Sprint. For shorter Sprints, the event is usually shorter. The Scrum Master ensures that the event takes place and that attendants understand its purpose. The Scrum Master teaches the Scrum Team to keep it within the time-box.

Sprint Planning answers the following:

- What can be delivered in the Increment resulting from the upcoming Sprint?
- How will the work needed to deliver the Increment be achieved?

Topic One: What can be done this Sprint?

The Development Team works to forecast the functionality that will be developed during the Sprint. The Product Owner discusses the objective that the Sprint should achieve and the Product Backlog items that, if completed in the Sprint, would achieve the Sprint Goal. The entire Scrum Team collaborates on understanding the work of the Sprint.

The input to this meeting is the Product Backlog, the latest product Increment, projected capacity of the Development Team during the Sprint, and past performance of the Development Team. The number of items selected from the Product Backlog for the Sprint is solely up to the Development Team. Only the Development Team can assess what it can accomplish over the upcoming Sprint.

During Sprint Planning the Scrum Team also crafts a Sprint Goal. The Sprint Goal is an objective that will be met within the Sprint through the implementation of the Product Backlog, and it provides guidance to the Development Team on why it is building the Increment.

Topic Two: how will the chosen work get done?

Having set the Sprint Goal and selected the Product Backlog items for the Sprint, the Development Team decides how it will build this functionality into a "Done" product Increment during the Sprint. The Product Backlog items selected for this Sprint plus the plan for delivering them is called the Sprint Backlog.

The Development Team usually starts by designing the system and the work needed to convert the Product Backlog into a working product Increment. Work may be of varying size, or estimated effort. However, enough work is planned during Sprint Planning for the Development Team to forecast what it believes it can do in the upcoming Sprint. Work planned for the first days of the Sprint by the Development Team is decomposed by the end of this meeting, often to units of one day or

less. The Development Team self-organizes to undertake the work in the Sprint Backlog, both during Sprint Planning and as needed throughout the Sprint.

The Product Owner can help to clarify the selected Product Backlog items and make trade-offs. If the Development Team determines it has too much or too little work, it may renegotiate the selected Product Backlog items with the Product Owner. The Development Team may also invite other people to attend to provide technical or domain advice.

By the end of the Sprint Planning, the Development Team should be able to explain to the Product Owner and Scrum Master how it intends to work as a self-organizing team to accomplish the Sprint Goal and create the anticipated Increment.

Sprint Goal

The Sprint Goal is an objective set for the Sprint that can be met through the implementation of Product Backlog. It provides guidance to the Development Team on why it is building the Increment. It is created during the Sprint Planning meeting. The Sprint Goal gives the Development Team some flexibility regarding the functionality implemented within the Sprint. The selected Product Backlog items deliver one coherent function, which can be the Sprint Goal. The Sprint Goal can be any other coherence that causes the Development Team to work together rather than on separate initiatives.

As the Development Team works, it keeps the Sprint Goal in mind. In order to satisfy the Sprint Goal, it implements functionality and technology. If the work turns out to be different than the Development Team expected, they collaborate with the Product Owner to negotiate the scope of Sprint Backlog within the Sprint.

Daily Scrum

The Daily Scrum is a 15-minute time-boxed event for the Development Team. The Daily Scrum is held every day of the Sprint. At it, the Development Team plans work for the next 24 hours. This optimizes team collaboration and performance by inspecting the work since the last Daily Scrum and forecasting upcoming Sprint work. The Daily Scrum is held at the same time and place each day to reduce complexity.

The Development Team uses the Daily Scrum to inspect progress toward the Sprint Goal and to inspect how progress is trending toward completing the work in the Sprint Backlog. The Daily Scrum optimizes the probability that the Development Team will meet the Sprint Goal. Every day, the Development Team should understand how it intends to work together as a self-organizing team to accomplish the Sprint Goal and create the anticipated Increment by the end of the Sprint.

The structure of the meeting is set by the Development Team and can be conducted in different ways if it focuses on progress toward the Sprint Goal. Some Development Teams will use questions, some will be more discussion based. Here is an example of what might be used:

- What did I do yesterday that helped the Development Team meet the Sprint Goal?
- What will I do today to help the Development Team meet the Sprint Goal?
- Do I see any impediment that prevents me or the Development Team from meeting the Sprint Goal?

The Development Team or team members often meet immediately after the Daily Scrum for detailed discussions, or to adapt, or replan, the rest of the Sprint's work.

The Scrum Master ensures that the Development Team has the meeting, but the Development Team is responsible for conducting the Daily Scrum. The Scrum Master teaches the Development Team to keep the Daily Scrum within the 15-minute time-box.

The Daily Scrum is an internal meeting for the Development Team. If others are present, the Scrum Master ensures that they do not disrupt the meeting.

Daily Scrums improve communications, eliminate other meetings, identify impediments to development for removal, highlight and promote quick decision-making, and improve the Development Team's level of knowledge. This is a key inspect and adapt meeting.

Sprint Review

A Sprint Review is held at the end of the Sprint to inspect the Increment and adapt the Product Backlog if needed. During the Sprint Review, the Scrum Team and stakeholders collaborate about what was done in the Sprint. Based on that and any changes to the Product Backlog during the Sprint, attendees collaborate on the next things that could be done to optimize value. This is an informal meeting, not a status meeting, and the presentation of the Increment is intended to elicit feedback and foster collaboration.

This is at most a four-hour meeting for one-month Sprints. For shorter Sprints, the event is usually shorter. The Scrum Master ensures that the event takes place and that attendees understand its purpose. The Scrum Master teaches everyone involved to keep it within the time-box.

The Sprint Review includes the following elements:

- Attendees include the Scrum Team and key stakeholders invited by the Product Owner;
- The Product Owner explains what Product Backlog items have been "Done" and what has not been "Done";
- The Development Team discusses what went well during the Sprint, what problems it ran into, and how those problems were solved;
- The Development Team demonstrates the work that it has "Done" and answers questions about the Increment;
- The Product Owner discusses the Product Backlog as it stands. He or she projects likely target and delivery dates based on progress to date (if needed);
- The entire group collaborates on what to do next, so that the Sprint Review provides valuable input to subsequent Sprint Planning;
- Review of how the marketplace or potential use of the product might have changed what is the most valuable thing to do next; and,
- Review of the timeline, budget, potential capabilities, and marketplace for the next anticipated releases of functionality or capability of the product.

The result of the Sprint Review is a revised Product Backlog that defines the probable Product Backlog items for the next Sprint. The Product Backlog may also be adjusted overall to meet new opportunities.

Sprint Retrospective

The Sprint Retrospective is an opportunity for the Scrum Team to inspect itself and create a plan for improvements to be enacted during the next Sprint.

The Sprint Retrospective occurs after the Sprint Review and prior to the next Sprint Planning. This is at most a three-hour meeting for one-month Sprints. For shorter Sprints, the event is usually shorter. The Scrum Master ensures that the event takes place and that attendants understand its purpose.

The Scrum Master ensures that the meeting is positive and productive. The Scrum Master teaches all to keep it within the time-box. The Scrum Master participates as a peer team member in the meeting from the accountability over the Scrum process.

The purpose of the Sprint Retrospective is to:

- Inspect how the last Sprint went with regards to people, relationships, process, and tools;

- Identify and order the major items that went well and potential improvements; and,
- Create a plan for implementing improvements to the way the Scrum Team does its work.

The Scrum Master encourages the Scrum Team to improve, within the Scrum process framework, its development process and practices to make it more effective and enjoyable for the next Sprint. During each Sprint Retrospective, the Scrum Team plans ways to increase product quality by improving work processes or adapting the definition of "Done", if appropriate and not in conflict with product or organizational standards.

By the end of the Sprint Retrospective, the Scrum Team should have identified improvements that it will implement in the next Sprint. Implementing these improvements in the next Sprint is the adaptation to the inspection of the Scrum Team itself. Although improvements may be implemented at any time, the Sprint Retrospective provides a formal opportunity to focus on inspection and adaptation.

16. Epics, stories, initiatives, themes

These simple structures help agile teams gracefully manage scope and structure work.

Let's say you and your team want to do something ambitious, like launch a rocket into space. To do so, you'll need to structure your work: from the largest objectives down to the minute details. You'll want to be able to respond to change, report your progress, and stick to a plan. Epics, stories, themes, and initiatives are precisely the tools you'll need to do so. By understanding how these popular agile methodologies help organize work, your team can strike a healthy balance between structure, flexibility, and launching rockets into space.

What are stories, epics, initiatives, and themes?

- **Stories**, also called "user stories," are short requirements or requests written from the perspective of an end user.
- **Epics** are large bodies of work that can be broken down into a number of smaller tasks (called stories).
- **Initiatives** are collections of epics that drive toward a common goal.
- **Themes** are large focus areas that span the organization.

Agile Epic vs Story

In a sense, stories and epics in agile are similar to stories and epics in film or literature. A story is one simple narrative; a series of related and interdependent stories makes up an epic. The same is true for your work management, where the completion of related stories leads to the completion of an epic. The stories tell the arc of the work completed while the epic shares a high-level view of the unifying objective.

On an agile team, stories are something the team can commit to finish within a one or two-week sprint. Oftentimes, developers would work on dozens of stories a month. Epics, in contrast, are few in number and take longer to complete. Teams often have two or three epics they work to complete each quarter.

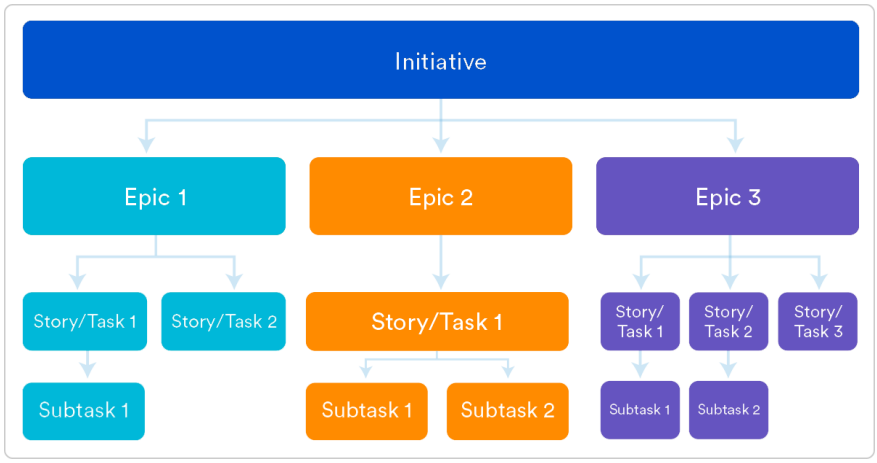
If your company was launching rockets into space, and wanted to improve the streaming service for your launches, you might structure your stories like the ones below.

Examples of an agile story:

- iPhone users need access to a vertical view of the live feed when using the mobile app.
- Desktop users need a "view fullscreen" button in the lower right hand corner of the video player.
- Android users need to be linked to apple store.
- The above stories are all related, and could all be considered individual tasks that drive toward the completion of a larger body of work (an epic). In this case, the epic might be "Improve Streaming Service for Q1 Launch."
- Organizing work into stories and epics also helps you and your team communicate effectively within the organization. If you were reporting your team's progress to the Head of Engineering, you'd be speaking in epics. If you were talking to a colleague on your development team, you'd speak at the story level.

Agile Epic vs Initiative

In the same way that epics are made up of stories, initiatives are made up of epics. Initiatives offer another level of organization above epics. In many cases, an initiative compiles epics from multiple teams to achieve a much broader, bigger goal than any of the epics themselves. While an epic is something you might complete in a month or a quarter, initiatives are often completed in multiple quarters to a year.



Example of epics in an initiative:

Let’s say your rocket ship company wants to decrease the cost per launch by 5% this year. That’s a great fit for an initiative, as no single epic could likely achieve that big of a goal. Within that initiative, there would be epics such as, “Decrease launch-phase fuel consumption by 1%,” “Increase launches per quarter from 3 to 4,” and “Turn all thermostats down from 71 to 69 degrees.”

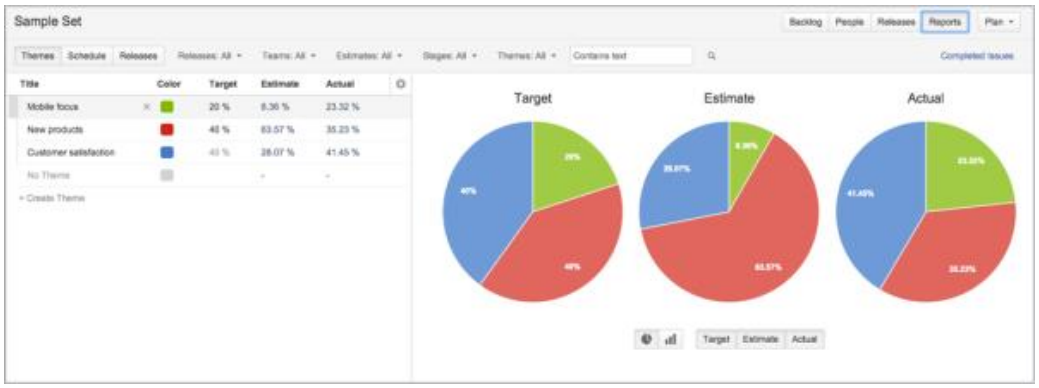
Initiatives vs. Themes

In many organizations the founders and management team will encourage the pursuit of some aspirational destination. These are the (sometimes super corny) goals announced each year or quarter, and themes are how you keep track of them.

- Initiatives are collections of epics
- Themes are labels that track high-level organizational goals

Initiatives have a structural design. They house epics, and the completion of those epics will lead to the completion of the initiative. Themes are an organizational tool that allows you to label backlog items, epics, and initiatives to understand what work contributes to what organizational goals. Themes should inspire the creation of epics and initiatives but don’t have a rigid 1-to-1 relationship with them. A theme for a rocket ship company would be something like “Safety First.”

This is what themes look like:



17. Using workflows for fun & profit

Everyone hates "process", but let's face it: without an established workflow, you're going nowhere fast.

Every software team has a process they use to complete work. Normalizing that process–i.e., establishing it as a workflow–makes it clearly structured and repeatable, which, in turn, makes it scalable. As an example, we take an iterative approach to workflow management because it helps us meet our goals faster and exemplifies our team culture. We are experts in agile workflow management (if we do say so ourselves), and we want to help you become experts too.

Start simple, start now

When implementing a workflow for the team, always start simple. Fight the temptation to spend weeks (over-)engineering it. Overly complex workflows are hard to understand and adopt–not to mention adapt. For software teams, we recommend these basic workflow states:

TO DO

Work that has not been started

IN PROGRESS

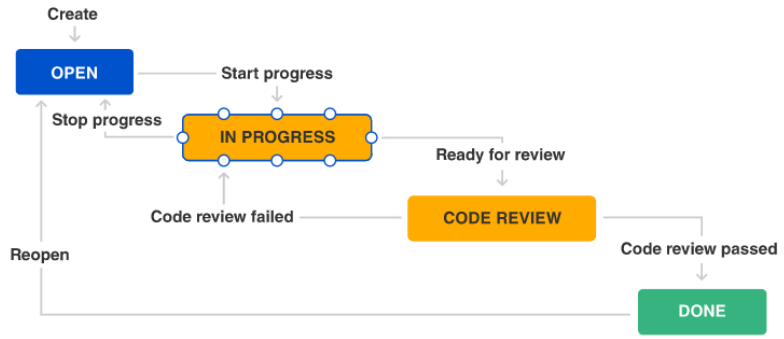
Work that is actively being looked at by the team.

CODE REVIEW

Work that is completed, but awaiting review.

DONE

Work that is completely finished and meets the team's definition of done.
In an issue tracker, these statuses flow from one to the next using transitions which structure the workflow.



Some software teams include additional states in their workflow that help them track the status of work more precisely.

AWAITING QA

Work that has been implemented, but is still waiting for a tester review (see our article on agile testing for more details).
READY TO MERGE

Code that has been reviewed and is ready to merge into the master or release branch.
Each state in the workflow doesn't need to be handled by a different person. As an agile team matures, developers handle more and more of the work–from design all the way through to delivery. An autonomous team that can handle heterogeneous work is one of the hallmarks of agility, after all.

Healthy workflows adapt to the needs of the team. Occasional pain is normal. Chronic pain is not.

Discuss each pain point in the team's retrospective, and keep in mind that each team will have slightly different values based on their project, technology stack, and method in which they like to work. That's why it's important to choose an issue tracker that has a flexible workflow configuration. Too many teams compromise their work style to fit a particular toolset, which is frustrating for everyone. So team members start to avoid using that tool altogether, compounding frustration across the team and generally wreaking havoc. And when morale falls, productivity suffers. That's a double whammy we all want to avoid!

Teams that are new to agile or that don't have cross-functional skills often end up with "mini waterfalls" in their workflow. For example, design kicks off a work item with a mock-up. Development does the implementation. Test confirms quality. Each state is blocked until the former state is complete. Sound familiar? That's waterfall. But we can do much better with agile workflows to unblock the team and make development easier.

Optimize the workflow

When you're comfortable with the basic workflow and are ready to customize it, create statuses for each type of work in a team's process. Ideation, design, development, code review, and test are functionally different and can be individual statuses. Aim for a lean set of statues that still clearly communicate what phase a piece of work is in.

Project statuses can also be shared with the rest of the organization. When building a workflow, think about which metrics are important to report on and what non-team members might be interested in learning. For example, a well designed workflow answers the following questions:

- What work has the team completed?
- Is the backlog of work increasing or keeping pace with the team?
- How many items are in each status?
- Are there any bottlenecks that are slowing the team down?
- How long does it take to complete an average task?
- How many work items didn't pass our quality standards the first time around?

The next step in optimizing the workflow is to ensure a steady stream of work through the workflow. Work-in-progress (WIP) limits dictate a minimum and maximum number of issues in a particular state of the workflow, making sure each state in the workflow has enough work to keep the team fully utilized, but not so much that they lose focus because they're juggling priorities. Enforcing work-in-progress limits will quickly show which processes in the team are slowing down the overall work through the pipeline. As the team learns to optimize around its work-in-progress limits, throughput will increase. (See the article on WIP limits for more details.)

The challenges of scaling a workflow

Organizations that have several agile teams face special challenges with workflows. Teams often want to optimize their own workflow to reflect their unique process and culture. Perfectly understandable. But it can create headaches when different teams use different processes but work on the same project.

Agile teams that work together can benefit from sharing the same workflow. Using the same workflow can make transitioning work between agile teams easier, because they use the same conventions for defining and delivering work. Creating a common process usually involves some give and take from both teams. That's good! They'll learn from one another and come out with a better workflow in the end.

18. The secret behind story points and agile estimation

Good estimation helps product owners optimize for efficiency and impact. That's why it's so important.

Estimation is hard. For software developers, it's among the most difficult–if not the most difficult–aspects of the job. It must take into account a slew of factors that help product owners make decisions that affect the entire team–and the business. With all that at stake, it's no wonder everyone from developers to upper management is prone to getting their undies in a bunch about it. But that's a mistake. Agile estimation is just that: an estimate. Not a blood-oath.

There's no requirement to work weekends in order to compensate for under-estimating a piece of work. That said, let's look at some ways to make agile estimates as accurate as possible.

Collaborating with the product owner

In agile development, the product owner is tasked with prioritizing the backlog–the ordered list of work that contains short descriptions of all desired features and fixes for a product. Product owners capture requirements from the business, but they don't always understand the details of implementation. So good estimation can give the product owner new insight into the level of effort for each work item, which then feeds back into their assessment of each item's relative priority. When the engineering team begins its estimation process, questions usually arise about requirements and user stories. And that's good: those questions help the entire team understand the work more fully. For product owners specifically, breaking down work items into granular pieces and estimates via story points helps them prioritize all (and potentially hidden!) areas of work. And once they have estimates from the dev team, it's not uncommon for a product owner to reorder items on the backlog.

Agile estimation is a team sport

Involving everyone (developers, designers, testers, deployers... everyone) on the team is key. Each team member brings a different perspective on the product and the work required to deliver a user story. For example, if product management wants to do something that seems simple, like support a new web browser, development and QA need to weigh in because their experience has taught them what dragons may be lurking beneath the surface.

Likewise, design changes require not only the design team's input, but that of development and QA as well. Leaving part of the broader product team out of the estimation process creates lower quality estimates, lowers morale because key contributors don't feel included, and compromises the quality of the software.

So don't let your team fall victim to estimates made in a vacuum. It's a fast track to failure!

Story points vs. hours

Traditional software teams give estimates in a time format: days, weeks, months. Many agile teams, however, have transitioned to story points. Story points rate the relative effort of work in a Fibonacci-like format: 0, 0.5, 1, 2, 3, 5, 8, 13, 20, 40, 100. It may sound counter-intuitive, but that abstraction is actually helpful because it pushes the team to make tougher decisions around the difficulty of work. Here are few reasons to use story points:

- Dates don't account for the non-project related work that inevitably creeps into our days: emails, meetings, and interviews that a team member may be involved in.
- Dates have an emotional attachment to them. Relative estimation removes the emotional attachment.
- Each team will estimate work on a slightly different scale, which means their velocity (measured in points) will naturally be different. This, in turn, makes it impossible to play politics using velocity as a weapon.
- Once you agree on the relative effort of each story point value, you can assign points quickly without much debate.
- Story points reward team members for solving problems based on difficulty, not time spent. This keeps team members focused on shipping value, not spending time.

Story points and planning poker

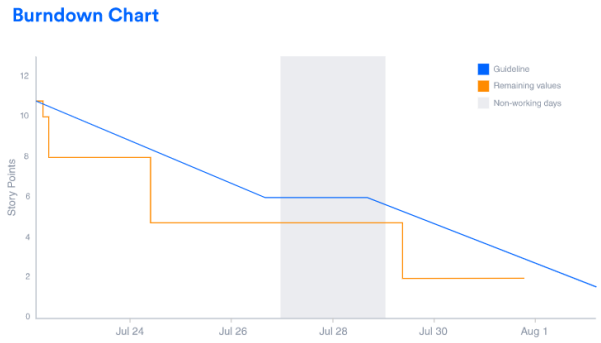
Teams starting out with story points use an exercise called planning poker. As an example, planning poker is a common practice across the company. The team will take an item from the backlog, discuss it briefly, and each member will mentally formulate an estimate. Then everyone holds up a card with the number that reflects their estimate. If everyone is in agreement, great! If not, take some time (but not too much time–just couple minutes) to understand the rationale behind

different estimates. Remember though, estimation should be a high level activity. If the team is too far into the weeds, take a breath, and up-level the discussion.

19. Agile metrics

Sprint Burndown

Scrum teams organize development into time-boxed sprints. At the outset of the sprint, the team forecasts how much work they can complete during a sprint. A sprint burndown report then tracks the completion of work throughout the sprint. The x-axis represents time, and the y-axis refers to the amount of work left to complete, measured in either story points or hours. The goal is to have all the forecasted work completed by the end of the sprint.
A team that consistently meets its forecast is a compelling advertisement for agile in their organization. But don't let that tempt you to fudge the numbers by declaring an item complete before it really is. It may look good in the short term, but in the long run only hampers learning and improvement.



ANTI-PATTERNS TO WATCH FOR

- The team finishes early sprint after sprint because they aren't committing to enough work.
- The team misses their forecast sprint after sprint because they're committing to too much work.
- The burndown line makes steep drops rather than a more gradual burndown because the work hasn't been broken down into granular pieces.
- The product owner adds or changes the scope mid-sprint.

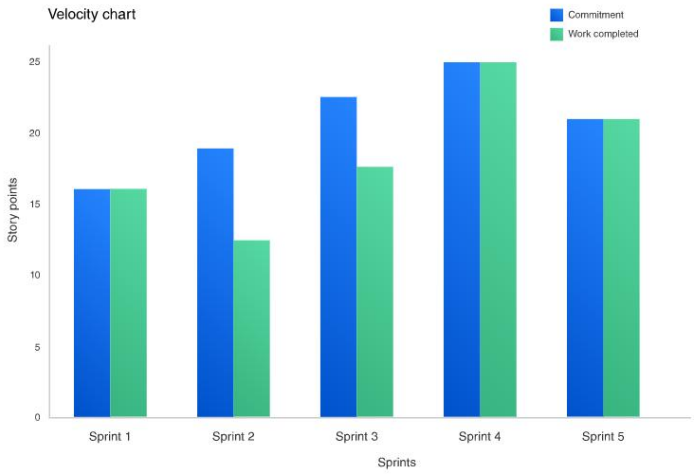
Velocity & Predictability

Velocity is the average amount of work a scrum team completes during a sprint, measured in either story points or hours, and is very useful for forecasting. The product owner can use velocity to predict how quickly a team can work through the backlog, because the report tracks the forecasted and completed work over several iterations–the more iterations, the more accurate the forecast.

Let's say the product owner wants to complete 500 story points in the backlog. We know that the development team generally completes 50 story points per iteration. The product owner can reasonably assume the team will need 10 iterations (give or take) to complete the required work.

It's important to monitor how velocity evolves over time. New teams can expect to see an increase in velocity as the team optimizes relationships and the work process. Existing teams can track their velocity to ensure consistent performance over

time, and can confirm that a particular process change made improvements or not. A decrease in average velocity is usually a sign that some part of the team's development process has become inefficient and should be brought up at the next retrospective.



ANTI-PATTERNS TO WATCH FOR

When velocity is erratic over a long period of time, always revisit the team's estimation practices. During the team's retrospective, ask the following questions:

- Are there unforeseen development challenges we didn't account for when estimating this work? How can we better break down work to uncover some of these challenges?
- Is there outside business pressure pushing the team beyond its limits? Is adherence to development best practices suffering as a result?
- As a team, are we overzealous in forecasting for the sprint?

Each team’s velocity is unique. If team A has a velocity of 50 and team B has a velocity of 75, it doesn't mean that team B has higher throughput. Since each team's estimation culture is unique, their velocity will be as well. Resist the temptation to compare velocity across teams. Measure the level of effort and output of work based on each team's unique interpretation of story points.