
Product Backlog

Once features are prioritized based on techniques like MoSCoW or RICE, the next step is to **organize them within the product backlog**. This ensures that all stakeholders understand what needs to be done and why. The product backlog serves as **a central repository** for everything related to a single product, from detailed tasks ready for immediate action to high-level concepts for future exploration. In other words, it includes all the work items that contribute to **the development**, **improvement**, **maintenance**, or **support** of that product.

Now let's explore what the product backlog might contain. Backlog is **a dynamic tool, encompassing both short-term tasks and long-term strategic objectives.** Each company or team might structure their backlog differently, tailoring it to their unique processes, goals, and workflows. Despite these differences, there are certain elements that are commonly found in most backlogs, which we will explore in detail.

To ensure the backlog is actionable and efficient, it needs **a clear hierarchy**. Let's start by examining columns.

- ***Title***

The first column, Title, contains a concise name for the backlog item, allowing the team to identify it at a glance.

- ***Description***

Then, we have a column Description that contains a short but clear explanation of what the item entails and why it's important.

- ***Effort Estimate***

Now, we have an interesting one called Effort Estimate, which gives an approximate measure of the effort or time required to complete a task. In agile, effort is often expressed in **story points** instead of hours. Rather than relying on specific time estimates or deadlines, story points assess the relative complexity and size of tasks. These points are subjective and vary by team, but they typically follow **a relative scale**:

Story Points:

- **1 Story Point**: A simple task, quick to complete
- **3 Story Points**: Moderately complex, requiring some time or thought
- **5 Story Points**: A complex task involving multiple steps
- **8 or More Story Points**: Very large or uncertain tasks

1 Story Point is a simple task that is quick to complete, such as fixing a typo on a webpage. **3 Story Points** get moderately complex tasks, requiring some time or thought like adding sign-up form for account. **5 Story Points** represent a complex task that involves multiple steps such as adding a feature to upload and manage profile pictures within their account. **And 8 or More Story Points** describe very large or uncertain tasks, often requiring further breakdown. For example, developing a dashboard that displays real-time analytics for user activity across multiple accounts.

Remember that these rough estimates are **associated with specific sprints or development cycles**. The focus is on **tasks that can be completed within the sprint**, aligning with the team's immediate goals and priorities.

- ***Priority***

The next column Priority indicates the importance of the item relative to others in the backlog. Priority levels might be "High", "Medium", "Low" or use numerical rankings.

- ***Status***

Lastly, we have column Status which tracks the item's current state in the workflow.

Now that we've covered the key columns that structure the product backlog, let's shift our focus to the types of items it contains. To make this section practical, we'll also provide real-life examples for each type of backlog item.

Backlog Item	Title	Description	Effort estimate	Priority	Status
Epic	Lead Scoring Algorithm	Develop an algorithm to assign scores to leads based on activity and data	13	Low	Backlog
Feature	Third-Party Integration	Allows users to sync data with external tools	8	Medium	To-do
User story	Password Reset for Admins	As an admin, I want to reset passwords for users so that I can manage security efficiently	5	High	Ready for sprint
Task	Build Reset Password Form	Create the front-end for password reset	3	High	In progress
Bug	Sign-up Form issue	A sign-up form allows invalid email addresses	2	Critical	To-do
Technical debt	Refactor Payment Module	Improve code readability and scalability	13	Low	Backlog
Non-functional requirement	API Response Time	Maintain response time under 200ms	5	High	Read for sprint

- ***Epics***

First, we have Epics. These are **large, overarching goals that often represent significant pieces of functionality**. In a SaaS application, an epic might be something like "Lead Scoring Algorithm". These items are too broad to tackle at once, so they are broken down into smaller components over time. Epics provide the team with **a clear vision of long-term objectives.**

- ***Features***

Next, features are more detailed than epics but still **represent significant, specific functionality**. For example, a SaaS product feature might be "User Account Management" or "Third-Party Integration". Features focus on **delivering specific value to users** by addressing their needs or improving their experience.

- ***User Stories***

Then we have User stories. These are **concise descriptions of functionality written from the end-user's perspective**. They follow the format: "As a [user type], I want [functionality] so that [benefit]". For instance, "As an admin, I want to reset passwords for users so that I can manage security efficiently". These stories clarify the purpose behind each feature and guide the team's development efforts.

- ***Tasks***

Tasks are **the smallest units of work in the product backlog.** They break down user stories

or features **into actionable items that can be completed within a sprint**. An example might be "Create the front-end for password reset". Tasks **make it easier to track progress and estimate the time required for completion**.

- ***Bugs***

It's common to encounter bugs in any product. These are **unexpected errors or issues that disrupt the product's functionality.** Bugs can vary in severity, from minor glitches in the user interface to critical failures that affect core features. An example could be “A sign-up form allows invalid email addresses”. Including bugs in the backlog ensures they are visible to the team and **prioritized appropriately alongside new features**.

- ***Technical Debt***

Product Backlog also contains Technical debts. These include shortcuts or compromises made during development that need addressing later. For instance, a team might decide to use a quick workaround for a bug rather than implementing a robust solution or skip writing thorough tests to release a feature quickly. These decisions, while practical in the moment, **can accumulate into problems that slow down future development or require costly fixes.**

- ***Non-Functional Requirements***

And finally, we have non-functional requirements. These items address performance, scalability, security, and compliance aspects of the product. Including these ensures the product **meets technical and legal standards**.

Product backlogs are typically managed in digital tools designed for agile workflows. Popular options are softwares like Jira, Clickup or Trello.

Backlog Refinement Process

Once the backlog is created, it becomes a living document that evolves with the product. Teams regularly revisit the backlog to review and update it, a process often referred to as **grooming**. This backlog refinement ensures the document stays relevant, manageable, and aligned with business objectives. Let’s explore the key aspects of this process.

a) Weekly Review

Weekly Reviews:

- Outdated items are removed
- Priorities are updated
- Large items are broken down
- Details are added to upcoming items

Firstly, teams usually conduct weekly reviews. These are **dedicated sessions where the team prepares tasks for upcoming sprints**. During these sessions they remove items that are no longer relevant or viable. The team also reorders backlog items based on new information, such as customer feedback or technical constraints. This ensures that the most impactful work is tackled first. High-level tasks like epics or large features are broken down into smaller, actionable tasks or user stories. This makes it easier to plan and execute during sprints. The team also refines items that are likely to be addressed soon. For example, they add clear descriptions, effort estimates, and other necessary details to ensure the tasks are well-defined and ready for implementation.

As we can see, weekly reviews are truly essential for keeping the backlog actionable and ensuring the team is always prepared for the next sprint.

b) Stakeholder Input

Stakeholder Input:

- Customer Feedback: Users often highlight pain points or propose valuable enhancements through surveys, interviews or support tickets.
- Sales Team Insights: Sales representatives interact directly with prospects and can highlight features or improvements that would address objections or boost conversions.
- Technical Team Input: Developers and engineers contribute insights on feasibility and technical dependencies, ensuring the team focuses on tasks that can be realistically achieved.

Another critical part of backlog refinement is **gathering input from stakeholders**. Backlogs are not created or maintained in isolation. They require insights from various perspectives to ensure the product delivers maximum value. So, teams engage with key stakeholders to align it with user needs, business goals, and technical feasibility. Let's examine how different stakeholders contribute.

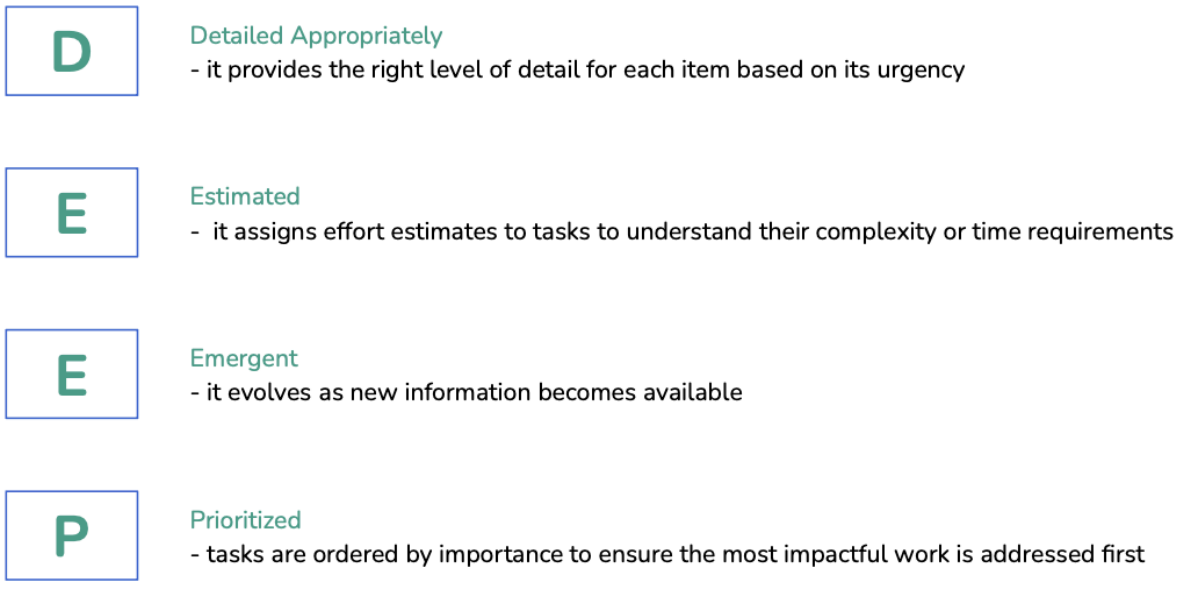
Firstly, teams can analyze customer feedback from surveys, interviews, or support tickets to

identify their pain points or requests. For example, if multiple users report challenges with navigation, a related feature or improvement is added to the backlog and prioritized accordingly. During grooming sessions, teams can revisit and re-prioritize features using frameworks like RICE or MoSCoW that we covered in the previous lecture. This ensures the backlog remains aligned with actual customer needs.

Another group of stakeholders is sales representatives, who frequently interact with prospects during calls or trials. These interactions offer valuable insights into customer needs. For example, prospects might request specific features or raise concerns about missing functionality or limitations. This helps the team identify improvements that address objections.

Developers and engineers in technical teams also provide valuable input. They assess whether a proposed feature or task can be built given the current system architecture, tools, and resources. Many backlog items rely on other tasks to be completed first. For example, before adding a new reporting feature, the underlying data collection mechanisms might need to be updated. Engineers map out these dependencies to ensure that work is sequenced logically. Developers also provide input on the complexity of each backlog item, often expressed in story points. This helps the team plan sprints and allocate resources effectively.

In the last part of this lecture, we'll explore what makes a backlog healthy and manageable. One widely recognized framework for evaluating backlog quality is the **DEEP** approach. A healthy backlog should be **Detailed Appropriately, Estimated, Emergent,** and **Prioritized**. Let's break down each of these qualities and explore what they mean in practice.



First, we have "detailed appropriately". This means providing **the right level of detail for**

each item based on its urgency. Items that the team plans to work on soon should have clear descriptions, or any other specifics to avoid confusion during implementation. For example, a near-term task like “Adding a password reset feature” would include details on design and functionality.. On the other hand, long-term ideas, such as “Exploring AI recommendations”, can remain high-level until they become more relevant.

The next characteristic is “Estimated”. Once items are sufficiently detailed, the next step is to **assign effort estimates**. This helps the team understand how complex or time-consuming a task might be. For instance, knowing that a feature like “User Profile Picture Upload” will take moderate effort might influence its placement in the sprint.

Then, the backlog should be “Emergent”. As we already know, the backlog is a living document that evolves as new information becomes available. Teams frequently make **grooming** based on changing priorities or customer feedback. This emergent quality ensures the backlog remains relevant and reflective of current goals.

And finally, prioritization ensures that the most important tasks are tackled first. For instance, resolving a critical bug affecting user logins would take precedence over implementing a minor enhancement. Clear prioritization ensures that the backlog drives meaningful progress.