User Stories:

-user stories are key components of agile software development.

-They are short, simple descriptions of a feature or functionality from the perspective of a user.

-used to capture requirements in an agile project.

-help the development team understand the needs and expectations of the users.

Here are some key characteristics of user stories:

1)User-centric: User stories focus on the needs of the user and what they want to achieve with the software.

2)Simple: User stories are short and simple descriptions of a feature or functionality.

3)Independent: User stories can stand on their own and do not rely on other user stories.

4)Negotiable: User stories are open to discussion and can be refined and modified based on feedback from stakeholders

5)Valuable: User stories provide value to the user and the business.

6)Estimable: User stories can be estimated in terms of time and effort required for implementation.

7)Testable: User stories can be tested to ensure they meet the needs of the user.

8)Prioritized: User stories are prioritized based on their importance to the user and the business goals.

9)Iterative: User stories are developed iteratively, allowing for feedback and changes throughout the development process

10)Consistent: User stories follow a consistent format, making them easy to understand and work with.

11)Contextual: User stories are written in a way that provides context to the development team, helping them understand the user’s needs and goals.

12)Acceptance criteria: User stories have clear and specific acceptance criteria that define when the story is considered “done” and ready for release.

13)Role-based: User stories are written from the perspective of a specific user role, helping to ensure that the development team is building features that are relevant and useful to that user.

14)Traceable: User stories are tracked and linked to specific features and functionality in the software, making it easy to trace back to the original user need.

-In agile software development, user stories are typically written on index cards or in a digital format, and are used to drive the development process.

\*\*\*Pattern of User Story\*\*\*\*

User stories are completely from the end-user perspective which follows the Role-Feature-Benefit pattern.

As a [ type of user ], I want [ an action ], so that [ some reason ]

\*\*\*For example:

As the project manager of a construction team, I want our team-messaging app to include file sharing and information update so that my team can collaborate and communicate with each other in real-time as a result the construction project development and completion will be fast.

\*\*\*What are agile user stories?

---A user story is the smallest unit of work in an agile framework. It’s an end goal, not a feature, expressed from the software user’s perspective.

---A user story is an informal, general explanation of a software feature written from the perspective of the end user or customer.

---The purpose of a user story is to articulate how a piece of work will deliver a particular value back to the customer.

Note that "customers" don't have to be external end users in the traditional sense,

they can also be internal customers or colleagues within your organization who depend on your team.

---Stories fit neatly into agile frameworks like scrum and kanban.

In scrum, user stories are added to sprints and “burned down” over the duration of the sprint.

Kanban teams pull user stories into their backlog and run them through their workflow.

It’s this work on user stories that help scrum teams get better at estimation and sprint planning, leading to more accurate forecasting and greater agility.

---User stories are also the building blocks of larger agile frameworks like epics and initiatives.

Epics are large work items broken down into a set of stories, and multiple epics comprise an initiative. These larger structures ensure that the day-to-day work of the development team (on stores) contributes to the organizational goals built into epics and initiatives.

INITIATIVES



Epic1 Epic2 Epic3

Story/task1 Story/task2

\*\*\*Why create user stories?\*\*\*

--For development teams new to agile, user stories sometimes seem like an added step.

Why not just break the big project (the epic) into a series of steps and get on with it?

But stories give the team important context and associate tasks with the value those tasks bring.

\*User stories serve a number of key benefits:

1)Stories keep the focus on the user: A to-do list keeps the team focused on tasks that need to be checked off, but a collection of stories keeps the team focused on solving problems for real users.

2)Stories enable collaboration: With the end goal defined, the team can work together to decide how best to serve the user and meet that goal.

3)Stories drive creative solutions: Stories encourage the team to think critically and creatively about how to best solve for an end goal.

4)Stories create momentum: With each passing story, the development team enjoys a small challenge and a small win, driving momentum.

\*\*\*Working with user stories\*\*\*

--Once a story has been written, it’s time to integrate it into your workflow. Generally a story is written by the product owner, product manager, or program manager and submitted for review.

\*\*How to write user stories?\*\*

Consider the following when writing user stories:

1)Definition of “done” — The story is generally “done” when the user can complete the outlined task, but make sure to define what that is.

2)Outline subtasks or tasks — Decide which specific steps need to be completed and who is responsible for each of them.

3)User personas — For whom? If there are multiple end users, consider making multiple stories.

4)Ordered Steps — Write a story for each step in a larger process.

5)Listen to feedback — Talk to your users and capture the problem or need in their words.

No need to guess at stories when you can source them from your customers.

6)Time — Time is a touchy subject.

Many development teams avoid discussions of time altogether, relying instead on their estimation frameworks.

Since stories should be completable in one sprint, stories that might take weeks or months to complete should be broken up into smaller stories or should be considered their own epic.

\*\*\*\*\*\*Acceptance criteria for User Stories\*\*

--a set of predefined requirements that must be met to mark a user story complete.

--Acceptance criteria are also sometimes called the “definition of done”

because they determine the scope and requirements that must be executed by developers to consider the user story finished.

\*\*What is acceptance criteria in user story with example?

The acceptance criteria are what should be done to solve their problem or achieve their goal.

In this way, the user story describes the “why” of the work, while the acceptance criteria describe the “what.” The “how” is decided by developers as they work through the sprint.

\*\*\*\*\*What are the 2 types of acceptance criteria?\*\*\*\*

There are two distinctive approaches to writing acceptance criteria:

1)scenario-based

2)and rule-based.

1)scenario-based

The acceptance criteria, when framed using Gherkin, follow a template with five main statements:

1. Scenario - the name for the behavior that will be described
2. Given - the beginning state of the scenario
3. When - specific action that the user makes
4. Then - the outcome of the action in “When”
5. And - used to continue any of three previous statements

2)and rule-based.

Sometimes, it’s challenging to fit acceptance criteria into the Given/When/Then structure. For instance, GWT would hardly be useful for the following circumstances:

* You’re working with user stories that describe system-level functionality that needs other [methods of quality assurance](https://www.altexsoft.com/whitepapers/quality-assurance-quality-control-and-testing-the-basics-of-software-quality-management/).
* The target audience for acceptance criteria doesn’t need precise details of the test scenarios.
* GWT scenarios don’t fit the described design and user experience constraints of a feature. Developers may miss some critical details.