User Stories

What is a User Story?

Represents a small piece of business value that a team can deliver on a sprint.

- The brief description of the need
- The conversations that happen during backlog and iteration planning
- The acceptance criteria that confirm the story's satisfactory completion

A well-formed story is:

Independent	We want to be able to develop in any sequence.
Negotiable	Avoid too much detail, keep them flexible so the team can adjust how much of the story to implement.
Valuable	Users or customers get some value from the story.
Estimatable	The team must be able to use them for planning.
Small	Large stories are harder to estimate and plan, by the time of the sprint planning, the story should be able to be designed, coded and tested within the sprint.
Testable	Document acceptance criteria, which leads to test cases.

Why Use User Stories?

- Keep expressing business value
- Avoid introducing detail too early that will prevent design options and avoid developers into one solution
- Avoid the appearance of false completeness and clarity
- Have small enough chunks that invite negotiation and movement in the backlog
- Leave the technical aspect to the developers.

How to Write a User Story?

As a <User Role>, I want to <Action/Function>, So I can <Value/Benefit>

Examples:

- As a Registered User, I want to login, So I can purchase an item.
- As an Admin, I want to run a report, So I can know which products are the most sold.

What size should a User Story be?

A user story should be small enough to be coded and tested within an sprint. When it is too large it's called an Epic.

Too broad

- As a Registered User I want to view products.

Too detailed

- As a Registered User I want to view a table of products with name, description, size, package type, items included, price, stock, sku.

Correct Story

- As a Registered User I want to view a list of products with its main fields.

Where to add detail to the User Story?

Use the Acceptance Criteria to provide the definition of Done for the story.

Example:

Must be able to login with either username or email

Password must be at least 8 characters long

Should provide a way to persist the session

On Success:

Must redirect to User's profile

Must show a success message

On Failure:

Must show an error message

Must show a captcha widget when failed attempts are > 3.

Steps to Write and Estimate User Stories

- Define User Roles and its main goals
- Define User Stories to accomplish each role main goals
- Talk through the requirements of the story
- Discuss and write down important things to implement the story
- Pocker Plan each story in the backlog with this checklist
    - Architecture: What we have to learn/have before we can start to work on this? (Design, technologies, …)
    - Coding: How much code we will need to write? Have we written similar code before?
    - Testing: Will any special setup (Mocking?) be required to test this story
    - Integration: Does this story have external dependencies?
    - Expertise: Does anyone have done something similar before?
    - Find some point of relative comparison
- Reach a consensus with the entire team
- Validate that the story estimates are internally consistent

It doesn’t matter if the estimates are correct or incorrect as long as they are **consistent**.

How do we estimate?

People are bad at absolute estimates but good at relative ones.

See the two objects below and tell me: How big is the first one and how big is the second (absolute estimate)?

Now answer how much bigger the second object is compared to the first one (relative estimate).

Usually, people are quicker and better with the second question. So it is quicker and therefore cheaper to use relative estimation techniques.

We do not have to re-estimate relative sizes with every change. If we estimated our items in days and something changes (e.g. we lose a programmer or get a new framework) we generally have to re-estimate the whole backlog. Let's say we need five minutes to estimate one item, our backlog consists of 100 items, and five people are involved in the estimation process: This means five person-days per estimation. With absolute numbers, we will need to re-invest five days whenever something changes. With relative estimation, we just need to invest this time once. Whenever something changes, the estimates remain the same and the only thing that changes is the velocity.

This velocity is measured and not estimated, so we don't need to spend a buck on getting it.

How do we measure the team velocity?

Velocity = Sum of the estimates of delivered stories per sprint.

In which unit are we estimating in?

The Story Point Scale: *1, 3, 5, 8, 13, 21, 40, 100*

If you ask a developer how much bigger something is compared to something else, he most often will tell you "twice as big". In Fibonacci there is no "twice as much" and thus people have to make a decision. Comparing the reasons leads to a discussion that clarifies many different aspects for the developers. This discussion constitutes the true value of the estimation process.

The team tells their Product Owner by showing "40" that "this item is too big and has to be refined" while "100" basically means "this will never fit into a Sprint, more refinement is needed".

The bigger the number is, the greater the gap between them. This helps to illustrate that estimates become less reliable with growing size. So Fibonacci numbers are excellent to facilitate discussions in the context of estimation.