Table of Contents

What is a User Story? - SSA

Work in progress

And “why do we use them?” And “what makes a 'good' user story?” And “how do we create a user story?” And “who creates a user story?”

Premise

User stories are a foundational aspect of an agile implementation.

The Basics

A “user story” captures the need for a new capability from the perspective of a “user”. User stories are normally written in the following format:

- “As a <user_role> I want some <capability> so I can get some <business_value_or_benefit>.”

Some examples:

- For a storefront application - “As a shopper, I want to save my shopping cart, so I can continue to shop later”.
- For an integration between two systems - “As a support analyst, I want to understand the latest status information from the development planning tool, so I can update my customers when they call.”

A user story describes:

- Who wants the capability, the role that person plays
- What they want (the capability)
- Why they want it (to get some business value or benefit)

It’s a valuable capability of the system that doesn’t exist yet.

The key part to understand is that this is not a task that we as implementers need to do, but rather the functionality that we need to provide to our end users. We already know that we will be demonstrating this capability to our end users at the end of a Sprint / iteration so if you find that you are not sure whether you have generated a user story or just a big task, ask yourself “What are we going to demonstrate to the end user?”. If you cannot figure out what you are going to demonstrate that the end

user would have good feedback for, then you probably have a task, not a user story.

It is useful to think about the capability starting this with a verb.

Why Do We Use This Format?

Understand Why We Want The Capability

Traditional requirements often come in the form of “I need this thing”. Sometimes they come as part of a spec. Other times they come as part of one-liner in an email – I don’t know how often this has been the case for me. The problem is that you’ve lost all the context. For example, if I understand why someone might want something, I might be able to offer up an alternative idea that would achieve the result at, for example, less cost. I can imagine a requirement such as the Amazon on-click approach – “as a shopper I want to save my shopping cart so I can continue to shop later”. There is a solution already implied in this “save the shopping cart” sounds like a button to me with “save” written on it. But then because I understand why someone wants this, perhaps I can say “actually we don’t need a save – we can just do a recall of the shopping cart based on the IP address – would that work for you?” And that’s why I now accidentally buy a bunch of things in a shopping cart that I didn’t really want …

Understand Who Wants The Capability

As the name suggests, an important part of the “user story” is the “user”. Why is this important? A user story describes how a customer or user employs the product; it is written from the user’s perspective in what some people call the “user’s voice”.

Computer requirements specifications are notorious in that they clump all users together. The joke is that there are only two industries that call their customers “users” – ours and the illegal drug industry – and this fact probably explains our lack of understanding of who we are doing the work for. Think about it. If you are writing code, or testing a capability and in your head you are assuming that a “database administrator” is going to be the end user of the system then you will make a bunch of assumptions as you do this implementation (for example, level of handholding required, type of error messages presented, command line interface vs GUI, etc) than if you assume the end user is your wayward brother-in-law.

If you don’t know who the users and customers are and why they would want to use the product, then you probably should not write any user stories. Carry out the necessary research. If you do not, there are a couple of problems that can occur:

- The team runs the risk of implementing a story based on their perception of the user. Given that they do not know who the user is, the picture they have in their mind will probably look like the developer themselves. There are literally hundreds of decisions being made by the technical team as they implement the capability. If the user they have in mind does not match the characteristics

of the real end user there is a high change that you will develop the wrong thing.
- If you are defining a user story and have no view of the actual user, you run the risk of writing speculative stories that are based on beliefs and ideas.

*** Personas** *

The 3 Cs - Card, Conversation, Confirmation

When people think about a “user story” the “as a, I want, so that” is quite literally how they think about it. Some people refer to this format as the “User's Voice” format. But a user story is more than that.

It is a placeholder, a reminder that we need to talk about the requirement before we start working on it. So the main purpose of a user story is to have a conversation.

In fact a “user story” really is about the 3 Cs:

- Card: In the early days of user stories, a user story was written on to an 5×3 index card. The card represented the requirement in all discussions. Clearly you cannot put a lot of information on a 5×3 card. And that was the point. Since there was not much information here, we needed to talk to each other about the requirement so we can have a common understanding of the what we are expected to do. This leads us to the second C …
- Conversation: This is how we flesh out the details of the requirement. We flesh out the details “just in time” as we are just about to begin working on the item. And through this discussion we begin to understand the boundaries of the requirement, to answer the question “how do we know when this work is considered “finished” from the perspective of the “need”. This leads us to the third C …
- Confirmation: The confirmation is the “acceptance criteria” or “Acceptance Criteria (or Conditions of Satisfaction)” for a story. The acceptance criteria basically answers the question “how will we know we have completed this functionality for our stakeholders” and forms the basis for determining what tests we need to run.

The key point is to have the conversation.

This does not mean that we don't capture information about the user story as we talk about it. We might capture links to documentation, reference information, all kinds of things as we talk about the work. But the idea is to iterate toward increased understanding based on the conversations, leveraging everyone's input, while making sure that everyone has a common understanding of the need.

Why Do We Use User Stories?

Lets think a moment why do user stories. In the past, development teams were required to develop to “specifications”. These would often start off as high level requirements, which would be progressively detailed as part of the “planning” phase of the software development project, until they were handed over to the development team for implementation and the testing team for validation. Often reams of

documentation would be produced on the assumption that this will improve the communication of intent.

There are lots of problems with this approach, but three problems stand out in particular:

- By the time the specific got to the team doing the work it would it would be so detailed a spec that it really was presented as a pre-packaged solution to the problem. So all that was left was to code and test the work. In other words, rather than treating the folks who implement the work as intelligent, contributing people they were treated as “code monkeys”. This lead to other problems …
- Programmers were treated as “code monkeys”; testers as “test monkeys”. When things went wrong the response would often be that they “did not follow the spec”. Here’s the problem with this thinking. Since specifications are written in English, they are open to interpretation. And sadly there is no amount of written words you can use to improve the discussion. There is a whole area of study at College devoted to the interpretation and re-interpretation of the same set of words – English literature. If English were precise, we would not have this. But it is not precise. Not only is it an ambiguous language (all human languages are) but also that there are people involved in the interpreting the words and they bring their own experience to the party.
- When (not if) things went wrong in the implementation of the capability, it was easy to assume that there was a problem in the requirements specification and planning process. As said, people would assume the way to address this problem was to increase the quantity of documentation. Have you ever played “Telegraph” or “Chinese Whispers”? If yes, then you know how successful this approach to communication can be.

Now while all this is true, these problems are not as big as the basic business problem this process creates:

*** Time to market** * Stale requirements Inability to respond quickly to changing positions Lack of feedback

How Do We Know We Have a Good User Story? The INVEST Criteria

How do we know if we have a “good” User Story? People use the mnemonic to make sure they have good Stories to work with – INVEST. INVEST stands for:

- **I**ndependant: Work item is modular and can de delivered with no (or at least manageable) dependencies.
- **N**egotiable: A basic solution is articulated, with room for options.
- **V**aluable: The value is clearly articulated. This is defined in the “so that” part of the User Story.
- **E**stimable: Provides just enough information so it can be sized compared to other work (relative sizing).
- **S**mall: Small enough to be completed in 1 Iteration (Sprint). Some call this attribute “sized to fit”.
- **T**estable: Acceptance criteria has been developed and are understood by the Team.

The initial draft of a Story will not have all these characteristics. Rather the mnemonic guides the

discussion to improve our understanding of the Story. Many teams establish a Definition of Ready (DoR) crtieria for a Story. In other words we are “READY” when the Story has the INVEST characteristics. Then as Story bubbles up the Backlog and it becomes a candidate for work, they refine the Story so that it has these characteristics helping to increase overall understanding.

→ Read more...

The “User Voice” Format of a User Story

In most cases teams write User Stories in the “user voice” format - the “As a <user role> I want <capability> so I get <value>” format. We do this is to ensure we are focusing on the business value to be delivered and it is clear when communicating with our stakeholders (by speaking in a common, business language.)

If you can determine a way to express the need in a user voice format then you should make the effort to do so. This will help us deliver real value to the business and increase the transparency of the work we do.

For example, perhaps we need to upgrade to a new version of an underlying database, as we expect it to improve the performance of the application. Instead of writing this as “Upgrade the database” we should say “As an administrator, I would like to have increased performance, so that I receive less support tickets as a result of perceived performance problems.” Not only will this increase transparency, but the team might come up with other ideas on how to improve the performance of the application.

Are All Stories About End User Value?

As much as possible we would like to express work that we do in terms of the benefit that we provide to our customers and there should be something that our end users can see at a demonstration and provide feedback.

But there are a number of cases where this may not make sense. There are really two basic types of stories – those that are aimed at describing the end user requirement – user stories - and those that are aimed at laying down the groundwork so we can work on user stories in the future. The second class of stories are called “enablers” or “technical” stories.

There are basically 3 types of enablers. The first type of enabler involves infrastructure aimed at improving how we implement our work. This is the development of things like build, testing and deployment frameworks, that lead to faster feedback and reduce errors and so lead to a better development process.

A second type of enabler that allows us to develop a new capability faster (or at all). In SAFe these are called “architectural enablers”. But the ideas is follow what Kent Beck said about development (although

he was talking about the practice of refactoring in XP) that when you implement something the first step is to “make the change easy, then make the easy change”.

The final type is when we “explore” what is needed – it enables use to build the right thing by first finding out answers to questions we have.

One word of caution. In software, there is no such thing as a “sure thing”. There are always unknowns. Sometimes they are big (we cannot get started at all) but most of the time they are just things we can figure out as we do work. I’ve seen some teams decided that everything is an “enabler” or a “spike” simply because they are not sure of everything. We need to be realistic about what kind of work we have.

And finally sometimes it does not make sense to turn all work into a “story” format. Defects are a good example of this. There is often no value in trying to force a bug into the story format. What are you going to say - “As a shopper I would like the application to not crash when I …” What matters about a defect is the conditions where it occurred. Since there is no additional value in turning it into this format, I would not bother unless fixing the defect resulted in some kind of change from the end user perspective that it would be good to feedback from.

Not Everything Should Be Transformed to the User Voice

However, not all work benefits from pushing it into a “user voice” format as sometimes there is no additional value in transforming the need.

General Case

The general documentation on User Stories has examples where it is hard to see the additional value that transforming to the “user voice” would have:

- Refactoring work: “Update MYSQL replication to latest release”
- Infrastructure: “Set up the new build server”
- Defects: “Fix the bug that makes the system crash”

While I am a fan in making sure we don't just do things “because”, there is sometimes no additional value in transforming these into “user voice” format in most cases. But please be careful here. Transparency is one of the basic values of agile. You should not use these more technical requirements to hide work. You should make the effort to have people understand why these things need to be done and where possible explain the need from an end user value perspective. For example, perhaps a “new build server” means that we will be able to do build every day instead of every week which means that we will have less errors in our builds, resulting in faster feedback and less rework time. All of this should be of interest to a business.

Special Cases

In addition, there are a number of specific situations where it would be wasteful if we converted the need into the “user voice” format.

Third Party Involvement

When IT organizations have a strategy to “use third party solutions where possible”, agile teams often do some work, then hand it to a third party. Once the third party does their work it is returned for validation of the results we see from the third party. If we tried to convert this work to the “user’s voice” it would cause confusion because the business value is only delivered when the final step is delivered. Separate Stories should be written for the work up to the handover and the work after the third party has returned the results – in effect these are big tasks.

Note that it is important that we track the impact that using third parties has on our work both to identify problem areas, to improve how we work with third parties, and to better forecast our work when it involves third parties. When you have this kind of “split” work, as you complete your work and hand it over to the third party, track the amount of time that it takes for the third party to respond. You will use this data to understand how long (cycle time) we are waiting for third parties so we can understand where we have issues that need to be addressed. See Why Should We Work Harder to Eliminate the Effect of Dependencies? for more information.

In-flight Work

Similarly, when we started working in the agile approach, much of the work is “in-flight” with, for example, only testing remaining. Again there is no value to be gained by putting this work into a “user voice” format. The Story should just talk about the work to be completed. Again each story is effectively a “big task”.

How To Write These “Big Tasks”

If you find you have to write a “big task” then you need to work on the name of the story so the result is as clear as possible. This improves communication with the stakeholders and focusses us on the result of the work.

Generally, this means starting with a verb summarizing the result of the work. For example:

1. “Resolve member age calculation defect (Incident # 123)”
2. “Complete regression testing for Release 28.3”

What Don't We Want To See?

It is hard to break habits built over the years. It is easy to slip back into traditional, non-agile ways of thinking. This is especially true when you are trying out a new practice. When people start working on user stories you will often see the same problems:

- Understand the real need. If the team and Product Owner lack the appropriate domain knowledge then find someone who can help – the Product Manager, a Subject Matter Expert, the End User, whoever. It makes no sense to start working on something without a base level of knowledge. Observing users in action and then asking follow-up questions is a great way to immerse yourself in the domain.
- Watch for stories that are defined in terms of the architectural layers of the product. Often teams create stories that reflect the architectural layers of the product (a story that says “deliver data access”, another that says “deliver business objects”) or focused on splitting stories as sequential components (where the components are systems). Instead try to figure out “who wants the capability” and “why”. In particular, ask yourself “what are we going to demonstrate to our stakeholders?” If you are asking people to provide feedback on the new schema you have developed then you probably have a problem. Remember, a good story is a slice through the architectural layers.
- Watch for Stories that are too big. Ideally it should be possible to complete a story (i.e., “done”) within 2-3 days. Remember the fundamental principle - working in small batches provides huge benefits through faster feedback, time to market, increase in quality, and so on.
- Watch for stories that are just big tasks. Many stories are capturing a set of tasks rather than describe desired capabilities or outcomes. User Stories should be in business domain language, not technical language. Again, focus on the “what”, not “how”. (There are some exceptions as noted above but these should be the exception, not the rule.)
- Related to this idea, in particular do not pretend to write stories that just mimic the old waterfall implementation approach where, for example, analysis is done in one Sprint, development in the next, and certification in the next. If you have this kind of approach to implementation, you really are not concentrating on the delivery of business value and, in most cases, you've just replaced the traditional development process with a couple of agile terms, but are not really using an agile approach at all.
- Use Spikes when you have uncertainty. Uncertainty can take many forms – requirements uncertainty, design uncertainty – or could also just be the need for some quick and dirty code or a quick test to try something out. Make sure the acceptance criteria are written so the result of the Spike is clear. For example, for something dealing with requirements uncertainty, the acceptance criteria might specify that it is done when we have a new set of user stories in READY state that can be worked in future Sprints.
- Watch for stories that have too many acceptance criteria. Remember the acceptance criteria is not the same thing as a test plan. You don’t have to document everything. If you find that you have too many acceptance criteria, you might actually be dealing with a very large story. If this is the case, then you can use the acceptance criteria to split it into smaller User Stories. This works well because acceptance criteria is written from a business perspective and so it means that it is a natural split for a User Story.
- Remember the purpose of a Story is a reminder to have a conversation. Make sure that teams have a conversation on every story and that everyone understands the acceptance criteria before you

start doing any work.

More information on how to split User Stories can be found at How Do We Split a User Story?

FAQ

How Do We Get More Training on User Stories?

User Stories, though simple in concept, are hard to do. The reason is simple - we are not used to thinking in terms of “delivering thin slices of end user value” and so it means we have to retrain our mind from the way we thought about the problem to the approach we want to take now (see The Backwards Bicycle to understand why this is so hard.)

What this means is that the best approach to learn how to do User Stories is the practice writing them. And it helps if you practice under the watchful eye of someone who can help steer the conversation, someone who has done it before. If you don't have that person, start asking the questions you see above, starting with “When we have finished this work what will we demonstrate to our end user / customer and what kind of feedback do we expect to get?” If you cannot answer this question, or the answer is “we will show the schema” then in most cases you have a big task, not a user story.

What Is Typically Captured as a "Ready" User Story on the Story Card?

The way to think about the information on a card is that, while you do not list out all the details you have, you use the card to capture the basics, to act as a reminder and to provide information on where to find more details if required. One way to think about the level of information is to think about an old library system to find books - an index card would allow you to find the information you need. And don't forget, I User Story is a “reminder to have a conversation”.

The following sample information follows the expectations from established as a result of following the the INVEST criteria, assuming you are capturing the results in a work management system (e.g. Jira, VersionOne, Rally, etc.):

- Title: 3-10 word descriptive title that clearly describes the outcome; make it meaningful. Experience shows that cards (and tools) get very “busy” if you use the user voice format of the story to reference it. After spending time with the story, people will naturally use short cuts to reference the information. This is the short cut and is used to reference the requirement in discussions and meetings. Titles work best when it completes the statement:
    - “I wish I had…” or

    - “I wish the system would…”Has a short, 3-10 word descriptive title that clearly describes the outcome; make it meaningful
- User Voice: Has a “user voice” form of the requirement (“As a <role> I want <capability> so that <I get this value>”)
- Acceptance Criteria: Has an acceptance criteria that helps us understand when we are done. Where possible these should be defined with examples (i.e. mostly written in Gherkin “Given <system is in this state> when <I take this action> then <I expect to see this result>”)
- Type: The issue type (story, defect, etc.) is correct for the work
- Estimate: Has an estimate based on completing to Definition of Done, and used by the Team to understand how big the piece of work is so they can plan appropriately. For Stories that are about to be worked on the estimate small enough to fit into an Iteration (Sprint) (estimate is typically sized below some threshold that means something to the Team e.g. 8's and below will fit into an Iteration)
- Dependencies: Have been identified and priority aligns with Iteration (Sprint) timing
- Prioritized: Items “expected” for the upcoming Iteration (Sprint) are prioritized appropriately, including improvement items from the Retrospective. In other words they are at the top of the Team (Product) Backlog

If you are doing this with physical 5×3“ index cards, the back of the card will typically have the acceptance criteria and other notes that come up during the discussion, while the front will have the title, description, estimates, etc.

Why Don’t We Use Written English to Communicate Requirements?

A base assumption of Agile is that understanding of requirements is best achieved by:

1. Going to the “gemba”: In other words, actually seeing what people are doing
2. Talking about the need: In other words, asking questions and working through examples and assumptions
3. Getting everyone involved right at the beginning - product management, architects, testers, developers, and designers.

In particular, we try to avoid using “written” English to communicate requirements, because written English is open to different interpretations. Consider the sentence “Mary had a little lamb” and think about all the meanings this could have based on where you put the emphasis on the sentence:

- Mary had a little lamb - it was Mary's lamb, not John's
- Mary had a little lamb - she doesn't have it any longer
- Mary had a little lamb - she only had one lamb; other people had more
- Mary had a little lamb - it really was surprisingly small
- Mary had a little lamb - she didn't have the curried chicken like everyone else

You can see a shift in your emphasis while reading will potentially change the whole meaning of the

sentence. Further, more words will often add to the problem.

And finally traditionally there is usually a series of documents that are used to produce different levels of documentation before the Team gets to work on them. Requirement documents lead to design documents and detailed design documents, which are then reinterpreted into testing documents and so on. Now we are compounding misunderstanding on misunderstanding. Is it any surprise that we might end up producing something that has no use to the person who asked for the capability?

How Do We Write User Stories When We Cannot Get All Development and Testing Done in One Sprint?

Work in progress

What we really want to avoid is water falling Sprints, where we dev in one sprint, and then test in the next. The reason user stories don't work well when you have this happen is that the purpose of a user story is to help deliver value (user's perspective). If you split a story based on tasks then you are not delivering business value. The result is that you are increasing risk (as you don't have good understanding of the true status of development) and are typically slowing things down (large batches of work trying to go through the system instead of small batches) and lower quality.

When teams tell me this my first question is "have we really exhausted all avenues for doing dev / test in sprint to product running tested features". Development often has this idea that we have to complete something for release and that takes Sprint to develop. Ask "what can I get in one week if we assume that not all the fields (for example) that will eventually be required would appear on the page."

This goes to idea that agile is about "(technical) implementation is a separate event (business decision) to release" or "deliver on a cadence; release on demand".

Or ask "what will you implement first?" and use this discussion to see if a "value oriented" breakdown appears. Mostly this kind of thing is caused by a develop holding on too long to the code, only wanting to hand it over to cert when everything is done.

In particular we need to establish a team process whereby a developer, for example, checks in code at the end of every day so that the tester can work on something. This will make the overall process less of a handover, and allow the team to work together better.

And finally, what is the approach to test automation? The dev sprint / test sprint thinking usually results when there is manual testing everywhere, rather than an approach to automation. Above two questions will help drive behavior, but if you have no test automation then you are going to be doing suboptimal things for a long time.

These are just ideas, based on things I've seen before. Let me say that there are hundreds of other things that could be happening, so I really don't know if this helps at all:-) Bottom line is that the aim is to get user stories being delivered every couple of days during a sprint, with the first one being completed say 3 days into a sprint. If you don't have this your user stories aren't small enough. If you are not there it is a symptom of a problem in how you execute.

Consultant, Tools, UserStory, Stories, FAQ, SubjectSpecificArticle, ToDo, FirstSprint